NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Design and Implementation of a Prototype Database for Part Information to Support the MK92 Fire Control System Maintenance Advisor Expert System

by

Susan G. Talley

March, 1994

Thesis Advisor: Magdi Kamel

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1994 March 30

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE DESIGN AND IMPLEMENTATION OF A PROTOTYPE DATABASE FOR PART INFORMATION TO SUPPORT THE MK92 FIRE CONTROL SYSTEM MAINTENANCE ADVISOR EXPERT SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S) Susan G. Talley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(maximum 200 words)*

The MK92 Fire Control System (FCS) is the heart of shipboard weapon systems found aboard U. S. Oliver Hazard Perry class FFGs. This system, based on 1970's technology, frequently requires extensive troubleshooting and supplemental shore-base support. A maintenance advisor expert system is being developed to assist shipboard technicians in correctly diagnosing system faults, providing expert advice concerning part replacement or further tests which should be made.

Additional information provided by the expert system includes documentation references, alternate location for a part, and part numbers. Storing such information in a relational database that communicates with the expert system would greatly improve its maintainability, modifiability, and accuracy.

This thesis addresses the design and implementation of a database to support the MK92 MOD 2 FCS Maintenance Advisor Expert System using Microsoft Access^{TM}. This database includes such functions as part and replacement information, database maintenance, and expert system support. Research revealed that the currently supported Windows^{TM} interprogram communications mechanism of Dynamic Data Exchange (DDE), as supported by the current versions of Access and Softsell Adept^{TM}, will not adequately support the database to expert system interface requirements. Suggestions for alternative interface solutions are provided in the thesis.

14. SUBJECT TERMS Database. Expert System. MK92 MOD 2 Fire Control System. Database Design and Implementation. Database Application.

15. NUMBER OF PAGES 199

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Design and Implementation of a Prototype Database for Part Information to Support the MK92 Fire Control System Maintenance Advisor Expert System

by

Susan G. Talley
Lieutenant Commander, United States Navy
B.S.M.E., University of Washington

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1994

Author: Susan G. Talley
Susan G. Talley

Approved by:
Magdi Kamel, Thesis Advisor

Martin J McCaffrey
Martin J. McCaffrey, Associate Advisor

David R. Whipple, Chairman
Department of Systems Management

ABSTRACT

The MK92 Fire Control System (FCS) is the heart of shipboard weapon systems found aboard U. S. Oliver Hazard Perry class FFGs. This system, based on 1970's technology, frequently requires extensive troubleshooting and supplemental shore-base support. A maintenance advisor expert system is being developed to assist shipboard technicians in correctly diagnosing system faults, providing expert advice concerning part replacement or further tests which should be made.

Additional information provided by the expert system includes documentation references, alternate location for a part, and part numbers. Storing such information in a relational database that communicates with the expert system would greatly improve its maintainability, modifiability, and accuracy.

This thesis addresses the design and implementation of a database to support the MK92 MOD 2 FCS Maintenance Advisor Expert System using Microsoft Access^{TM}. This database includes such functions as part and replacement information, database maintenance, and expert system support. Research revealed that the currently supported Windows interprogram communications mechanism of Dynamic Data Exchange (DDE), as supported by the current versions of Access and Softsell Adept^{TM}, will not adequately support the database to expert system interface requirements. Suggestions for alternative interface solutions are provided in the thesis.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The MK92 Fire Control System (FCS) is the heart of shipboard weapon systems found aboard U. S. Oliver Hazard Perry class FFGs and some U. S. Coast Guard and Australian vessels. Based on 1970's technology, the system requires a great deal of effort to correctly identify components causing system faults. Shipboard technicians spend valuable man hours and often replace good components, resulting in significant costs and/or extended system down time. In addition, shipboard technicians may not have the necessary expertise, and technical assist visits from shore-based technicians are often required to get the system back into operation. (Smith, 1993, p. 1)

A maintenance advisor expert system is being developed to enhance fault diagnosis and calibration of the MK92 MOD 2 FCS. Its purpose is to reduce the amount of time and money spent on system diagnostics and to reduce overall system down time. More significant, this expert system has the potential for reducing the dependence on shore-based systems support, which is not likely to be available during at-sea operations or war when it is critical for the MK92 FCS to be fully operational.

