1983

HILDA: The Flexible Design and Implementation of a Database Machine Executive

Paul Anthony Fishwick
College of William & Mary - Arts & Sciences

HILDA : THE FLEXIBLE DESIGN AND IMPLEMENTATION OF A DATABASE MACHINE EXECUTIVE

A Thesis

Presented to

The Faculty of the Department of Mathematics and Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Master of Science

---

by

Paul A. Fishwick

1983

APPROVAL SHEET

This thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science

Author

Approved, July 1983

Stefan Feyock

Robert Noonan

Kathy Samms

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS
LIST OF FIGURES
LIST OF APPENDICES
ABSTRACT
INTRODUCTION. 2

PART I. A BASIS FOR THE THESIS RESEARCH

CHAPTER I. DATA BASE MANAGEMENT SYSTEMS 3
CHAPTER II. DATABASE MACHINES. 10
CHAPTER III. THE INTEL DATA BASE PROCESSOR. 13
CHAPTER IV. THE DEVELOPMENT OF HILDA 24

PART II. THE DESIGN AND IMPLEMENTATION OF HILDA

CHAPTER V. LAYER 1: THE DATA COMMUNICATIONS PROTOCOL. . . .28
CHAPTER VI. LAYER 2: A SEMANTICS SPECIFICATION PACKAGE . . .43
CHAPTER VII. LAYER 3: A VIEW-ORIENTED QUERY LANGUAGE. 51
CHAPTER VIII.CONCLUDING REMARKS 67

GLOSSARY OF ACRONYMS. .73
APPENDIX A. SPP SOURCE
APPENDIX B. A SAMPLE TRANSMISSION TRACE
APPENDIX C. DBPSSP SOURCE
APPENDIX D. DBPSSP EXAMPLES
APPENDIX E. DBPQL CONCEPTUAL PROCEDURES
APPENDIX F. DBPQL GRAMMAR FILE
APPENDIX G. A SAMPLE DBPQL USER DIALOG
FIGURES
REFERENCES

ACKNOWLEDGEMENTS

The author would like to acknowledge the help of the thesis review committee for providing suggestions and constructive criticisms concerning the earlier drafts of the thesis. The comments and questions of Stefan Feyock, Bob Noonan, and Kathy Samms during the thesis defense were very helpful. In particular, the advice and numerous suggestions of my thesis advisor(Stefan Feyock) are much appreciated. His advice and numerous suggestions have made this thesis a more comprehensible one. The author would also like to thank the IPAD office at NASA Langley Research Center for providing a work order so that the thesis research could be successfully completed. Specifically, discussions with Floyd Shipman, Timothy Rau, and Bob Fulton are appreciated.

Finally the author acknowledges the support of his wife, Martha during the long hours. Her continuous support has helped to make this thesis possible.

LIST OF FIGURES

Figure 1 - The Physical DBP Environment

Figure 2 - Relational Commands for Manipulating Views

Figure 3 - HILDA : A general flow chart

Figure 4 - HILDA : A sample query

Figure 5 - Layers within HILDA and SPP

Figure 6 - General form for Host-DBP interaction

Figure 7 - VAX Asynchronous Communications Parameters

Figure 8 - Threaded Data Structure of SPP

Figure 9 - Request Module Form

Figure 10 - A sample assembly for "REMARK"

LIST OF APPENDICES

APPENDIX A - SPP SOURCE

APPENDIX B - A SAMPLE TRANSMISSION TRACE

APPENDIX C - DBPSSP SOURCE

APPENDIX D - DBPSSP EXAMPLES

APPENDIX E - DBPQL GRAMMAR FILE

APPENDIX F - A SAMPLE DBPQL USER DIALOG

APPENDIX G - DBPQL CONCEPTUAL PROCEDURES

ABSTRACT

The design and implementation of a three-layer executive is described for the Intel Data Base Processor. The executive is termed "HILDA" which stands for High Level Data Abstraction System. The layered components of the executive include an asynchronous error-correcting protocol, a semantics specification package, and a high-level interactive query language. Relevant source listings and interactive results are included in the appendices.

HILDA : THE FLEXIBLE DESIGN AND IMPLEMENTATION OF A DATABASE MACHINE EXECUTIVE

INTRODUCTION

The management of large quantities of scientific data presents special problems. There is a great need for the engineering analyst to be able to easily and efficiently access vast amounts of data associated with engineering computer programs. A project called IPAD(Integrated Programs for Aerospace Vehicle Design)[1,2,3] was created at NASA Langley Research Center so that research could be initiated to address the problem of engineering data base management.

Due to a joint IPAD/ICASE(Institute for Computer Applications in Engineering) effort, an Intel Data Base Processor(DBP) was obtained to aid in the research associated with scientific and engineering data management. This thesis describes the design and development of a flexible set of tools which allow the scientific user to efficiently use the DBP.

PART I

A BASIS FOR THE THESIS RESEARCH

CHAPTER I

DATA BASE MANAGEMENT SYSTEMS

The Engineering Data Handling Problem

With the vast amounts of scientific analysis data being manipulated in the engineering work environment, a need exists for adequately managing the data. In addition to the problems associated with managing large quantities of data, there also exist integration problems when engineers are required to move information from one computer program to another in a reliable manner. Over the course of a project several different application programs will have been used since it is rare to find one program which will satisfy the needs of every engineer. The data from these programs must be efficiently integrated and managed to insure the success of the project.

There are currently many programs oriented towards engineering applications which attempt to solve this data management problem by directly using sequential or random files. This primitive type of data management is not sufficient when large quantities of data are accessed, since standard disk files do not represent a flexible method of accessing data. There also exists an efficiency problem with respect to data access times when using sequential or random access files. In the IPAD project, it was seen that a new type of data management for scientific data was necessary so that the engineer could have a flexible and easy-to-use mechanism for working with design and analysis data.

Current Data Base Management Systems

Currently available data base management technology was surveyed to obtain an initial solution to the data management problem. The three major data models were carefully reviewed to determine the most appropriate given the engineering work environment. It is assumed at this point that the reader has a basic understanding of certain database principles as presented in references 4 and 5. The three data models are briefly described :

1. Hierarchical - Involves storing a group of data records in a hierarchical(or tree-like) fashion. Data sets formed using this data model can be referred to as "owner" or "member" sets since the data is organized into trees.

2. Network - This data model revolves around the network data structure. Any item may be linked with any other item in a database using explicit links(or pointers). A major difference between the network data model and the hierarchical data model is that a member record may have more than one owner in the network model.

3. Relational - This data model is represented by a collection of two-dimensional tables called relations. Each relation is composed of a set of named columns(attributes) and rows(tuples). Each item is declared to be of a certain data type such as integer, floating point, or ASCII.

The relational model was chosen as the most appropriate in the scientific/engineering environment for the following reasons:

1. Most scientists and engineers are familiar with data that is presented in tabular form. Mathematics and engineering texts often contain appendices which include tabular data.

2. The relational model does not burden the user with the task of having to keep track of explicit links which connect the sets of data. With large amounts of engineering data, the number of pointers in a hierarchical or network data base can be staggering. The relational data model promotes a pointer-free method of manipulating data. Relationships among the data items may be easily formed using relational algebra upon the tables. Examples of this algebra will be included throughout the thesis.

3. The algebra that is used to initiate queries for a relational data base is very user-oriented. In many cases, the written query syntax resembles an English language command as it would be verbalized. This English command syntax promotes a short "learning curve" when the engineer needs to access data areas.

A relational data base system called RIM(Relational Information Manager)[6] was created within the IPAD project to address the data management problem. RIM permits the engineer to easily manipulate data with English-like commands.

RIM is specifically oriented towards the engineering environment. In addition to containing the usual primitive data types such as integer and ASCII, RIM contains double precision, vector and matrix data types which are more common in the engineering environment. Tolerances, which are fairly common in the scientific world(and not so much in the business world), are also handled in RIM. One may specify retrieval of data items based, not on an exact floating point value, but on a reasonable approximation based on tolerance.

It was decided that there were many more issues to be investigated concerning scientific data management that go beyond the RIM development. Specifically, when large quantities of data are stored in a RIM database, a performance problem arises. Selecting and retrieving certain data becomes cumbersome due to a time lag brought on by the overhead of RIM on the host VAX 11/780 computer. For example, a typical query performed on a relation with five thousand tuples may cause a thirty second wait depending on the system load. Also, there are certain facets of RIM which needed improving :

1. RIM does not contain views("views" are described in chapter 3). When a new relation is formed through relational algebra, this relation does not contain any logical connection to the underlying relation from which it was formed. This means, for instance, that when data is stored into the new relation, this change is not reflected in the underlying relation.

2. RIM has a FORTRAN interface so that data manipulation may be controlled from within a user's application program. The interface operates differently from the easy-to-use interactive command language. One uses

the program interface by calling a set of subroutines that contain many different arguments (which refer to relation and attribute names, attribute values, etc.). The interactive command language, on the other hand, requires a more English-like command specification. It would be beneficial if the program interface mode operated more similarly to the interactive mode in terms of command syntax. Specifically, it would be nice if the program interface supported a single subroutine whose single argument would contain the command text that would normally be entered interactively to produce a given result.

3. RIM does not contain the mechanism that allows the sophisticated user to build network and hierarchical data structures based on the relations. It is possible to have a relational system which contains performance-oriented pointers and hash tables that are present in the schema yet transparent to the casual user who wishes only to see the pointer-free tabular output. RIM does not contain many of these performance capabilities and therefore performance problems arise when large amounts of engineering data are involved.

It was decided that since performance was such a critical issue in scientific data management, the use of database hardware(i.e. data base machines) would provide an interesting avenue of research. Data base machines will be discussed in the next chapter.

CHAPTER II

DATA BASE MACHINES

Data Base Machines are hardware devices that perform data base functions normally associated with software data base management systems. Typically the data base machine is physically connected to other computer systems in a network fashion or as a back-end data base engine to a host computer.

In a recent survey paper, Maryanski[7] points out some of the benefits derived from data base machines :

1. Performance - The data base machine improves the throughput of a loaded host computer since the data base management functionality is removed from the host and delegated to the data base machine. The data base machine operates concurrently with the host computer to achieve optimal performance.

2. Cost - The attachment of a data base machine to an existing host computer is often cheaper than the purchase of a larger host computer(such as a mainframe system) for handling data base chores. As data base machines proliferate, the cost of the machines will gradually fall.

3. Security - Since the data base hardware is physically separate from the host computer, the programmer is forced to perform all data access through the channel connecting the host and data base machine. This channeling supports a structured, secure means of doing data base management. When the data base functionality is all on the host computer, it is often possible to bypass the normal conventions in accessing data base files (either inadvertently or intentionally).

Over the past nine years, a number of data base machine prototypes have been built. The first prototype, named XDMS, was constructed at Bell Laboratories by Canaday et al.[8] in 1974. The purpose of the XDMS research was to demonstrate the feasibility of the back-end data base machine concept. The term "back-end" refers to the fact that the data base machine is acting as back-end to the front-end host computer which gives all the orders. The back-end data base machine simply receives orders and responds to them. It does not initiate orders.

Presently, there are very few commercially available data base machines. The most notable are the Britton-Lee IDM(Intelligent Database Machine) and the Intel DBP(Data Base Processor). Both machines are similarly priced and perform in a similar manner. Britton-Lee includes a query language and communications protocol to allow the user to talk to the IDM. Intel is planning on supplying a query capability and communications software in the near future. Both the IDM and DBP operate by being passed a sequence of low-level data base management operations from the host computer.

In an agreement involving Intel Corporation, ICASE(Institute for Computer Applications in Science and Engineering) and NASA Langley Research Center, an Intel DBP was donated so that the effect of the data base machine in the engineering work environment could be adequately studied.

CHAPTER III

THE INTEL DATA BASE PROCESSOR

This chapter presents an overview of the Intel Data Base Processor(DBP). To appreciate the remainder of this thesis, it is important that the reader gain an understanding of the purpose and capabilities of the DBP.

The Intel DBP acts as an "intelligent mass storage controller" whose primary purpose is to relieve the host computer of time-consuming data base chores. On the majority of current computer systems the mass storage controller that controls the disk units contains little logic or functionality. Most disk controllers, for instance, contain only the capability to read/write tracks and sectors. An "intelligent" mass storage controller, on the other hand, provides a wide range of functionality to users and programs which access it, such as the capability to manage databases and manipulate entities within the databases. The entity types that the DBP handles is the subject of a subsequent section.

The DBP may be used in a variety of environments. It can be used as a back-end device connected to a host computer which drives it or as a server acting as the data base manager node in a local area network. For purposes of the work described in this thesis, the DBP was used as a back-end data base machine which was directly connected to a VAX 11/780 minicomputer. This arrangement is portrayed in figure 1.

In the back-end environment, the DBP is connected to a host computer which gives orders to the DBP. Each order is in the form of a contiguous set of commands called a "request module". A program on the host sends a request module to the DBP and the DBP sends back one or more "response modules". There are many different types of commands that one may include within a request module to be delivered to the DBP. Some of these types are identified below. Note that some of the key terms used within the command overviews, such as the database entities "session", "file", and "view" will be described in the following sections.

1. Administration Commands : Allow the user to create, delete, and make modifications to databases, files, and views. Typical commands include the capability to define file schemas (the organization of the file), integrity constraints (keeping tabs on the consistency of the data), and views(windows created

from existing files). These commands do not manipulate the data contained within the files, but rather the status and structure associated with files and other major DBP entities.

2. Resource Control Commands : Provide access to the DBP entities. For instance, views may be either attached or detached (freed) from the users current application session. Locks may be placed at different levels on certain entities. The creation of these locks and the keys with which to open the locks are part of the resource control mechanism.

3. Performance Enhancement Commands: Allow the analyst to enhance the performance of the DBP using certain techniques such as pointers, hash tables for a given item, and indexes.

4. Data Manipulation Commands : Perform manipulation of the data stored within a given database file. Manipulation may involve fetching, storing, or modifying data.

5. Flow Control Commands : Allows control structures to be included within the command block. Conditional execution of certain DBP requests are facilitated with "IF...ELSE...ENDIF". Iterative execution is accomplished by setting up "LOOPS".

The DBP executes request modules using "sessions". A session is defined to be a set of host-resident application programs which are functionally related. Programs which involve the manipulation of an engineering drawing might be considered to be a session. A set of programs which keep track of the inventory for the drawings would be in another session. There are two types of sessions: control and application. Control sessions are used for DBP administrative purposes such as creating one or more application sessions. Application sessions are used for the majority of DBP commands including commands which define and manipulate data base entities.

When one thinks of the primary entities which are stored on the disks attached to the DBP, one thinks of three particular entities : databases, files, and views. A database is a collection of files. In a scientific environment, a database might contain all of the data relevant to a real model(such as a structural model of an airplane wing).

In the DBP, one may have two types of files: unstructured and structured. Structured files are relations which look like two-dimensional tables and are composed of a set of tuples and items(rows and columns). Unstructured files are simply byte streams(that is, there is no structured tabular format). All disk files on current computer systems may be considered to be unstructured since there is no underlying structure to the file: the operating system looks at the file as a sequential string of bytes.

In the following pages, it will be shown that the DBP permits the two different types of files, unstructured and structured, to be manipulated together using the special operations SUBSTRING and CONNECT. This thesis is most concerned with discussing structured files rather than unstructured files since structured files represent relations, which are familiar entities. Each item within a structured file may be of a certain data type such as signed/unsigned integer, ASCII, stringpointer, or recordpointer.

Views are pieces of the data within files which are sectioned out so that the user can "see" the data which is relevant at any particular time. Files can be quite large, and it is often necessary to look at only a portion of the files. A view is therefore a "window" into one or more files. Views are formed using relational algebra[9] on other previously created views. Note that when a view is "created", it is created only in a virtual sense. That is, views are typically created by implementing a hidden table full of pointers which directly point to the file rows and columns that the user has chosen using relational algebra. Therefore, when data is stored into a view, it is actually stored into the underlying file from which the view was "created". Retrieving data from a view is similar. Data is retrieved via the view directly from the underlying file. Examples of using relational algebra and creating views will subsequently be shown. First, a description of each type of view is presented along with accompanying figures 2a-2g:

1. JOIN - A join view is created from two existing source views. The two existing views are "joined"(or concatenated) together based on common values within a single item in each source view. The concatenation may be seen as a "column-wise" transformation, since the new view will contain the total number of columns from both source views.

2. SELECT - A select view is created by applying a constraint(or constraints) to the total number of rows in a source view. After the constraint has been applied, the newly formed select view will contain a subset of the rows in the original source view.

3. PROJECT - A project view is similar to a select view except that a constraint is applied to the columns in the source view. That is, the project view is formed by specifying a subset of the original number of columns found in the source view.

4. ORDER - An order view is created by sorting on certain items within a source view using either a ascending or descending order.

5. UNION - A union view is similar to a join view in that two source views are concatenated together to form a new view. The difference is that the concatenation is performed row-wise so that the new view contains the total number of rows from the original source views added together. It is required that the two source views have the same number of

items of identical data types.

6. SUBSTRING - A substring view is created using an unstructured file and a pattern-matching sequence. An example of the use of SUBSTRING may be seen when looking at a word processing application. If a pattern containing CRLF(carriage return, line feed) were specified, then one could form a structured SUBSTRING view delineating the sentences within any given piece of text.

7. CONNECT - A connect view is similar the previously defined join view except that two views are connected using an item with the specific "stringpointer" data type. This pointer is automatically updated when items are loaded into the view.

Since views are so important to understanding the function of the DBP, some examples relating to the scientific/engineering environment are presented. The example files represent typical entities which would be found in a "finite element modeling" database. A finite element model is a geometric model of a real structure which is composed of connected two, three, and four -noded elements. A "node" is a point which is used to join together several finite elements. In finite element modeling, two-noded elements are called either "beams" or "rods", while three-noded elements are denoted "triangular elements" and four-noded elements are denoted "quadrilateral elements". Real structures such as bridges, airplane wings, and electrical transmission towers may be analyzed by breaking the structure into a finite number of elements. In the finite element modeling database we may define four sample files called, "BEAMS","TRIANGLES", "QUADS", and "NODES". The files BEAMS, TRIANGLES, and QUADS contain finite element data whereas NODES contains the 3D model coordinate data. Using this finite element modeling example, the formation of certain views are presented. For these examples, two files "BEAMS" and "NODES" will be used:

```
+--------------+
¶ FILE : BEAMS ¶
+--------------+
```

GROUP	ELEMENT	NODE1	NODE2	EL-TYPE	NOM-SIZE	MATERIAL
1	1	1	2	WFL	8x8	ALUMINUM
1	2	3	4	I	3x2	TITANIUM
2	3	5	6	WFL	3x2	GRAPHITE

```
+--------------+
¶ FILE : NODES ¶
+--------------+
```

NODE	X	Y	Z
1	5.3	6.22	0.0
2	6.7	10.20	0.0
3	1.0	1.05	0.0
4	2.3	5.39	0.0
5	5.4	8.21	2.0

6	8.4	21.00	2.0

When these two files are initially created they automatically contain "identity" views which represent the base views upon which we may create other views. For instance, we can create a new view of the beam element connectivities by executing the following conceptual DBP command:

```
create project view CONNECTIVITIES from beams
       including NODE1 NODE2 EL-TYPE
```

If we then wanted to work just with the connectivities of wide-flange(WFL) beams, we could define another view on top of the view "CONNECTIVITIES":

```
create select view WFL-CONNECT from CONNECTIVITIES
       where EL-TYPE = WFL
```

yielding :

```
+-------------------+
¶ VIEW: WFL-CONNECT ¶
+-------------------+
```

NODE1	NODE2	EL-TYPE
1	2	WFL
5	6	WFL

Now, supposing we wanted to work with the Z-values associated with the element connectivities. This new view will require that we use both "BEAMS" and "NODES". Starting from scratch, we might do the following:

```
create join view J1 from BEAMS NODE1 NODES NODE

create project view P1 from J1 including ELEMENT NODE1 Z NODE2

create join view J1 from P1 NODE2 NODES NODE

create project view Z-VALUES from J1 excluding NODE X Y
```

yielding:

```
+-------------------+
¶  VIEW : Z-VALUES  ¶
+-------------------+
```

ELEMENT	NODE1	Z	NODE2	Z
1	1	0.0	2	0.0
2	3	0.0	4	0.0
3	5	2.0	6	2.0

It is important to note that views are <u>virtual</u> database entities as opposed to relations which are <u>real</u> entities. When a view is created using relational algebra it should be referenced as a channel into one or more database relations. Views which have been created and are no longer needed may easily be deleted from the user's session. Note that we could store data into view "Z-VALUES" and this change would be reflected back in the original identity views upon which Z-VALUES was created, namely "BEAMS" and "NODES".

The DBP supports all three major data models: Relational, Hierarchical, and Network. When one is setting up a schema for files within a database, "pointer" items may be created. These pointers permit underlying hierarchical and network data structures to be defined. Items in one file are linked with items in other files using the "recordpointer" data type. Therefore, many-to-one and one-to-many pointer relationships may be defined inside the database files. After having defined the schemas, the user is free to access the data relationally using views. Hence, the DBP may be thought of as a "relational data base machine" since relational algebra is the primary mechanism during data accesses. However, as just mentioned, the other two data models can be accommodated by careful modification of the underlying file structure with pointers.

CHAPTER IV

THE DEVELOPMENT OF HILDA

The existing host-resident interfaces to the currently available data base machines were found to be inflexible with respect to the modification of syntax and semantics associated with query processing and command encoding. The purpose of this thesis is to present an extensible and flexible means of specifying the syntax and semantics for a data base machine.

This thesis represents an initial investigation into a flexible and high-level method of communicating with a data base machine. A system called HILDA, which stands for High Level Data Abstraction System, has been designed and implemented by the author in an attempt to bridge the gap between the somewhat rigid data base machine and the user. In this sense, HILDA is an executive (or operating system), since it represents a collection of tools which allow for a high-level interface mechanism to the data base machine resource. The general structure of HILDA may be seen in figures 3 and 4. Figure 3 displays the method by which one may flexibly modify the syntax and semantics for the data base machine. Figure 4 shows the anatomy of a sample query made to the data base machine.

The specific data base machine used during the research was the Intel Data Base Processor(or DBP). It should be noted, however, that HILDA has been designed so that it may easily be integrated into other research efforts utilizing different vendors' data base machines. The DBP is connected to a VAX 11/780 host computer via a serial link. To use the DBP, one must send a set of encoded commands to the DBP and then receive a set of encoded responses. Most of the source code within HILDA is transportable to other host machines. The machine-dependent text is identified as such to aid in the transportation problem.

HILDA is functionally structured into three layers. The bottom layer is the closest to the actual data base machine. This layer is represented by the SPP program, an asynchronous error-correcting protocol. The middle layer represents a semantic specification for the data base machine. Using a small set of rigorously specified semantic procedures a developer may easily form DBP request blocks which are, in effect, pieces of machine code that the DBP understands. The highest layer of HILDA is represented by the query language DBPQL. DBPQL permits both casual and sophisticated users to reap the benefits of the data base machine via a simple-to-use view mechanism. The unusual aspect of DBPQL lies in its flexible mechanisms for syntax and semantics modification. Each of these layers will be discussed in-depth throughout the remainder of the thesis.

Some of the points that will be addressed in the thesis along with certain data base machine issues are highlighted below:

1. As previously mentioned, HILDA has been divided into three distinct layers. The layered approach seemed to be the best method for adequately testing the various software component modules.

2. The data base machine takes a load off the host computer by performing the time-consuming chores associated with data management. The performance of both the data base machine and the host during communications should be measured. The tools necessary for this measurement are included in this thesis.

3. The data base machine speaks a particular low-level language, much like any given microprocessor. Both require some type of semantic specification which is used during communications. In the case of the microprocessor, the semantics language might be the assembler. The semantics specification language for the DBP is slightly different. This specification is discussed.

4. The most important design element discussed within this thesis is the flexibility associated with the creation of the syntax and semantics in HILDA. The

flexible features found within HILDA will be identified in each chapter.

At the end of each chapter, the results of the particular research performed will be discussed. Problems encountered and things learned from the work will be emphasized in the results section. After all the chapters, there is a concluding remarks section which summarizes the overall results of the research. A brief description of each chapter associated with the design and development of HILDA is shown below:

1. Chapter V - A description of the asynchronous protocol design and implementation. The protocol includes cyclic error recovery and allows a host computer to communicate with the DBP. Some useful low-level utilities will also be outlined.

2. Chapter VI - A semantics specification package for the DBP. This package is composed of a small set of procedures which permit an efficient construction of request modules to be sent to the DBP.

3. Chapter VII - A user-oriented interactive query language. The query language is based on relational algebra which operates on objects called "views". Many of the essential but arcane DBP functions are hidden from the user.

PART II

THE DESIGN AND IMPLEMENTATION OF HILDA

CHAPTER V

LAYER 1 : THE DATA COMMUNICATIONS PROTOCOL

The design and implementation of a data communications protocol for the Intel Data Base Processor (DBP) is defined in this chapter. SPP is an asynchronous data communications protocol that has been designed and implemented for use with the Intel Data Base Processor. The protocol is termed SPP (Service Port Protocol) since it enables data transfer between the host computer and the DBP service port. The service port may be connected either to a host computer or to an interactive terminal which is used as a console monitor. The service port is currently the sole means of communicating with the DBP. The data rate using the service port is limited to 9600 baud. Therefore, it should be noted that even though the complete functionality of the DBP may be studied, the performance of the DBP will be slow. Efforts to implement a high-speed channel link(Ethernet) are currently underway at Intel.

The protocol implementation is extensible in that it is explicitly layered and the protocol functionality is hierarchically organized. Extensive trace and performance capabilities have been supplied with the protocol software to permit optional efficient monitoring of the data transfer between the host and the Intel data base processor. The design of SPP is fairly typical of communications protocols which use cyclic error recovery.

Machine independence was considered to be an important attribute during the design and implementation of SPP. Most of the protocol is therefore written in FORTRAN so that it may be portable among different machines. The protocol source code is fully commented and is included in Appendix A of this report. All source text which contains machine dependent constructs is marked to aid the analyst operating in another data base machine research project.

AN OVERVIEW OF SPP

SPP is the supporting first layer within HILDA. The other two layers, DBPSSP and DBPQL, rely completely on the correct operation of the protocol during data transmissions. The protocol permits complete usage of the DBP functionality. The physical environment in which the DBP operates consists of the host DEC VAX 11/780 minicomputer with VMS level 3 operating system, the Intel Data Base Processor, and an RS-232 connection. At its most abstract interpretation, SPP is composed of the two procedures "Send

Request" and "Receive Response". The SPP user may send a request (composed of a contiguous set of encoded commands) and receive a set of responses which may be in the form of ASCII text or a more general binary form. "Send Request" and "Receive Response" activate a hierarchy of hand-shaking primitives which include error detection and correction capabilities using cyclic redundancy checking on both the host and DBP sides.

SPP may be viewed as a three-layer protocol. The "layer" within the protocol should not, however, be confused with the layers within HILDA (see figure 5). The SPP layers may therefore be construed to be sub-layers of the HILDA data communications layer. The three sub-layers of SPP are outlined below and described more completely on the succeeding pages.

1. Application/Session:

   The sub-layer representing the highest level interface between the application software on the host computer and the DBP.

2. Data Link:

   A middle protocol sub-layer representing structured data transmission handshaking implemented with error detection and correction.

3. Physical Link:

The sub-layer closest to the DBP, representing a primitive block I/O capability.

It is important to note that all procedures within the layers of the SPP protocol operate strictly on the host computer. The Intel DBP has its own embedded set of protocol layers in firmware. Each of the SPP protocol layers will be separately discussed.

The Application/Session Sub-Layer

This is the protocol layer closest to the actual DBMS(Data Base Management System) application software accessing the data base machine. The application/session sub-layer is composed primarily of two procedures, "SEND_REQUEST" and "RECV_RESPONSE" which perform as demonstrated below (note that "PCB" stands for Parameter Control Block which is described in the section on data structures):

Function	Arguments	Description
SEND_REQUEST	MODULE	Byte array to be sent
	NBYTES_SENT	Number of bytes in 'MODULE'
	PCBTYPE	Control or application PCB flag
	APPLICATION_ID	A host-assigned id #
	REQUEST_ID	Id # of the session making the

		request
RECV_RESPONSE	MODULE	Byte array received from DBP
	NBYTES_RECV	Number of bytes received
	PCBTYPE	Control or application PCB flag
	MORE_TO_COME	Boolean flag representing whether all DBP data has been received

The "APPLICATION_ID" argument (in SEND_REQUEST) is a host-assigned number identifying the application program which will be sending the request to the DBP. "REQUEST_ID" (or session id) refers to the DBP-assigned number identifying the application program. A program that is sending a request to the control session must use a REQUEST_ID of zero, whereas programs sending application session requests may use the REQUEST_ID numbers 1 to 4 which are assigned by the DBP when the host creates application sessions. The request module contains "NBYTES_SENT" bytes of DBP machine code. It should be noted that the response module returned may be null (that is, NBYTES_RECV is zero) since many DBP operations do not yield a response. An example of the use of the above procedures may be shown in the form of the DBP conceptual command "REMARK <HOST> <HELLO>" which is performed after having started up the DBP with "DBPSTART":

```
C
C VMS( VAX OPERATING SYSTEM ) FORTRAN EXAMPLE
C
      BYTE MODULE( 512 )
      PARAMETER APPLICATION = 1
      DATA MODULE
     X     /'3A'X,'01'X,'01'X,'05'X,
     X      '48'X,'45'X,'4C'X,'4C'X,'4F'X,'FF'X,'00'X/
      CALL SEND_REQUEST( MODULE,11,APPLICATION,1,1 )
      CALL RECV_RESPONSE( MODULE,NBYTES_RECV,APPLICATION,MORE_TO_COME
```

Figure 6 graphically depicts the general form of the Host-DBP interaction occurring during the SEND_REQUEST and RECV_RESPONSE procedures. Note that each DBP request module is prefixed by the "APPLICATION_ID" and "REQUEST_ID". This four-byte prefix is inserted by the SEND_REQUEST procedure. The prefix need not be placed within the request module itself. A list of the valid machine codes and formats for request and response modules may be found in the DBP Reference Manual[10].

The Data Link Sub-Layer

The data link sub-layer is composed of the two operations "READ_BLOCK" and "WRITE_BLOCK". Data "blocks" may be viewed as the error-free transfer medium used during I/O with the DBP. A Cyclic Redundancy Check (using the

CRC16 polynomial[11]) has been implemented so that the data within the block is re-transmitted if an error is detected during transmission. The format of the two data link procedures is shown below :

Function	Arguments	Description
READ_BLOCK	BLOCK	Block of data to read from DBP
	NBYTES	Number of bytes to read
	NBYTES_RECV	Number of actual bytes read (including header, data, and trailing bytes)
	BASE	Base address for I/O
	OFFSET	Offset from BASE
WRITE_BLOCK	BLOCK	Block of data to write to DBP
	NBYTES	Number of bytes to write
	BASE	Base address for I/O
	OFFSET	Offset from BASE

The BASE and OFFSET arguments are added together(by first multiplying the BASE times 16) to form the complete address for I/O purposes. The data link layer routines are activated several times within each of the application/session layer routines: this is due to the PCB, PCB vector and request/response data area accesses which need to take place within the application/session layer.

The Physical Link Sub-Layer

The physical link layer is the protocol layer closest to the DBP. It represents the actual serial I/O on the channel. At this level, there is no error correction. For correct operation it is imperative that the TTY port and channel be configured correctly, otherwise ambiguities are sure to occur. Figure 7 displays the appropriate communications parameters which need to be set for the VAX. The physical link layer is represented by two procedures "Q_INPUT" and "Q_OUTPUT"(The VMS operating system assigns queues to each port[12]). The following table summarizes the format for the "Q_INPUT" and "Q_OUTPUT" operations:

Function	Arguments	Description
Q_INPUT	BYTES	Byte array received from DBP
	NBYTES_RECV	Number of received bytes
Q_OUTPUT	BYTES	Byte array to be sent to the DB
	NBYTES	Number of bytes to be sent

THE SPP THREADED DATA STRUCTURE

The DBP Service Port Protocol uses a simple memory mapped I/O scheme to handle the DBMS control and application functions. The host and DBP communicate by addressing the same section of DBP memory. The core of this scheme is represented as the PCB (Parameter Control Block) Vector. This vector contains pointers to the control and application address blocks. Depending on the type of DBMS function to be performed (control or application) the DBP commands are sent using the appropriate I/O addresses. All addresses are specified in a base:offset (4 bytes) format. Access to the data areas, whether the data is request or response data, is obtained by 'threading' through the PCB Vector and specific PCB (see figure 8).

OPERATION OF SPP

This section defines the actual operation of SPP in the implementation. The protocol should be used at the application/session layer level, that is, using the two session procedures "SEND_REQUEST" and "RECV_RESPONSE". The procedure for successfully communicating with the DBP is shown below :

Program	Procedures Activated	Description
DBPSTART	INIT_COMM	Initialize communications
	CREATE_CONTROL	Create control session

	CREATE_APPLICATION	Create application session
.communicate...	INIT_COMM	Initialize communications
	SEND_REQUEST	Send request module
	RECV_RESPONSE	Receive response module
DBPSTOP	INIT_COMM	Initialize communications
	DELETE_APPLICATION	Delete application session
	DELETE_CONTROL	Delete control session

SPP UTILITIES

SPP contains two primary utilities which are useful in conjunction with the protocol operation. The two available utilities are tracing and performance monitoring. "Tracing" refers to a map containing detailed data transmission information including snapshots of the PCB Vector and Control/Application PCBs. The entire handshaking sequence within SPP may be studied with the aid of the trace utility. "Performance Monitoring" refers to the collection of certain execution statistics during host-DBP transmissions. By monitoring the DBP, the software analyst may study both the effect of SPP on VMS and the elapsed time during host-DBP requests and responses.

Both utilities may be used within any of the three SPP layers. The depth of trace and performance information may therefore be set by the analyst if only a subset of the SPP operations require monitoring.