There are several potential uses of a database in conjunction with the expert system. A database is required to manage information concerning replacement part information, locations of identical parts within the system, and documentation references which are part of the expert system conclusions or recommendations. Since this information is used in more than one place within the expert system, storing it within a database will make it more easy to modify than if it were hard wired within the expert system itself. A second use of a database is to provide a supply support and inventory function to the technicians, to facilitate procurement of parts when they are required. Another possible benefit of using a database is using it to store and report usage/historical information, with respect to the use of the expert system, for future analysis and planning.

B. PURPOSE/OBJECTIVES

The purpose of this thesis is to design and implement a prototype database system which will work in conjunction with the MK92 FCS Maintenance Advisor Expert System (MAES). This database will primarily act as a repository for detailed information on replacement parts which will be available to the user, through the expert system. An easy to use interface will be provided to the users which allows them to maintain database information. In addition, the database will perform a supply support function for the technicians.

C. RESEARCH QUESTIONS

The following are the research questions that this thesis is addressing:

1. Does the use of off-the-shelf databases with expert systems add to the functionality of expert systems?

2. Does the use of databases with expert systems facilitate the maintenance of the currently developed expert systems?

3. What is the viability of the integration of databases and expert systems in the Windows environment?

D. SCOPE AND LIMITATIONS OF THESIS

This thesis defines, designs and implements primary functionality of a relational database system for use in conjunction with the MK92 Fire Control System MAES. Methods for integration of the database and expert system were explored and tested.

E. METHODOLOGY

This thesis uses the database life cycle and prototyping approach to develop the database application. This methodology of software development combines formal requirements and design techniques with one which utilizes a series of adaptive prototypes to test feasibility and to use for evolutionary requirements analysis.

The initial design and prototype focuses on the requirements generated by the interaction between the database and the expert system. When necessary, database complexity was limited to keep communications between the database and expert system as simple as possible.

F. THESIS ORGANIZATION

This thesis is organized in the following manner:

Chapter II describes the system requirements (Requirements Analysis Phase). Data and process requirements are discussed and represented using an Entity-Relationship (E-R) diagram and a leveled set of Data Flow Diagrams (DFD).

Chapter III covers how the requirements are converted into a database design (Design Phase). Data and process design are discussed, including the design of menus, forms, and reports.

Chapter IV discusses the implementation of the database (Implementation Phase). In this phase, data and processes are discussed in terms of implementation within a specific database software program. This chapter covers the construction of a Microsoft Access^{TM} application from a generic design.

Chapter V discusses the integration of the database and expert system. Dynamic data exchange (DDE) is the primary method covered, with other possible mechanisms briefly discussed.

Chapter VI presents lessons learned from the system development and future work requirements.

II. REQUIREMENTS ANALYSIS

Requirements analysis consists of determining two types of system requirements: data requirements and process requirements. Determining data requirements specifies what data needs to be stored in the system, while process requirements specify the processes which operate on the data in order to provide the required database functionality. In the requirements analysis phase of software design, initial design requirements were obtained by analyzing the functionality of the expert system and gathering the requirements and capabilities requested by the program manager, the Naval Surface Warfare Center (NSWC), Port Hueneme Detachment (PHD).

Prior to beginning the database design, its requirements had been discussed by the MK92 FCS MAES project team during several meetings, with the primary requirement determined to be presenting the user particular amplifying information concerning the expert system result nodes. NSWC PHD provided this information in the form of a list for the calibration portion of the expert system. Data requirements were developed primarily from this list and the knowledge representation diagrams.

In addition, discussion with NSWC personnel and the project advisors included interest in the possibility of storing data concerning the usage of the expert system to provide justification for the development of the system. Initial requirements were based on the possible information this sort of system would store, along with likely input and output processes, but the detailed design and the implementation of these requirements will not be covered in this thesis. Follow-on work may contain this functionality if it remains a system requirement.