The Trace Utility

A trace facility has been designed into SPP so that all Host-DBP communications may optionally be monitored. The trace output may be re-directed to any logical output unit including the terminal, if desired. Tracing may be accomplished by using the following two routines :

1. TRACE_START(UNIT) where UNIT = logical output file unit
2. TRACE_STOP

Snapshots of the PCB Vector and PCB are displayed on the trace output to aid the analyst. Appendix B displays all communications that transpire during the "CREATE_CONTROL" and "CREATE_APPLICATION" procedures (activated when DBP_START is called). For further information on interpreting the trace see the DBP Operations Manual[13].

The Performance Monitoring Utility

The analyst may wish to invoke the performance monitoring facility when using the other routines. The statistics that are currently monitored are listed below :

1. Elapsed Clock time

2. Elapsed VAX CPU time

3. Number of VMS buffered I/O requests

4. Number of VMS direct I/O requests

5. Number of VMS page faults

The following two routines may be used to obtain the above statistics :

1. PERFORM_START

2. PERFORM_STOP(CLOCK,CPU,BIO,DIO,PAGE) - where each argument directly corresponds to each item listed above.

One of the purposes of the data base machine is to enhance the performance of the host machine by allowing a back-end data base machine to exercise many time-consuming data base management chores normally assumed by the host. The addition of the performance monitoring utility is thought to be essential(along with the trace utility) in maintaining a flexible front-end to the data base machine.

LIMITATIONS OF SPP

SPP is currently fully operational using a 9600 baud physical link to the DBP service port. SPP is limited in that only one host may be used at any one time. It should be realized, however, that several host application sessions may be instantiated permitting multiple host simulation studies if desired.

In the future, Intel is planning on supporting the Ethernet link between multiple hosts and the DBP. Ethernet is a hardware communications package composed of a co-axial cable (connecting two or more hardware devices together) and an interface for each device. The Ethernet environment forms a local area network. Since the Ethernet hardware is attached to the high-speed buses in the computers, a very high-speed access rate to the DBP will be possible. The extensive host link protocol[14] (corresponding to the recent ISO protocol standard) will be used with Ethernet. The Ethernet implementation will permit fast DBP access which will be essential for multiple-user and embedded DBMS applications.

RESULTS

SPP has been implemented so that it may be separated from the HILDA system for use in other research efforts. The construction of a machine-independent protocol was considered important since the data base machine may be connected to a wide variety of hosts. The essential machine dependencies in SPP are clearly marked to aid the implementor in a non-DEC computer environment.

The functional, layered design of SPP supports the concept of extensibility so that an individual may easily make modifications and enhancements to the existing implementation.

Finding bugs in the protocol software during development was often quite difficult, and necessitated the creation of an extensive tracing mechanism. The trace output including I/O sequences and PCB snapshots proved to be indispensible in spotting design errors in the protocol software.

The performance evaluation tool is very useful: one should note, however, that the statistics gathered reflect different aspects of VMS overhead, and not DBP overhead. That is, there are no commands that may be sent to the DBP which request statistical information that the DBP has compiled during internal processing. Intel does include an event log capability; however, this should be treated as an accounting function and not a performance function. In the future, Intel may want to consider including a set of performance commands in their command repertoire. It is inevitable that the data base machine users will want this type of capability.

The author has gained a much greater appreciation for communications protocols after having written one. The various aspects of host-slave synchronization and error correction go unnoticed to all except the implementor. This is as it should be.

CHAPTER VI

LAYER 2 : A SEMANTICS SPECIFICATION PACKAGE

A DBP Semantics Specification Package for the Intel Data Base Processor(DBP) is defined within this chapter. DBPSSP serves as a collection of cross-assembly tools that allow the analyst to assemble request blocks on the host computer for passage to the DBP. The assembly tools may be effectively used in conjunction with a DBP-compatible data communications protocol to form a query processor, precompiler or file management system for the data base processor. The SPP protocol, as defined in the previous chapter, is used within HILDA. It is important to note that even though DBPSSP may be used with SPP, the assembly primitives and procedures within DBPSSP are independent of SPP. That is, another data communications protocol may be effectively used with DBPSSP if necessary. The source modules representing the components of DBPSSP are fully commented and included as Appendix C.

AN OVERVIEW OF DBPSSP

DBPSSP is a package containing procedures which are used on the host computer to construct request modules that are to be sent to the Intel DBP. DBPSSP functions as a cross assembler in that Intel DBP "machine code" is assembled on the host computer and then directed to the data base machine for execution. Each request module sent to the DBP is of the form shown in figure 9. Every module contains an arbitrary number of commands. A command is always composed of exactly three primary sections:

1. Opcode Byte - the operation to perform on the DBP (fetch, store, define database, etc.)

2. Parameters/Data - parameters and data which relate to the operation being performed.

3. Terminator Bytes - two bytes which represent the end of the current operation which is to be performed by the DBP.

The following sections outline the capabilities and suggested usage for the DBPSSP component modules. DBPSSP should be thought of as a collection of procedures(or subroutines) that permit the software developer to easily construct data base requests to the Intel DBP. In this manner, the analyst is free to develop a flexible front-end interpreter or compiler to the data base machine. Some highlights of DBPSSP are as follows:

1. Relative and Absolute Offsets - When assembling machine code for the DBP, it is necessary to "place" the code at the proper offset within the request module. In many cases, one may build the request module sequentially from start to finish. This sequential mode of assembling is termed "relative" offsetting, since the current assembled code is simply "tacked on" to the previously assembled code. One may choose, however, to assemble code at a specific offset within the command block. This random mode of assembling is termed "absolute" offsetting. The mode used by the software developer depends on the overlying front-end driver accessing the assembly tools. A particular parsing method(for a query language, for instance) used for constructing a driver may dictate the use of one offset method over another.

2. Primitive and High-Order Procedures - DBPSSP is composed of a set of general primitive procedures and a set of high-order procedures which are based on the primitives. The high-order procedures are similar in appearance to assembler mnemonics for a given microprocessor: they have short names and contain few operands.

3. Macro Capability - Since DBPSSP is a set of procedures, it is straightforward and useful to develop parameterized procedures which access the fundamental DBPSSP procedures. This feature plays a role similar to the "macro" capability found in many assemblers.

THE COMPONENTS OF DBPSSP

DBPSSP is composed of a minimal set of general primitives and a set of high-order procedures. Each set is divided into "Control" modules and "Assembly" modules. The control modules effect the data communications options, while the assembly modules are pure assembly directives pertaining specifically to the construction of the command blocks. The control modules are discussed in the previous chapter. Modules that are dependent on the specific data communications protocol used are denoted "(D)" next to the respective module name. Each module set is depicted below:

PRIMITIVES

- INIT_COMM(D) - initialize DBP communications
- DBP_SEND(D) - send a request module to the DBP
- DBP_RECV(D) - receive a response module from the DBP
- TRACE_START(D) - start tracing
- TRACE_STOP(D) - stop tracing
- PERFORM_START - start performance monitoring
- PERFORM_STOP - stop performance monitoring(gather statistics)
- DBP_BEGIN - start to assemble a command block
- DBP_BITS_BEGIN - start bit masking operations
- DBP_BITS - perform logical 'or'ing of bits
- DBP_BITS_END - end bit masking operations
- DBP_BYTES - assemble an ASCII string

DBP_INTEGER - assemble a 1,2, or 4 byte integer

HIGH-ORDER PROCEDURES

INIT(D) - Initialize DBP communications

SEND(D) - send the built request module to the DBP

RECV(D) - receive a response module from the DBP

TRON(D) - start trace

TROFF(D) - stop trace

PRON - start performance monitoring

PROFF - stop performance monitoring(gather statistics)

START - Start encoding a command block

TERMINATE - Add the two terminator bytes to the command block being constructed.

BITSB - begin bit masking(relative offset)

BITS - logical 'or' on command block(relative offset)

BITSE - end bit masking(relative offset)

BITSB_A - same as 'BITSB'(absolute offset)

ASC - assemble an ascii string(relative offset)

ASC_A - same as 'ASC'(absolute offset)

INT1 - assemble a 1-byte integer(relative offset)

INT1_A - same as 'INT1'(absolute offset)

INT2 - assemble a 2-byte integer(relative offset)

INT2_A - same as 'INT2'(absolute offset)

INT4 - assemble a 4-byte integer(relative offset)

INT4_A - same as 'INT4'(absolute offset)

Details on using the above procedures may be found in the source which is included as Appendix C.

THE ASSEMBLY PROCESS

Figure 10 displays the assembly process occurring for a "REMARK <HOST> <HELLO>" DBP command. It is assumed in figure 10 that the user has developed a parsing method which will activate the semantic assembly primitives within DBPSSP (START,INT1,ASC, and SEND) when the REMARK command is encountered. A more substantial program is presented in Appendix D that performs the conceptual DBP operations listed below (variable arguments are bracketed for identification):

1. SUBMIT KEYS <ADMIN>

   Submit the ADMIN key to the session keyring.

2. DEFINE DATABASE <TESTING>

   Define a new database called TESTING.

3. KEEP DATABASE <TESTING>

   Make the database TESTING a permanent database.

4. DEFINE FILE <FILE1> <DBPSYS>

   Define a new file called FILE1 on the system disk(DBPSYS).

5. DEFINE SCHEMA <INT1> INT*4 <INT2> INT*4 <INT3> INT*4 <FILE1>

   Define a schema containing exactly three 4-byte integers(INT1, INT2, and INT3) for the previously defined file FILE1.

6. KEEP FILE <FILE1> <TESTING>

   Make the file FILE1 a permanent file.

7. LIST DATABASE <TESTING>

   Show the schema descripton for files within database TESTING

For further information on the command syntax see the DBP Reference Manual[10]. The program containing the above commands is presented in two languages, FORTRAN 77 and Pascal(in Appendix D). The FORTRAN program is coded using the primitives, while the Pascal program uses the high-order procedures. In general, the software designer will want to use the high-order procedures since the high-order procedures are more functionally precise with fewer parameters. Note the relative compactness of the Pascal program. The assembled modules and received DBP responses obtained after having sent the completed requests to the DBP are shown at the end of Appendix D.

RESULTS

The procedures within DBPSSP provide the developer with all of the necessary tools to construct an interactive query processor, compiler, or file management system for the Intel DBP. It was useful to create the semantics procedures in a strictly hierarchical fashion. Therefore, there are very few low-level primitives and all of the high-order procedures activate one or more of the primitives. This designed hierarchical construct turned out to be extremely powerful in forming new pieces of DBP "object code".

The merits of absolute vs. relative offsetting is unclear. It was necessary to use only the relative offsetting during the development of the query language DBPQL(to be described in the next chapter). Relative offsetting was simpler since it was not necessary to keep constant track of the offset pointer. Also, since the command line tokens and default arguments were all stored in symbol tables, it was usually convenient to construct the final request module all at once after having collected the required module data. Absolute offsetting may be valuable in a situation where the construction of symbol tables is seen as being difficult or detrimental.

DBPSSP permits the definition of a complete semantics specification associated with any given command or language syntax that the developer may choose. For example, the third layer of HILDA in conjuction with PARGEN specifies the syntax for an intepretive language "DBPQL" and the associated semantics defined using DBPSSP. DBPQL will be discussed in the next chapter.

CHAPTER VII

LAYER 3 : A VIEW-ORIENTED QUERY LANGUAGE

A query language interpreter named "DBPQL"(Data Base Processor Query Language) has been designed to allow the user to effectively and easily communicate with the data base machine. DBPQL was built using PARGEN[15] and other tools available within the MYSTRO system. The PARGEN program stands for "PARser GENerator" and will be discussed in the next section. DBPQL is designed to be utilized by both casual data management users and experienced users.

The DBPQL / DBP Conceptual Command Dichotomy

The DBP Reference Manual contains an in-depth description of the format for the request and response modules. The manual also includes a BNF-type "conceptual" command scheme which relates to the DBMS functions using a one-to-one mapping. That is, each internal DBP command may be conceptually defined by an external user-oriented BNF syntax production. The conceptual command is similar to a machine instruction on a conventional machine in that it represents the lowest, indivisible level of DBP functionality.

DBPQL was designed with the idea that the vast majority of data base machine users are not interested in exploring the functionality at the level described in the reference manual. This implies that there is a definite one-to-many mapping of DBPQL query commands to Intel DBP conceptual commands. The semantics afforded by each conceptual DBP command are encapsulated within the file "DBPCMD.DAT" which is included as Appendix E. This file includes a set of procedures which are called by the semantics within the DBPQL grammar file. Each procedure represents a single conceptual DBP command with the arguments necessary to build the request block portion relating to that command.

A strong attempt was made to shield the casual user of the data base machine from abstruse and often confusing functions such as the following:

1. FREE - free a currently attached view from the session.

2. ATTACH - attach a view to the user's session.

3. KEEP - make an entity (database,file,view) permanent.

4. SUBMIT KEYS - submit keys to the session's keyring.

5. DEFINE FILE - define a file within a given database.

These functions may be quite useful to the sophisticated user, but they should be transparent to a user who simply wishes to easily and quickly manipulate data. Most of the functions such as those listed above are required during data base management queries. In order to make them transparent, their functionality is woven into the semantic definitions of the relevant queries.

The data within the user's database is manipulated through relational algebra which is performed on views[16]. Views in DBPQL are defined to be windows which map onto the general set of data, therefore allowing the user to see only the relevant data sections within the database. Views may easily created from other views via relational commands such as the following:

```
create project view CONNECTIVITIES from QUADS
       including NODE1 NODE2 NODE3 NODE4
```

In this example, the user wishes to see only the connectivity information present within the relation "QUADS". This information is extracted from items, "NODE1", "NODE2", "NODE3", and "NODE4" and a new view named "CONNECTIVITIES" is created in the process.

The advanced user of the data base machine is accommodated through the use of "options". Options permit the experienced user to be specific about certain database and relation creation parameters, such as internal page size, variable item area size within files, and allocation sizes. The inexperienced user can simply assume the defaults in most cases and not be unduly affected.

In the following sections within this chapter, the method which is used to develop DBPQL will be shown. This same method may be applied in other research efforts to develop an entirely different high-level user interface. The first section describes PARGEN, a programming tool used to develop the syntax and semantics of DBPQL.

PARGEN

PARGEN stands for PARser GENerator and is a program contained within the MYSTRO[15] system developed at the College of William and Mary. As its name implies, PARGEN is used to develop(or generate) compilers and query processors which contain parsers. PARGEN expects two inputs before it can execute:

1. Grammar File - contains the syntax and semantics for the language to be generated. The syntax is specified in terms of BNF(Backus-Naur Form) productions. The semantics are written in Pascal

directly following the syntax production to which they are related.

2. Skeleton Compiler or Query Processor - contains a minimal language compiler/query processor which has embedded tags to aid PARGEN in the correct insertion of certain variables and the synthesize case statement. The case statement is used during the parsing phase to activate the semantics associated with a specific rule being fired.

As output, PARGEN produces the new compiler/query processor which contains everything necessary to correctly parse the user's particular source program. PARGEN also produces the parse tables which are used by the compiler/query processor during the parsing phase.

It should be noted that PARGEN can handle certain grammar ambiguities such as shift-reduce and reduce-reduce conflicts which occur regularly when designing new languages. In addition, productions may contain semantic conditions which must be true for the production to be applied. These semantic conditions may be used to resolve a given reduce-reduce conflict in the user's grammar. The class of grammars that can be handled by PARGEN is the NQLALR(1) type[17].

The execution of PARGEN may best be portrayed with an example. An arithmetic expression grammar is shown as an example of the grammar input file to PARGEN:

```
?ALL
?CRUSHER
?ERC
<GOAL> ::= <EXPR> <EOLN>
  WRITELN('THE ANSWER IS = ',SSTACK[MP].IVAL);
<EXPR> ::= <EXPR> + <FULL_TERM>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL + SSTACK[SP].IVAL;
<EXPR> ::= <EXPR> - <FULL_TERM>
  SSTACK[MP].IVAL := SSTACK[MP].IVAL - SSTACK[SP].IVAL;
<EXPR> ::= <FULL_TERM>
 ;
<FULL_TERM> ::= <TERM>
 ;
<FULL_TERM> ::= + <TERM>
 ;
<FULL_TERM> ::= - <TERM>
 SSTACK[MP].IVAL := -SSTACK[SP].IVAL;
<TERM> ::= <TERM> * <FACTOR>
 SSTACK[MP].IVAL := SSTACK[MP].IVAL * SSTACK[SP].IVAL;
<TERM> ::= <FACTOR>
 ;
<TERM> ::= <TERM> / <FACTOR>
 SSTACK[MP].IVAL := SSTACK[MP].IVAL DIV SSTACK[SP].IVAL;
<FACTOR> ::= <PRIMARY> ** <PRIMARY>
 ;
<FACTOR> ::= <PRIMARY>
 SSTACK[MP].IVAL := SSTACK[SP].IVAL;
<PRIMARY> ::= <NO>
 ;
<PRIMARY> ::= ( <EXPR> )
 SSTACK[MP] := SSTACK[MP+1];
```

The semantic text is able to "pick off" the appropriate command/source line tokens by accessing the semantics stack which is maintained in the compiler. The semantics stack variable "SSTACK" may be referenced as follows:

Suppose that the user types in the expression: 2*34

One production that would fire during the parsing of this expression would be:

```
<TERM> ::= <TERM> * <FACTOR>
 SSTACK[MP].IVAL := SSTACK[MP].IVAL * SSTACK[SP].IVAL;
```

Note the semantics for this production rule. Three items are expected on the top of the stack:

```
  Stack

  <FACTOR> = 34 -¶
  *             ¶-----   is reduced to -->    <TERM> = 68
  <TERM>   = 2  -¶
```

Through the semantics, the three stack items are replaced by the result of the multiplication. The variable "MP" refers to the left-hand side production symbol(LHS), and the variable "SP" refers to the last token in the right-hand side(RHS):

```
  <FACTOR>          SSTACK[SP]   or  SSTACK[MP+2]
  *                 SSTACK[SP-1] or  SSTACK[MP+1]
  <TERM>            SSTACK[SP-2] or  SSTACK[MP]
```

The above expression grammar, when run through PARGEN, will produce an expression evaluator program. The evaluator will ask the user for a given arithmetic expression, and then produce the result. Note that this rather compact grammar can handle quite sophisticated input such as the following :

```
-> 2*(3 +4)/(3**(3*(4+1)) + 1)
```

The order of operator precedence is contained within the proper "parsing order" inherent within the syntax productions.

For further in depth information on PARGEN reference the PARGEN User's Manual[15]. In the reference manual there are several other options which have not been mentioned here.

AN OVERVIEW OF DBPQL

DBPQL(Data Base Processor Query Language) has been designed and developed with the aid of PARGEN. The primary purpose of DBPQL is to allow the user a simple and flexible access tool in communicating with the Intel Data Base Machine. Since DBPQL is intimately related to PARGEN, a system developer maintains the flexibility to easily create an entirely new query grammar or modify the existing one. As the needs of the data base machine users change, the developer may change and adapt the query processing language accordingly.

DBPQL is entirely "view-oriented", as the chapter heading suggests. This means that all data to be placed into or retrieved from the database is done via a view. The entire procedure necessary to work with DBPQL may be best described using a sequence of steps:

1. Create a database - The database will hold the data which is to be defined and transformed later.

2. Create a relation - A relation is similar to a table with a set of rows(tuples) and columns(attributes,items). The relation identifies an underlying table which represents the "structure" within the database. There may be many relations within a single database. When one creates a relation, an "identity" view is immediately assigned for that relation. When one is "looking" at the identity view, one is viewing the entire relation as originally defined.

3. Create a view - A user will inevitably wish to look at the relation(s) in a different way than they were originally defined. Relational algebra is used(the syntax of which will be defined later) to aid in viewing the data differently. Through the use of relational algebra the user may create views upon other previously defined views. The identity view is considered to be the base view upon which all other views are constructed. Data may be henceforth retrieved and stored by directly accessing an appropriate view.

4. Display a view - Display a given view in tabular form.

5. Load a view - Load data into a given view. It should be noted that data may be loaded not only into the identity view, but also into a view that was created from other views.

6. List information - Provide a list of information about views and databases.

7. Trace - Trace the encoding and decoding of blocks to/from the DBP.

8. Performance Monitoring - Measure the performance of given DBP operations.

9. Delete a view - Delete a view that is no longer needed.

10. Delete a relation - Delete a relation that is no longer needed.

11. Delete a database - Delete a database that is no longer needed.

A view may seen to be analogous to a window(either in the real world or as in computer graphics). One is actually sectioning off a particular part of the world of data(or database) and using this modular new section for further communications.

THE DBPQL GRAMMAR

DBPQL is a context-free query language which is represented in the BNF form specified by PARGEN. The complete grammar file is included in this thesis as Appendix F, however, a more concise form is listed below(without semantics):

1. <CREATE_DATABASE> ::= CREATE DATABASE <DBNAME>

2. <CREATE_RELATION> ::= CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA <SCHEMA> <OPTIONS>

3. <CREATE_CONNECT_VIEW> ::= CREATE CONNECT VIEW <NEW_VIEW> FROM <SOURCE_VIEW1> <STRING_PTR> <SOURCE_VIEW2>

4. <CREATE_JOIN_VIEW> ::= CREATE JOIN VIEW <NEW_VIEW> FROM <SOURCE_VIEW1> <ITEM1> <SOURCE_VIEW2> <ITEM2>

5. <CREATE_ORDER_VIEW> ::= CREATE ORDER VIEW <NEW_VIEW> FROM <SOURCE_VIEW> <ITEMS...> <DIRECTION>

6. <CREATE_PROJECT_VIEW> ::= CREATE PROJECT VIEW <NEW_VIEW> FROM <SOURCE_VIEW> [INCLUDING ¶ EXCLUDING] <ITEMS...>

7. <CREATE_SELECT_VIEW> ::= CREATE SELECT VIEW <NEW_VIEW> WHERE <WHERE_CLAUSE> <OPTIONS>

8. <DELETE_VIEW> ::= DELETE VIEW <VIEW>

9. <DELETE_RELATION> ::= DELETE RELATION <RELATION>

10. <DELETE_DATABASE> ::= DELETE DATABASE <DATABASE>

11. <DISPLAY> ::= DISPLAY <VIEW>

12. <HELP> ::= HELP [<DBPQL_COMMAND>]

13. <INPUT> ::= INPUT

14. <LISTDB> ::= LISTDB <DATABASE> ¶ ALL

15. <LISTDBS> ::= LISTDBS

16. <LIST_VIEW> ::= LISTVIEW <VIEW>

17. <LIST_VIEWS> ::= LISTVIEWS

18. <LOAD> ::= LOAD <VIEW> <ITEMS_TO_LOAD>

19. <PERFORMANCECOMMAND> ::= PERFON ¶ PERFOFF

20. <TRACE_COMMAND> ::= TRACEON ¶ TRACEOFF

Once a view has been created, the user may either display the view(using DISPLAY) or load data into it(using LOAD). The structure of the database and views may be shown using the LISTVIEW and LISTVIEWS commands. Appendix G contains an actual DBPQL/user dialog during the creation of a finite-element model database. Also shown in Appendix G is the function of the (TRACEON,TRACEOFF) commands which permit an optional display of the request and response modules that are being transmitted between the host and the data base machine. The trace commands enable the developer to easily verify the command encodings and the proper interpretation of the DBP responses.

The general form of a DBPQL query statement

When referencing the DBPQL grammar file in Appendix F, one will notice a consistent structure in the formation of the syntax productions. This general structure is shown below:

<QUERY> ::= <KEYWORD> <QUERY_REST> <OPTIONS>

1. Start the encoding of the request module
2. Remove the command line tokens from the symbol tables and call the appropriate conceptual command procedures.
3. Send the request module to the DBP.
4. Process the response module

<KEYWORD> ::= XXXXXX

1. Initialize counter variables.
2. Set all defaults now.

<QUERY_REST> ::= <QUERY_DEPENDENT_ARGUMENTS>

1. Take the tokens from the semantics stack and store them in the appropriate symbol tables.

<OPTIONS> ::= <OPTIONAL_CLAUSES>

1. Usually involves setting flag variables which override the default settings.

The inclusion of the "<OPTIONS>" production allows the sophisticated user to tailor a specific database environment to his needs. On the other hand, the casual user is not forced to supply the system with complicated details, since the details are optional.

RESULTS

The syntax of DBPQL is not unusual. There are many examples of query languages whose syntax closely resembles the DBPQL syntax. The unusual aspect of the DBPQL development resides with the use of two concepts which will be discussed in the following paragraphs.

The first concept is the parser generator. The parser generator, PARGEN, used in forming DBPQL is considered to be an integral, embedded part of the DBPQL system. PARGEN is not to be used solely by the developer of the initial query language. PARGEN is designed to accompany DBPQL(or another language) if DBPQL is distributed, so that the end-users have a choice in modifying or enhancing the grammar to suit their local needs. Many of the data base management packages currently available are quite inflexible with regard to changes in syntax and semantics. Most of the packages have "built-in" parsers. With the DBPQL research, it is hoped that the utility of having the parser generator and the query language together as a packaged system has been shown.

The second concept is that of a rigorous semantics specification. At first, the DBPSSP semantic procedures were used directly in the grammar file. Then, after designing several syntax productions, it became evident that there was a cleaner method of accomplishing the task of coding the semantics. The meaning(or semantics) of each DBP conceptual command is captured in a one-to-one relationship with a "conceptual procedure". The conceptual procedures contain DBPSSP semantic procedures, while the grammar file contains calls to the conceptual procedures. Implementing the conceptual procedures seemed to make the task of preparing a grammar file a simple one. Also, since many different queries will contain the same semantics(such as to attach and free session views), the grammar file is more compact and comprehensible.

In developing the grammar file for DBPQL, it was annoying to constantly have to invent new variables which act as symbol tables during parsing. This meant that it was necessary to modify the skeleton query processor to insert the variable declarations. Perhaps the symbol tables ought to reside on the DBP side inside a symbol table relation. This would mean slower data access to the DBP-resident symbol tables, but the independence of the grammar file and skeleton query processor would be facilitated. In other words, one would not have to modify the skeleton processor to include the variable definitions for symbol table storage and manipulation.

CHAPTER VIII

CONCLUDING REMARKS

First, some conclusions obtained from the current state of the HILDA development will be discussed, then a few future enhancements and research efforts stemming from the current work will be outlined.

The portability of the HILDA components was seen as being an important facet of the research. As more data base machines become commercially available, the aspect of portability of host software will become an important one. To a greater extent, the very idea of a "database machine" promotes the notion of machine independence of data base software, since the data base machine serves as a separate module which "plugs in" to any particular host. The machine dependence of the database management functionality resides in the modular data base machine, as opposed to a piece of host-resident software.

The layered design of HILDA proved to be useful. By dividing the entire functionality into separate(but communicating) layers, the software design cycle time was minimized. It was easy to concentrate on single modules rather than having to constantly work with one huge program unit. It should be noted that it was important to completely test each layer individually before going to implement the next higher one. It took a long time to locate errors when they were due to a supposedly correct lower layer.

When implementing the third layer, DBPQL, it was amazing to notice how quickly the syntax and semantics for various query commands could be generated. This speed in design is derived from the flexiblility associated with the semantics specification language and the integral parser generator program. During the DBPQL development, it was decided that instead of the multitude of currently available programming languages, it would be nice if there existed a single multi-purpose language skeleton driven by the syntax and semantics data which could be stored in a manipulable database file. The run-time symbol tables and synthesize procedures could also be stored in the database. Some very interesting initial work(albeit done with database software) has been developed with respect to storing and manipulating program text[18]. This data-driven language could be the basis for interesting development as data base machines become more prevalent.

The primary aim of the research was to develop a very <u>flexible</u> syntax/semantics interface to the data base machine. It is the author's belief that HILDA is an example of one such interface. It was important to design DBPQL such that the casual user would be able to excercise the data base machine and control his own databases. Too often, with database administrators and complicated sub-schemas, many computing environments shut out the user from the data management process. The point is clear - a user should be able to easily manage his own data. At the same time, there should be an inherent functionality in a high-level interface which supports the sophisticated user. If data is shared among many users, the addition of integrity constraints should be a made a simple task. The DBPQL interpreter is successful as a user-friendly interactive language and directly addresses these issues. Many of the required low-level DBP conceptual functions are hidden - it was really annoying to have to manually attach and free session views, for example.

With regards to future enhancements and modifications to HILDA, there are several points to be made:

1. There is an interesting question that arises when implementing some of the DBP conceptual commands: "How should the tasks which may be assumed by either the host or the DBP be distributed?". For instance, there is a "LOOP WHILE" DBP conceptual command which, when sent to the DBP, causes a conditional iterative evaluation of the commands in the loop body. Is it more efficient to let the DBP do this, or should the iteration use the host CPU? Also, some of the symbol table manipulation could be done on the DBP. Even though this would be more inefficient than having the symbol table in memory, the data abstraction features of the DBP make it an attractive device for all kinds

of data manipulation. These are questions that are unanswered and pose intriguing research problems.

2. Much work needs to be done in the user interface area. For example, the help facility with DBPQL is typical of many facilities currently resident within other interactive data base management systems. What happens if the user types in, "CREATE <carriage-return>". Normally this query would generate a syntax error in the parsing process. It might be interesting to assign certain "help procedures" which would fire when a particular state has been reached within the automaton generated from the grammar[19]. Then, the system might be able to respond with "What should I create ?".

3. Intel is planning on shrinking the size of the DBP so that it may be inserted into a given microcomputer as a single-board disk controller. This idea of an "intelligent" disk controller is very powerful:

    1. Microcomputers incorporating this type of intelligent mass storage control would be able to perform reasonably complex data management tasks. This small-scale data management capability will have far-reaching effects especially when considering a local area network environment using Ethernet.

2. Current file i/o mechanisms in programming languages reflect the underlying disk controller architecture. That is, one may read and write sequentially or randomly. With the new content-addressable disk controller, programming languages will change to reflect the new capabilities. Many new forms of data abstraction, such as "relation","tuple", and "view" will appear within the new languages as standard data types. Instead of using the simple i/o to which we are presently accustomed, we may be routinely performing relational commands on variables previously declared as type "view". The onset of the intelligent disk controller will lend credence to the higher level abstractions in language designs. Most languages do not presently have these abstractions as embedded features because of the performance overhead associated with the mappings of the high-level functions to the currently available controllers.

3. We currently think of the topics, "File Management" and "Data Base Management" as disparate. In reviewing the Intel DBP's capability, one may see that these two topics are one and the same. "Files" may be either structured or unstructured, allowing for all kinds of powerful relational operations.

"Databases" are simply collections of "files"(or relations). Unstructured files are very similar to files as we presently use them. The reason behind labelling HILDA as a data abstraction system rather than a data base management system lies with this idea of file/database homogeneity. As users of database hardware, we should be able to think of data abstractions in our programs, instead of "databases", "views", "directories", and "files". How will the introduction of the data base machine effect the file management functions within an operating system ? What types of new functions should be present to assist the user in managing data ? These topics have yet to be explored.

4. The problems of data integrity and distributed data will continue to be as much of a problem with data base machines as they have been with data base software. Data base machines seem to be begging for a distributed environment, especially as the data base machine prices decrease.

Much more work needs to be done in the area of data base machines. It is hoped that the development of HILDA and the notes within this thesis have served a useful contribution to the search for new methods of data management and abstraction using data base machines.

GLOSSARY OF ACRONYMS

1. BNF - Backus Naur Form. A formal method of specifying the syntax for a given language.

2. CRC - Cyclic Redundancy Check. CRC byte(s) are built from a packet of data which is transmitted from one computer to another. These byte(s) are also built on the computer receiving the data packet. The two CRC byte groups are checked for equality. If the bytes are not equal then the original data packet is re-transmitted from the originating computer. CRC16 is a special case of cyclic redundancy checking which uses a 16-bit word(two bytes).

3. DBMS - Data Base Management System. A system used for storing, retrieving, and manipulating data.

4. DBP - Data Base Processor. Intel's data base machine(processor).

5. DBPQL - Data Base Processor Query Language. Top layer within HILDA. An interactive view-oriented query language for the Intel DBP.

6. DBPSSP - Data Base Processor Semantics Specification Package. Middle layer within HILDA. This package is a group of procedures which enable users to easily form DBP request blocks.

7. HILDA - High Level Data Abstraction System. A system composed of three layers(SSP, DBPSSP, and DBPQL) which allows a user to use the functionality of the Intel DBP. The main emphasis of HILDA is the flexible formulation of syntax and semantics associated with a given high-level language.

8. ICASE - Institute for Computer Applications in Science and Engineering.

9. IPAD - Integrated Programs for Aerospace Vehicle Design.

10. MYSTRO - A collection of tools for language development conceived at the College of William and Mary.

11. PARGEN - Parser Generator. A program permitting the user to create a compiler or query processor by specifying a skeleton compiler/processor and a grammar file.

12. PCB - Parameter Control Block. A block of DBP memory which includes pointers to data buffers and specific protocol information. Used within the SPP protocol software.

13. RIM - Relational Information Manager. A relational database management system built within the IPAD project.

14. SPP - Service Port Protocol. The bottom layer within HILDA which allows the host computer(VAX 11/780) and DBP to communicate with each other.

15. VAX - Digital Equipment Corporation's VAX minicomputer. A VAX 11/780 was used to develop HILDA.

16. VMS - Virtual memory management operating system used on the VAX minicomputer.

APPENDICES

APPENDIX A - SPP Source

SPP has been implemented using VAX VMS FORTRAN 77. The 'SPP' program module specifies implementation notes which refer to certain computer dependencies of SPP. Subroutines which contain at least one source of VAX/VMS machine dependence are flagged with '*** MACHINE DEPENDENT ***' at the head of the routine.

```
®
$ @typeq
      PROGRAM SPP
C========================================================
C
C PURPOSE :
C
C   'SPP' IS A SERVICE PORT PROTOCOL TO BE USED IN
C   ACCESSING THE INTEL DBP
C
C ARGUMENTS :
C
C   NONE
C
C DIAGNOSTIC TRACE OPTION FOR PROTOCOL :
C
C USE TRACE_START AND TRACE_STOP
C
C PERFORMANCE MONITORING OPTION :
C
C USE PERFORM_START AND PERFORM_STOP
C
C
C SPP FUNCTIONAL COMPONENTS :
C
C PROGRAMS
C
C      DBP_START  - USED TO START I/O WITH THE DBP.
C
C      SPP        - THIS PROGRAM IS JUST A SAMPLE PROGRAM
C                   WRITTEN TO SHOW THE CORRECT FORM
C                   FOR SPP OPERATION.
C
C      DBP_STOP   - USED TO END I/O WITH THE DBP.
C
C PROTOCOL SUBROUTINES :
C
C      INIT_COMM          - INITIALIZE COMMUNICATIONS WITH DBP
C      END_COMM           - END COMMUNICATIONS WITH DBP
C      CREATE_CONTROL     - CREATE A DBP CONTROL SESSION
C      DELETE_CONTROL     - DELETE THE DBP CONTROL SESSION
C      CREATE_APPLICATION- CREATE A DBP APPLICATION SESSION
C      DELETE_APPLICATION- DELETE THE DBP APPLICATION SESSION
C      RECV_RESPONSE      - RECEIVE A DBP RESPONSE
C      SEND_REQUEST       - SEND A REQUEST TO THE DBP
C      READ_BLOCK         - READ A DATA BLOCK FROM THE DBP
C      WRITE_BLOCK        - WRITE A DATA BLOCK TO THE DBP
C      Q_INPUT            - RECEIVE A BYTE BUFFER FROM THE DBP
C      Q_OUTPUT           - SEND A BYTE BUFFER TO THE DBP
C      LOW16              - RETURN LOW ORDER BYTE FROM 16-BIT WORD
C      LOW32              - RETURN LOW ORDER BYTE FROM 32-BIT WORD
C      HIGH16             - RETURN HIGH ORDER BYTE FROM 16-BIT WORD
C      HIGH32             - RETURN HIGH ORDER BYTE FROM LOWER-HALF
C                           OF 32-BIT WORD
C      GLUE               - RETURN A 16-BIT WORD FORMED FROM 2 BYTES
C
C UTILITY SUBROUTINES :
C
C      TRACK              - IF TRACE MODE HAS BEEN ENABLED, DISPLAY THE
C                           TWO DATA STRUCTURE FORMATS( PCB &
```

```
C                            PCB VECTOR )
C       TRACE_START        - ENABLE TRACE MODE
C       TRACE_STOP         - DISABLE TRACE MODE
C       PERFORM_START      - ENABLE PERFORMANCE TRACING
C       PERFORM_STOP       - DISABLE PERFORMANCE TRACING
C
C
C
C MACHINE DEPENDENCIES :
C
C   THIS SOURCE TEXT REPRESENTS A TESTED VAX/VMS
C   VERSION OF SPP.
C
C   SPP HAS BEEN IMPLEMENTED SO THAT THE MACHINE
C   DEPENDENCIES INHERENT WITHIN THE SOURCE TEXT
C   ARE CLEARLY MARKED TO AID THE IMPLEMENTOR IN
C   A NON-DEC COMPUTER ENVIRONMENT.
C
C   THE FOLLOWING IS A LIST OF THINGS TO WATCH OUT FOR
C   IF A NON-DEC MACHINE IS BEING USED :
C
C   1.) THE FOLLOWING ROUTINES CONTAIN VMS MACRO CALLS
C       WHICH ARE USED MAINLY FOR TTY I/O PURPOSES :
C
C
C       ROUTINE                DEPENDENCIES
C
C       INIT_COMM              LIB$CRC_TABLE,SYS$ASSIGN
C       Q_INPUT                SYS$QIOW
C       Q_OUTPUT               SYS$QIOW
C       READ_BLOCK             LIB$CRC
C       WRITE_BLOCK            LIB$CRC
C       END_COMM               SYS$DASSGN
C
C       WHERE:
C
C          LIB$CRC_TABLE - Initialize a table for further CRC16
C                          calculations
C          LIB$CRC       - Calculate CRC16 for a given ASCII string
C          SYS$ASSIGN    - Assign an I/O channel
C          SYS$DASSGN    - De-assign an I/O channel
C          SYS$QIOW      - Block I/O routine for serial I/O
C
C
C       THE TYPES OF FUNCTIONS PRESENT WITHIN THESE
C       ROUTINES IS USUALLY FOUND WITHIN MOST OPERATING
C       SYSTEM SERVICE MANUALS.
C
C   2.) HEXADECIMAL VALUES FOR THE VAX ARE SPECIFIED AS
C       FOLLOWS :
C
C       'DE'X     'FF'X     etc.
C
C       THIS REPRESENTATION MAY DIFFER ON ANOTHER COMPUTER.
C
C   3.) DATA TYPE 'BYTE' - ON THE VAX, THE MOST NATURAL WAY
C       TO REPRESENT PURE BYTE STREAMS IS USING THE DATA TYPE
C       'BYTE'. ON OTHER MACHINES, ONE MAY USE 'LOGICAL*1' OR
C       'CHARACTER*1'. KEEP IN MIND, HOWEVER, THAT CHARACTER
C       DATA IS GENERALLY STORE DIFFERENTLY( VMS CALLS THIS
```

```
C       A DESCRIPTOR TYPE ).
C
C
C    4.) IDENTIFIER LENGTHS - THE FORTRAN VARIABLE NAME LENGTHS ARE
C        LONGER THAN MAY BE SUPPORTED WITH SOME FORTRAN COMPILERS.
C        THEY ARE LONG TO AID IN THE READING AND COMPREHENSION OF
C        THE SOURCE.
C
C    5.) 'INCLUDE' STATEMENT - MOST FORTRANS SUPPORT A METHOD FOR
C         INCLUDING/INSERTING A DISK FILE WITHIN THE SOURCE PRIOR
C         TO COMPILATION.
C
C DATE:
C
C   APRIL 12, 1983
C
C AUTHOR:
C
C   PAUL A. FISHWICK
C   KENTRON TECHNICAL CENTER
C   3221 NORTH ARMISTEAD RD.
C   HAMPTON, VA. 23666
C   ( 804 )-865-3195
C
C=================================================
      INTEGER*4 BIO,DIO,PAGEF
      INCLUDE 'SPPCOM.TXT'
C
C NOTE: THIS IS AN EXAMPLE USE OF 'SPP'. THE USER MUST
C HAVE STARTED COMMUNICATIONS BY ACTIVATING THE PROGRAM
C 'DBPSTART' PRIOR TO THIS. THE FOLLOWING SET OF BYTES
C REPRESENTS THE CONCEPTUAL 'DEFINE DATABASE <TESTING>'
C DBP COMMAND. THE DIAGNOSTIC AND PERFORMANCE TRACING
C OPTIONS HAVE BEEN UTILIZED.
C
      CALL TRACE_START( 9 )
      CALL INIT_COMM
      MODULE(1) = '60'X
      MODULE(2) = '07'X
      MODULE(3) = '54'X
      MODULE(4) = '45'X
      MODULE(5) = '53'X
      MODULE(6) = '54'X
      MODULE(7) = '49'X
      MODULE(8) = '4E'X
      MODULE(9) = '47'X
      MODULE(10)= 'FF'X
      MODULE(11)= '00'X
      CALL PERFORM_START
      CALL SEND_REQUEST(MODULE,11,1,1,1)
      CALL RECV_RESPONSE(MODULE,NBYTES_RECV,1,MORE_TO_COME)
      CALL PERFORM_STOP( CLOCK,CPU,BIO,DIO,PAGEF )
      CALL TRACE_STOP
      CALL EXIT
      END
C
C COMMON FOR SPP( SERVICE PORT PROTOCOL )
C
      BYTE BYTES(1024),BLOCK(1024),MODULE(1024)
      BYTE MODULE2(1024)
```

```
      INTEGER*2 BASE,OFFSET,IOSB(4),NBYTES,NBYTES_RECV
      INTEGER*2 TTY_CHANNEL
      INTEGER*4 STATUS,CRC_TABLE(16),CRC
      CHARACTER STRING*512
      COMMON/CRCCOM/ CRC,CRC_TABLE
      COMMON/COMM/ TTY_CHANNEL
C
      LOGICAL*4 MORE_TO_COME
C
C SYSTEM SERVICE PARAMETERS
C
C READ PARAMETERS
      PARAMETER IO$M_NOECHO = '00000040'X
      PARAMETER IO$M_PURGE  = '00000800'X
      PARAMETER IO$M_TIMED  = '00000080'X
      PARAMETER IO$_TTYREADALL = '0000003A'X
C STATUS INDICATORS
      PARAMETER SS$_NORMAL  = '00000001'X
C WRITE PARAMETERS
      PARAMETER IO$_WRITEVBLK = '00000030'X
C
C DEBUG(TRACE) VARIABLES
C
      INTEGER*4 UNIT
      LOGICAL*4 DEBUG
      COMMON/TRACECOM/ DEBUG,UNIT
      DATA DEBUG/.FALSE./
      PROGRAM DBP_START
C=====================================================================
C
C PURPOSE :
C
C   START OPERATIONS FOR THE DBP
C   THIS INCLUDES ALLOCATING THE CHANNEL TO
C   BE USED FOR HOST <-> DBP COMMUNICATIONS
C
C
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
      INCLUDE 'SPPCOM.TXT'
C
C SET UP COMMUNICATIONS
C
      PRINT *,'** START DBP COMMUNICATIONS **'
```

```
      CALL TRACE_START( 9 )
      CALL INIT_COMM
C
C CREATE CONTROL,APPLICATION SESSIONS
C
      PRINT *,'** CREATING CONTROL SESSION **'
      CALL CREATE_CONTROL
      PRINT *,'** CREATING APPLICATION SESSION **'
      CALL CREATE_APPLICATION
C
      CALL TRACE_STOP
      CALL EXIT
      END
      PROGRAM DBP_STOP
C=============================================================
C
C PURPOSE :
C
C   STOP OPERATIONS FOR THE DBP
C
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=============================================================
      INCLUDE 'SPPCOM.TXT'
C
C SET UP COMMUNICATIONS
C
      PRINT *,'** START DBP COMMUNICATIONS **'
      CALL TRACE_START( 9 )
      CALL INIT_COMM
C
C DELETE THE APPLICATION SESSION
C AND CONTROL SESSION
C ( TERMINATE DBP )
C
      PRINT *,'** DELETING THE APPLICATION SESSION **'
      CALL DELETE_APPLICATION
C
      PRINT *,'** DELETING THE CONTROL SESSION **'
      CALL DELETE_CONTROL
C
      CALL TRACE_STOP
      CALL EXIT
      END
      SUBROUTINE INIT_COMM
```

```
C=======================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   INITIALIZE COMMUNICATIONS PARAMETERS PRIOR TO ACTUALLY
C   TRANSMITTING DATA BACK AND FORTH
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=======================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*4 SYS$ASSIGN
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Initialize iDBP Communications **')
C
C INITIALIZE A CRC-16 TABLE FOR ERROR DETECTION
C ( THE VAX 'CRC' MACHINE INSTRUCTION IS USED )
C
      CALL LIB$CRC_TABLE( '120001'O,CRC_TABLE )
C
C ASSIGN AN I/O CHANNEL USING A TTY PORT
C
      STATUS = SYS$ASSIGN( 'REMOTE',TTY_CHANNEL,, )
      IF( STATUS.NE.SS$_NORMAL ) THEN
        WRITE( UNIT,300 ) STATUS
300     FORMAT(' Error,unable to assign the DBP I/O Channel'/,
     X          ' Status is ',z8,/,' See : INIT_COMM' )
      ENDIF
C
C SEND A CONTROL-C TO FLUSH THE TYPE-AHEAD BUFFER
C AND INITIALIZE DBP COMMUNICATIONS
C
      BYTES(1) = '03'X
      CALL Q_OUTPUT( BYTES,1 )
      NBYTES_RECV = 16
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      RETURN
      END
      SUBROUTINE END_COMM
C=======================================================
C
C *** MACHINE DEPENDENT ***
```

```
C
C PURPOSE :
C
C   END COMMUNICATIONS TO THE DBP. DEASSIGN CHANNEL.
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C======================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*4 SYS$DASSGN
C
C DEASSIGN THE PREVIOUSLY ASSIGNED CHANNEL
C
      STATUS = SYS$DASSGN( TTY_CHANNEL )
      IF( STATUS.NE.SS$_NORMAL ) THEN
        WRITE( UNIT,100 ) STATUS
100     FORMAT(' Error,unable to de-assign the DBP Channel'/,
     X         ' Status is ',z8,/' See: END_COMM' )
      ENDIF
      RETURN
      END
      SUBROUTINE CREATE_CONTROL
C======================================================
C
C PURPOSE :
C
C   CREATE A CONTROL SESSION
C   NOTE : THIS IS THE FIRST FUNCTION TO BE PERFORMED
C          TO ACCESS THE DBP. THE 'MONITOR' BUTTON MUST
C          BE PUSHED PRIOR TO CALLING THIS ROUTINE.
C
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
```

```
C
C   APRIL 12,1983
C
C===============================================================
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 BASE_CTRL,OFFSET_CTRL
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Create Control Session **' )
C
C READ THE PCB ADDRESS VECTOR
C
      BASE = 'EE0C'X
      OFFSET = 0
      CALL READ_BLOCK( BLOCK,10,NBYTES_RECV,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,0 )
      CALL GLUE( BLOCK(3),BLOCK(4),OFFSET_CTRL )
      CALL GLUE( BLOCK(5),BLOCK(6),BASE_CTRL )
C
C READ THE CONTROL SESSION PCB
C
      CALL READ_BLOCK( BLOCK,43,NBYTES_RECV,BASE_CTRL,OFFSET_CTRL )
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
      IF( BLOCK(15).EQ.4 ) THEN
C HOST TO SEND 'ENABLE SERVICE PORT'
        BLOCK(16) = '11'X
        CALL WRITE_BLOCK( BLOCK,43,BASE_CTRL,OFFSET_CTRL )
        IF( DEBUG ) CALL TRACK( BLOCK,1 )
      ELSE
        WRITE( UNIT,100 ) BLOCK(15)
100     FORMAT(' Error, DBP''s Wait on Enable is not set.'/,
     X          ' DBP Status is ',z2'h' )
      ENDIF
C
C RETURN CONTROL TO THE DBMS
C
      BYTES(1) = '47'X
      BYTES(2) = '0D'X
      CALL Q_OUTPUT( BYTES,2 )
      NBYTES_RECV = 29
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      RETURN
      END
      SUBROUTINE CREATE_APPLICATION
C===============================================================
C
C PURPOSE :
C
C   CREATE AN APPLICATION SESSION
C
C
C ARGUMENTS :
C
C   NONE
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
```

```
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
      INCLUDE 'SPPCOM.TXT'
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Create Application Session **')
C
C PERFORM 'CREATE APPLICATION SESSION'
C
      MODULE(1)='E4'X
      MODULE(2)='01'X
      MODULE(3)='FE'X
      MODULE(4)='FF'X
      MODULE(5)='00'X
C
      CALL SEND_REQUEST( MODULE,5,0,1,0 )
C
C RECEIVE THE APPLICATION #
C
      CALL RECV_RESPONSE( MODULE,NBYTES_RECV,0,MORE_TO_COME )
      IF( DEBUG ) THEN
        WRITE( UNIT,200 ) (MODULE(I),I=1,NBYTES_RECV)
200     FORMAT( ' ** Create Application Response **'//,
     X          16(1X,Z2.2) )
      ENDIF
C
C RETURN CONTROL TO THE DBMS
C
      BYTES(1) = '47'X
      BYTES(2) = '0D'X
      CALL Q_OUTPUT( BYTES,2 )
      NBYTES_RECV = 29
      CALL Q_INPUT( BYTES,NBYTES_RECV )
C
      RETURN
      END
      SUBROUTINE DELETE_CONTROL
C=====================================================================
C
C PURPOSE :
C
C   DELETE A CONTROL SESSION
C   NOTE : THIS IS THE LAST FUNCTION TO BE PERFORMED
C          WHEN THE DBP IS TO BE STOPPED
C
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
```

```
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
      INCLUDE 'SPPCOM.TXT'
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Delete Control Session **')
C
C PERFORM 'TERMINATE DBP'
C
      MODULE(1)='ED'X
      MODULE(2)='FF'X
      MODULE(3)='00'X
C
      CALL SEND_REQUEST( MODULE,3,0,1,0 )
C
C RECEIVE THE 'TERMINATE DBP' RESPONSE
C
      CALL RECV_RESPONSE( MODULE,NBYTES_RECV,0,MORE_TO_COME )
C
      RETURN
      END
      SUBROUTINE DELETE_APPLICATION
C=====================================================================
C
C PURPOSE :
C
C   DELETE AN APPLICATION SESSION
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 BASE_APP,OFFSET_APP
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Delete Application Session **' )
C
C READ THE PCB ADDRESS VECTOR
```

```
C
      BASE = 'EE0C'X
      OFFSET = 0
      CALL READ_BLOCK( BLOCK,10,NBYTES_RECV,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,0 )
      CALL GLUE( BLOCK(7),BLOCK(8),OFFSET_APP )
      CALL GLUE( BLOCK(9),BLOCK(10),BASE_APP )
C
C CHECK THE INDEX FIELD FOR POSSIBLE ERRORS
C
      IF( (BLOCK(1).GE.'A0'X).AND.
     X    (BLOCK(1).LE.'DF'X) ) THEN
          WRITE( UNIT,100 ) BLOCK(1)
100       FORMAT(' Error, Couldn''t Delete Application Session'/,
     X           ' Index Field(low) is ',z2,'h' )
          RETURN
      ENDIF
C
C READ THE APPLICATION SESSION PCB
C
      CALL READ_BLOCK( BLOCK,43,NBYTES_RECV,BASE_APP,OFFSET_APP)
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
      IF( BLOCK(15).EQ.7 ) THEN
C HOST TO SEND 'OK FIN'
        BLOCK(16) = '05'X
        CALL WRITE_BLOCK( BLOCK,43,BASE_APP,OFFSET_APP )
        IF( DEBUG ) CALL TRACK( BLOCK,1 )
      ELSE
        WRITE( UNIT,200 ) BLOCK(15)
200     FORMAT(' Error, Application Session cannot be deleted.'/,
     X         ' DBP Status is ',z2'h' )
      ENDIF
C
C RETURN CONTROL TO THE DBMS
C
      BYTES(1) = '47'X
      BYTES(2) = '0D'X
      CALL Q_OUTPUT( BYTES,2 )
      NBYTES_RECV = 29
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      RETURN
      END
      SUBROUTINE RECV_RESPONSE( MODULE,TOTAL_BYTES,PCBTYPE,
     X                          MORE_TO_COME )
C==================================================================
C
C PURPOSE :
C
C   RECEIVE RESPONSE MODULE FROM THE DBP
C
C ARGUMENTS :
C
C   MODULE      - RECEIVED RESPONSE MODULE
C   NBYTES      - # OF BYTES IN RESPONSE MODULE RECEIVED
C   PCBTYPE     - TYPE OF PCB TO RECEIVE RESPONSE MODULE
C
C               = 0 --> CONTROL PCB
C               = 1 --> APPLICATION PCB
C
C   MORE_TO_COME - TRUE, IF THERE IS MORE DATA TO BE RECEIVED
```

```
C                  AFTER THIS ROUTINE HAS BEEN CALLED
C
C                - FALSE, IF ALL DATA HAS BEEN RECEIVED FROM
C                  THE DBP
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
C============================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 NSEGMENTS,BUFFER1_LENGTH,BUFFER2_LENGTH,PCBTYPE
      INTEGER*2 BUFFER1_BASE,BUFFER2_BASE,BUFFER1_OFFSET
      INTEGER*2 BUFFER2_OFFSET,TOTAL_BYTES
C
C
C GET PCB ADDRESS VECTOR
C
      IF( DEBUG ) WRITE( UNIT,5 )
5     FORMAT(' ** Receive Response **')
50    TOTAL_BYTES = 0
60    BASE = 'EE0C'X
      OFFSET = 0
      CALL READ_BLOCK( BLOCK,10,NBYTES_RECV,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,0 )
C
C LOOK AT THE INDEX FIELD
C
      IF( (BLOCK(1).GE.'A0'X).AND.
     X    (BLOCK(1).LE.'DF'X) ) THEN
        WRITE( UNIT,100 ) BLOCK(1)
100     FORMAT(' Error in RECV_RESPONSE, Index(low) is ',z2,'h')
        RETURN
      ELSE IF ( (BLOCK(1).EQ.'FF'X).AND.
     X          (BLOCK(2).EQ.'FF'X) ) THEN
        IF( DEBUG ) WRITE( UNIT,125 )
125     FORMAT(' Error, iDBP is suspended. Index is FFFFh')
        GO TO 9999
      ENDIF
C
C RECEIVE RESPONSE USING CONTROL OR APPLICATION PCB ?
C
      IF( PCBTYPE.EQ.0 ) THEN
         CALL GLUE( BLOCK(3),BLOCK(4),OFFSET )
         CALL GLUE( BLOCK(5),BLOCK(6),BASE )
      ELSE
         CALL GLUE( BLOCK(7),BLOCK(8),OFFSET )
         CALL GLUE( BLOCK(9),BLOCK(10),BASE )
      ENDIF
      CALL READ_BLOCK( BLOCK,43,NBYTES_RECV,BASE,OFFSET )
```

```
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
C
C TEST THE DBP STATUS FIELD, FIRST
C
      IF( BLOCK(15).EQ.7 ) THEN
C
C UPDATE THE PCB
C
        BLOCK( 16 ) = 1
        CALL WRITE_BLOCK( BLOCK,43,BASE,OFFSET )
        IF( DEBUG ) CALL TRACK( BLOCK,1 )
C
C RETURN CONTROL TO DBMS
C
        BYTES(1) = '47'X
        BYTES(2) = '0D'X
        CALL Q_OUTPUT( BYTES,2 )
        NBYTES_RECV = 29
        CALL Q_INPUT( BYTES,NBYTES_RECV )
        TOTAL_BYTES = 0
        MORE_TO_COME = .FALSE.
        GO TO 9999
      ENDIF
C
C READY TO RECEIVE SEGMENT(S)
C
      NSEGMENTS = BLOCK( 31 )
C RECEIVE THE FIRST BUFFER( SEGMENT )
      CALL GLUE( BLOCK(32),BLOCK(33),BUFFER1_OFFSET )
      CALL GLUE( BLOCK(34),BLOCK(35),BUFFER1_BASE )
      CALL GLUE( BLOCK(36),BLOCK(37),BUFFER1_LENGTH )
      TOTAL_BYTES = TOTAL_BYTES + BUFFER1_LENGTH
      CALL READ_BLOCK( MODULE(1),BUFFER1_LENGTH,NBYTES_RECV,
     X                 BUFFER1_BASE,BUFFER1_OFFSET )
C RECEIVE THE SECOND BUFFER( SEGMENT ), IF ANY
      IF( NSEGMENTS.NE.2 ) GO TO 200
      CALL GLUE( BLOCK(38),BLOCK(39),BUFFER2_OFFSET )
      CALL GLUE( BLOCK(40),BLOCK(41),BUFFER2_BASE )
      CALL GLUE( BLOCK(42),BLOCK(43),BUFFER2_LENGTH )
      IF(BUFFER2_LENGTH.GT.0) CALL READ_BLOCK( MODULE(TOTAL_BYTES+1),
     X      BUFFER2_LENGTH,NBYTES_RECV,BUFFER2_BASE,BUFFER2_OFFSET)
      TOTAL_BYTES = TOTAL_BYTES + BUFFER2_LENGTH
C
C UPDATE THE PCB
C
200   BLOCK( 16 ) = 1
      CALL WRITE_BLOCK( BLOCK,43,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
C
C RETURN CONTROL TO DBMS SOFTWARE
C
      BYTES( 1 ) = '47'X
      BYTES( 2 ) = '0D'X
      CALL Q_OUTPUT( BYTES,2 )
      NBYTES_RECV = 29
      CALL Q_INPUT( BYTES,NBYTES_RECV )
C
C ARE ALL MODULES READ FROM THE DBP ?
C IF NOT, FLAG THE CALLER
C
```

```
      IF( BLOCK(15).EQ.6 ) THEN
        MORE_TO_COME = .FALSE.
        IF( DEBUG ) WRITE( UNIT,300 )
300     FORMAT(' ** All data has been received **' )
      ELSE
        MORE_TO_COME = .TRUE.
        IF( DEBUG ) WRITE( UNIT,400 )
400     FORMAT(' ** There is more data to come **')
      ENDIF
C
C DONE READING ALL RESPONSES
C
C ADJUST THE MODULE ARRAY( RETURNED RESPONSE )
C TO GET RID OF THE HEADER BYTES
C
      DO 500 I = 1,TOTAL_BYTES
500   MODULE( I ) = MODULE( I+4 )
      TOTAL_BYTES = TOTAL_BYTES - 4
C
9999  RETURN
      END
      SUBROUTINE SEND_REQUEST( MODULE,NBYTES_SENT,PCBTYPE,
     X                         APPLICATION_ID,REQUEST_ID )
C==================================================================
C
C PURPOSE :
C
C   SEND REQUEST MODULE TO THE DBP
C
C ARGUMENTS :
C
C   MODULE     - REQUEST MODULE
C   NBYTES     - # OF BYTES IN REQUEST MODULE TO SEND
C   PCBTYPE    - TYPE OF PCB TO RECEIVE RESPONSE MODULE
C
C              = 0 --> CONTROL PCB
C              = 1 --> APPLICATION PCB
C
C
C   APPLICATION_ID - ARBITRARILY ASSIGNED HOST APPLICATION ID
C
C   REQUEST_ID - THIS IS THE ID # OF THE SESSION MAKING
C                THE REQUEST. WHEN AN APPLICATION IS FIRST
C                CREATED, THE CONTROL SESSION( = 0 ) ID IS
C                THE 'REQUEST_ID'. AFTER THAT, THE APPLICATION
C                ID ISSUING THE REQUEST IS THE 'REQUEST_ID'.
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   APPLICATION
C
C DATE :
C
C   APRIL 12,1983
C
```

```
C========================================================
C
      INCLUDE 'SPPCOM.TXT'
      BYTE BUFFER1( 512 ),BUFFER2( 512 ),TEMP_BYTE
      INTEGER*2 PCBTYPE,NSEGMENTS,BUFFER1_LENGTH,BUFFER2_LENGTH
      INTEGER*2 BUFFER1_BASE,BUFFER1_OFFSET
      INTEGER*2 BUFFER2_BASE,BUFFER2_OFFSET
      INTEGER*2 TOTAL_SENT,NBYTES_SENT
      INTEGER*4 APPLICATION_ID,REQUEST_ID
C
C STICK IN HOST APPLICATION ID & SESSION ID
C
50    NBYTES = NBYTES_SENT
      LEFTOVER_BYTES = .FALSE.
      DO 2 I = NBYTES,1,-1
2     MODULE(I+4) = MODULE(I)
      CALL LOW32( APPLICATION_ID,TEMP_BYTE )
      MODULE(1) = TEMP_BYTE
      MODULE(2) = '00'X
      CALL LOW32( REQUEST_ID,TEMP_BYTE )
      MODULE(3) = TEMP_BYTE
      MODULE(4) = '00'X
      NBYTES = NBYTES + 4
C
C GET PCB ADDRESS VECTOR
C
70    IF( DEBUG ) WRITE( UNIT,80 )
80    FORMAT(' ** Send Request **')
      BASE = 'EE0C'X
      OFFSET = 0
      CALL READ_BLOCK( BLOCK,10,NBYTES_RECV,BASE,OFFSET,MORE_TO_COME )
      IF( DEBUG ) CALL TRACK( BLOCK,0 )
C
C LOOK AT THE INDEX FIELD
C
      IF( (BLOCK(1).GE.'A0'X).AND.
     X    (BLOCK(1).LE.'DF'X) ) THEN
        WRITE( UNIT,100 ) BLOCK(1)
100     FORMAT(' Error in SEND_REQUEST, Index(low) is ',z2,'h')
        RETURN
      ELSE IF ( (BLOCK(1).EQ.'FF'X).AND.
     X          (BLOCK(2).EQ.'FF'X) ) THEN
        BYTES(1) = '03'X
        CALL Q_OUTPUT( BYTES,1 )
        NBYTES_RECV = 16
        CALL Q_INPUT( BYTES,NBYTES_RECV )
        GO TO 50
      ENDIF
C
C SEND REQUEST USING CONTROL OR APPLICATION PCB ?
C
      IF( PCBTYPE.EQ.0 ) THEN
         CALL GLUE( BLOCK(3),BLOCK(4),OFFSET )
         CALL GLUE( BLOCK(5),BLOCK(6),BASE )
      ELSE
         CALL GLUE( BLOCK(7),BLOCK(8),OFFSET )
         CALL GLUE( BLOCK(9),BLOCK(10),BASE )
      ENDIF
      CALL READ_BLOCK( BLOCK,43,NBYTES_RECV,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
```

```
C
C TEST THE DBP STATUS FIELD, FIRST
C
      IF( (BLOCK(15).EQ.5).OR.(BLOCK(15).EQ.6)) THEN
140     IF( DEBUG ) WRITE( UNIT,150 ) BLOCK(15)
150     FORMAT(' ** Warning **'/,
     X         ' ** Had to receive a response during ',
     X         'this SEND_REQUEST'/,
     X         ' iDBP Status is ',z2,'h' )
        CALL RECV_RESPONSE( MODULE2,NBYTES_RECV,PCBTYPE,MORE_TO_COME )
        IF( MORE_TO_COME ) GO TO 140
      ENDIF
C
C CAN SEND THE MODULE
C
      NSEGMENTS = BLOCK( 31 )
C
C GO AHEAD AND TAKE CARE OF THE FIRST BUFFER
C
      CALL GLUE( BLOCK(32),BLOCK(33),BUFFER1_OFFSET )
      CALL GLUE( BLOCK(34),BLOCK(35),BUFFER1_BASE )
      CALL GLUE( BLOCK(36),BLOCK(37),BUFFER1_LENGTH )
      IF( NBYTES.LT.BUFFER1_LENGTH ) THEN
        LENGTH = NBYTES
      ELSE
        LENGTH = BUFFER1_LENGTH
      ENDIF
      LEFTOVER = NBYTES - LENGTH
      DO 200 I = 1,LENGTH
200   BUFFER1( I ) = MODULE( I )
C WRITE THE FIRST BUFFER
      CALL WRITE_BLOCK( BUFFER1,LENGTH,BUFFER1_BASE,
     X                  BUFFER1_OFFSET )
      TOTAL_SENT = LENGTH
C
C IF TWO SEGMENTS ARE REQUESTED, SEND THE OTHER BUFFER
C
      IF( NSEGMENTS.EQ.2 ) THEN
        CALL GLUE( BLOCK(38),BLOCK(39),BUFFER2_OFFSET )
        CALL GLUE( BLOCK(40),BLOCK(41),BUFFER2_BASE )
        CALL GLUE( BLOCK(42),BLOCK(43),BUFFER2_LENGTH )
        DO 300 I = 1,LEFTOVER
300     BUFFER2(I) = MODULE( I + LENGTH  )
C WRITE THE SECOND BUFFER
        CALL WRITE_BLOCK( BUFFER2,LEFTOVER,BUFFER2_BASE,
     X                    BUFFER2_OFFSET )
        TOTAL_SENT = TOTAL_SENT + LEFTOVER
        LEFTOVER = 0
      ENDIF
C
C UPDATE THE PCB &
C SET 'REQUEST LENGTH' FIELD
C
      IF( LEFTOVER.GT.0 ) THEN
C
C SEND REQUEST
C BUFFER THIS REQUEST UNTIL THE REST OF THE
C REQUEST DATA CAN BE SENT
C
          BLOCK( 16 ) = 1
```

```
      ELSE
C
C SEND REQUEST WITH EOM
C I.E. THE COMPLETED REQUEST IS SENT
C
          BLOCK( 16 ) = 3
      ENDIF
      CALL LOW16( TOTAL_SENT,BLOCK(29) )
      CALL HIGH16( TOTAL_SENT,BLOCK(30) )
      CALL WRITE_BLOCK( BLOCK,43,BASE,OFFSET )
      IF( DEBUG ) CALL TRACK( BLOCK,1 )
C
C RETURN CONTROL TO DBMS SOFTWARE
C
      BYTES( 1 ) = '47'X
      BYTES( 2 ) = '0D'X
      CALL Q_OUTPUT( BYTES,2 )
      NBYTES_RECV = 29
      CALL Q_INPUT( BYTES,NBYTES_RECV )
C
C CHECK IF THE HOST NEEDS TO SEND ANY
C LEFTOVER BYTES
C
      IF( LEFTOVER.GT.0 ) THEN
        IF( DEBUG ) WRITE( UNIT,600 ) NBYTES-LENGTH
600     FORMAT(/' ** Process ',I3,' leftover bytes **'/)
        DO 750 I = LENGTH+1,NBYTES
750     MODULE(I-LENGTH) = MODULE( I )
        NBYTES = NBYTES - LENGTH
        GO TO 70
      ENDIF
C
      RETURN
      END
      SUBROUTINE READ_BLOCK( BLOCK,NBYTES,NBYTES_RECV,BASE,OFFSET )
C==================================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   READS DATA FROM THE DBP
C
C ARGUMENTS :
C
C   BLOCK      - DATA READ FROM DBP
C   NBYTES     - # OF BYTES READ FROM THE DBP
C   BASE       - BASE PART OF I/O ADDRESS
C   OFFSET     - OFFSET PART OF I/O ADDRESS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
```

```
C   APRIL 12,1983
C
C===========================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 COUNT
      BYTE LOWBYTE,HIGHBYTE
      BYTE INIT( 3 )
      DATA INIT/ '55'X,'52'X,'0D'X /
C
C INITIATE READ
C
50    IF( DEBUG ) WRITE( UNIT,55 )
55    FORMAT(' ** Initiate a READ_BLOCK **' )
      DO 75 I = 1,3
75    BYTES( I )  = INIT( I )
C
C SEND COUNT,OFFSET,AND BASE
C
      CALL LOW16( NBYTES,BYTES(4) )
      CALL HIGH16( NBYTES,BYTES(5) )
      CALL LOW16( OFFSET,BYTES(6) )
      CALL HIGH16( OFFSET,BYTES(7) )
      CALL LOW16( BASE,BYTES(8) )
      CALL HIGH16( BASE,BYTES(9) )
      WRITE( STRING,110 ) (BYTES(I),I=4,9 )
110   FORMAT( 6A1 )
      CRC = LIB$CRC( CRC_TABLE,0,STRING(1:6) )
      CALL LOW32( CRC,BYTES(10) )
      CALL HIGH32( CRC,BYTES(11) )
C
C SEND THE BYTES
C
      CALL Q_OUTPUT( BYTES,11 )
C
C RECEIVE RESPONSE
C
      NBYTES_RECV = NBYTES + 15
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      IF( (BYTES(1).NE.'55'X).OR.
     X    (BYTES(2).NE.'52'X).OR.
     X    (BYTES(3).NE.'0D'X).OR.
     X    (BYTES(4).NE.'0A'X) ) THEN
        IF(DEBUG) WRITE( UNIT,125 ) (BYTES(I),I=1,4)
125     FORMAT(' Error, Expected to find 55h,52h,0Dh,0Ah.'/,
     X         ' Instead found ',z2,'h',3(',',z2,'h') )
        GO TO 50
      ENDIF
C
C CHECK THE REMAINDER OF THE DATA BYTES
C
      WRITE( STRING,150 ) (BYTES(I),I=5,10)
150   FORMAT( 6A1 )
      CRC = LIB$CRC( CRC_TABLE,0,STRING(1:6) )
C
C CHECK CRC-1
C
      CALL LOW32( CRC,LOWBYTE )
      CALL HIGH32( CRC,HIGHBYTE )
      IF( (BYTES(11).NE.LOWBYTE).OR.
```

```
     X     (BYTES(12).NE.HIGHBYTE) ) THEN
C CRC'S DO NOT MATCH
         IF( DEBUG ) WRITE( UNIT,200 ) HIGHBYTE,LOWBYTE,
     X                 BYTES(12),BYTES(11)
200      FORMAT(' Error, CRC16 :',/,
     X            ' Host CRC( High,Low ) : ',z2,1x,z2,/,
     X            ' DBP  CRC( High,Low ) : ',z2,1x,z2/ )
           GO TO 50
      ENDIF
C
C PROCESS REST OF DATA
C
      CALL GLUE( BYTES(5),BYTES(6),NBYTES_RECV )
      DO 400 I = 1,NBYTES_RECV+3
400   BLOCK(I) = BYTES(I+12)
      WRITE( STRING,410 ) (BLOCK(I),I=1,NBYTES_RECV)
410   FORMAT( <NBYTES_RECV>A1 )
      CRC = LIB$CRC( CRC_TABLE,0,STRING(1:NBYTES_RECV) )
C
C CHECK CRC-2
C
      CALL LOW32( CRC,LOWBYTE )
      CALL HIGH32( CRC,HIGHBYTE )
      IF( (BLOCK(NBYTES_RECV+1).NE.LOWBYTE ).OR.
     X     (BLOCK(NBYTES_RECV+2).NE.HIGHBYTE) ) THEN
        IF( DEBUG ) WRITE( UNIT,500 ) HIGHBYTE,LOWBYTE,
     X     BLOCK(NBYTES_RECV+2),BLOCK(NBYTES_RECV+1)
500     FORMAT(' Error, CRC16 :',/,
     X            ' Host CRC( High,Low ) : ',z2,1x,z2,/,
     X            ' DBP  CRC( High,Low ) : ',z2,1x,z2/ )
        GO TO 50
      ENDIF
C
C SUCCESSFUL READ_BLOCK OPERATION
C
      RETURN
      END
      SUBROUTINE WRITE_BLOCK( BLOCK,NBYTES,BASE,OFFSET )
C=======================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   WRITES DATA FROM THE HOST TO THE DBP
C
C ARGUMENTS :
C
C   BLOCK         - DATA TO BE WRITTEN TO THE DBP
C   NBYTES        - # OF BYTES IN 'BLOCK' TO BE SENT
C   BASE          - BASE PART OF I/O ADDRESS
C   OFFSET        - OFFSET PART OF I/O ADDRESS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
```

```
C
C DATE :
C
C   APRIL 12,1983
C
C===========================================================================
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 COUNT
      BYTE INIT( 3 )
      DATA INIT/ '55'X,'57'X,'0D'X /
C
C INITIATE WRITE
C
50    IF( DEBUG ) WRITE( UNIT,60 )
60    FORMAT(' ** Initiate a WRITE_BLOCK **')
      DO 75 I = 1,3
75    BYTES( I ) = INIT( I )
C
C SEND COUNT,OFFSET, AND BASE
C
      CALL LOW16( NBYTES,BYTES(4) )
      CALL HIGH16( NBYTES,BYTES(5) )
      CALL LOW16( OFFSET,BYTES(6) )
      CALL HIGH16( OFFSET,BYTES(7) )
      CALL LOW16( BASE,BYTES(8) )
      CALL HIGH16( BASE,BYTES(9) )
      WRITE( STRING,100 ) ( BYTES(I),I=4,9 )
100   FORMAT( 6A1 )
      CRC = LIB$CRC( CRC_TABLE,0,STRING(1:6) )
      CALL LOW32( CRC,BYTES(10) )
      CALL HIGH32( CRC,BYTES(11) )
C
C SEND THE BYTES
C
      CALL Q_OUTPUT( BYTES,11 )
C
C RECEIVE ACKNOWLEDGMENT
C
      NBYTES_RECV = NBYTES + 15
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      IF( (BYTES(1).EQ.'55'X).AND.
     X    (BYTES(2).EQ.'57'X).AND.
     X    (BYTES(3).EQ.'0D'X).AND.
     X    (BYTES(4).EQ.'0A'X) ) THEN
        IF( BYTES(5).NE.'06'X ) THEN
          IF(DEBUG) WRITE( UNIT,200 ) BYTES( 5 )
200       FORMAT(' Error, WRITE_BLOCK 1st Receive Ack.'/,
     X           ' Expecting to find 06h, instead found ',z2,'h')
          GO TO 50
        ENDIF
      ELSE
        IF(DEBUG) WRITE( UNIT,300 ) (BYTES(I),I=1,4)
300     FORMAT(' Error, WRITE_BLOCK 1st Receive Ack.'/,
     X         ' Expecting to find 55h,57h,0Dh,0Ah.' /,
     X         ' Instead found ',z2,'h',3(',',z2,'h') )
      GO TO 50
      ENDIF
C
C SEND DATA
C
```

```
      CRC = 0
      IF( NBYTES.EQ.0 ) GO TO 650
C
C BUFFER THE CRC
C
      WRITE( STRING,625 ) (BLOCK(I),I=1,NBYTES)
625   FORMAT(<NBYTES>A1 )
      CRC = LIB$CRC( CRC_TABLE,0,STRING(1:NBYTES) )
650   CALL LOW32( CRC,BLOCK(NBYTES+1) )
      CALL HIGH32( CRC,BLOCK(NBYTES+2) )
      CALL Q_OUTPUT( BLOCK,NBYTES+2 )
C
C RECEIVE ACKNOWLEDGEMENT
C
      NBYTES_RECV = 2
      CALL Q_INPUT( BYTES,NBYTES_RECV )
      IF( BYTES(1).NE.'06'X ) THEN
        IF(DEBUG) WRITE( UNIT,700 ) BYTES(1)
700     FORMAT(' Error, in WRITE_BLOCK 2nd Receive Ack.'/,
     X         ' Expecting 06h, instead found ',z2,'h' )
        GO TO 50
      ENDIF
C
C SUCCESSFUL WRITE_BLOCK OPERATION
C
      RETURN
      END
      SUBROUTINE Q_INPUT( BYTES,NBYTES_RECV )
C=====================================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   QUEUE A SEQUENCE OF BYTES TO THE INPUT CHANNEL
C   'Q_INPUT' WAITS UNTIL DATA APPEARS ON THE CHANNEL
C
C ARGUMENTS :
C
C   BYTES      - THE ARRAY( SEQUENCE ) OF BYTES RECEIVED
C   NBYTES_RECV    - THE NUMBER OF BYTES TO RECEIVE &
C                    THE NUMBER OF ACTUAL BYTES RECEIVED
C
C
C NOTE :
C
C   Q_INPUT WAITS FOR THE DBP TO SEND 'NBYTES_RECV' BYTES.
C   IF 'NBYTES_RECV' BYTES HAVE NOT BEEN SENT BY THE TIME
C   THAT THE TIME-OUT VALUE( CURRENTLY 5 SECONDS ) HAS
C   OCCURRED, THE ROUTINE EXITS WITH THE DATA THAT WAS
C   RECEIVED.
C
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
```

```
C   PHYSICAL
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*4 SYS$QIOW,TERMINATOR(2),TIME_OUT
      BYTE PRBYTES( 1024 ), MASK( 6 )
C
      TIME_OUT = 1
C
C SET UP THE TERMINATOR BYTES
C
      TERMINATOR(1) = 0
      TERMINATOR(2) = 0
C
C INITIATE THE INPUT OPERATION
C ( WAIT FOR THE DBP TO SPEAK )
C
5     IOSB(2) = 0
      STATUS = SYS$QIOW( ,%VAL( TTY_CHANNEL ),
     X   %VAL(IO$_TTYREADALL+IO$M_NOECHO+IO$M_TIMED),
     X   IOSB,,,BYTES(1),%VAL(NBYTES_RECV),%VAL(5),TERMINATOR,,)
      IF( STATUS.NE.SS$_NORMAL ) THEN
        WRITE( UNIT,10 ) STATUS
10      FORMAT(' Error, Q_INPUT failure.'/,
     X         ' Return Status is ',z8 )
      ENDIF
      NBYTES_RECV = IOSB(2)
      IF( NBYTES_RECV.EQ.0 ) THEN
         IF( TIME_OUT.EQ.10 ) THEN
            IF( DEBUG ) WRITE( UNIT,18 )
18          FORMAT(' --- Max Time Out''s Encountered ---')
            RETURN
         ELSE
C
C RETURN TO GET INPUT ONCE MORE
C
            IF( DEBUG ) WRITE( UNIT,20 ) TIME_OUT
20          FORMAT(' --- Time Out # ',I2,' ---' )
            TIME_OUT = TIME_OUT + 1
            GO TO 5
         ENDIF
      ENDIF
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C NOTE: NON-PRINTABLE CHARACTERS ARE DENOTED
C WITH A PERIOD( '2E'X )
C
C
      DO 50 I = 1,NBYTES_RECV
      IF( (BYTES(I).LT.'20'X).OR.
     X    (BYTES(I).GT.'7E'X)) THEN
         PRBYTES(I) = '2E'X
      ELSE
         PRBYTES(I) = BYTES(I)
```

```
      ENDIF
50    CONTINUE
      WRITE( UNIT,100 ) NBYTES_RECV
100   FORMAT(' == Q_INPUT =='/' # of bytes is ',I5,
     X       /,' Byte Stream :'/ )
      MULTIPLE16 = ( NBYTES_RECV/16 )*16
      LEFTOVER   = NBYTES_RECV - MULTIPLE16
      IF( MULTIPLE16.GT.0 ) THEN
        DO 200 I = 1,MULTIPLE16,16
        WRITE( UNIT,150 ) (BYTES(I1),I1=I,I+15),(PRBYTES(I2),I2=I,I+15)
150     FORMAT(16(1X,Z2.2),2X,16A1)
200     CONTINUE
      ENDIF
      IF( LEFTOVER.GT.0 ) THEN
        WRITE( UNIT,250 ) (BYTES(I1),I1=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER)
250     FORMAT( <LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X           <LEFTOVER>A1 )
      ENDIF
      WRITE( UNIT,400 )
400   FORMAT(/)
      ENDIF
      RETURN
      END
      SUBROUTINE Q_OUTPUT( BYTES,NBYTES )
C===========================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   QUEUE A SEQUENCE OF BYTES TO THE OUTPUT TTY CHANNEL
C
C ARGUMENTS :
C
C   BYTES     - THE ARRAY( SEQUENCE ) OF BYTES TO BE TRANSFERRED
C   NBYTES    - # OF BYTES TO BE TRANSFERRED IN ARRAY 'BYTES'
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
C   APRIL 12,1983
C
C===========================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*4 SYS$QIOW
      BYTE PRBYTES( 1024 )
C
C INITIATE THE OUTPUT OPERATION
C ( TALK TO THE DBP )
C
```

```
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C
      DO 50 I = 1,NBYTES
      IF( (BYTES(I).LT.'20'X).OR.
     X    (BYTES(I).GT.'7E'X)) THEN
         PRBYTES(I) = '2E'X
      ELSE
         PRBYTES(I) = BYTES(I)
      ENDIF
50    CONTINUE
      WRITE( UNIT,90 ) NBYTES
90    FORMAT(' == Q_OUTPUT =='/' # of bytes is ',I5,
     X       /,' Byte Stream :'/ )
      MULTIPLE16 = (NBYTES/16)*16
      LEFTOVER   = NBYTES - MULTIPLE16
      IF( MULTIPLE16.GT.0 ) THEN
        DO 200 I = 1,MULTIPLE16,16
        WRITE( UNIT,150 ) (BYTES(I1),I1=I,I+15),(PRBYTES(I2),I2=I,I+15)
150     FORMAT(16(1X,Z2.2),2X,16A1)
200     CONTINUE
      ENDIF
      IF( LEFTOVER.GT.0 ) THEN
        WRITE( UNIT,250 )(BYTES(I1),I1=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER)
250     FORMAT(<LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X          <LEFTOVER>A1 )
      ENDIF
      WRITE( UNIT,300 )
300   FORMAT(/)
C
      ENDIF
      STATUS = SYS$QIOW( , %VAL(TTY_CHANNEL),
     X                    %VAL(IO$_WRITEVBLK),IOSB,,,
     X                    BYTES(1),%VAL(NBYTES),,%VAL(0),, )
      IF( STATUS.NE.SS$_NORMAL ) THEN
        WRITE( UNIT,400 ) STATUS
400     FORMAT(' Error, Q_OUTPUT failure.',/,
     X         ' Return Status is ',z8 )
      ENDIF
      RETURN
      END
```