Requirements for a supply function were based on personal experience with technicians and their needs for quick information for ordering parts. This system will be developed further based examination of the prototype by NSWC in follow-on work.

Some type of interface for maintenance is required of all databases. While much of the maintenance is performed by an administrator, it may also be practical for data to be maintained by the users, as well. If the users are required to maintain data, the design of a database maintenance interface is considerably more important. This interface must provide the capability to maintain the data items subject to change and at the same time present the inadvertent corruption of the database, to the greatest extent possible.

A. DATA REQUIREMENTS

Data requirements may be stated in the form of an entity-relationship (E-R) model, which consists of entities, attributes, and relationships. The entity-relationship (E-R) diagram is used to visually describe the entities and the relationships between them, and is provided in Appendix A, Section C. Data requirements are described below.

1. Entities and Attributes

a. General Description

The basic object in an E-R model is an **entity**, which is defined as "something important to the users in the context of the system that is to be built" (Kroenke, 1992, p. 98). Each entity has specific properties called **attributes**, which are characteristics that describe it. Each instance of an entity is a unique occurrence of that entity, which can be specified by a particular attribute or identifier.

b. Specific System Entities and Attributes

One of the primary entities for this system is the NODE, which is the result node within the expert system requiring information on part replacement from the database. Attributes of a NODE are a Node-number and a Module-reference, with Node-number being the identifier, or unique attribute.

The second primary entity is REPLACEMENT. This entity describes a particular part and its location within the system, which may be replaced by the technician troubleshooting faults within the MK92 FCS. Attributes of a REPLACEMENT are a Circuit-card-location-reference, Alternate-location, and Notes. The Circuit-card-location-reference, commonly called the UD number within the system technical manuals as well as the expert system, is the identifier for the Replacement entity.

The next primary entity is PART, which describes a particular electronic part in the system, another item of interest to the technician. Attributes of PART are the Part-number, Stock-number, Price, Part-allowance, Parts-on-hand, and Parts-on-Order. Part-number is the identifier for the PART entity.

The fourth entity is NODE-REPL, which provides a link between the NODE and the REPLACEMENT entities. The identifier is a composite attribute consisting of Part-number (from PART) and Circuit-card-location-reference (UD#) (from REPLACEMENT). There is also one attribute, circuit-reference, which is the document reference for a particular combination of NODE and REPLACEMENT identifiers.

The last entity is USAGE, which would store a record of the actual usage of the expert system to allow management to analyze its effectiveness and perform more accurate cost/benefit analysis. The initial definition of the attributes of USAGE are the Usage-number, Usage-date, Part-replaced, and Usage-notes. Usage-number is the identifier for the USAGE entity.

Other entities may be added to the system as development progresses. Appendix A, Sections A-B contains a listing of all system entities, their attributes, and their definitions.

2. Relationships

a. General Description

The association between two entities is called a **relationship**. A relationship can be characterized on several dimensions. The first dimension is the degree of the relationship. Most relationships involve only two entities and are called binary relationships.

The second dimension is cardinality, which specifies how many instances of each entity may be associated with the other entity in the relationship. There are three main types of binary relationships, 1:1 (one-to-one), 1:N (one-to-many), and N:M (many-to-many).

A third dimension is participation. The participation constraint tells whether the relationship between one entity and another is required (mandatory) or not required (optional). If every member of an entity set must be related to another entity, then the participation constraint is mandatory, or total. If members of an entity can exist without being related to another entity, then the participation constraint is optional, or partial. (Elmsari and Navathe, pp. 50-51)

b. Specific System Relationships

The relationship between the NODE entity and the REPLACEMENT entity is a N:M relationship, that is, a NODE may use more than one REPLACEMENT and likewise, a REPLACEMENT may be used by more than one NODE. This relationship contains the attribute circuit-reference, as the documentation reference is associated with neither Node nor Replacement, but the combination of the two. Since this relationship cannot be implemented directly, it has instead, been broken into two 1:N relationships, as discussed below.