®

```
$ @typeq
      SUBROUTINE LOW16( WORD16,LOWBYTE )
C================================================================
C
C PURPOSE :
C
C   RETURN LOW ORDER BYTE FROM 16 BIT WORD
C
C ARGUMENTS :
C
C   WORD16    - 16 BIT WORD
C   LOWBYTE   - LOW ORDER 8 BITS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
C   APRIL 12,1983
C
C================================================================
C
      BYTE LOWBYTE,WORD16(2)
      LOWBYTE = WORD16(1)
      RETURN
      END
      SUBROUTINE LOW32( WORD32,LOWBYTE )
C================================================================
C
C PURPOSE :
C
C   RETURN LOW ORDER BYTE FROM 32 BIT WORD
C
C ARGUMENTS :
C
C   WORD32    - 32 BIT WORD
C   LOWBYTE   - LOW ORDER 8 BITS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
C   APRIL 12,1983
C
C================================================================
C
      BYTE LOWBYTE,WORD32(4)
      LOWBYTE = WORD32(1)
```

```
      RETURN
      END
      SUBROUTINE HIGH16( WORD16,HIGHBYTE )
C=======================================================================
C
C PURPOSE :
C
C   RETURN HIGH ORDER BYTE FROM 16 BIT WORD
C
C ARGUMENTS :
C
C   WORD16   - 16 BIT WORD
C   HIGHBYTE -  HIGH ORDER 8 BITS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
C   APRIL 12,1983
C
C=======================================================================
C
      BYTE HIGHBYTE,WORD16(2)
      HIGHBYTE = WORD16(2)
      RETURN
      END
      SUBROUTINE HIGH32( WORD32,HIGHBYTE )
C=======================================================================
C
C PURPOSE :
C
C   RETURN HIGH ORDER BYTE FROM LOWER HALF OF A
C   32-BIT WORD
C
C ARGUMENTS :
C
C   WORD32   - 32 BIT WORD
C   HIGHBYTE -  HIGH ORDER 8 BITS
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   DATA LINK
C
C DATE :
C
C   APRIL 12,1983
C
C=======================================================================
C
      BYTE HIGHBYTE,WORD32(4)
```

```
      HIGHBYTE = WORD32(2)
      RETURN
      END
      SUBROUTINE GLUE( LOWBYTE,HIGHBYTE,GLUED )
C=====================================================================
C
C PURPOSE :
C
C   GLUE TWO BYTES TOGETHER TO FORM A 16-BIT WORD
C
C ARGUMENTS :
C
C   LOWBYTE     - LOW ORDER 8 BITS
C   HIGHBYTE    - HIGH ORDER 8 BITS
C   GLUED       - 16-BIT WORD
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
      BYTE LOWBYTE,HIGHBYTE,GLUED(2)
      GLUED(1) = LOWBYTE
      GLUED(2) = HIGHBYTE
      RETURN
      END
      SUBROUTINE TRACE_START( TRACE_UNIT )
C=====================================================================
C
C PURPOSE :
C
C   INITIALIZE A FILE FOR DIAGNOSTIC TRACE OUTPUT
C
C ARGUMENTS :
C
C   TRACE_UNIT     - LOGICAL OUTPUT UNIT FOR TRACE INFORMATION
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL : TRACE UTILITY
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================================
C
      INCLUDE 'SPPCOM.TXT'
```

```
C
C OPEN A DEBUG FILE, IF WE ARE NOT TALKING
C TO THE TERMINAL
C
      IF( UNIT.NE.6 ) OPEN( UNIT=TRACE_UNIT,FILE='TRACE.DBP',
     X                      STATUS='NEW' )
      UNIT = TRACE_UNIT
      DEBUG = .TRUE.
C
      RETURN
      END
      SUBROUTINE TRACE_STOP
C=======================================================
C
C PURPOSE :
C
C   STOP THE TRACE OUTPUT
C
C ARGUMENTS :
C
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL : TRACE UTILITY
C
C DATE :
C
C   APRIL 12,1983
C
C=======================================================
C
      INCLUDE 'SPPCOM.TXT'
C
      DEBUG = .FALSE.
C
      RETURN
      END
      SUBROUTINE PERFORM_START
C=======================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   START TRACKING THE FOLLOWING PERFORMANCE STATISTICS :
C
C      1. VAX CPU TIME ELAPSED
C      2. VAX CLOCK TIME ELAPSED
C      3. VAX BUFFERED I/O
C      4. VAX DIRECT I/O
C      5. VAX PAGE FAULT COUNT
C
C
C ARGUMENTS :
C
```

```
C   NONE
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL : PERFORMANCE UTILITY
C
C DATE :
C
C   APRIL 12,1983
C
C==================================================
C
      INTEGER*4 BUFIO,CPUTIME,DIO,PAGEF
      INTEGER*4 BUFIO_ADR,CPUTIME_ADR,DIO_ADR,PAGEF_ADR
      INTEGER*4 ZERO1,ZERO2,ZERO3,ZERO4,ZERO5
      INTEGER*4 SYS$GETJPI,STATUS
C
      INTEGER*2 LENGTH1,LENGTH2,LENGTH3,LENGTH4
      INTEGER*2 BUFIO_CODE,CPUTIME_CODE,DIO_CODE,PAGEF_CODE
C
      COMMON/STATCOM/ CLOCK_TIME,BUFIO,CPUTIME,DIO,PAGEF
      COMMON/JPICOM/ LENGTH1,BUFIO_CODE,BUFIO_ADR,ZERO1,
     X               LENGTH2,CPUTIME_CODE,CPUTIME_ADR,ZERO2,
     X               LENGTH3,DIO_CODE,DIO_ADR,ZERO3,
     X               LENGTH4,PAGEF_CODE,PAGEF_ADR,ZERO4,ZERO5
      DATA BUFIO_CODE/ 1036 /
      DATA CPUTIME_CODE/ 1031 /
      DATA DIO_CODE/ 1035 /
      DATA PAGEF_CODE/ 1034 /
      DATA LENGTH1,LENGTH2,LENGTH3,LENGTH4/4,4,4,4/
C
C INITIALIZE THE STATISTIC VARIABLES
C
      CLOCK_TIME = SECNDS( 0.0 )
      BUFIO_ADR  = %LOC( BUFIO )
      CPUTIME_ADR= %LOC( CPUTIME )
      DIO_ADR    = %LOC( DIO )
      PAGEF_ADR  = %LOC( PAGEF )
C
C GET THE PROCESS INFORMATION
C
      STATUS = SYS$GETJPI( ,,,LENGTH1,,,)
      IF( STATUS.NE.1 ) WRITE( 6,100 ) STATUS
100   FORMAT(' Error with SYS$GETJPI, status is ',Z8,'h' )
C
      RETURN
      END
      SUBROUTINE PERFORM_STOP( NEW_CLOCK,NEW_CPU,NEW_BUFF,
     X                         NEW_DIRECT,NEW_PAGE )
C==================================================
C
C *** MACHINE DEPENDENT ***
C
C PURPOSE :
C
C   STOP THE TRACKING OF THE PERORMANCE STATISTICS
```

```
C   AND RETURN THE VALUES
C
C
C ARGUMENTS :
C
C   CLOCK     -   VAX CLOCK TIME ELAPSED
C   CPU       -   VAX CPU TIME ELAPSED
C   BUFFERED -    VAX BUFFERED I/O
C   DIRECT    -   VAX DIRECT I/O
C   PAGE      -   VAX PAGE FAULT COUNT
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL : PERFORMANCE UTILITY
C
C DATE :
C
C   APRIL 12,1983
C
C=====================================================
C
      INTEGER*4 BUFIO,CPUTIME,DIO,PAGEF
      INTEGER*4 BUFFERED,CPU_INT,DIRECT,PAGE
      INTEGER*4 NEW_BUFF,NEW_DIRECT,NEW_PAGE
      REAL NEW_CLOCK,NEW_CPU
      INTEGER*4 BUFIO_ADR,CPUTIME_ADR,DIO_ADR,PAGEF_ADR
      INTEGER*4 ZERO1,ZERO2,ZERO3,ZERO4,ZERO5
      INTEGER*4 SYS$GETJPI,STATUS
C
      INTEGER*2 LENGTH1,LENGTH2,LENGTH3,LENGTH4,LENGTH5
      INTEGER*2 BUFIO_CODE,CPUTIME_CODE,DIO_CODE,PAGEF_CODE
C
      COMMON/STATCOM/ CLOCK_TIME,BUFIO,CPUTIME,DIO,PAGEF
      COMMON/JPICOM/ LENGTH1,BUFIO_CODE,BUFIO_ADR,ZERO1,
     X              LENGTH2,CPUTIME_CODE,CPUTIME_ADR,ZERO2,
     X              LENGTH3,DIO_CODE,DIO_ADR,ZERO3,
     X              LENGTH4,PAGEF_CODE,PAGEF_ADR,ZERO4,ZERO5
      DATA BUFIO_CODE/ 1036 /
      DATA CPUTIME_CODE/ 1031 /
      DATA DIO_CODE/ 1035 /
      DATA PAGEF_CODE/ 1034 /
      DATA LENGTH1,LENGTH2,LENGTH3,LENGTH4/4,4,4,4/
C
C DETERMINE THE STATISTICS
C
      BUFIO_ADR   = %LOC( BUFFERED )
      CPUTIME_ADR= %LOC( CPU_INT )
      DIO_ADR     = %LOC( DIRECT )
      PAGEF_ADR   = %LOC( PAGE )
C
      STATUS = SYS$GETJPI( ,,,LENGTH1,,, )
      IF( STATUS.NE.1 ) WRITE( 6,100 ) STATUS
100   FORMAT(' Error, SYS$GETJPI, status is ',z8,'h' )
C
C RETURN THE APPROPRIATE STATISTICS
```

```
C
      NEW_CLOCK = SECNDS( CLOCK_TIME )
      NEW_CPU   = ( CPU_INT - CPUTIME )/100.0
      NEW_DIRECT= DIRECT - DIO
      NEW_PAGE  = PAGE - PAGEF
      NEW_BUFF  = BUFFERED - BUFIO
C
      RETURN
      END
      SUBROUTINE TRACK( BLOCK,DATA_TYPE )
C===========================================================
C
C PURPOSE :
C
C   DISPLAY THE FORMAT OF THE REQUESTED DATA STRUCTURE
C
C   TWO DATA STRUCTURES ARE DISPLAYED -
C
C       1.) PCB VECTOR
C       2.) PCB
C
C
C ARGUMENTS :
C
C   BLOCK       - THE ARRAY CONTAINING THE DATA
C   DATA_TYPE   - THE DATA STRUCTURE TYPE
C
C               = 0 IF PCB VECTOR
C               = 1 IF PCB
C
C
C PROTOCOL :
C
C   SERVICE PORT
C
C LAYER :
C
C   ALL
C
C DATE :
C
C   APRIL 12,1983
C
C===========================================================
C
      INCLUDE 'SPPCOM.TXT'
      INTEGER*2 REQUEST_LENGTH
      INTEGER*2 BUFFER1_LENGTH,BUFFER2_LENGTH
      INTEGER*4 DBP_STATUS(4),HOST_STATUS(6),DATA_TYPE
      CHARACTER*40 DBP_MESSAGE(4),HOST_MESSAGE(6),DBP,HOST
      DATA DBP_STATUS
     X /4,5,6,7/
      DATA HOST_STATUS
     X /0,1,2,3,5,17/
      DATA DBP_MESSAGE
     X /'WAIT ON ENABLE',
     X   'READ RESPONSE',
     X   'READ RESPONSE WITH EOM',
     X   'WRITE REQUEST' /
      DATA HOST_MESSAGE
```

```
     X  /'SUSPEND SESSION',
     X   'READ/WRITE OK',
     X   'ERROR ENCOUNTERED',
     X   'WRITE OK WITH EOM',
     X   'OK FIN',
     X   'ENABLE SERVICE PORT' /
C
C DETERMINE THE NECESSARY DECIMAL VALUES
C
      CALL GLUE( BLOCK(29),BLOCK(30),REQUEST_LENGTH )
      CALL GLUE( BLOCK(36),BLOCK(37),BUFFER1_LENGTH )
      CALL GLUE( BLOCK(42),BLOCK(43),BUFFER2_LENGTH )
C
C OUTPUT THE PCB VECTOR OR PCB
C
      IF( DATA_TYPE.EQ.1 ) THEN
C
C PROCESS A PCB DATA STRUCTURE
C
        DO 50 I = 1,4
50      IF( BLOCK(15).EQ.DBP_STATUS(I)) GO TO 75
        DBP = 'UNKNOWN DBP STATUS'
        GO TO 80
75      DBP = DBP_MESSAGE(I)
80      DO 100 I = 1,6
100     IF( BLOCK(16).EQ.HOST_STATUS(I)) GO TO 125
        HOST = 'UNKNOWN HOST STATUS'
        GO TO 130
125     HOST = HOST_MESSAGE(I)
130     WRITE( UNIT,200 ) (BLOCK(I1),I1=1,14),BLOCK(15),DBP,
     X        BLOCK(16),HOST,(BLOCK(I2),I2=17,28),REQUEST_LENGTH,
     X        BLOCK(31),(BLOCK(I3),I3=35,32,-1),
     X        BUFFER1_LENGTH,(BLOCK(I4),I4=41,38,-1),
     X        BUFFER2_LENGTH
200     FORMAT(' +-------------------+'/,
     X          ' ¶         PCB        ¶'/,
     X          ' +-------------------+'//,
     X          ' RESERVED',T25,14(Z2.2,1X),/,
     X          ' iDBP STATUS',T25,Z2.2,1X,A,/,
     X          ' HOST STATUS',T25,Z2.2,1X,A,/,
     X          ' RESERVED',T25,12(Z2.2,1X),/,
     X          ' REQUEST LENGTH',T25,I4,/,
     X          ' NUMBER OF SEGMENTS',T25,I1,/,
     X          ' BUFFER 1 PTR',T25,4(Z2.2)/,
     X          ' BUFFER 1 LENGTH',T25,I4,/,
     X          ' BUFFER 2 PTR',T25,4(Z2.2)/,
     X          ' BUFFER 2 LENGTH',T25,I4,//)
       ELSE
C
C PROCESS A PCB VECTOR DATA STRUCTURE
C
        WRITE( UNIT,300 ) (BLOCK(I1),I1=2,1,-1),
     X          (BLOCK(I2),I2=6,3,-1),(BLOCK(I3),I3=10,7,-1)
300     FORMAT(' +-------------------+'/,
     X          ' ¶    PCB    VECTOR   ¶'/,
     X          ' +-------------------+'//,
     X          ' INDEX ',T30,2Z2.2,/,
     X          ' CONTROL PCB ADDRESS',T30,4Z2.2,/,
     X          ' APPLICATION PCB ADDRESS',T30,4Z2.2,// )
       ENDIF
```

```
      RETURN
      END
$ !
$ ! THIS IS THE COMMAND FILE USED TO RUN PROGRAM 'SPP'
$ ! THE VMS TTY PORT 'TTB0:' IS USED FOR COMMUNICATIONS
$ !
$ DBPTERM := _TTB0:
$ ! ALLOCATE THE PORT FOR ACCESS
$ ALLOCATE 'DBPTERM'
$ SET PROTECTION=(W:RW)/DEVICE 'DBPTERM'
$ !
$ ! SET TERMINAL CHARACTERISTICS FOR TTB0:
$ ! SEE FIGURE 2 OF THIS REPORT
$ !
$ SET TERMINAL 'DBPTERM'/NOWRAP/WIDTH=80/SPEED=9600/PASSALL/EIGHT_BIT/PERM
$ ASSIGN/USER 'DBPTERM' REMOTE
$ ASSIGN/USER TT: SYS$INPUT
$ RUN [INTEL.SPP]SPP
$ ! DEALLOCATE TTB0:
$ DEALLOCATE 'DBPTERM'
$
```

APPENDIX B

A sample transmission trace

```
$ type trace.dbp
** Initialize iDBP Communications **
== Q_OUTPUT ==
# of bytes is     1
Byte Stream :

03


== Q_INPUT ==
# of bytes is    16
Byte Stream :

0D 0A 2A 43 6F 6E 74 72 6F 6C 20 43 2A 0D 0A 2E  ..*Control C*...


** Create Control Session **
** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 0A 00 00 00 0C EE 85 E6                 UR.........


== Q_INPUT ==
# of bytes is    25
Byte Stream :

55 52 0D 0A 0A 00 00 00 0C EE 85 E6 00 00 00 00  UR..............
4D 98 00 00 00 00 2E 01 2E                       M........


+--------------------+
|   PCB    VECTOR    |
+--------------------+

INDEX                           0000
CONTROL PCB ADDRESS             984D0000
APPLICATION PCB ADDRESS         00000000


** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 2B 00 00 00 4D 98 32 E1                 UR.+...M.2.


== Q_INPUT ==
# of bytes is    58
Byte Stream :

55 52 0D 0A 2B 00 00 00 4D 98 32 E1 00 00 00 00  UR..+...M.2.....
00 00 00 00 00 00 00 00 00 00 04 00 00 02 00 00  ................
00 00 00 00 00 04 00 00 00 00 01 BD 1F 03 00 80  ................
00 FF FF FF 00 00 00 B3 51 2E                    ........Q.
```

```
+--------------------+
|        PCB         |
+--------------------+

RESERVED               00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS            04 WAIT ON ENABLE
HOST STATUS            00 SUSPEND SESSION
RESERVED               00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH            0
NUMBER OF SEGMENTS     1
BUFFER 1 PTR           00031FBD
BUFFER 1 LENGTH         128
BUFFER 2 PTR           00FFFFFF
BUFFER 2 LENGTH           0


** Initiate a WRITE_BLOCK **
== Q_OUTPUT ==
# of bytes is     11
Byte Stream :

55 57 0D 2B 00 00 00 4D 98 32 E1                      UW.+...M.2.


== Q_INPUT ==
# of bytes is      5
Byte Stream :

55 57 0D 0A 06                                        UW...


== Q_OUTPUT ==
# of bytes is     45
Byte Stream :

00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 11       ................
00 02 00 00 00 00 00 00 00 04 00 00 00 00 01 BD       ................
1F 03 00 80 00 FF FF FF 00 00 00 9E 51                ............Q


== Q_INPUT ==
# of bytes is      2
Byte Stream :

06 2E                                                 ..


+--------------------+
|        PCB         |
+--------------------+

RESERVED               00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS            04 WAIT ON ENABLE
HOST STATUS            11 ENABLE SERVICE PORT
RESERVED               00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH            0
NUMBER OF SEGMENTS     1
BUFFER 1 PTR           00031FBD
BUFFER 1 LENGTH         128
```

```
BUFFER 2 PTR            00FFFFFF
BUFFER 2 LENGTH            0


== Q_OUTPUT ==
# of bytes is     2
Byte Stream :

47 0D                                               G.


== Q_INPUT ==
# of bytes is    29
Byte Stream :

47 0D 0A 0D 0A 2A 42 52 45 41 4B 2A 20 61 74 20  G....*BREAK* at
33 30 41 36 3A 30 33 42 41 20 0D 0A 2E           30A6:03BA ...


** Create Application Session **
** Send Request **
** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 0A 00 00 00 0C EE 85 E6                 UR.........


== Q_INPUT ==
# of bytes is    25
Byte Stream :

55 52 0D 0A 0A 00 00 00 0C EE 85 E6 00 00 00 00  UR..............
4D 98 00 00 00 00 2E 01 2E                       M........


+--------------------+
|   PCB    VECTOR    |
+--------------------+

INDEX                         0000
CONTROL PCB ADDRESS           984D0000
APPLICATION PCB ADDRESS       00000000


** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 2B 00 00 00 4D 98 32 E1                 UR.+...M.2.


== Q_INPUT ==
# of bytes is    58
Byte Stream :

55 52 0D 0A 2B 00 00 00 4D 98 32 E1 00 00 00 00  UR..+...M.2.....
00 00 00 00 00 00 00 00 00 00 07 00 00 02 00 00  ................
```

```
00 00 00 00 00 04 00 00 00 00 01 B9 1F 00 30 84   ..............0.
00 FF FF 00 F0 00 00 97 46 2E                     ........F.


+---------------------+
|         PCB         |
+---------------------+

RESERVED                 00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS              07 WRITE REQUEST
HOST STATUS              00 SUSPEND SESSION
RESERVED                 00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH              0
NUMBER OF SEGMENTS       1
BUFFER 1 PTR             30001FB9
BUFFER 1 LENGTH           132
BUFFER 2 PTR             F000FFFF
BUFFER 2 LENGTH             0


** Initiate a WRITE_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 57 0D 09 00 B9 1F 00 30 14 17                  UW......0..


== Q_INPUT ==
# of bytes is     5
Byte Stream :

55 57 0D 0A 06                                    UW...


== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

01 00 00 00 E4 01 FE FF 00 5C 7A                  .........\z


== Q_INPUT ==
# of bytes is     2
Byte Stream :

06 2E                                             ..


** Initiate a WRITE_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 57 0D 2B 00 00 00 4D 98 32 E1                  UW.+...M.2.


== Q_INPUT ==
# of bytes is     5
Byte Stream :
```

```
55 57 0D 0A 06                                  UW...


== Q_OUTPUT ==
# of bytes is    45
Byte Stream :

00 00 00 00 00 00 00 00 00 00 00 00 00 00 07 03  ................
00 02 00 00 00 00 00 00 00 04 00 00 09 00 01 B9  ................
1F 00 30 84 00 FF FF 00 F0 00 00 58 80           ..0........X.


== Q_INPUT ==
# of bytes is     2
Byte Stream :

06 2E                                            ..


+---------------------+
|         PCB         |
+---------------------+

RESERVED               00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS            07 WRITE REQUEST
HOST STATUS            03 WRITE OK WITH EOM
RESERVED               00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH            9
NUMBER OF SEGMENTS     1
BUFFER 1 PTR           30001FB9
BUFFER 1 LENGTH         132
BUFFER 2 PTR           F000FFFF
BUFFER 2 LENGTH           0


== Q_OUTPUT ==
# of bytes is     2
Byte Stream :

47 0D                                            G.


== Q_INPUT ==
# of bytes is    29
Byte Stream :

47 0D 0A 0D 0A 2A 42 52 45 41 4B 2A 20 61 74 20  G....*BREAK* at
33 30 41 36 3A 30 33 42 41 20 0D 0A 2E           30A6:03BA ...


** Receive Response **
** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 0A 00 00 00 0C EE 85 E6                 UR.........
```

```
== Q_INPUT ==
# of bytes is    25
Byte Stream :

55 52 0D 0A 0A 00 00 00 0C EE 85 E6 00 00 00 00  UR..............
4D 98 00 00 C5 68 7C BF 2E                       M....h|..


+---------------------+
|    PCB    VECTOR    |
+---------------------+

INDEX                        0000
CONTROL PCB ADDRESS          984D0000
APPLICATION PCB ADDRESS      68C50000


** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 2B 00 00 00 4D 98 32 E1                 UR.+...M.2.


== Q_INPUT ==
# of bytes is    58
Byte Stream :

55 52 0D 0A 2B 00 00 00 4D 98 32 E1 00 00 00 00  UR..+...M.2.....
00 00 00 00 00 00 00 00 00 00 06 00 00 02 00 00  ................
00 00 00 00 00 04 00 00 09 00 01 B9 1F 00 30 0B  ..............0.
00 FF FF 00 F0 00 00 D6 61 2E                    ........a.


+---------------------+
|         PCB         |
+---------------------+

RESERVED                00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS             06 READ RESPONSE WITH EOM
HOST STATUS             00 SUSPEND SESSION
RESERVED                00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH             9
NUMBER OF SEGMENTS      1
BUFFER 1 PTR            30001FB9
BUFFER 1 LENGTH           11
BUFFER 2 PTR            F000FFFF
BUFFER 2 LENGTH            0


** Initiate a READ_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 52 0D 0B 00 B9 1F 00 30 15 F5                 UR......0..


== Q_INPUT ==
```

```
# of bytes is    26
Byte Stream :

55 52 0D 0A 0B 00 B9 1F 00 30 15 F5 01 00 00 00  UR.......0......
FC 03 E4 01 00 FF 00 08 DA 2E                    ..........


** Initiate a WRITE_BLOCK **
== Q_OUTPUT ==
# of bytes is    11
Byte Stream :

55 57 0D 2B 00 00 00 4D 98 32 E1                 UW.+...M.2.


== Q_INPUT ==
# of bytes is     5
Byte Stream :

55 57 0D 0A 06                                   UW...


== Q_OUTPUT ==
# of bytes is    45
Byte Stream :

00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 01  ................
00 02 00 00 00 00 00 00 00 04 00 00 09 00 01 B9  ................
1F 00 30 0B 00 FF FF 00 F0 00 00 D5 E1           ..0..........


== Q_INPUT ==
# of bytes is     2
Byte Stream :

06 2E                                            ..


+----------------------+
|         PCB          |
+----------------------+

RESERVED               00 00 00 00 00 00 00 00 00 00 00 00 00 00
iDBP STATUS            06 READ RESPONSE WITH EOM
HOST STATUS            01 READ/WRITE OK
RESERVED               00 02 00 00 00 00 00 00 00 04 00 00
REQUEST LENGTH            9
NUMBER OF SEGMENTS     1
BUFFER 1 PTR           30001FB9
BUFFER 1 LENGTH          11
BUFFER 2 PTR           F000FFFF
BUFFER 2 LENGTH           0


== Q_OUTPUT ==
# of bytes is     2
Byte Stream :

47 0D                                            G.
```

```
== Q_INPUT ==
# of bytes is    29
Byte Stream :

47 0D 0A 0D 0A 2A 42 52 45 41 4B 2A 20 61 74 20  G....*BREAK* at
33 30 41 36 3A 30 33 42 41 20 0D 0A 2E           30A6:03BA ...


** All data has been received **
** Create Application Response **

FC 03 E4 01 00 FF 00
== Q_OUTPUT ==
# of bytes is     2
Byte Stream :

47 0D                                            G.


== Q_INPUT ==
# of bytes is    29
Byte Stream :

47 0D 0A 0D 0A 2A 42 52 45 41 4B 2A 20 61 74 20  G....*BREAK* at
33 30 41 36 3A 30 33 42 41 20 0D 0A 2E           30A6:03BA ...


$
```

APPENDIX C - DBPSSP Source

DBPSSP has been implemented using VAX VMS FORTRAN 77. The file "DBPSSP.FOR" contains all of the high-order procedures which will be used most often.

```
$ @typeq
C=======================================================
C
C CONTENTS :
C
C   THIS FILE CONTAINS A SET OF ASSEMBLY TOOLS NECESSARY
C   TO EFFICIENTLY CONSTRUCT REQUEST MODULES FOR THE
C   DBP. EACH PROCEDURE ACTIVATES ONE OR MORE PRIMITIVE
C   ASSEMBLY PROCEDURES.
C
C DATE :
C
C   APRIL 20,1983
C
C=======================================================
C
      SUBROUTINE INIT
C
C INITIALIZE DBP COMMUNICATIONS VIA
C SPP( SERVICE PORT PROTOCOL )
C
      CALL INIT_COMM
      RETURN
      END
      SUBROUTINE START
C
C START ENCODING A REQUEST MODULE
C
      CALL DBP_BEGIN
      RETURN
      END
      SUBROUTINE BITSB
C
C PREPARE FOR INSERTING AN 'OR'ED VALUE WITHIN
C THE REQUEST MODULE
C
      CALL DBP_BITS_BEGIN(-1)
      RETURN
      END
      SUBROUTINE BITSB_A( OFFSET )
C
C *** ABSOLUTE OFFSET ***
C PREPARE FOR INSERTING AN 'OR'ED VALUE WITHIN
C THE REQUEST MODULE
C
      CALL DBP_BITS_BEGIN( OFFSET )
      RETURN
      END
      SUBROUTINE BITS( BYTE_VALUE )
C
C PERFORM AN 'OR' OPERATION OF 'BYTE_VALUE' ON THE
C CURRENT BYTE WITHIN THE REQUEST MODULE.
C
      BYTE BYTE_VALUE
      CALL DBP_BITS( BYTE_VALUE )
      RETURN
      END
      SUBROUTINE BITSE
C
```

```
C STOP THE 'OR'ING PROCESS FOR THE CURRENT BYTE BEING
C FORMED WITHIN THE REQUEST MODULE
C
      CALL DBP_BITS_END
      RETURN
      END
      SUBROUTINE ASC( STRING,LENGTH )
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C ALSO PLACE THE 'LENGTH' DIRECTLY IN FRONT OF THE
C ASCII BYTES
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH
C
      CALL DBP_INTEGER( -1,LENGTH,1 )
      CALL DBP_BYTES( -1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASCX( STRING,LENGTH )
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C DO NOT PLACE THE 'LENGTH' WITHIN THE REQUEST MODULE
C BEING BUILT
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH
C
      CALL DBP_BYTES( -1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASC_A( OFFSET,STRING,LENGTH )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH,OFFSET
C
      CALL DBP_INTEGER( OFFSET,LENGTH,1 )
      CALL DBP_BYTES( OFFSET+1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASCX_A( OFFSET,STRING,LENGTH )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH,OFFSET
C
      CALL DBP_BYTES( OFFSET+1,STRING,LENGTH )
      RETURN
      END
```

```
      SUBROUTINE INT1( VALUE )
C
C INSERT A ONE-BYTE INTEGER
C
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,1 )
      RETURN
      END
      SUBROUTINE INT1_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A ONE-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,1 )
      RETURN
      END
      SUBROUTINE INT2( VALUE )
C
C INSERT A TWO-BYTE INTEGER
C
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,2 )
      RETURN
      END
      SUBROUTINE INT2_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A TWO-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,2 )
      RETURN
      END
      SUBROUTINE INT4( VALUE )
C
C INSERT A FOUR-BYTE INTEGER
C
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,4 )
      RETURN
      END
      SUBROUTINE INT4_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A FOUR-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,4 )
      RETURN
      END
      SUBROUTINE TRON
C
C START THE DIAGNOSTIC TRACE UTILITY
C USE UNIT #9
C
      CALL TRACE_START(9)
```

```
      RETURN
      END
      SUBROUTINE TROFF
C
C STOP THE TRACE UTILITY
C
      CALL TRACE_STOP
      RETURN
      END
      SUBROUTINE PRON
C
C START THE PERFORMANCE MONITORING UTILITY
C
      CALL PERFORM_START
      RETURN
      END
      SUBROUTINE PROFF( CLOCK,CPU,BIO,DIO,PAGE )
C
C STOP THE PERFORMANCE MONITORING UTILITY AND
C RETRIEVE THE EXECUTION STATISTICS SINCE THE
C LAST ACTIVATION OF 'PRON'
C
      INTEGER*4 BIO,DIO,PAGE
      CALL PERFORM_STOP( CLOCK,CPU,BIO,DIO,PAGE )
      RETURN
      END
      SUBROUTINE TERMINATE
C
C INSERT THE TERMINATOR BYTES INTO THE REQUEST STREAM
C
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      RETURN
      END
      SUBROUTINE SEND
C
C SEND THE BUILT REQUEST MODULE TO THE DBP
C
      CALL DBP_SEND
      RETURN
      END
      SUBROUTINE RECV( RESPONSE,NBYTES_RECV,MORE )
C
C RECEIVE THE MESSAGE FROM THE DBP. IF 'MORE' IS
C TRUE THEN WE SHOULD RE-ACTIVATE 'SEND'
C
      LOGICAL MORE
      BYTE RESPONSE(1024)
      INTEGER*4 NBYTES_RECV
C
      CALL DBP_RECV( RESPONSE,NBYTES_RECV,MORE )
      RETURN
      END
      SUBROUTINE PERFON
C
C TURN ON THE PERFORMANCE MONITORING
C
      LOGICAL PERF_DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
C
```

```
      PERF_DEBUG = .TRUE.
      RETURN
      END
      SUBROUTINE PERFOFF
C
C TURN OFF THE PERFORMANCE MONITORING
C
      LOGICAL PERF_DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
C
      PERF_DEBUG = .FALSE.
      RETURN
      END
      SUBROUTINE TRACEON
C
C TURN ON THE TRACE TO DISPLAY THE ENCODED SEND
C AND REQUEST MODULES BEING TRANSFERRED
C
      LOGICAL DEBUG
      COMMON/DEBUGMODE/ DEBUG
C
      DEBUG = .TRUE.
      RETURN
      END
      SUBROUTINE TRACEOFF
C
C TURN OFF THE TRACE TO DISPLAY ENCODED SEND
C AND REQUEST MODULES BEING TRANSFERRED
C
      LOGICAL DEBUG
      COMMON/DEBUGMODE/ DEBUG
C
      DEBUG = .FALSE.
      RETURN
      END
      SUBROUTINE DBP_BEGIN
C=======================================================
C
C PURPOSE :
C
C   START THE ENCODING PROCESS NECESSARY TO BUILD
C   A COMMAND BLOCK FOR PASSAGE TO THE DBP
C
C ARGUMENTS :
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
C
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 CURRENT_OFFSET
      COMMON/OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C RESET THE CURRENT OFFSET COUNTER
C
      CURRENT_OFFSET = 0
```

```
      RETURN
      END
      SUBROUTINE DBP_INTEGER( OFFSET,VALUE,LENGTH )
C=======================================================
C
C PURPOSE :
C
C   TO INSERT A 1,2, OR 4 BYTE INTEGER INTO THE COMMAND BLOCK
C   BEING CONSTRUCTED.
C
C ARGUMENTS
C
C   OFFSET         - OFFSET FROM START OF THE COMMAND BLOCK
C                    BEING BUILT. STARTS AT ZERO.
C
C   VALUE          - VALUE TO BE INSERTED INTO THE COMMAND BLOCK
C
C   LENGTH         - NUMBER OF BYTES IN INTEGER 'VALUE'
C                    = 1,2, OR 4
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BYTE_ARRAY(4)
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,VALUE,VALUE2,LENGTH,CURRENT_OFFSET
      INTEGER*4 POSITION
      EQUIVALENCE( VALUE2,BYTE_ARRAY(1) )
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
      VALUE2 = VALUE
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
      DO 100 I = 1,LENGTH
100   BUILT_MODULE( POSITION+I ) = BYTE_ARRAY(I)
C
C UPDATE THE OFFSET COUNTER
C
      CURRENT_OFFSET = POSITION + LENGTH
      RETURN
      END
      SUBROUTINE DBP_BYTES( OFFSET,STRING,LENGTH )
C=======================================================
C
C PURPOSE :
C
C   TO INSERT A CHARACTER STRING OF LENGTH 'LENGTH' INTO
C   THE COMMAND BLOCK BEING CONSTRUCTED
C
C ARGUMENTS
C
```

```
C   OFFSET        - OFFSET FROM START OF THE COMMAND BLOCK
C                   BEING BUILT. STARTS AT ZERO.
C
C   STRING        - CHARACTER STRING TO BE INSERTED INTO THE
C                   COMMAND BLOCK
C
C   LENGTH        - NUMBER OF BYTES IN CHARACTER STRING.
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,LENGTH,CURRENT_OFFSET
      INTEGER*4 POSITION
      CHARACTER*(*) STRING
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
C
C INSERT THE STRING INTO THE MODULE BEING BUILT
C
      READ( STRING,100 ) ( BUILT_MODULE(I),I=POSITION+1,
     X                     POSITION+LENGTH )
100   FORMAT( <LENGTH>A1 )
C
C UPDATE THE OFFSET COUNTER
C
      CURRENT_OFFSET = POSITION + LENGTH
      RETURN
      END
      SUBROUTINE DBP_BITS( BYTE_VALUE )
C=======================================================
C
C PURPOSE :
C
C   TO 'OR' THE GIVEN BYTE VALUE WITH THE BYTE
C   VALUE ALREADY PRESENT
C
C NOTE:
C
C   THE CURRENT_OFFSET COUNTER IS NOT INCREMENTED
C   THIS PERMITS MULTIPLE OR'S. WHEN LOGICAL 'OR'ING
C   IS DONE, USE ROUTINE 'DBP_BITS_END'
C
C
C ARGUMENTS
C
C
C   BYTE_VALUE    - BYTE VALUE TO 'OR'
C
C DATE :
```

```
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 CURRENT_OFFSET
      BYTE BYTE_VALUE
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C OR THE GIVEN BYTE WITH THE BYTE ALREADY THERE
C
      BUILT_MODULE(CURRENT_OFFSET+1 ) = BUILT_MODULE(CURRENT_OFFSET+1 ).OR.
     X        BYTE_VALUE
      RETURN
      END
      SUBROUTINE DBP_BITS_BEGIN( OFFSET )
C=======================================================
C
C PURPOSE :
C
C   TO INITIALIZE THE GIVEN BYTE WITHIN 'BUILT_MODULE'.
C   FUTURE 'OR'ING IS EXPECTED ON THE CURRENT BYTE,
C   SO THE CURRENT OFFSET COUNTER IS NOT
C   INCREMENTED.
C
C ARGUMENTS
C
C   OFFSET       - OFFSET FROM START OF THE COMMAND BLOCK
C                  BEING BUILT. STARTS AT ZERO.
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,CURRENT_OFFSET
      INTEGER*4 POSITION
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
C
      BUILT_MODULE( POSITION+1 ) = 0
      RETURN
      END
      SUBROUTINE DBP_BITS_END
C=======================================================
C
C PURPOSE :
C
C   SIGNIFIES THAT THE 'OR'ING PROCESS ON THE CURRENT
C   MODULE BYTE IS DONE. TIME TO CONTINUE CONSTRUCTION
C   OF THE REST OF THE MODULE. INCREMENT THE CURRENT
```

```
C   OFFSET COUNTER.
C
C ARGUMENTS
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,CURRENT_OFFSET
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
      CURRENT_OFFSET = CURRENT_OFFSET + 1
      RETURN
      END
      SUBROUTINE DBP_SEND
C=======================================================
C
C PURPOSE :
C
C   SEND THE COMMAND BLOCK TO THE DBP.
C
C NOTE :
C
C   THIS ROUTINE CALLS THE 'SPP' PACKAGE
C   ( SERVICE PORT PROTOCOL )
C   TO PERMIT HOST-DBP COMMUNICATION
C
C ARGUMENTS
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 ),PRBYTES( 1024 )
      INTEGER*4 CURRENT_OFFSET,TOTAL_BYTES,UNIT
      LOGICAL*4 MORE_TO_COME,DEBUG,PERF_DEBUG
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
      COMMON/ DEBUGMODE/ DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
      DATA PERF_DEBUG/.FALSE./,DEBUG/.FALSE./,UNIT/6/
C
C 1. SEND THE BUILT COMMAND BLOCK TO THE DBP
C 2. LOOP TO RECEIVE ALL DBP RESPONSES
C
C
C OUTPUT THE REQUEST BLOCK IF IN DEBUG MODE
C
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C
      DO 50 I = 1,CURRENT_OFFSET+1
      IF( (BUILT_MODULE(I).LT.'20'X).OR.
```

```
     X    (BUILT_MODULE(I).GT.'7E'X)) THEN
            PRBYTES(I) = '2E'X
         ELSE
            PRBYTES(I) = BUILT_MODULE(I)
         ENDIF
50       CONTINUE
         WRITE( UNIT,100 ) CURRENT_OFFSET
100      FORMAT(' == DBP REQUEST =='/' # of bytes is ',I5,
     X        /,' Byte Stream :'/ )
         MULTIPLE16 = (CURRENT_OFFSET/16)*16
         LEFTOVER   = CURRENT_OFFSET - MULTIPLE16
         IF( MULTIPLE16.GT.0 ) THEN
           DO 200 I = 1,MULTIPLE16,16
           WRITE( UNIT,150 ) (BUILT_MODULE(I1),I1=I,I+15),
     X      (PRBYTES(I2),I2=I,I+15)
150        FORMAT(16(1X,Z2.2),2X,16A1)
200        CONTINUE
         ENDIF
         IF( LEFTOVER.GT.0 ) THEN
           WRITE( UNIT,210 ) (BUILT_MODULE(I1),I1=MULTIPLE16+1,
     X     MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X     MULTIPLE16+LEFTOVER)
210        FORMAT(<LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X            <LEFTOVER>A1 )
         ENDIF
         WRITE( UNIT,250 )
250      FORMAT(/)
         ENDIF
         IF( PERF_DEBUG ) CALL PRON
         CALL SEND_REQUEST( BUILT_MODULE,CURRENT_OFFSET,1,1,1 )
C
C FINISHED WITH THIS COMMAND
C
9999     RETURN
         END
         SUBROUTINE DBP_RECV( RESPONSE,TOTAL_BYTES,MORE )
C=======================================================
C
C PURPOSE :
C
C   RECEIVE THE RESPONSE FROM THE DBP
C
C NOTE :
C
C   THIS ROUTINE CALLS THE 'SPP' PACKAGE
C   ( SERVICE PORT PROTOCOL )
C   TO PERMIT HOST-DBP COMMUNICATION
C
C ARGUMENTS
C
C   RESPONSE     - THE BYTE RESPONSE FROM THE DBP
C   MORE         - = .TRUE. IF THERE IS MORE TO COME
C                    FROM THE DBP
C
C                - = .FALSE. IF ALL THE DATA FROM THE
C                    DBP HAS BEEN RECEIVED
C
C
C DATE :
C
```

```
C   APRIL 20,1983
C
C=======================================================
      BYTE RESPONSE( 1024 ),PRBYTES( 1024 )
      INTEGER*4 DIO,BIO,PAGE
      INTEGER*4 CURRENT_OFFSET,TOTAL_BYTES,UNIT
      LOGICAL*4 MORE,DEBUG,PERF_DEBUG
      COMMON/DEBUGMODE/ DEBUG
      COMMON/PERFMODE/ PERF_DEBUG
      DATA PERF_DEBUG/.FALSE./,DEBUG/.FALSE./,UNIT/6/
C
      CALL RECV_RESPONSE( RESPONSE,TOTAL_BYTES,1,MORE )
      IF( PERF_DEBUG ) THEN
        CALL PROFF( CLOCK,CPU,BIO,DIO,PAGE )
        WRITE(6,10) CLOCK,CPU,BIO,DIO,PAGE
10     FORMAT(/' Clock ',F12.5/,' CPU ',F12.5/,
     X   ' Buffered I/O count ',I6/,' Direct I/O count ',I6,/,
     X   ' Page Fault count ',I6 ,//)
      ENDIF
      IF( TOTAL_BYTES.GT.1024 ) THEN
        WRITE( 6,25 ) TOTAL_BYTES
25      FORMAT(' Error, DBP says that it has ',
     X  I7,' bytes to send back.',
     X  /' This exceeds the limit of 1024.' )
      GO TO 9999
      ENDIF
C
C OUTPUT THE RESPONSE IF IN DEBUG MODE
C
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C
      DO 500 I = 1,TOTAL_BYTES
      IF( (RESPONSE(I).LT.'20'X).OR.
     X    (RESPONSE(I).GT.'7E'X)) THEN
         PRBYTES(I) = '2E'X
      ELSE
         PRBYTES(I) = RESPONSE(I)
      ENDIF
500   CONTINUE
      WRITE( UNIT,600 ) TOTAL_BYTES
600   FORMAT(' == DBP RESPONSE =='/' # of bytes is ',I5,
     X       /,' Byte Stream :'/ )
      MULTIPLE16 = (TOTAL_BYTES/16)*16
      LEFTOVER   = TOTAL_BYTES - MULTIPLE16
      IF( MULTIPLE16.GT.0 ) THEN
        DO 700 I = 1,TOTAL_BYTES,16
        WRITE( UNIT,650 ) (RESPONSE(I1),I1=I,I+15),
     X   (PRBYTES(I2),I2=I,I+15)
650     FORMAT(16(1X,Z2.2),2X,16A1)
700     CONTINUE
      ENDIF
      IF( LEFTOVER.GT.0 ) THEN
        WRITE( UNIT,675 ) (RESPONSE(I1),I1=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X  MULTIPLE16+LEFTOVER)
675     FORMAT(<LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X          <LEFTOVER>A1 )
      ENDIF
```

```
      WRITE( UNIT,750 )
750   FORMAT(/)
      ENDIF
C
C FINISHED WITH THIS COMMAND
C
9999  RETURN
      END
$
```

APPENDIX D - DBPSSP Examples

A FORTRAN and Pascal example are given to aid the reader in evaluating the utility of DBPSSP. A brief trace of the requests and responses is also included.

```
$ type testfor.for
      PROGRAM TESTFOR
C
C A FORTRAN EXAMPLE USING THE DBPSSP PRIMITIVE
C ROUTINES
C
      BYTE RESPONSE( 1024 )
      INTEGER*4 BYTES_RECV
      LOGICAL MORE
C
      CALL TRACE_START( 9 )
      CALL TRACEON
      CALL INIT_COMM
C
C SUBMIT KEYS 'ADMIN'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'07'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'ADMIN',5 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE DATABASE CALLED 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'60'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C KEEP DATABASE 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'64'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE FILE CALLED 'FILE1'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'40'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '1000'X )
      CALL DBP_BITS_END
```

```
      CALL DBP_INTEGER( -1,6,1 )
      CALL DBP_BYTES( -1,'DBPSYS',6 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,10,2 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,0,2 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE SCHEMA ON PERMANENT FILE 'FILE1'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'49'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0000'X )
      CALL DBP_BITS( '0000'X )
      CALL DBP_BITS_END
C
C SCHEMA SPECIFICATION - SET UP AS
C
C INT1 INTEGER*4
C INT2 INTEGER*4
C INT3 INTEGER*4
C
      CALL DBP_INTEGER( -1,0,1 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,20,2 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,20,2 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT1',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,4,1 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT2',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,4,1 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT3',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
```

```
      CALL DBP_INTEGER( -1,4,1 )
C DONE
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C KEEP FILE
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'41'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C LIST DATABASE 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'90'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,'F0'X,1 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
      CALL TRACE_STOP
      CALL TRACEOFF
      CALL EXIT
      END
$
```

```
$ type testpas.pas
PROGRAM TESTPAS( INPUT,OUTPUT );

(* TEST OF DBPSSP *)

TYPE LIMIT = ARRAY[ 1..1024 ] OF CHAR;
VAR RESPONSE: LIMIT;
    TOTAL_BYTES: INTEGER;
    MORE: BOOLEAN;
    I : INTEGER;



(* DBPSSP SUPPORT PROCEDURES - EXTERNAL *)

PROCEDURE INIT; FORTRAN;
PROCEDURE START; FORTRAN;
PROCEDURE TRON; FORTRAN;
PROCEDURE TROFF; FORTRAN;
PROCEDURE BITSB; FORTRAN;
PROCEDURE BITS( BYTEVALUE:INTEGER ); FORTRAN;
PROCEDURE BITSE; FORTRAN;
PROCEDURE ASC( %STDESCR STRING:PACKED ARRAY[INTEGER]
               OF CHAR; LENGTH:INTEGER ); FORTRAN ;
PROCEDURE INT1( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT2( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT4( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE TERMINATE; FORTRAN;
PROCEDURE SEND; FORTRAN;
PROCEDURE RECV( VAR RESPONSE:LIMIT;
                VAR TOTAL_BYTES:INTEGER;
                VAR MORE:BOOLEAN ); FORTRAN;
PROCEDURE TRACEON; FORTRAN;
PROCEDURE TRACEOFF; FORTRAN;



BEGIN
   TRACEON;
   TRON;
   INIT;

(* SUBMIT KEYS 'ADMIN' *)

   START;
   INT1(7);
   ASC('ADMIN',5);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE DATABASE CALLED 'TESTING' *)

   START;
   INT1(96);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );
```

```
(* KEEP DATABASE 'TESTING' *)

   START;
   INT1(100);
   ASC('TESTING',7);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE FILE CALLED 'FILE1' *)

   START;
   INT1(64);
   ASC('FILE1',5);
   INT1(1);
   BITSB; BITS(8); BITSE;
   ASC('DBPSYS',6);
   INT1(2);
   INT2(10);
   INT1(2);
   INT2(0);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE SCHEMA ON PERMANENT FILE 'FILE' *)

   START;
   INT1(73);
   ASC('FILE1',5);
   INT1(1);
   BITSB; BITS(0); BITSE;

(* SCHEMA SPECIFICATION - SET UP AS

   INT1 INTEGER*4
   INT2 INTEGER*4
   INT3 INTEGER*4

*)

   INT1(0);
   INT1(2);
   INT2(20);
   INT1(2);
   INT2(20);

   ASC('INT1',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);

   ASC('INT2',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);
```

```
   ASC('INT3',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);

   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* KEEP FILE *)

   START;
   INT1(65);
   ASC('FILE1',5);
   ASC('FILE1',5);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* LIST DATABASE 'TESTING' *)

   START;
   INT1(144);
   ASC('TESTING',7);
   INT1(1);
   INT1(240);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

   TROFF
END.
$
```

```
$ @testfor
  _TTB0: allocated
== DBP REQUEST ==
# of bytes is     9
Byte Stream :

07 05 41 44 4D 49 4E FF 00                             ..ADMIN..


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    11
Byte Stream :

60 07 54 45 53 54 49 4E 47 FF 00                       `.TESTING..


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    19
Byte Stream :

64 07 54 45 53 54 49 4E 47 07 54 45 53 54 49 4E  d.TESTING.TESTIN
47 FF 00                                         G..


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    24
Byte Stream :

40 05 46 49 4C 45 31 01 00 06 44 42 50 53 59 53  @.FILE1...DBPSYS
02 0A 00 02 00 00 FF 00                          ........


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    45
Byte Stream :
```

```
49 05 46 49 4C 45 31 01 00 00 02 14 00 02 14 00   I.FILE1.........
04 49 4E 54 31 01 01 01 04 04 49 4E 54 32 01 01   .INT1.....INT2..
01 04 04 49 4E 54 33 01 01 01 04 FF 00            ...INT3......


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    23
Byte Stream :

41 05 46 49 4C 45 31 05 46 49 4C 45 31 07 54 45   A.FILE1.FILE1.TE
53 54 49 4E 47 FF 00                              STING..


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    13
Byte Stream :

90 07 54 45 53 54 49 4E 47 01 F0 FF 00            ..TESTING....


== DBP RESPONSE ==
# of bytes is    55
Byte Stream :

F8 02 90 F0 01 00 01 01 06 02 03 00 00 00 00 07   ................
54 45 53 54 49 4E 47 01 03 06 02 03 03 00 05 00   TESTING.........
05 46 49 4C 45 31 01 00 06 02 03 03 00 05 00 05   .FILE1..........
46 49 4C 45 31 FF 00 15 2D 2E 00 15 2D 2E 00 00   FILE1..
46 49 4C 45 31 FF 00                              FILE1..


$
```