The relationship between the NODE entity and the NODE-REPL entity is a 1:N relationship, that is, a NODE instance may be associated with more than one instance of NODE-REPL, but each instance of NODE-REPL may be associated with only one instance of NODE. Similarly, the relationship between REPLACEMENT entity and the NODE-REPL entity is a 1:N relationship. A REPLACEMENT instance may be associated with several instances of NODE-REPL, but each instance of NODE-REPL may be associated with only one instance of NODE. The participation constraints are such that there is a mandatory requirement for each NODE-REPL instance to be associated with a NODE and a REPLACEMENT instance. On the other hand, NODE and REPLACEMENT instances can exist without an associated NODE-REPL instance, therefore those constraints are optional.

The relationship between the PART entity and the REPLACEMENT entity is a 1:N relationship, since a PART instance may be associated with more than one REPLACEMENT instance, but each REPLACEMENT instance is associated with only one PART instance. The participation constraint is optional both ways, that is a REPLACEMENT instance may have a related PART instance and a PART instance may exist without being related to a REPLACEMENT instance.

B. PROCESS REQUIREMENTS

Process requirements are the second component of the overall system requirement. Processes can be modeled in terms of how the data flows through the system and the processing that is performed on the data. Data flow models are used to depict the processes and how they interact with one another, and how the data flows between processes. (Whitten, et. al., 1989, p.275)

Process modeling begins with factoring a system into subsystems and functions, using a top-down functional decomposition diagram. Logical data flow diagrams (DFDs) are then constructed, corresponding to each level in the decomposition diagram. Middle level DFDs show details about key subsystems, and the primitive level diagrams show explicit data flows and processes for a single functional piece of the system. (Whitten, et. al., 1989, pp. 284-321)

1. Process Decomposition

The decomposition of the process requirements for this system is shown in the decomposition diagrams provided in Appendix B. This system is broken into three main subsystems: the **Part Information** subsystem, the **Data Store Maintenance** subsystem, and the **System Usage** subsystem (Figure B-1).

These subsystems have been further broken down into subprocesses, which are activities corresponding to various system transactions, data maintenance functions, and reports. Further decomposition is shown in Figures B-2 through B-4. There is a fourth component of the database, which is associated with the interface between the database and the expert system. This component is discussed in detail in Chapter V.

2. System Data Flow Diagrams (DFDs)

Logical data flow diagrams (DFDs) are used to show detailed processing and associated data flows. Higher level DFDs correspond to the higher levels in the decomposition diagram, and give a more general illustration of what the subsystems do. The lower level DFDs show the detailed processing requirements of the primitive level functions. (Whitten, et. al., 1989, p. 289)

There are three main components of a data flow diagram: the external entities to the system, the logical data flows, and the logical processes. The external entities define the system boundaries, are the agents with which the system interacts, and include the end-users of the system. These end-users may be either sources of data or recipients of system information, or both. (Whitten, et. al., 1989, pp. 277-8)

In this system, the two major entities are 1) the **Technician** (or Expert System User), and 2) the **System Administrator** (System Admin). Two separate entities are used because, while the majority of the system processes are of possible use to both entities, there are separate processes which are designed for use by personnel maintaining the expert system and/or the database system (system administration). Changes to the expert system may require related changes to the database, where the user may be required to perform some simple database maintenance tasks if information in the data stores changes. A simplified interface is provided for the primary end-user (technician).

a. Context Diagram

The highest level DFD is the context diagram. This diagram "defines the scope and boundary for the system and project, (Whitten, et. al., 1989, p. 289)" and in this case is shown in Figure B-5. In this diagram, the only process shown is the root process. In addition, this diagram shows the external entities and the major data flows. Since details are not shown, the flows in this diagram represent a collection (or consolidation) of flows between the system and the entities.