APPENDIX E - DBPQL conceptual procedures

```
$ type [intel.dbpql]dbpcmd.dat

(* SUPPORT PROCEDURES FOR DBPQL - EXTERNAL *)

PROCEDURE INIT; FORTRAN;
PROCEDURE START; FORTRAN;
PROCEDURE TRON; FORTRAN;
PROCEDURE TROFF; FORTRAN;
PROCEDURE BITSB; FORTRAN;
PROCEDURE BITS( BYTEVALUE:INTEGER ); FORTRAN;
PROCEDURE BITSE; FORTRAN;
PROCEDURE ASC( %STDESCR STRING:PACKED ARRAY[INTEGER]
               OF CHAR; LENGTH:INTEGER ); FORTRAN ;
PROCEDURE ASCX( %STDESCR STRING:PACKED ARRAY[INTEGER]
                OF CHAR; LENGTH:INTEGER ); FORTRAN;

PROCEDURE INT1( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT2( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT4( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE TERMINATE; FORTRAN;
PROCEDURE SEND; FORTRAN;
PROCEDURE RECV( VAR RESPONSE:LIMIT;
                VAR TOTAL_BYTES:INTEGER;
                VAR MORE:BOOLEAN ); FORTRAN;
PROCEDURE PERFON; FORTRAN;
PROCEDURE PERFOFF; FORTRAN;
PROCEDURE TRACEON; FORTRAN;
PROCEDURE TRACEOFF; FORTRAN;

(* UTILITY PROCEDURES *)

PROCEDURE NUM_TO_ASCII( NUMBER:INTEGER;VAR ASCII_NUMBER:IDENT_STRING;
                        VAR ASCII_NUMBERL:INTEGER );

(* CONVERT AN INTEGER TO ASCII *)

VAR  COUNT    : INTEGER;
     COUNT2   : INTEGER;
     DIGIT    : INTEGER;
     WORKING_NUMBER: INTEGER;
     STRING   : IDENT_STRING;

BEGIN
   WORKING_NUMBER := NUMBER;
   COUNT := 0;
   REPEAT
     COUNT := COUNT + 1;
     DIGIT := WORKING_NUMBER - ( WORKING_NUMBER DIV 10 )*10;
     WORKING_NUMBER := WORKING_NUMBER DIV 10;
     STRING[ COUNT ] := CHR( DIGIT + 48 );
   UNTIL WORKING_NUMBER = 0;
   (* REVERSE THE DIGITS *)
   FOR COUNT2 := 1 TO COUNT DO
      ASCII_NUMBER[ COUNT2 ] := STRING[ COUNT - COUNT2 + 1 ];
   ASCII_NUMBERL := COUNT
END;
```

```
(* RECEIVE 'DESCRIBE VIEW' RESPONSE *)

PROCEDURE DV_RESPONSE( VAR VIEW:IDENT_STRING;VAR VIEWL:INTEGER;
          VAR VIEW2:IDENT_STRING;VAR VIEW2L:INTEGER;
          VAR NUM_ITEMS:INTEGER;
          VAR ITEM_NAME:IDTYPE;VAR ITEM_NAMEL:NUMTYPE;
          VAR ITEM_TYPE:NUMTYPE;VAR ITEM_LENGTH:NUMTYPE );


VAR    COUNT :INTEGER;
       COUNT2:INTEGER;
       OFFSET:INTEGER;

BEGIN
   OFFSET := 16;
   VIEWL := ORD( RESPONSE[ OFFSET ]);
   VIEW := BLANK_IDENT;
   FOR COUNT := 1 TO VIEWL DO
       VIEW[ COUNT ] := RESPONSE[OFFSET+COUNT];
   (* POINT OFFSET TO 'VIEW-OWNER' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   VIEW2L := ORD( RESPONSE[ OFFSET ]);
   VIEW2 := BLANK_IDENT;
   FOR COUNT := 1 TO VIEW2L DO
       VIEW2[ COUNT ] := RESPONSE[OFFSET+COUNT];
   (* POINT OFFSET TO 'READ-LOCK' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'WRITE-LOCK' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'READ-WRITE' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
   (* POINT OFFSET TO 'FILE-COUNT' *)
   OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;

   NUM_ITEMS := ORD(RESPONSE[OFFSET+3]);
   OFFSET := OFFSET + 5;
   FOR COUNT := 1 TO NUM_ITEMS DO
   BEGIN
      ITEM_TYPE[ COUNT ] := ORD(RESPONSE[OFFSET+2]);
      ITEM_LENGTH[ COUNT ] := ORD(RESPONSE[OFFSET+3]);
      ITEM_NAMEL[ COUNT ] := ORD(RESPONSE[OFFSET+4]);
      ITEM_NAME[ COUNT ] := BLANK_IDENT;
      FOR COUNT2 := 1 TO ITEM_NAMEL[ COUNT ] DO
          ITEM_NAME[ COUNT,COUNT2 ] := RESPONSE[ OFFSET+COUNT2+4];
      (* UPDATE OFFSET TO THE NEXT ITEM *)

      OFFSET := OFFSET + ITEM_NAMEL[ COUNT ] + 5;
      OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1;
      OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1
   END
END;



(* QUERY *)

PROCEDURE QUERY;


TYPE  HEXTYPE = ARRAY[1..2] OF CHAR;
```

```
VAR   NUM_ITEMS: INTEGER;
      COUNT    : INTEGER;
      COUNT2   : INTEGER;
      COUNTER  : INTEGER;
      COUNTER2 : INTEGER;
      OFFSET   : INTEGER;
      ITEM_COUNT : INTEGER;
      DBCOUNT : INTEGER;
      VIEWCOUNT : INTEGER;
      FILECOUNT : INTEGER;
      SPACE_FILL : INTEGER;
      HEXSTRING : HEXTYPE;


PROCEDURE TOHEX( BYTE_VALUE:CHAR; VAR HEXDIGITS:HEXTYPE );

  FUNCTION HEXDIGIT( NUMBER:INTEGER ):CHAR;

  BEGIN
     CASE NUMBER OF

     0,1,2,3,4,5,6,7,8,9: HEXDIGIT := CHR( NUMBER+48 );

     10 : HEXDIGIT := 'A';
     11 : HEXDIGIT := 'B';
     12 : HEXDIGIT := 'C';
     13 : HEXDIGIT := 'D';
     14 : HEXDIGIT := 'E';
     15 : HEXDIGIT := 'F'

     END
  END;


BEGIN
   HEXDIGITS[1] := HEXDIGIT( ORD( BYTE_VALUE ) DIV 16  );
   HEXDIGITS[2] := HEXDIGIT( ORD( BYTE_VALUE ) MOD 16  )
END;


BEGIN
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );
   IF TOTAL_BYTES = 0 THEN
      WRITELN('Ok')
   ELSE
   BEGIN
      CASE ORD( RESPONSE[ 1 ] ) OF

      (* FETCH RESPONSE *)

      (* F1 *) 241: BEGIN
                       OFFSET := 1;
                       REPEAT
                       ITEM_COUNT := 0;
                       OFFSET := OFFSET + 3;
                       REPEAT
                          ITEM_COUNT := ITEM_COUNT + 1;
                          FOR COUNT2 := 1 TO ORD(RESPONSE[OFFSET]) DO
```

```
                WRITE( RESPONSE[OFFSET+COUNT2]);
            COUNTER := ITEMS1L[ ITEM_COUNT ]-ORD(RESPONSE[OFFSET]);
            IF COUNTER > 0 THEN
               FOR COUNTER2 := 1 TO COUNTER DO
                   WRITE(' ');
            WRITE('  ');
            OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1;
         UNTIL ORD(RESPONSE[OFFSET]) = 255;
         WRITELN;
         OFFSET := OFFSET + 2;
         UNTIL ORD(RESPONSE[OFFSET]) = 246;
      END;

(* COMPLETION CODE RESPONSE *)

(* F6 *) 246: BEGIN
         WRITE('Completion Code is ');
         TOHEX(RESPONSE[6],HEXSTRING);
         FOR COUNT := 1 TO 2 DO WRITE(HEXSTRING[COUNT]);
         WRITE(' ');
         TOHEX(RESPONSE[5],HEXSTRING);
         FOR COUNT := 1 TO 2 DO WRITE(HEXSTRING[COUNT]);
         WRITELN;
         WRITELN('--> Refer to the iDBP reference manual.');
         WRITELN;WRITELN;
      END;

(* LIST RESPONSE *)

(* F8 *) 248: BEGIN
      IF ORD(RESPONSE[7]) <> 255 THEN
      BEGIN

        WRITELN;WRITELN;
        CASE ORD( RESPONSE[3]) OF

       144 : BEGIN
                OFFSET := 7;
                DBCOUNT := 0;
                REPEAT

                CASE ORD(RESPONSE[OFFSET+1]) OF

                 0: BEGIN
                      VIEWCOUNT := VIEWCOUNT + 1;
                      WRITE('          view # ',VIEWCOUNT:2,' ')
                   END;
                 1: BEGIN
                      FILECOUNT := 0;
                      DBCOUNT := DBCOUNT + 1;
                      WRITE('database # ',DBCOUNT:2,' ')
                   END;
                 3: BEGIN
                      VIEWCOUNT := 0;
                      FILECOUNT := FILECOUNT + 1;
                      WRITE('      file # ',FILECOUNT:2,' ' )
                   END;
                END;
                OFFSET := OFFSET + 9;
                FOR COUNT := 1 TO ORD(RESPONSE[OFFSET]) DO
```

```
                WRITE(RESPONSE[OFFSET+COUNT]);
            WRITELN;
            OFFSET := OFFSET + ORD(RESPONSE[OFFSET]) + 1

            UNTIL ORD(RESPONSE[OFFSET])=255;
            WRITELN
          END;

    146 : BEGIN
             WRITELN('List of Views :');WRITELN;
             OFFSET := 7;
             REPEAT
                OFFSET := OFFSET + 9;
                FOR COUNT := 1 TO ORD(RESPONSE[OFFSET]) DO
                    WRITE(RESPONSE[OFFSET+COUNT]);
                WRITELN;
                OFFSET := OFFSET + ORD(RESPONSE[OFFSET])+1
             UNTIL ORD(RESPONSE[OFFSET])=255;
             WRITELN
          END;
    END
    END
    ELSE
      WRITELN('I have never heard of that database.')
  END;

(* DESCRIBE VIEW RESPONSE *)

(* F9 *) 249: BEGIN
    IF ORD( RESPONSE[7]) <> 255 THEN
    BEGIN
     DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
        ITEMS1,ITEMS1L,VALS1,VALS2 );
     WRITELN;
     WRITE('View : ');
     FOR COUNT := 1 TO ITEM1L DO
         WRITE( ITEM1[ COUNT ]);
     WRITELN;
     WRITE('Underlying Relation : ');
     FOR COUNT := 1 TO ITEM2L DO
         WRITE( ITEM2[ COUNT ]);
     WRITELN;
     WRITELN('# of items = ',NUM_ITEMS:3);
     WRITELN;
     FOR COUNT := 1 TO NUM_ITEMS DO
     BEGIN
        FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
            WRITE( ITEMS1[ COUNT,COUNT2 ]);
        SPACE_FILL := 20 - ITEMS1L[ COUNT ];
        FOR COUNT2 := 1 TO SPACE_FILL DO
            WRITE(' ');
        CASE VALS1[ COUNT ] OF

        0 : WRITE('Unsigned Integer    ');
        1 : WRITE('Signed Integer      ');
        2 : WRITE('Uninterpreted       ');
        3 : WRITE('ASCII Alphanumeric  ');
        7 : WRITE('Record Pointer      ');
        9 : WRITE('String Pointer      ');
        64: WRITE('Zero Integer        ');
```

```
                                    END;
                                    WRITELN( VALS2[ COUNT ] )
                                 END
                                END
                                ELSE
                                   WRITELN('I have never heard of that view.')
                              END;

      (* REMARK RESPONSE *)

      (* FC *) 252: BEGIN
                                 WRITE('Echo : ');
                                 FOR COUNTER := 1 TO ORD(RESPONSE[4])-1 DO
                                     WRITE( RESPONSE[ COUNTER+6 ]);
                                 WRITELN;WRITELN
                              END;

      OTHERWISE
               ;
      END
   END
END;


(*-------------------------------------------------*)

(* ATTACH *)

PROCEDURE ATTACH( ITEM:IDTYPE;ITEML:NUMTYPE;NUMBER_OF_ITEMS,
          OWN_USE,OTHERS_USE,WAIT_FLAG,ABORT_FLAG:INTEGER );

VAR   COUNT : INTEGER;


BEGIN
   INT1(0);
   INT1(1);
   BITSB;
   BITS( WAIT_FLAG*128 );
   BITS( ABORT_FLAG*64 );
   BITSE;
   FOR COUNT := 1 TO NUMBER_OF_ITEMS DO
   BEGIN
      ASC( ITEM[ COUNT ],ITEML[ COUNT ] );
      INT1(1);
      BITSB;
      BITS( OWN_USE*16 );
      BITS( OTHERS_USE );
      BITSE
   END;
   TERMINATE
END;


(* DEFINE DATABASE *)

PROCEDURE DEFINE_DATABASE( DBNAME:IDENT_STRING;
          DBNAMEL:INTEGER );
BEGIN
```

```
   INT1(96);
   ASC(DBNAME,DBNAMEL);
   TERMINATE
END;


(* DEFINE FILE *)

PROCEDURE DEFINE_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          PAGESIZE:INTEGER;VOLUMEID:IDENT_STRING;VOLUMEIDL:INTEGER;
          INITALLOC,MAXALLOC:INTEGER );

BEGIN
   INT1(64);
   ASC(FILENAME,FILENAMEL);
   INT1(1);
   INT1(PAGESIZE*128);
   ASC(VOLUMEID,VOLUMEIDL);
   INT1(2);
   INT2(INITALLOC);
   INT1(2);
   INT2(MAXALLOC);
   TERMINATE
END;


(* DEFINE SCHEMA *)

PROCEDURE DEFINE_SCHEMA( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          DSTYPE,REUSE,EXPAREA,SUBSTRLEN:INTEGER;
          VARITEM:INTEGER;VIEW:IDENT_STRING;VIEWL:INTEGER;
          NUM_ITEMS:INTEGER;
          DS_ID:IDTYPE;DS_IDL:NUMTYPE;
          FIXEDVAR:FIXEDVAR_TYPE;ITEMTYPE:ITEM_TYPE;
          ITEML:ITEML_TYPE );


VAR   COUNT: INTEGER;

BEGIN
   INT1(73);
   ASC(FILENAME,FILENAMEL);
   INT1(1);
   BITSB;BITS(DSTYPE*16);BITS(REUSE*8);BITSE;
   IF VIEWL <> 0 THEN
   (* A VIEW HAS BEEN SPECIFIED FOR THE SCHEMA *)
      ASC( VIEW,VIEWL )
   ELSE
   (* A SCHEMA HAS BEEN EXPLICITLY DEFINED *)
   BEGIN
      INT1(0);
      INT1(2);
      IF EXPAREA = 0 THEN
         INT2(SUBSTRLEN)
      ELSE
         INT2(EXPAREA);
      IF DSTYPE <> 0 THEN
         INT1(0)
      ELSE
      BEGIN
```

```
         INT1(2);
         INT2(VARITEM)
      END;
      FOR COUNT := 1 TO NUM_ITEMS DO
      BEGIN
         ASC(DS_ID[ COUNT ],DS_IDL[ COUNT ]);
         INT1(1);
         BITSB;BITS(FIXEDVAR[COUNT]*128);
         BITS( ITEMTYPE[ COUNT ] );BITSE;
         IF DSTYPE <> 0 THEN
            INT1(0)
         ELSE
         BEGIN
            INT1(1);
            INT1(ITEML[ COUNT ])
         END
      END
   END;
   TERMINATE
END;


(* DEFINE VIEW - CONNECT *)

PROCEDURE DEFINE_VIEW_CONNECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE1:IDENT_STRING;SOURCE1L:INTEGER;
          STRINGPTR:IDENT_STRING;STRINGPTRL:INTEGER;
          SOURCE2:IDENT_STRING;SOURCE2L:INTEGER );

BEGIN
   INT1(80);
   ASC( NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(32);
   ASC( SOURCE1,SOURCE1L );
   INT1(255);
   INT1(5);
   ASC( STRINGPTR,STRINGPTRL );
   ASC( SOURCE2,SOURCE2L );
   TERMINATE
END;

(* DEFINE VIEW - JOIN *)

PROCEDURE DEFINE_VIEW_JOIN( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
                            SOURCE1:IDENT_STRING;SOURCE1L:INTEGER;
                            ITEM1:IDENT_STRING;ITEM1L:INTEGER;
                            SOURCE2:IDENT_STRING;SOURCE2L:INTEGER;
                            ITEM2:IDENT_STRING;ITEM2L:INTEGER );

BEGIN
   INT1(80);
   ASC( NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(16);
   ASC( SOURCE1,SOURCE1L );
   INT1(255);
   INT1(5);
   ASC( ITEM1,ITEM1L );
   ASC( SOURCE2,SOURCE2L );
```

```
   INT1(255);
   INT1(5);
   ASC( ITEM2,ITEM2L );
   TERMINATE
END;


(* DEFINE VIEW - ORDER *)

PROCEDURE DEFINE_VIEW_ORDER( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          ORDER_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE;
          ASC_OR_DESC:INTEGER );


VAR   COUNT: INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL );
   INT1(1);
   INT1(64);
   ASC(SOURCE,SOURCEL);
   FOR COUNT := 1 TO ORDER_NUM DO
   BEGIN
      INT1(255);
      INT1(5);
      ASC(ITEMS1[ COUNT ],ITEMS1L[ COUNT ]);
      INT1(1);
      INT1(ASC_OR_DESC*128)
   END;
   TERMINATE
END;


(* DEFINE VIEW - PROJECT *)

PROCEDURE DEFINE_VIEW_PROJECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          INC_EXC:INTEGER;SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          ALL_INDICATOR:BOOLEAN;PROJECT_NUM:INTEGER;
          ITEM:IDTYPE;ITEML:NUMTYPE );


VAR   COUNT:  INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL);
   INT1(1);
   IF INC_EXC = 0 THEN
     INT1(96)
   ELSE
     INT1(112);
   ASC(SOURCE,SOURCEL);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
```

```
   BEGIN
      FOR COUNT := 1 TO PROJECT_NUM DO
      BEGIN
         INT1(255);
         INT1(5);
         ASC(ITEM[ COUNT ],ITEML[ COUNT ])
      END
   END;
   TERMINATE
END;

(* DEFINE VIEW - SELECT *)

PROCEDURE DEFINE_VIEW_SELECT( NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER;
          SOURCE:IDENT_STRING;SOURCEL:INTEGER;
          COMPARATOR:NUMTYPE;REC_SEARCH:INTEGER;
          COMPARE_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE;
          ITEMS2:IDTYPE;ITEMS2L:NUMTYPE );

VAR   COUNT:  INTEGER;

BEGIN
   INT1(80);
   ASC(NEWVIEW,NEWVIEWL);
   INT1(1);
   INT1(48);
   ASC(SOURCE,SOURCEL);

   (* CREATE A BLOCK FOR EACH COMPARATOR W/OPERANDS *)

   FOR COUNT := 1 TO COMPARE_NUM DO
   BEGIN
      IF COMPARATOR[ COUNT ] IN [ 1,11 ] THEN
      BEGIN
         INT1(2);
         INT1( COMPARATOR[ COUNT ] );
         INT1( REC_SEARCH*128);
         INT1(255);
         INT1(5);
         ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ])
       END

      ELSE

      BEGIN
         INT1(2);
         INT1( COMPARATOR[ COUNT ] );
         INT1( REC_SEARCH*128 );
         INT1(255);
         INT1(5);
         ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ]);
         ASC( ITEMS2[ COUNT ],ITEMS2L[ COUNT ])
      END;
      IF COUNT < COMPARE_NUM THEN
      (* INSERT AN 'AND' OPERATION *)

      BEGIN
         INT1(2);
         INT1(250);
         INT1(0)
```

```
      END
   END;

   TERMINATE
END;

(* DELETE DATABASE *)

PROCEDURE DELETE_DATABASE( DBNAME:IDENT_STRING;DBNAMEL:INTEGER );

BEGIN
   INT1(97);
   ASC(DBNAME,DBNAMEL);
   TERMINATE
END;

(* DELETE FILE *)

PROCEDURE DELETE_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER );

BEGIN
   INT1(66);
   ASC(FILENAME,FILENAMEL);
   TERMINATE
END;


(* DELETE VIEW *)

PROCEDURE DELETE_VIEW( VIEW:IDENT_STRING;VIEWL:INTEGER );

BEGIN
   INT1(82);
   ASC( VIEW,VIEWL );
   TERMINATE
END;


(* DESCRIBE VIEW *)

PROCEDURE DESCRIBE_VIEW( VIEW:IDENT_STRING;VIEWL:INTEGER;
          ALL_INDICATOR:BOOLEAN );

BEGIN
   INT1(154);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( VIEW,VIEWL );
   TERMINATE
END;


(* DETACH *)

PROCEDURE DETACH( VIEW:IDENT_STRING;VIEWL:INTEGER;ALL_INDICATOR:BOOLEAN );
```

```
BEGIN
   INT1(1);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( VIEW,VIEWL );
   TERMINATE
END;

(* END CURSOR *)

PROCEDURE END_CURSOR( CURSOR_NUM:INTEGER;ALL_INDICATOR:BOOLEAN );
BEGIN
   INT1(3);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
   BEGIN
      INT1(1);
      INT1(CURSOR_NUM)
   END;
   TERMINATE
END;


(* FETCH *)

PROCEDURE FETCH( CURSOR_NUM:INTEGER;COUNT:INTEGER );

BEGIN
   INT1(16);
   INT1(1);
   INT1(CURSOR_NUM);
   IF COUNT = 0 THEN

   (* FETCH ALL *)

   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE

   (* FETCH A SPECIFIC NUMBER OF RECORDS *)

   BEGIN
      INT1(2);
      INT2(COUNT)
   END;
   TERMINATE
END;


(* FREE *)
```

```
PROCEDURE FREE( VIEW:IDENT_STRING;VIEWL:INTEGER;ALL_INDICATOR:BOOLEAN );

BEGIN
   INT1(1);
   IF ALL_INDICATOR THEN
   (* FREE ALL VIEWS *)
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
   (* FREE A SPECIFIC VIEW *)
      ASC( VIEW,VIEWL );
   TERMINATE
END;


(* KEEP DATABASE *)

PROCEDURE KEEP_DATABASE( OLDDB:IDENT_STRING;OLDDBL:INTEGER;
          NEWDB:IDENT_STRING;NEWDBL:INTEGER );

BEGIN
   INT1(100);
   ASC( OLDDB,OLDDBL );
   ASC( NEWDB,NEWDBL );
   TERMINATE
END;


(* KEEP FILE *)

PROCEDURE KEEP_FILE( OLDFILE:IDENT_STRING;OLDFILEL:INTEGER;
          NEWFILE:IDENT_STRING;NEWFILEL:INTEGER;
          DATABASE:IDENT_STRING;DATABASEL:INTEGER );

BEGIN
   INT1(65);
   ASC( OLDFILE,OLDFILEL );
   ASC( NEWFILE,NEWFILEL );
   IF DATABASEL = 0 THEN
      INT1(0)
   ELSE
      ASC( DATABASE,DATABASEL );
   TERMINATE
END;


(* KEEP VIEW *)

PROCEDURE KEEP_VIEW( OLDVIEW:IDENT_STRING;OLDVIEWL:INTEGER;
          NEWVIEW:IDENT_STRING;NEWVIEWL:INTEGER );

BEGIN
   INT1(81);
   ASC( OLDVIEW,OLDVIEWL );
   ASC( NEWVIEW,NEWVIEWL );
   TERMINATE
END;
```

```
(* LIST DATABASE *)

PROCEDURE LIST_DATABASE( DBNAME:IDENT_STRING;DBNAMEL:INTEGER;
          ALL_ENTITIES,ALL_DATABASES: BOOLEAN );

BEGIN
   INT1(144);
   IF ALL_DATABASES THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( DBNAME,DBNAMEL );
   INT1(1);
   IF NOT ALL_ENTITIES THEN
      INT1(0)
   ELSE
      INT1(160);
   TERMINATE
END;


(* LIST FILE *)

PROCEDURE LIST_FILE( FILENAME:IDENT_STRING;FILENAMEL:INTEGER;
          ALL_INDICATOR:BOOLEAN );
BEGIN
   INT1(145);
   IF ALL_INDICATOR THEN
   BEGIN
      INT1(255);
      INT1(6)
   END
   ELSE
      ASC( FILENAME,FILENAMEL );
   INT1(1);
   INT1(255);
   TERMINATE
END;


(* LIST VIEWS *)

PROCEDURE LIST_VIEWS;

BEGIN
   INT1(146);
   INT1(255);
   INT1(6);
   TERMINATE
END;

(* REMARK *)

PROCEDURE REMARK( A_WORD:IDENT_STRING;A_WORDL:INTEGER;DESTINATION:INTEGER );

BEGIN
```

```
   INT1(58);
   INT1(1);
   INT1(DESTINATION);
   ASC(A_WORD,A_WORDL);
   TERMINATE
END;


(* SUBMIT KEYS *)

PROCEDURE SUBMIT_KEYS( KEY:IDENT_STRING;KEYL:INTEGER );

BEGIN
   INT1(7);
   ASC( KEY,KEYL );
   TERMINATE
END;

(* START CURSOR *)

PROCEDURE START_CURSOR( CURSOR_NUM:INTEGER; VIEW:IDENT_STRING;
          VIEWL:INTEGER;MODE:INTEGER;DIRECTION:INTEGER;
          RETEST:INTEGER );

BEGIN
   INT1(2);
   INT1(1);
   INT1(CURSOR_NUM);
   ASC(VIEW,VIEWL);
   INT1(1);
   BITSB;
   BITS(MODE*128);
   BITS(DIRECTION*64);
   BITS(RETEST*32);
   BITSE;
   TERMINATE
END;

PROCEDURE STORE( CURSOR_NUM:INTEGER;INTEGRITY:BOOLEAN;
                 ITEM_NUM:INTEGER;ITEMS1:IDTYPE;ITEMS1L:NUMTYPE );

VAR   COUNT:INTEGER;

BEGIN
   INT1(18);
   INT1(1);
   INT1(CURSOR_NUM);
   INT1(1);
   IF INTEGRITY THEN
      INT1(0)
   ELSE
      INT1(128);
   FOR COUNT := 1 TO ITEM_NUM DO
   BEGIN
      ASC( ITEMS1[ COUNT ],ITEMS1L[ COUNT ])
   END;
   TERMINATE
END;

$
```

APPENDIX F - DBPQL Grammar File

```
$ type [intel.dbpql]dbpql.grm
<GOAL> ::= <QUERY> <EOLN>
 ;
*===================================
* QUERY TYPES
*===================================
<QUERY> ::=
 ;
<QUERY> ::= <ATTACH>
 ;
<QUERY> ::= <CREATE_DATABASE>
 ;
<QUERY> ::= <CREATE_RELATION>
 ;
<QUERY> ::= <CREATE_VIEWC>
 ;
<QUERY> ::= <CREATE_VIEWJ>
 ;
<QUERY> ::= <CREATE_VIEWO>
 ;
<QUERY> ::= <CREATE_VIEWP>
 ;
<QUERY> ::= <CREATE_VIEWS>
 ;
<QUERY> ::= <DELETE_DATABASE>
 ;
<QUERY> ::= <DELETE_RELATION>
 ;
<QUERY> ::= <DELETE_VIEW>
 ;
<QUERY> ::= <DETACH>
 ;
<QUERY> ::= <DISPLAY>
 ;
<QUERY> ::= <ECHO>
 ;
<QUERY> ::= <HELP>
 ;
<QUERY> ::= <INPUT>
 ;
<QUERY> ::= <LIST_DBS>
 ;
<QUERY> ::= <LIST_DB>
 ;
<QUERY> ::= <LIST_VIEWS>
 ;
<QUERY> ::= <LIST_VIEW>
 ;
<QUERY> ::= <LOAD>
 ;
<QUERY> ::= <PERFORM_COMMAND>
 ;
<QUERY> ::= <TRACE_COMMAND>
 ;
*===================================
* ATTACH
*===================================
<ATTACH> ::= <ATTACHK> <VIEWS> <PERMISSION>
 BEGIN
```

```
    START;
    FREE( BLANK_IDENT,1,TRUE );
    ATTACH( ITEMS3,ITEMS3L,IDCOUNT,PERMISSION,0,0,0 );
    QUERY
 END;
<ATTACHK> ::= ATTACH
 IDCOUNT := 0;
<VIEWS> ::= <ATTACH_VIEW>
 ;
<VIEWS> ::= <VIEWS> <ATTACH_VIEW>
 ;
<ATTACH_VIEW> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS3[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS3L[ IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<PERMISSION> ::= READ
 PERMISSION := 1;
<PERMISSION> ::= WRITE
 PERMISSION := 2;
<PERMISSION> ::= RW
 PERMISSION := 3;
<PERMISSION> ::= ADMIN
 PERMISSION := 4;
*====================================
* CREATE DATABASE
*====================================
<CREATE_DATABASE> ::= <CREATE_DATABASEK> <CD_DBNAME>
 BEGIN
    START;
    DEFINE_DATABASE(DBNAME,DBNAMEL);
    KEEP_DATABASE(DBNAME,DBNAMEL,DBNAME,DBNAMEL);
    QUERY
 END;
<CREATE_DATABASEK> ::= CREATE DATABASE
 ;
<CD_DBNAME> ::= <ID>
 BEGIN
    DBNAME := SSTACK[SP].IDNAME;
    DBNAMEL:= SSTACK[SP].IDLEN
 END;
*====================================
* CREATE RELATION
*====================================
<CREATE_RELATION> ::= <CREATE_RELATIONK> <DF_REST>
 BEGIN
    START;
    DEFINE_FILE( DF_FILENAME,DF_FILENAMEL,DF_PAGESIZE,
                 DF_VOLUMEID,DF_VOLUMEIDL,
                 DF_INITALLOC,DF_MAXALLOC );

    DEFINE_SCHEMA( DF_FILENAME,DF_FILENAMEL,DS_TYPE,DS_REUSE,
                   DS_EXPAREA,DS_SUBSTRLEN,DS_VARITEM,DS_VIEW,DS_VIEWL,
                   IDCOUNT,DS_ID,DS_IDL,DS_FIXEDVAR,
                   DS_ITEMTYPE,DS_ITEML );

    KEEP_FILE( DF_FILENAME,DF_FILENAMEL,DF_FILENAME,DF_FILENAMEL,
               DBNAME,DBNAMEL );
    WRITELN('Ok, View ',DF_FILENAME:DF_FILENAMEL,' has been created.');
```

```
    QUERY
 END;
<CREATE_RELATIONK> ::= CREATE RELATION
 BEGIN

    (* DEFAULTS *)

    (* DEFINE FILE DEFAULTS *)

    DF_FILENAME := 'FILE1       ';
    DF_FILENAMEL := 5;
    DF_PAGESIZE := 0;
    DF_VOLUMEID := 'DBPSYS      ';
    DF_VOLUMEIDL := 6;
    DF_INITALLOC := 20;
    DF_MAXALLOC := 0;

    (* DEFINE SCHEMA DEFAULTS *)

    DS_TYPE := 0;
    DS_REUSE := 0;
    DS_VIEWL := 0;
    DS_EXPAREA := 20;
    DS_SUBSTRLEN := 80;
    DS_VARITEM := 20;
    IDCOUNT := 0;

 END;
<DF_REST> ::= <ID> IN <ID> <DF_REST2>
 BEGIN
    DF_FILENAME := SSTACK[MP].IDNAME;
    DF_FILENAMEL:= SSTACK[MP].IDLEN;
    DBNAME := SSTACK[MP+2].IDNAME;
    DBNAMEL:= SSTACK[MP+2].IDLEN
 END;
<DF_REST2> ::= USING SCHEMA <SCHEMA> <OPTIONS>
 ;
<DF_REST2> ::= USING VIEW <ID>
 BEGIN
    DS_VIEW := SSTACK[SP].IDNAME;
    DS_VIEWL:= SSTACK[SP].IDLEN;
 END;
<SCHEMA> ::= <ITEM>
 ;
<SCHEMA> ::= <SCHEMA> <ITEM>
 ;
<ITEM> ::= <ITEM_ID> <DATA_TYPE> <FIXED_VAR> <ITEM_LENGTH>
 ;
<ITEM_ID> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    DS_ID[ IDCOUNT ] := SSTACK[MP].IDNAME;
    DS_IDL[ IDCOUNT ] := SSTACK[MP].IDLEN
 END;
<FIXED_VAR> ::= FIXED
 DS_FIXEDVAR[ IDCOUNT ] := 0;
<FIXED_VAR> ::= VAR
 DS_FIXEDVAR[ IDCOUNT ] := 1;
<ITEM_LENGTH> ::= <NO>
```

```
 DS_ITEML[ IDCOUNT ] := SSTACK[SP].IVAL;
<DATA_TYPE> ::= UNSIGNED_INT
 DS_ITEMTYPE[ IDCOUNT ] := 0;
<DATA_TYPE> ::= SIGNED_INT
 DS_ITEMTYPE[ IDCOUNT ] := 1;
<DATA_TYPE> ::= INTEGER
 DS_ITEMTYPE[ IDCOUNT ] := 1;
<DATA_TYPE> ::= UNINTERPRET
 DS_ITEMTYPE[ IDCOUNT ] := 2;
<DATA_TYPE> ::= ASCII
 DS_ITEMTYPE[ IDCOUNT ] := 3;
<DATA_TYPE> ::= RECORD_PTR
 DS_ITEMTYPE[ IDCOUNT ] := 7;
<DATA_TYPE> ::= STRING_PTR
 DS_ITEMTYPE[ IDCOUNT ] := 9;
<OPTIONS> ::=
 ;
<OPTIONS> ::= <OPTIONS> <OPTION>
 ;
<OPTION> ::= SMALLPAGE
 DF_PAGESIZE := 0;
<OPTION> ::= LARGEPAGE
 DF_PAGESIZE := 1;
<OPTION> ::= VOLUME <ID>
 BEGIN
    DF_VOLUMEID := SSTACK[SP].IDNAME;
    DF_VOLUMEIDL:= SSTACK[SP].IDLEN;
 END;
<OPTION> ::= INITALLOC <NO>
 DF_INITALLOC := SSTACK[SP].IVAL;
<OPTION> ::= EXPANDFILE
 DF_MAXALLOC := 0;
<OPTION> ::= MAXALLOC <NO>
 DF_MAXALLOC := SSTACK[SP].IVAL;
<OPTION> ::= STRUCTURED
 DS_TYPE := 0;
<OPTION> ::= UN_CLEAR
 DS_TYPE := 1;
<OPTION> ::= UN_COMPLEX
 DS_TYPE := 2;
<OPTION> ::= UN_UNINTERP
 DS_TYPE := 3;
<OPTION> ::= UN_BACKUP
 DS_TYPE := 4;
<OPTION> ::= UN_ROLLF
 DS_TYPE := 5;
<OPTION> ::= RE_USE
 DS_REUSE := 0;
<OPTION> ::= NORE_USE
 DS_REUSE := 1;
<OPTION> ::= EXP_AREA <NO>
 DS_EXPAREA := SSTACK[SP].IVAL;
<OPTION> ::= SUBSTR_LEN <NO>
 DS_SUBSTRLEN := SSTACK[SP].IVAL;
<OPTION> ::= VARITEM <NO>
 DS_VARITEM := SSTACK[SP].IVAL;
*==================================
* CREATE VIEW - CONNECT
*==================================
<CREATE_VIEWC> ::= <CREATE_VIEWCK> <CVC_REST>
```

```
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEWS *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ITEMS3[2] := SOURCE2;
    ITEMS3L[2] := SOURCE2L;
    ATTACH( ITEMS3,ITEMS3L,2,3,0,0,0 );
    DEFINE_VIEW_CONNECT( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                STRPTR,STRPTRL,SOURCE2,SOURCE2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWCK> ::= CREATE CONNECT VIEW
 ;
<CVC_REST> ::= <NEWVIEW> FROM <SOURCE1> <STRPTR> <SOURCE2>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN;
    STRPTR := SSTACK[MP+3].IDNAME;
    STRPTRL:= SSTACK[MP+3].IDLEN;
    SOURCE2 := SSTACK[SP].IDNAME;
    SOURCE2L := SSTACK[SP].IDLEN
 END;
<SOURCE1> ::= <ID>
 ;
<SOURCE2> ::= <ID>
 ;
<NEWVIEW> ::= <ID>
 ;
<STRPTR> ::= <ID>
 ;
*===================================
* CREATE VIEW - JOIN
*===================================
<CREATE_VIEWJ> ::= <CREATE_VIEWJK> <CVJ_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEWS *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ITEMS3[2] := SOURCE2;
    ITEMS3L[2] := SOURCE2L;
    ATTACH( ITEMS3,ITEMS3L,2,3,0,0,0 );
    DEFINE_VIEW_JOIN( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                ITEM1,ITEM1L,SOURCE2,SOURCE2L,
                ITEM2,ITEM2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWJK> ::= CREATE JOIN VIEW
 ;
<CVJ_REST> ::= <NEWVIEW> FROM <SOURCE1> <ID> <SOURCE2> <ID>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
```

```
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN;
    ITEM1 := SSTACK[MP+3].IDNAME;
    ITEM1L:= SSTACK[MP+3].IDLEN;
    SOURCE2 := SSTACK[MP+4].IDNAME;
    SOURCE2L := SSTACK[MP+4].IDLEN;
    ITEM2 := SSTACK[SP].IDNAME;
    ITEM2L:= SSTACK[SP].IDLEN
 END;
*====================================
* CREATE VIEW - ORDER
*====================================
<CREATE_VIEWO> ::= <CREATE_VIEWOK> <CVO_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_ORDER( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,
                IDCOUNT,ITEMS1,ITEMS1L,ASC_OR_DESC );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWOK> ::= CREATE ORDER VIEW
 BEGIN
    IDCOUNT := 0;
    ASC_OR_DESC := 0
 END;
<CVO_REST> ::= <NEWVIEW> FROM <SOURCE1> <ORDER_ITEMS> <ASC_OR_DESC>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<ORDER_ITEMS> ::= <ORDER_ITEM>
 ;
<ORDER_ITEMS> ::= <ORDER_ITEMS> <ORDER_ITEM>
 ;
<ORDER_ITEM> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<ASC_OR_DESC> ::=
 ;
<ASC_OR_DESC> ::= ASCENDING
 ASC_OR_DESC := 0;
<ASC_OR_DESC> ::= DESCENDING
 ASC_OR_DESC := 1;
*====================================
* CREATE VIEW - PROJECT
*====================================
<CREATE_VIEWP> ::= <CREATE_VIEWPK> <CVP_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
```

```
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_PROJECT( NEWVIEW,NEWVIEWL,INC_EXC,SOURCE1,SOURCE1L,
                         ALL_INDICATOR,IDCOUNT,ITEMS1,ITEMS1L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWPK> ::= CREATE PROJECT VIEW
 BEGIN
    IDCOUNT := 0;
    ALL_INDICATOR := FALSE;
    INC_EXC := 0
 END;
<CVP_REST> ::= <NEWVIEW> FROM <SOURCE1> <INC_EXC> <PROJECT_IDS>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<PROJECT_IDS> ::= ALL-ITEMS
 ALL_INDICATOR := TRUE;
<PROJECT_IDS> ::= <PROJECT_ID>
 ;
<PROJECT_IDS> ::= <PROJECT_IDS> <PROJECT_ID>
 ;
<PROJECT_ID> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<INC_EXC> ::=
 ;
<INC_EXC> ::= INCLUDING
 INC_EXC := 0;
<INC_EXC> ::= EXCLUDING
 INC_EXC := 1;
*====================================
* CREATE VIEW - SELECT
*====================================
<CREATE_VIEWS> ::= <CREATE_VIEWSK> <CVS_REST>
 BEGIN
    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SOURCE1;
    ITEMS3L[1]:= SOURCE1L;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    DEFINE_VIEW_SELECT( NEWVIEW,NEWVIEWL,SOURCE1,SOURCE1L,COMPARATOR,
                REC_SEARCH,IDCOUNT,ITEMS1,ITEMS1L,ITEMS2,ITEMS2L );
    KEEP_VIEW( NEWVIEW,NEWVIEWL,NEWVIEW,NEWVIEWL );
    QUERY
 END;
<CREATE_VIEWSK> ::= CREATE SELECT VIEW
 BEGIN
    REC_SEARCH := 0;
    IDCOUNT := 0
```

```
 END;
<CVS_REST> ::= <NEWVIEW> FROM <SOURCE1> WHERE <WHERE_CLAUSE> <SELECT_OPTIONS>
 BEGIN
    NEWVIEW := SSTACK[MP].IDNAME;
    NEWVIEWL:= SSTACK[MP].IDLEN;
    SOURCE1 := SSTACK[MP+2].IDNAME;
    SOURCE1L:= SSTACK[MP+2].IDLEN
 END;
<WHERE_CLAUSE> ::= <SINGLE_CLAUSE>
 ;
<WHERE_CLAUSE> ::= <WHERE_CLAUSE> AND <SINGLE_CLAUSE>
 ;
<SINGLE_CLAUSE> ::= <FIRST> <BINARY> <SECOND>
 ;
<FIRST> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[ SP ].IDNAME;
    ITEMS1L[ IDCOUNT ] := SSTACK[ SP ].IDLEN
 END;
<SECOND> ::= <ID>
 BEGIN
    ITEMS2[ IDCOUNT ] := SSTACK[ SP ].IDNAME;
    ITEMS2L[ IDCOUNT ] := SSTACK[ SP ].IDLEN
 END;
<SECOND> ::= <NO>
 BEGIN
    NUM_TO_ASCII( SSTACK[SP].IVAL,ITEM1,ITEM1L );
    ITEMS2[ IDCOUNT ] := ITEM1 ;
    ITEMS2L[ IDCOUNT ] := ITEM1L
 END;
<SINGLE_CLAUSE> ::= <FIRST> <UNARY>
 ;
<BINARY> ::= =
 COMPARATOR[ IDCOUNT ] := 20;
<BINARY> ::= <>
 COMPARATOR[ IDCOUNT ] := 30;
<BINARY> ::= <
 COMPARATOR[ IDCOUNT ] := 40;
<BINARY> ::= <=
 COMPARATOR[ IDCOUNT ] := 60;
<BINARY> ::= >=
 COMPARATOR[ IDCOUNT ] := 50;
<BINARY> ::= >
 COMPARATOR[ IDCOUNT ] := 70;
<UNARY> ::= IS VALUED
 COMPARATOR[ IDCOUNT ] := 1;
<UNARY> ::= IS NULL
 COMPARATOR[ IDCOUNT ] := 11;
<SELECT_OPTIONS> ::= PERFORM
 REC_SEARCH := 0;
<SELECT_OPTIONS> ::= NOPERFORM
 REC_SEARCH := 1;
<SELECT_OPTIONS> ::=
 ;
*==================================
* DELETE DATABASE
*==================================
<DELETE_DATABASE> ::= <DELETE_DATABASEK> <ID>
 BEGIN
```

```
    START;
    DELETE_DATABASE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_DATABASEK> ::= DELETE DATABASE
 ;
*====================================
* DELETE RELATION
*====================================
<DELETE_RELATION> ::= <DELETE_RELATIONK> <ID>
 BEGIN
    START;
    (* ATTACH THE IDENTITY VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SSTACK[SP].IDNAME;
    ITEMS3L[1] := SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,4,0,0,0 );
    DELETE_FILE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_RELATIONK> ::= DELETE RELATION
 ;
*====================================
* DELETE VIEW
*====================================
<DELETE_VIEW> ::= <DELETE_VIEWK> <ID>
 BEGIN
    (* FIRST FIND THE UNDERLYING FILE- IDENTITY VIEW NAME *)
    (* THIS IDENTITY VIEW MUST BE ATTACHED WITH 'ADMIN ' *)

    START;
    DESCRIBE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,FALSE );
    SEND;
    RECV( RESPONSE,TOTAL_BYTES,MORE );
    DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
                 ITEMS1,ITEMS1L,VALS1,VALS2 );
    START;
    FREE( BLANK_IDENT,1,TRUE );
    ITEMS3[1] := ITEM2;
    ITEMS3L[1] := ITEM2L;
    ATTACH( ITEMS3,ITEMS3L,1,4,0,0,0 );
    DELETE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN );
    QUERY
 END;
<DELETE_VIEWK> ::= DELETE VIEW
 ;
*====================================
* DETACH
*====================================
<DETACH> ::= <DETACHK> <DETACH_WHAT>
 BEGIN
    START;
    DETACH( ITEM1,ITEM1L,ALL_INDICATOR );
    QUERY
 END;
<DETACHK> ::= DETACH
 ALL_INDICATOR := FALSE;
<DETACH_WHAT> ::= <ID>
 BEGIN
    ITEM1 := SSTACK[SP].IDNAME;
```

```
   ITEM1L:= SSTACK[SP].IDLEN
 END;
<DETACH_WHAT> ::= ALL
 ALL_INDICATOR := TRUE;
*====================================
* DISPLAY
*====================================
<DISPLAY> ::= DISPLAY <ID>
 BEGIN
    START;
    DESCRIBE_VIEW( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,FALSE );
    SEND;
    RECV( RESPONSE,TOTAL_BYTES,MORE );
    DV_RESPONSE( ITEM1,ITEM1L,ITEM2,ITEM2L,NUM_ITEMS,
                 ITEMS1,ITEMS1L,VALS1,VALS2 );
    (* PROVIDE A HEADER FOR THE VIEW DISPLAY *)
    WRITELN;
    WRITELN;
    FOR COUNT := 1 TO NUM_ITEMS DO
    BEGIN
       FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
           WRITE( ITEMS1[ COUNT,COUNT2 ] );
       COUNTER := VALS2[ COUNT ] - ITEMS1L[ COUNT ];
       IF COUNTER > 0 THEN
       BEGIN
          FOR COUNTER2 := 1 TO COUNTER DO
              WRITE(' ')
       END;
       WRITE('  ')
    END;
    WRITELN;
    (* DRAW THE UNDERLINING *)
    FOR COUNT := 1 TO NUM_ITEMS DO
    BEGIN
       FOR COUNT2 := 1 TO ITEMS1L[ COUNT ] DO
           WRITE('_');
       COUNTER := VALS2[ COUNT ] - ITEMS1L[ COUNT ];
       IF COUNTER > 0 THEN
       BEGIN
          FOR COUNTER2 := 1 TO COUNTER DO
              WRITE(' ')
       END;
       WRITE('  ')
    END;
    WRITELN;
    WRITELN;

    START;
    (* ATTACH THE SOURCE VIEW *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1]  := SSTACK[SP].IDNAME;
    ITEMS3L[1]:= SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    START_CURSOR(1,SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,
          0,0,1 );
    FETCH(1,0);
    END_CURSOR(1,TRUE);
    QUERY
 END;
*====================================
```

```
* ECHO
*===================================
<ECHO> ::= <ECHOK> <ID>
 BEGIN
    START;
    REMARK( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,1);
    QUERY
 END;
<ECHOK> ::= ECHO
 ;
*===================================
* HELP
*===================================
<HELP> ::= <HELPK> <HELP_COMMAND>
 ;
<HELPK> ::= HELP
 ;
<HELP_COMMAND> ::=
 BEGIN
   WRITELN;WRITELN;
   WRITELN('The following commands are available :');
   WRITELN;
   WRITELN('CREATE   DELETE   DISPLAY   ECHO  INPUT   LIST');
   WRITELN('LOAD     PERFON   PERFOFF   TRACEON');
   WRITELN('TRACEOFF');
   WRITELN;
   WRITELN('General Notes :');WRITELN;
   WRITELN('1. To continue lines of input use a dash( - ) at the end');
   WRITELN('   of each line to be continued.');
   WRITELN('2. To exit DBPQL, just enter X at the prompt.');
   WRITELN;
   WRITELN('For a specific help, type HELP <command>')
 END;
<HELP_COMMAND> ::= CREATE
 BEGIN
    WRITELN;WRITELN;
    WRITELN('You can create several different entities :');WRITELN;
    WRITELN('CREATE DATABASE');
    WRITELN('CREATE RELATION');
    WRITELN('CREATE CONNECT VIEW');
    WRITELN('CREATE JOIN VIEW');
    WRITELN('CREATE PROJECT VIEW');
    WRITELN('CREATE SELECT VIEW');
    WRITELN('CREATE ORDER VIEW');WRITELN;
    WRITELN('To get more help on any one of these,');
    WRITELN('type HELP <one of the above lines>')
 END;
<HELP_COMMAND> ::= CREATE DATABASE
 BEGIN
    WRITELN;WRITELN;
    WRITELN('CREATE DATABASE <DBNAME>');
    WRITELN('where <DBNAME> is the name of the database to be created.')
 END;
<HELP_COMMAND> ::= CREATE RELATION
 BEGIN
    WRITELN;WRITELN;
    WRITELN('You can define a new schema for a relation as follows:');
    WRITELN('CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA');
    WRITELN('  { <ITEMNAME> <TYPE1> <TYPE2> <LENGTH> }');
    WRITELN('  where : <TYPE1> = SIGNED_INT or INTEGER');
```

```
   WRITELN('                    UNSIGNED_INT');
   WRITELN('                    UNINTERPRET');
   WRITELN('                    ASCII');
   WRITELN('                    RECORD_PTR');
   WRITELN('                    STRING_PTR');
   WRITELN('          <TYPE2> = FIXED');
   WRITELN('                    VAR');
   WRITELN('          <LENGTH>= of item in bytes');
   WRITELN;
   WRITELN('You can also create a new relation which uses the');
   WRITELN('already defined schema of any given view :');
   WRITELN;
   WRITELN('CREATE RELATION <RELNAME> USING VIEW <OLD_VIEW>')
 END;
<HELP_COMMAND> ::= CREATE CONNECT VIEW
 BEGIN
   WRITELN;WRITELN;
   WRITELN('CREATE CONNECT VIEW <NEW_VIEW> <REST>');
   WRITELN('<REST>      = FROM <SOURCE_VIEW1> <STRING_PTR> <SOURCE_VIEW2>')
 END;
<HELP_COMMAND> ::= CREATE JOIN VIEW
 BEGIN
   WRITELN;WRITELN;
   WRITELN('CREATE JOIN VIEW <NEW_VIEW> <REST>');
   WRITELN('<REST>      = FROM <SOURCE_VIEW1> <ITEM1> <SOURCE_VIEW2> <ITEM2>')
 END;
<HELP_COMMAND> ::= CREATE PROJECT VIEW
 BEGIN
   WRITELN;WRITELN;
   WRITELN('CREATE PROJECT VIEW <NEW_VIEW> <REST>');
   WRITELN('<REST>      = FROM <SOURCE_VIEW> <INC_EXC> <PROJECT_ITEMS>');
   WRITELN('where <INC_EXC>= INCLUDING( default ) or EXCLUDING');
   WRITELN('and <PROJECT_ITEMS> = sequence of items to project')
 END;
<HELP_COMMAND> ::= CREATE SELECT VIEW
 BEGIN
   WRITELN;WRITELN;
   WRITELN('CREATE SELECT VIEW <NEW_VIEW> <REST>');
   WRITELN('<REST>      = FROM <SOURCE_VIEW> WHERE <WHERE_CLAUSE> <OPTIONS>');
   WRITELN;
   WRITELN('<WHERE_CLAUSE>  = sequence of <BINARY> or <UNARY> clause(s),');
   WRITELN('separated by AND');WRITELN;
   WRITELN('where <BINARY>  = <ITEM> <BINARY_OP> <VALUE>');
   WRITELN('<BINARY_OP> =    = , <> , < , > , <= , >= ');WRITELN;
   WRITELN('and   <UNARY>   = <ITEM> <UNARY_OP>');
   WRITELN('<UNARY_OP>  =    IS VALUED or IS NULL')
 END;
<HELP_COMMAND> ::= CREATE ORDER VIEW
 BEGIN
   WRITELN;WRITELN;
   WRITELN('CREATE ORDER VIEW <NEW_VIEW> <REST>');
   WRITELN('where <REST>   = FROM <SOURCE_VIEW> <ORDER_ITEMS> <ASC_OR_DESC>');
   WRITELN;
   WRITELN('<ORDER_ITEMS>  = sequence of items to sort');
   WRITELN('<ASC_OR_DESC>  = ASCENDING( default ) or DESCENDING')
 END;
<HELP_COMMAND> ::= DELETE
 BEGIN
   WRITELN;WRITELN;
   WRITELN('You may delete a DATABASE, RELATION, or VIEW by');
```

```
      WRITELN('saying DELETE <which-type> <identifier of thing to delete>')
   END;
 <HELP_COMMAND> ::= DISPLAY
   BEGIN
      WRITELN;WRITELN;
      WRITELN('This command allows you to see a view on your terminal.');
      WRITELN('Just say, DISPLAY <view name>')
   END;
 <HELP_COMMAND> ::= ECHO
   BEGIN
      WRITELN;WRITELN;
      WRITELN('This command serves as a simple test to see if the DBP');
      WRITELN('is up and running. Say ECHO <any-word> and you should');
      WRITELN('receive an echo of the word you received. If you do not');
      WRITELN('then the communications from the VAX to the DBP has not');
      WRITELN('been initialized correctly.')
   END;
 <HELP_COMMAND> ::= INPUT
   BEGIN
      WRITELN;WRITELN;
      WRITELN('Take all further command input from the file referenced');
      WRITELN('using logical name DBPIN. Assign the logical name prior');
      WRITELN('to invoking DBPQL. As an example :');WRITELN;
      WRITELN('$ ASSIGN DBP.DAT DBPIN');
      WRITELN('will assign the file DBP.DAT to the logical name DBPIN.');
   END;
 <HELP_COMMAND> ::= LIST
   BEGIN
      WRITELN;WRITELN;
      WRITELN('Type help <one-of-the-following> for further help on list:');
      WRITELN('LISTVIEW');
      WRITELN('LISTVIEWS');
      WRITELN('LISTDB');
      WRITELN('LISTDBS');
   END;
 <HELP_COMMAND> ::= LISTVIEWS
   BEGIN
      WRITELN;WRITELN;
      WRITELN('LISTVIEWS gives a list of all available views.')
   END;
 <HELP_COMMAND> ::= LISTDB
   BEGIN
      WRITELN;WRITELN;
      WRITELN('Two forms: LISTDB <DBNAME>  and  LISTDB ALL' );
      WRITELN('The database entities : file and view are listed for');
      WRITELN('the given database(s).');WRITELN;
      WRITELN('See HELP LISTDBS for a brief form which lists only database names.')
   END;
 <HELP_COMMAND> ::= LISTDBS
   BEGIN
      WRITELN;WRITELN;
      WRITELN('LISTDBS lists all currently available iDBP databases.');
      WRITELN('For a list containing file and view names, see HELP LISTDB.')
   END;
 <HELP_COMMAND> ::= LISTVIEW
   BEGIN
      WRITELN;WRITELN;
      WRITELN('LISTVIEW <view name> gives the structure of the specified view.');
      WRITELN('The item names, data types, and item lengths are printed.')
   END;
```

```
<HELP_COMMAND> ::= LOAD
 BEGIN
    WRITELN;WRITELN;
    WRITELN('LOAD <view name> <tuples>');WRITELN;
    WRITELN('where <tuples> is a sequence of tuples and each tuple');
    WRITELN('is in the form :  [ value1 value2 value3 ... ]');
    WRITELN;WRITELN('Note the brackets must be included.')
 END;
<HELP_COMMAND> ::= PERFON
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns on the Performance Tracing.');
    WRITELN('The following statistics are measured :');WRITELN;
    WRITELN('1. Elapsed Clock Time');WRITELN('2. Elapsed CPU Time');
    WRITELN('3. Buffered I/O Count');WRITELN('4. Direct I/O Count');
    WRITELN('5. Page Fault Count')
 END;
<HELP_COMMAND> ::= PERFOFF
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the performance tracing off --> see HELP PERFON')
 END;
<HELP_COMMAND> ::= TRACEON
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the byte tracing mechanism on. This mechanism allows');
    WRITELN('the system developer to view the exact form of the');
    WRITELN('request and response blocks sent/received over the');
    WRITELN('communications line between the VAX and the DBP.')
 END;
<HELP_COMMAND> ::= TRACEOFF
 BEGIN
    WRITELN;WRITELN;
    WRITELN('Turns the byte tracing mechanism off --> see HELP TRACEON')
 END;
*====================================
* INPUT FROM DEVICE
*====================================
<INPUT> ::= INPUT
 BEGIN
    INPUT_FIRST := TRUE;
    INPUT_FILE := TRUE
 END;
*====================================
* LIST DATABASES
*====================================
<LIST_DBS> ::= LISTDBS
 BEGIN
    START;
    LIST_DATABASE(BLANK_IDENT,1,FALSE,TRUE);
    QUERY
 END;
*====================================
* LIST DATABASE
*====================================
<LIST_DB> ::= <LISTDBK> <LISTDB_REST>
 QUERY;
<LISTDBK> ::= LISTDB
 START;
<LISTDB_REST> ::= <ID>
```

```
 LIST_DATABASE( SSTACK[SP].IDNAME,SSTACK[SP].IDLEN,TRUE,FALSE );
<LISTDB_REST> ::= ALL
 LIST_DATABASE( BLANK_IDENT,1,TRUE,TRUE );
*===================================
* LIST VIEWS
*===================================
<LIST_VIEWS> ::= <LIST_VIEWSK>
 BEGIN
    START;
    LIST_VIEWS;
    QUERY
 END;
<LIST_VIEWSK> ::= LISTVIEWS
 ;
*===================================
* LIST VIEW
*===================================
<LIST_VIEW> ::= <LIST_VIEWK> <WHICH_VIEW>
 BEGIN
    START;
    DESCRIBE_VIEW( ITEM1,ITEM1L,FALSE );
    QUERY
 END;
<LIST_VIEWK> ::= LISTVIEW
 ;
<WHICH_VIEW> ::= <ID>
 BEGIN
    ITEM1 := SSTACK[SP].IDNAME;
    ITEM1L:= SSTACK[SP].IDLEN
 END;
*===================================
* LOAD
*===================================
<LOAD> ::= <LOADK> <LV_REST>
 BEGIN
    END_CURSOR(1,TRUE);
    QUERY
 END;
<LOADK> ::= LOAD <ID>
 BEGIN
    START;
    SC_MODE := 1; (* RANDOM *)
    SC_DIRECTION := 0; (* FORWARD-ONLY *)
    SC_RETEST := 0;

    (* START A CURSOR FOR LOADING THIS VIEW *)

    (* ATTACH THE SOURCE VIEW, FIRST *)
    FREE(BLANK_IDENT,1,TRUE);
    ITEMS3[1] := SSTACK[SP].IDNAME;
    ITEMS3L[1]:= SSTACK[SP].IDLEN;
    ATTACH( ITEMS3,ITEMS3L,1,3,0,0,0 );
    START_CURSOR( 1,SSTACK[MP+1].IDNAME,SSTACK[MP+1].IDLEN,
          SC_MODE,SC_DIRECTION,SC_RETEST )
 END;
<LV_REST> ::= <TUPLES>
 ;
<TUPLES> ::= <TUPLE>
 ;
<TUPLES> ::= <TUPLES> <TUPLE>
```

```
 ;
<TUPLE> ::= <START_TUPLE> <LV_ITEMS> <END_TUPLE>
 STORE( 1,INTEGRITY,IDCOUNT,ITEMS1,ITEMS1L );
<START_TUPLE> ::= [
 IDCOUNT := 0;
<END_TUPLE> ::= ]
 ;
<LV_ITEMS> ::= <LV_ITEM>
 ;
<LV_ITEMS> ::= <LV_ITEMS> <LV_ITEM>
 ;
<LV_ITEM> ::= <ID>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    ITEMS1[ IDCOUNT ] := SSTACK[SP].IDNAME;
    ITEMS1L[ IDCOUNT ] := SSTACK[SP].IDLEN
 END;
<LV_ITEM> ::= <NO>
 BEGIN
    IDCOUNT := IDCOUNT + 1;
    NUM_TO_ASCII( SSTACK[SP].IVAL,ITEM1,ITEM1L );
    ITEMS1[ IDCOUNT ] := ITEM1;
    ITEMS1L[ IDCOUNT ] := ITEM1L
 END;
*===================================
* PERFORMANCE ON
*===================================
<PERFORM_COMMAND> ::= PERFON
 BEGIN
    WRITELN;
    WRITELN('Performance Monitoring is turned on.');
    PERFON
 END;
*===================================
* PERFORMANCE OFF
*===================================
<PERFORM_COMMAND> ::= PERFOFF
 BEGIN
    WRITELN;
    WRITELN('Performance Monitoring is turned off.');
    PERFOFF
 END;
*===================================
* TRACE ON
*===================================
<TRACE_COMMAND> ::= TRACEON
 BEGIN
    WRITELN;
    WRITELN('Trace is turned on.');
    TRACEON
 END;
*===================================
* TRACE OFF
*===================================
<TRACE_COMMAND> ::= TRACEOFF
 BEGIN
    WRITELN;
    WRITELN('Trace is turned off.');
    TRACEOFF
 END;
```

APPENDIX G - A sample DBPQL user dialog

```
$
$ ql
  _TTBO: allocated
DBPQL : DBP Query Language Version 1.0
Ok

QL> help

The following commands are available :

CREATE   DELETE   DISPLAY   ECHO  INPUT   LIST
LOAD     PERFON   PERFOFF   TRACEON
TRACEOFF

General Notes :

1. To continue lines of input use a dash( - ) at the end
   of each line to be continued.
2. To exit DBPQL, just enter X at the prompt.

For a specific help, type HELP <command>

QL> help create

You can create several different entities :

CREATE DATABASE
CREATE RELATION
CREATE CONNECT VIEW
CREATE JOIN VIEW
CREATE PROJECT VIEW
CREATE SELECT VIEW
CREATE ORDER VIEW

To get more help on any one of these,
type HELP <one of the above lines>

QL> create database fem
Ok

QL> help create relation

You can define a new schema for a relation as follows:
CREATE RELATION <RELNAME> IN <DBNAME> USING SCHEMA
  { <ITEMNAME> <TYPE1> <TYPE2> <LENGTH> }
  where : <TYPE1> = SIGNED_INT or INTEGER
                    UNSIGNED_INT
                    UNINTERPRET
                    ASCII
                    RECORD_PTR
                    STRING_PTR
          <TYPE2> = FIXED
                    VAR
          <LENGTH>= of item in bytes

You can also create a new relation which uses the
already defined schema of any given view :

CREATE RELATION <RELNAME> USING VIEW <OLD_VIEW>
```

```
QL> create relation beams in fem using schema -
QL> group integer fixed 4 -
QL> element integer fixed 4 -
QL> node1 integer fixed 4 -
QL> node2 integer fixed 4 -
QL> el-type ascii fixed 4 -
QL> nom-size ascii fixed 4 -
QL> material ascii fixed 8
Ok, View BEAMS has been created.
Ok

QL> listview beams
View : BEAMS
Underlying Relation : BEAMS
# of items =   7

GROUP                 Signed Integer            4
ELEMENT               Signed Integer            4
NODE1                 Signed Integer            4
NODE2                 Signed Integer            4
EL-TYPE               ASCII Alphanumeric        4
NOM-SIZE              ASCII Alphanumeric        4
MATERIAL              ASCII Alphanumeric        8

QL>
QL>
QL> create relation nodes in fem using schema -
QL> node integer fixed 4 -
QL> xcoord integer fixed 4 -
QL> ycoord integer fixed 4 -
QL> zcoord integer fixed 4
Ok, View NODES has been created.
Ok

QL> listviews

List of Views :

BEAMS
FILE1
NODES
PEOPLE


QL> perfon
Performance Monitoring is turned on.

QL> load beams -
QL> [ 1 1 1 2 wfl w8x8 aluminum ] -
QL> [ 1 2 3 4 i i3x2 titanium ] -
QL> [ 2 3 5 6 wfl w8x8 graphite ]

Clock      27.44922
CPU      0.15000
Buffered I/O count     40
Direct I/O count      0
Page Fault count      0

Ok
```

```
QL>
QL> perfoff
Performance Monitoring is turned off.

QL> load nodes -
QL> [ 1 53 62 0 ] -
QL> [ 2 67 10 0 ] -
QL> [ 3 10 11 0 ] -
QL> [ 4 23 53 0 ] -
QL> [ 5 54 82 2 ] -
QL> [ 6 84 21 2 ]
Ok

QL> display beams
```

GROUP	ELEMENT	NODE1	NODE2	EL-TYPE	NOM-SIZE	MATERIAL
1	1	1	2	WFL	W8X8	ALUMINUM
1	2	3	4	I	I3X2	TITANIUM
2	3	5	6	WFL	W8X8	GRAPHITE

```
QL> display nodes
```

NODE	XCOORD	YCOORD	ZCOORD
1	53	62	0
2	67	10	0
3	10	11	0
4	23	53	0
5	54	82	2
6	84	21	2

```
QL> traceon
Trace is turned on.

QL> display nodes
== DBP REQUEST ==
# of bytes is     9
Byte Stream :

9A 05 4E 4F 44 45 53 FF 00                              ..NODES..


== DBP RESPONSE ==
# of bytes is   112
Byte Stream :

F9 02 9A 00 01 00 01 00 06 04 03 07 00 05 00 05  ................
4E 4F 44 45 53 05 4E 4F 44 45 53 00 00 00 01 01  NODES.NODES.....
02 04 00 03 00 01 04 04 4E 4F 44 45 00 05 4E 4F  ........NODE..NO
44 45 53 03 01 01 04 06 58 43 4F 4F 52 44 00 05  DES.....XCOORD..
4E 4F 44 45 53 03 02 01 04 06 59 43 4F 4F 52 44  NODES.....YCOORD
00 05 4E 4F 44 45 53 03 03 01 04 06 5A 43 4F 4F  ..NODES.....ZCOO
52 44 00 05 4E 4F 44 45 53 FF 0A 00 00 00 FF 00  RD..NODES.......


NODE   XCOORD   YCOORD   ZCOORD
```

```
== DBP REQUEST ==
# of bytes is    43
Byte Stream :

01 FF 06 FF 00 00 01 00 05 4E 4F 44 45 53 01 30   .........NODES.0
FF 00 02 01 01 05 4E 4F 44 45 53 01 20 FF 00 10   ......NODES. ...
01 01 FF 06 FF 00 03 FF 06 FF 00                  ...........


== DBP RESPONSE ==
# of bytes is   106
Byte Stream :

F1 01 01 01 31 02 35 33 02 36 32 01 30 FF 00 F1   ....1.53.62.0...
01 01 01 32 02 36 37 02 31 30 01 30 FF 00 F1 01   ...2.67.10.0....
01 01 33 02 31 30 02 31 31 01 30 FF 00 F1 01 01   ..3.10.11.0.....
01 34 02 32 33 02 35 33 01 30 FF 00 F1 01 01 01   .4.23.53.0......
35 02 35 34 02 38 32 01 32 FF 00 F1 01 01 01 36   5.54.82.2......6
02 38 34 02 32 31 01 32 FF 00 F6 08 10 00 00 64   .84.21.2.......d
00 00 1F 00 02 00 00 00 FF 00 0A 11 2E D9 0A 11   ................
00 00 1F 00 02 00 00 00 FF 00                     ..........


1      53      62      0
2      67      10      0
3      10      11      0
4      23      53      0
5      54      82      2
6      84      21      2

QL> traceoff
Trace is turned off.

QL> create join view j1 from beams node1 nodes node
Ok


listview j1
View : J1
Underlying Relation : FEM
# of items =  11

GROUP              Signed Integer            4
ELEMENT            Signed Integer            4
NODE1              Signed Integer            4
NODE2              Signed Integer            4
EL-TYPE            ASCII Alphanumeric        4
NOM-SIZE           ASCII Alphanumeric        4
MATERIAL           ASCII Alphanumeric        8
NODE               Signed Integer            4
X                  Signed Integer            4
Y                  Signed Integer            4
Z                  Signed Integer            4
```

```
create select view sel1 from beams where el-type= wfl
Ok

QL> display sel1

GROUP  ELEMENT  NODE1  NODE2  EL-TYPE  NOM-SIZE  MATERIAL
-----  -------  -----  -----  -------  --------  --------

1      1        1      2      WFL      W8X8      ALUMINUM
2      3        5      6      WFL      W8X8      GRAPHITE

QL>
DBPQL - Goodbye.
$
```

FIGURES

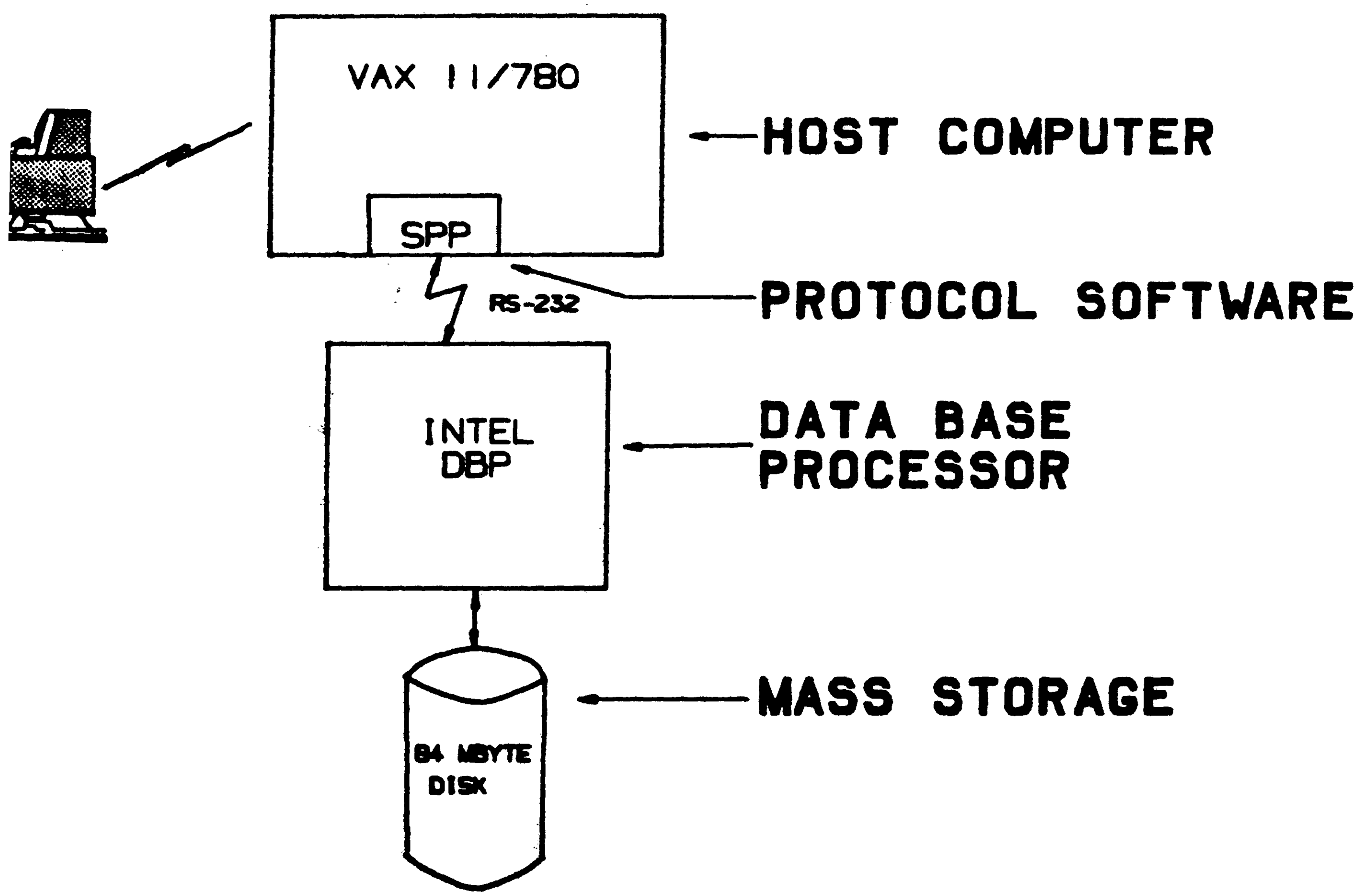

Figure 1 - The Physical DBP Environment

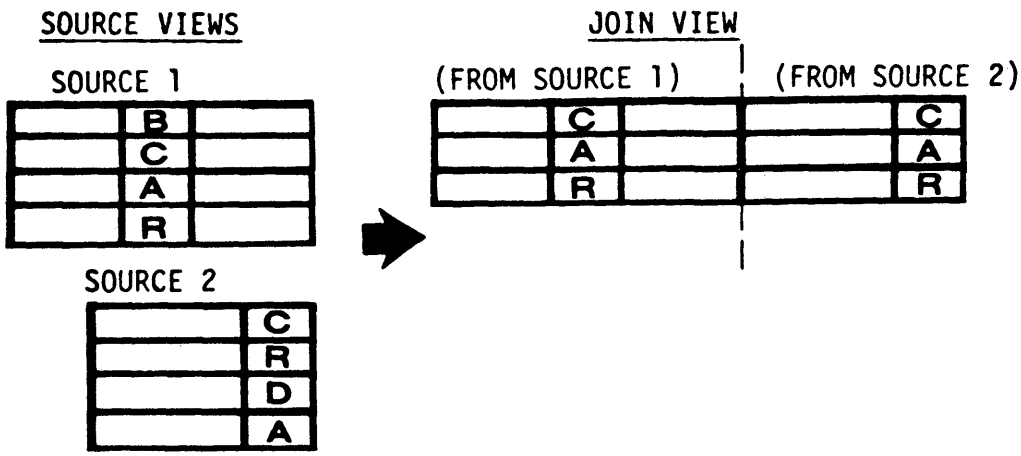

Figure 2a - The JOIN Command

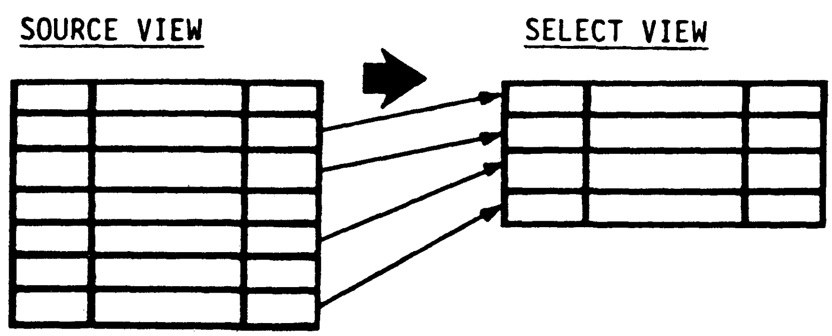

Figure 2b - The SELECT Command

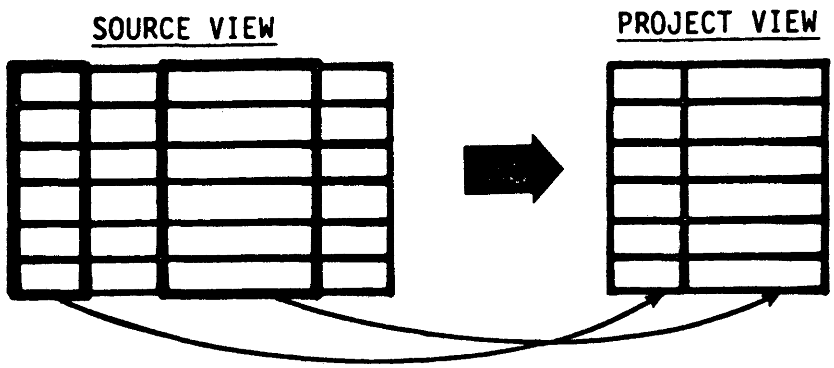

Figure 2c - The PROJECT Command

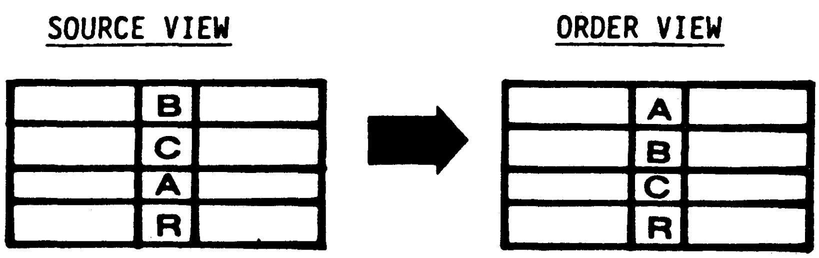

Figure 2d - The ORDER Command

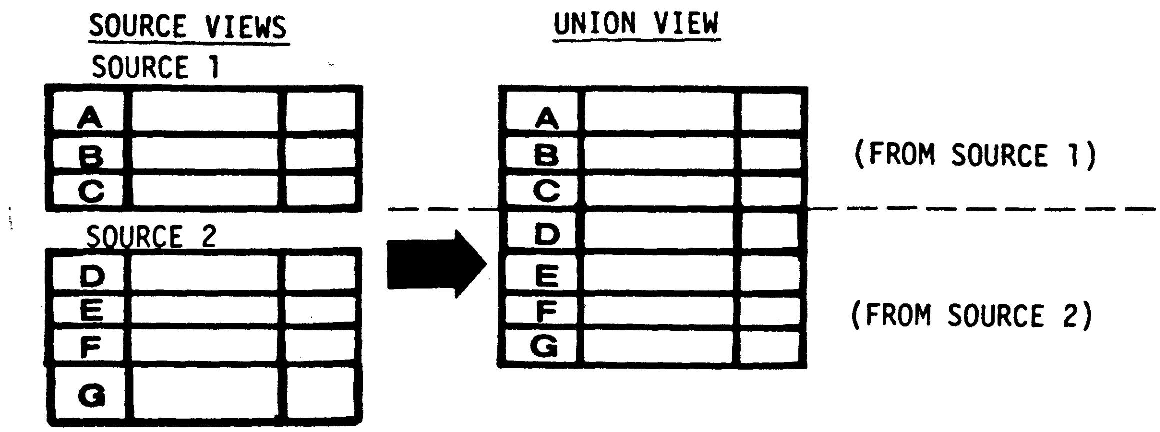

Figure 2e - The UNION Command

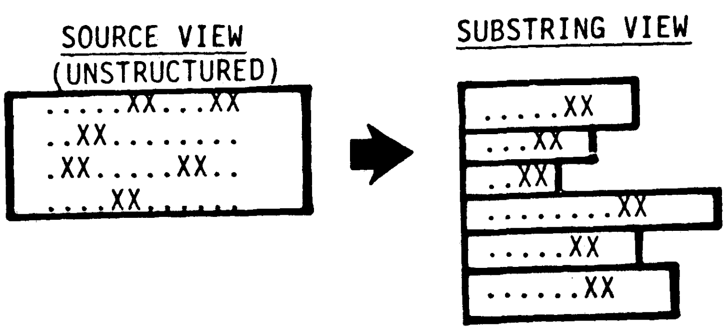

Figure 2f - The SUBSTRING Command

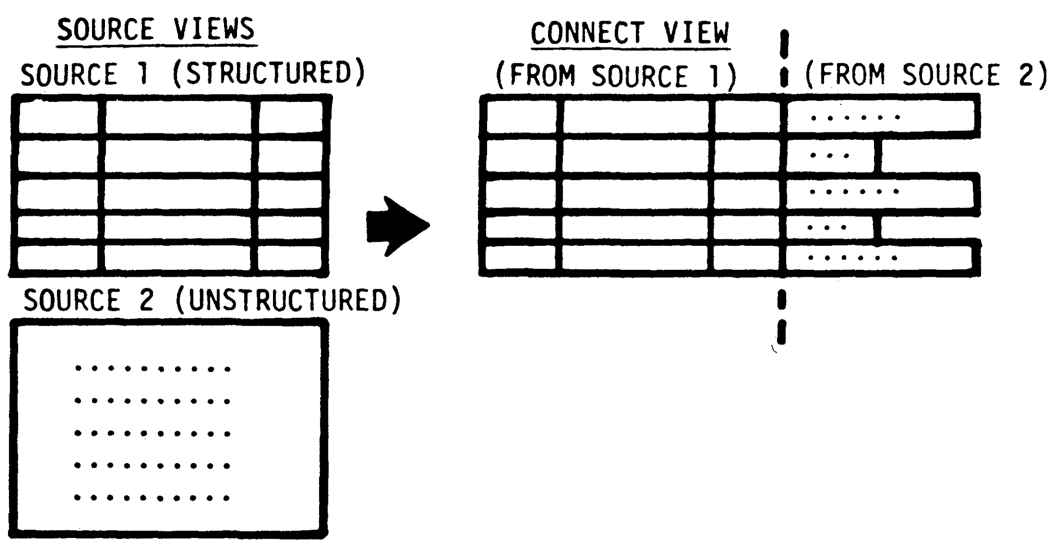

Figure 2g - The CONNECT Command

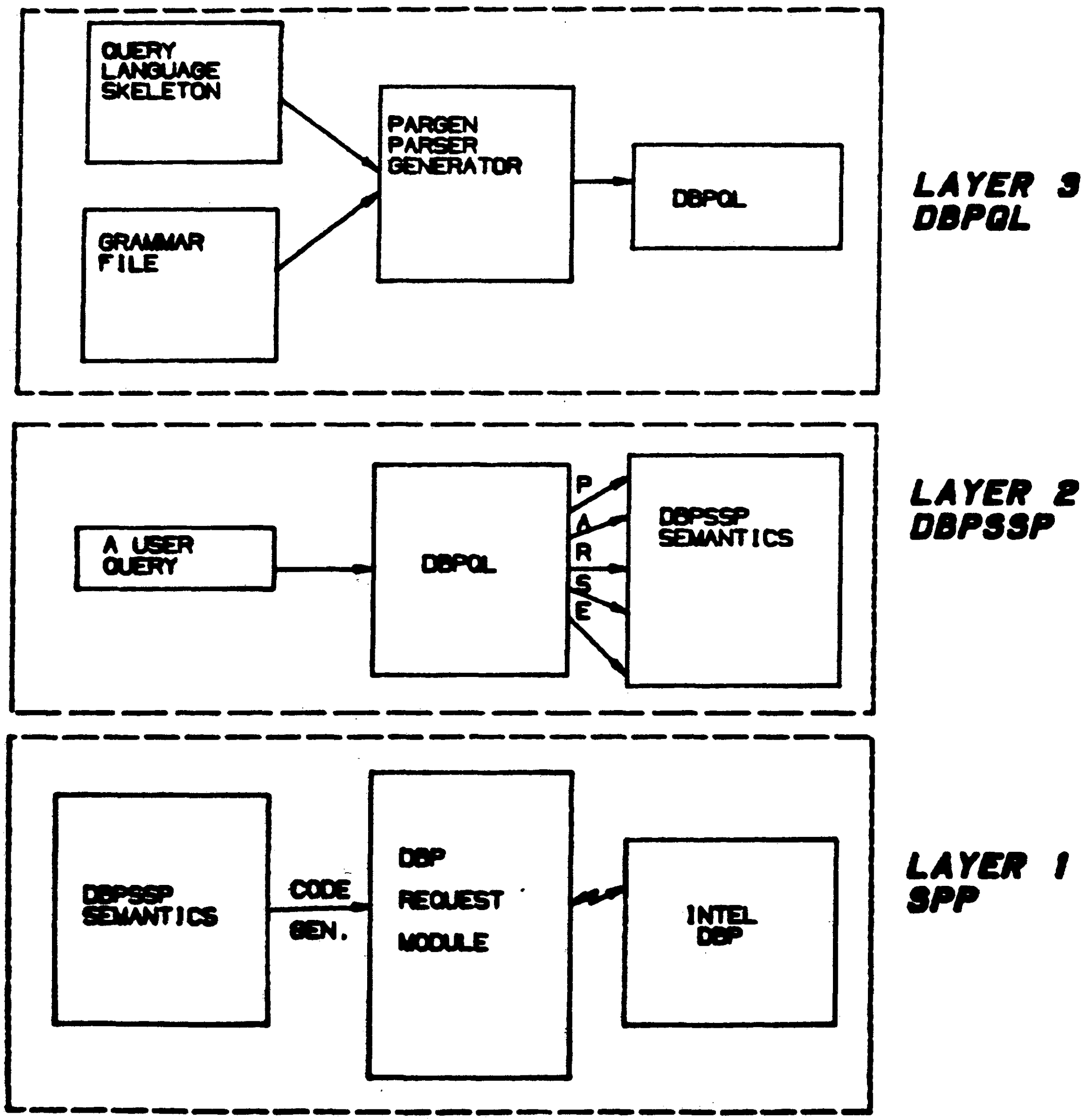

Figure 3 - HILDA : A general flow chart

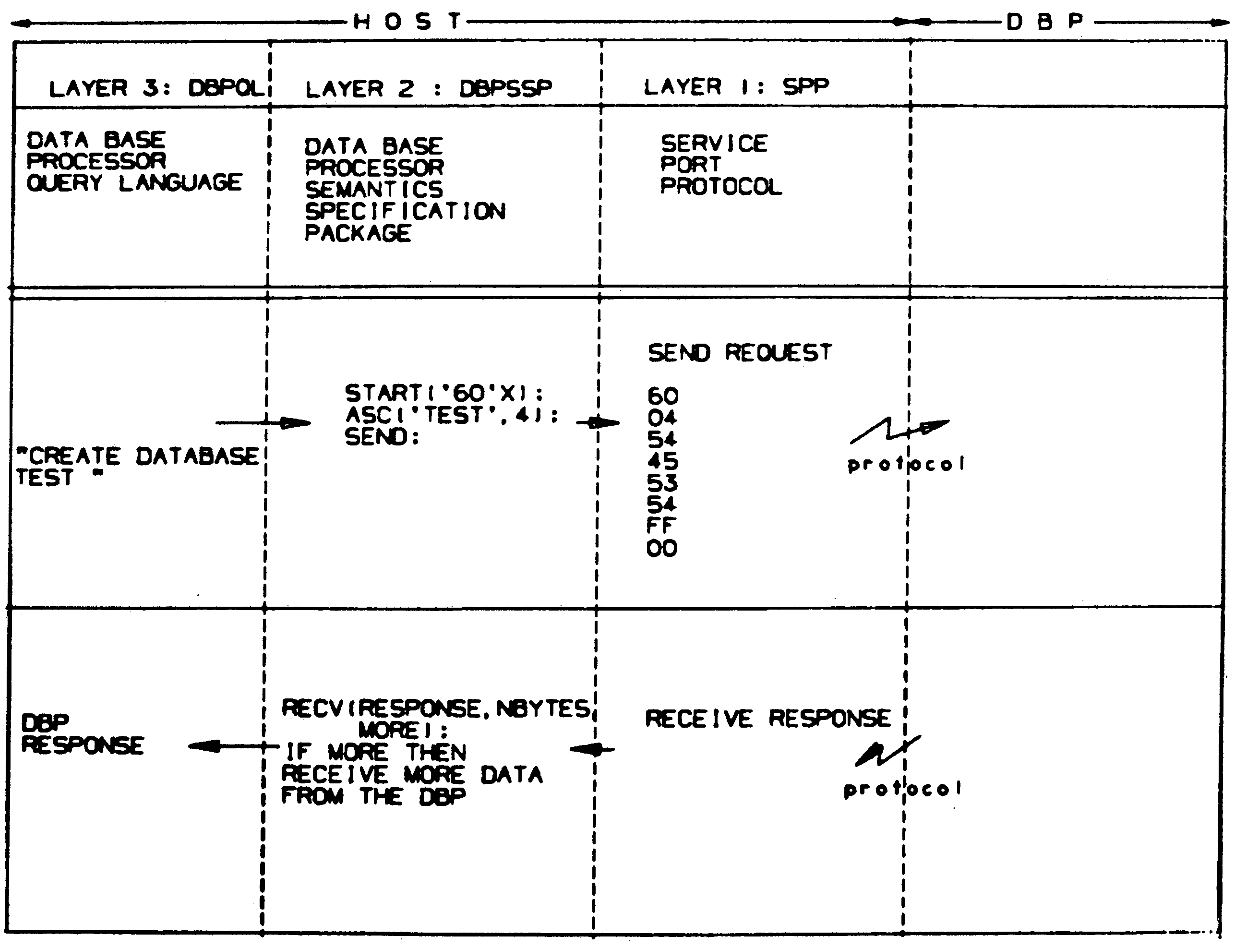

Figure 4 - HILDA : A sample query

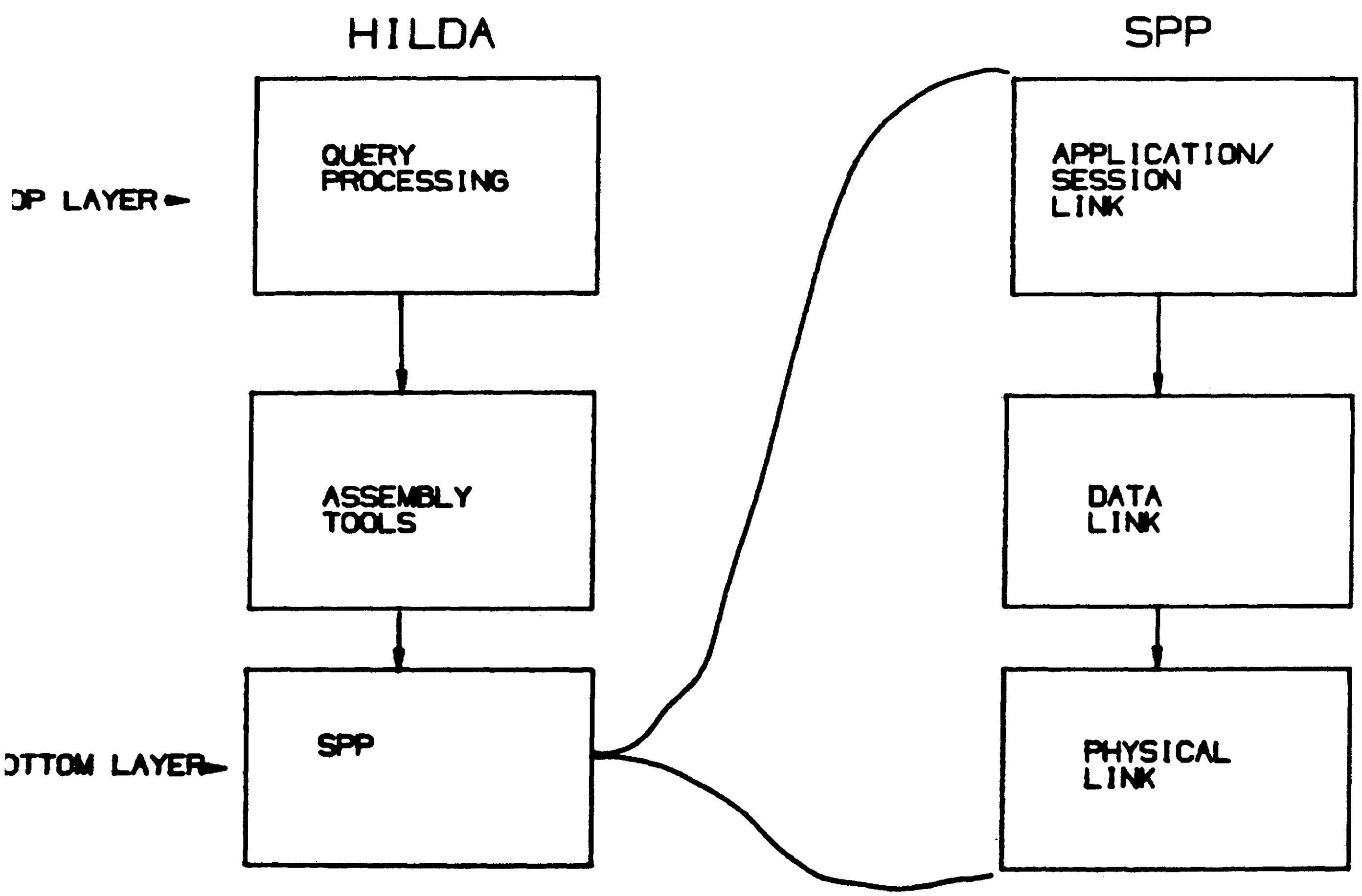

Figure 5 - Layers within HILDA and SPP

If the host sends the following request module to iDBP:

```
|IDID|--------Command-1--------|---------Command-2---------|-Command-3-|
```

Then the host and iDBP will transmit the following segments:
(values are for example only)

```
            HOST                                      iDBP

1.                                         Number of Segments = 1,
                                           Buffer 1 Length = 4

+-----+-----+
| ID  | ID  | 4-byte               ---->
+-----+-----+ header

2.                                         Number of Segments = 1,
                                           Buffer 1 Length = 512

+---------------------------+
|        Command-1-a        | 512  ---->
+---------------------------+ bytes

3.                                         Number of Segments = 1,
                                           Buffer 1 Length = 512

+---------------------------+
| Command-1-b Command-2-a   | 512  ---->
+---------------------------+ bytes

4.                                         Number of Segments = 2,
                                           Buffer 1 Length = 4,
                                           Buffer 2 Length = 512

                                                  +-----------+
                                   <----   4-byte | ID  | ID  |
                                           header +-----------+

                                                +---------------------------+
                                   <----    512 |        Response-A         |
                                          bytes +---------------------------+
```

Figure 6 - General Form for Host-DBP Interaction

```
Terminal: _TTB0:            Device_Type: VT52           Owner: No Owner

   Input:   9600      LFfill:  0      Width:  80      Parity: None
   Output:  9600      CRfill:  0      Page:   24

Terminal Characteristics:
   Passall            Echo                Type_ahead         No Escape
   No Hostsync        TTsync              Lowercase          No Tab
   No Wrap            Scope               No Remote          No Holdscreen
   Eightbit           Broadcast           No Readsync        No Form
   Fulldup            No Modem            No Local_echo      No Autobaud
   No Hangup          No Brdcstmbx        No DMA             No Altypeahd
   Set_speed          No ANSI_CRT         No Regis           No Block_mode
   No Advanced_video  No Edit_mode        No DEC_CRT
```

Figure 7 - VAX Asynchronous Communications Parameters

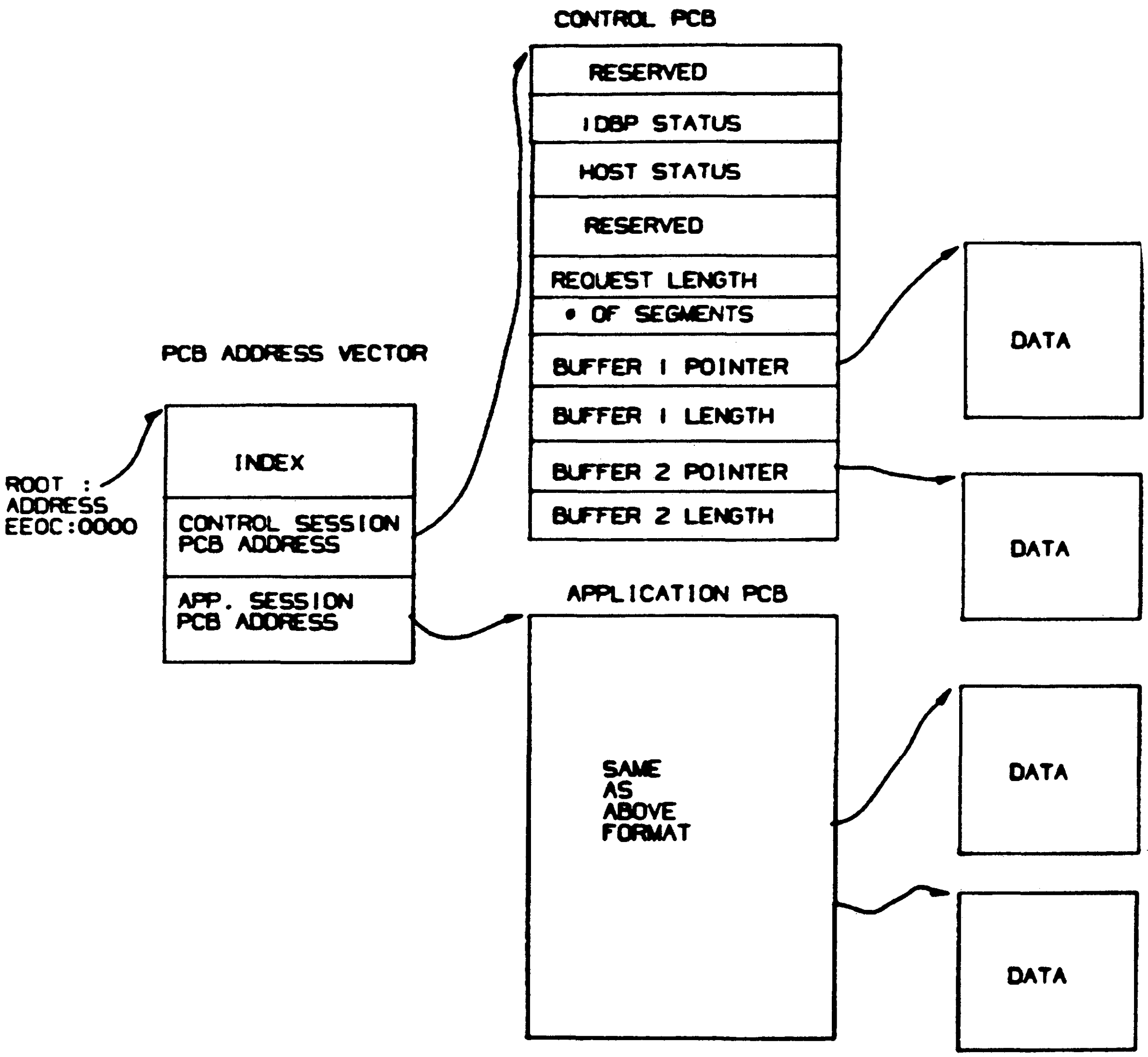

Figure 8 - Threaded Data Structure of SPP

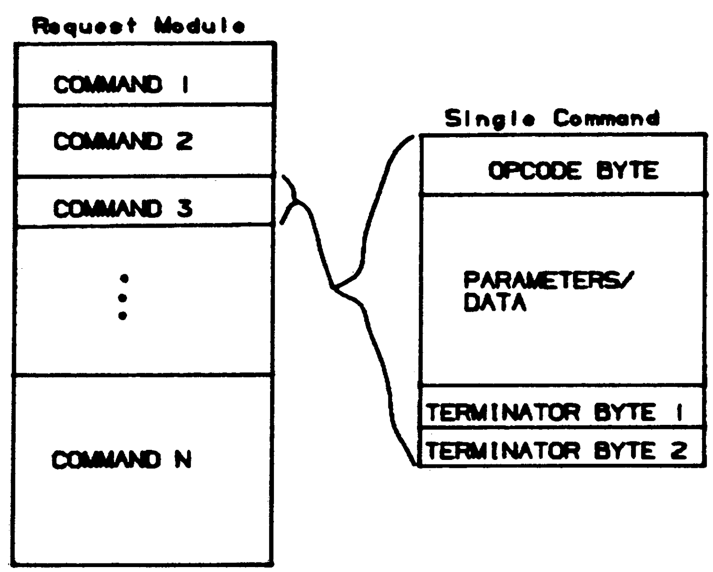

Figure 9 - Request Module Form

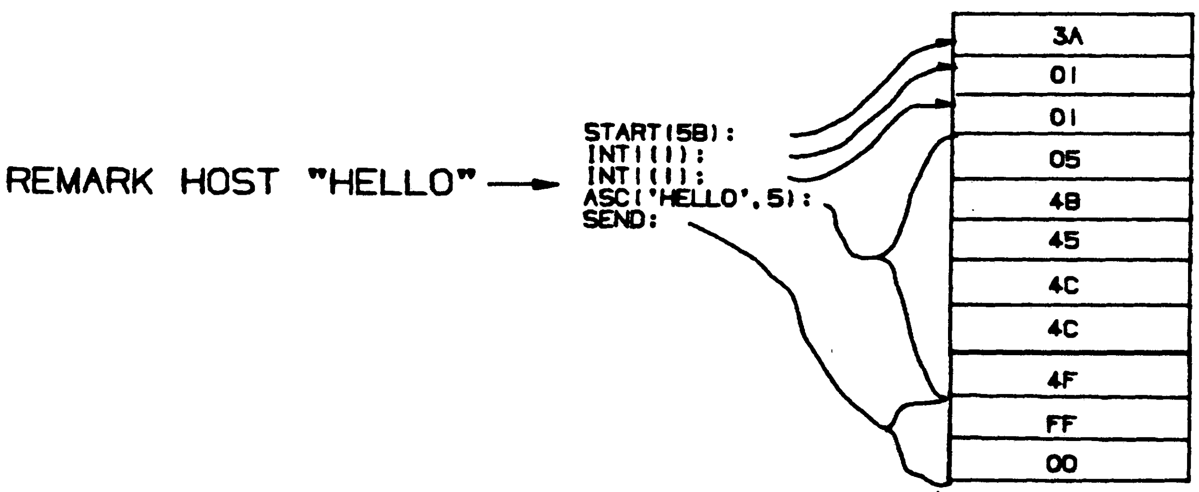

Figure 10 - A sample assembly for "REMARK"

REFERENCES

1. Fulton, R. E.: "IPAD Project Overview," NASA Conference Publication 2143, Sept. 17-19, 1980.

2. Blackburn, C. L.; Storaasli, O. O.; and Fulton, R. E.: "The Role and Application of Data Base Management in Integrated Computer-Aided Design," Proceedings of the AIAA/ASME/ASCE/AHS 23rd Structures, Structural Dynamics, and Materials Conference, New Orleans, LA, May 10-12, 1982.

3. Fishwick, P. A.; and Blackburn, C. L.: "Managing Engineering Data Bases: The Relational Approach," (CIME) Computers in Mechanical Engineering, Vol. 1, No. 3, Jan. 1983.

4. Martin, J. : Computer Data Base Organization, 2nd Ed., Englewood Cliffs, N.J., Prentice-Hall, 1977.

5. Date, C. J. : An Introduction to Data Base Systems, 2nd Ed., New York, NY., Addison-Wesley, 1977.

6. RIM Users Guide, Academic Computer Center, University of Washington, W33, Jan. 1980.

7. Maryanski, Fred J.: "Backend Database Systems," Computing Surveys, Vol. 12, No.1, March 1980.

8. Canaday, R. E.;Harrison, R. D.;Ivie, E. L.; Ryder, J. L.; and Wehr, L. A.: "A Back-End Computer for Data Base Management," Communications of the ACM, Vol. 10, pp. 575-582, Oct. 1974.

9. Codd, E. F.: "Relational Data Base: A Practical Foundation for Productivity," Communications of the ACM, Vol. 25, No. 2, Feb. 1982.

10. DBP DBMS Reference Manual. Intel Corporation, Austin, TX, Revision 001, Order No. 222100-001, August 1982.

11. Davenport, William P. : Modern Data Communication - Concepts , Language, and Media. Hayden Book Company, 1971.

12. VAX/VMS I/O User's Guide(Volume 1). Digital Equipment Corporation, Maynard , MA., Software Version 3.0, May 1982.

13. DBP Operations Manual. Intel Corporation, Austin, TX, Revision 001, Order No. 222101-001, August 1982.

14. DBP Host Link Reference Manual. Intel Corporation, Austin, TX, Revision 001, Order No. 222102-001, August 1982.

15. Noonan, Robert E.; and Collins, Robert: "The Mystro Parser Generator PARGEN User's Manual: Version 6.2," Aug. 1982.

16. Adiba, M.:"Derived Relations: A Unified Mechanism for Views, Snapshots, and Distributed Data," Proceedings

of the Seventh International Conference on Very Large Data Bases, Cannes, France, Sept. 1981.

17. DeRemer, Frank; and Pennello, Thomas J.: "Efficient Computation of LALR(1) Look-ahead Sets," Proceedings of SIGPLAN Symposium on Compiler Construction, pp. 176-187, Aug. 1979.

18. Powell, M. L. and Linton, M. A.: "Database Support for Programming Environments," Proceedings of the Annual Database Week Meeting, Engineering Design Applications, San Jose, May 23-26, 1983.

19. Feyock, Stefan: "Transition diagram-based CAI/HELP systems", International Journal of Man-Machine Studies, Vol. 9, pp. 399-413, 1977.

VITA

Paul Anthony Fishwick

Born in Bebington, Cheshire, England, July 18, 1955. Graduated from Downingtown High School, Pennsylvania, B.S. Mathematics, Pennsylvania State University. M.S. candidate, College of William and Mary, 1981-1983, with a concentration in Computer Science. The course requirements for this degree have been completed, but not the thesis: HILDA: The Flexible Design of a Data Base Machine Executive.

The author has had work experience at Newport News Shipbuilding and Dry Dock Co., Virginia working as a software analyst in the in-house computer aided ship design project. The author is now employed by Kentron Technical Center, Virginia where he is currently performing integrated computer-aided design and data base machine research at NASA Langley Research Center.