System maintenance information flows from the System Administrator to the system, and parts information flows from the system to the System Administrator. User information (including maintenance information) flows from the Technician to the system and parts/supply information flows from the system to the Technician.

b. System Diagram

(1) General Description. The **system diagram** is an explosion of the context diagram into a more detailed picture of the system, and is the second level DFD. This diagram shows the major subsystems and how they interact with one another. This system diagram is shown in Figure B-6, and shows the three primary subsystems (1.0 Part Info, 2.0 Data Store Maintenance, 3.0 System Usage) which are the second level of the decomposition diagram. In addition to showing data flows between the systems and the external entities, this and lower level DFDs also show communications between the processes and the data stores. Multiple data stores and entities of the same name are used only to keep the diagram readable; symbols using the same name represent the same entity

or data store. In some cases, a single "data model" is used to represent all systems data stores for simplification. (Whitten, et. al., 1989, pp. 291-294)

In a few instances, the data flows shown on this diagram still represent composite flows, which are exploded further in lower level diagrams. In addition, most communication between the system and the user is two-way, yet may be oniy shown in one direction. To simplify DFDs, only the *net* data flow is shown; for example, in an inquiry, the result is shown but not the request.

(2) Subsystem Descriptions. The **Part Info Subsystem** receives part supply input and provides (local) part information to the Technician and provides (system) part information to the System Administrator.[1] This subsystem uses the Part and the Replacement data stores.

The **Data Store Maintenance Subsystem** receives (system) maintenance information from the System Administrator and (local) maintenance information from the Technician. This subsystem maintains the Replacement, Node, Part, and Node-Repl data stores.

The **System Usage Subsystem** receives usage information from and provides usage reports to the Technician. This data is stored in the Usage data store.

c. *Middle Level and Primitive Level DFDs*

(1) General Description. Each of the processes on the systems diagram is further exploded to show more of the subsystem details. In the case of this system there is only one level DFD between the system level DFD and the lowest or primitive level DFD. Each diagram will show progressively more detail concerning flows

[1] The term "local" is used with respect to the data flows to generally denote information going between processes and the Technician entity, and the term "system" is likewise used to denote information between processes and the System Admin entity.

until reaching the primitive level. At the primitive level, all data flows are shown and composite flows are broken down into their individual components. The letter *P* is added to the identification number for primitive processes to show that this process does not explode to another DFD.

(2) Parts Information Subsystem. The Parts Information Subsystem is exploded from the systems diagram into two levels (Figures B-7 and B-8). There are three processes in the first level, two of which are primitive level processes. These correspond to the decomposition diagram for this subsystem.

(a) **Browse Part Info** process (1.1P). This allows the technician to look at part information for a particular part. Part Supply Details are retrieved from the Part data store, and then provided to the Technician, by selecting a particular part-number.

(b) **Update Supply Status** process (1.2P). This allows the technician to change supply status information for a particular part. Changes are provided to the process, which then updates details in the Part data store.

(c) **Report Part Info** process (1.3). This allows the technician and system administrator to retrieve Part Reports and System Part Reports, respectively. This process uses details from the Part and Replacement data stores. This process is broken down into the following primitive processes (Figure 8):

(i) **Not On Hand Report** process (1.3.1P). This provides a report to the Technician, of parts which are not in stock, using the part-number element from the Part data store and the circuit-card-location-ref (UD#) element from the Replacement data store. This report provides all UD#s which are related to a particular part.

(ii) **Parts On Order Report** process (1.3.2P). This provides a Parts On Order Report to the Technician, using the part-number, and the parts-on-order element from the Part data store, and the circuit-card-location-ref (UD#) element from the Replacement data store.

(iii) **Parts Under Stock Report** process (1.3.3P). This provides a list of all parts which are under allowance level to the Technician, using details from the Part data store.

(iv) **System Parts List** process (1.3.4P). This provides a list of all parts in the system to the System Administrator, using the part-number element from the Part data store, and the circuit-card-location-ref (UD#) element from the Replacement data store. All UD#s corresponding to a particular part are listed.

(3) Data Store Maintenance Subsystem. The Data Store Maintenance Subsystem is exploded from the systems diagram into two levels (Figures 9-13). There are three processes in the first level (Figure 9), which all explode into lower level processes.

(a) **Node Maintenance** process (2.1). This allows the System Administrator and the Technician to perform maintenance on the Node data store, as well as related data stores (Replacement and Node-Repl), by selecting a particular Node#. This process explodes into the following primitive processes (Figure B-10):

(i) **Update Node/UD Info by Node** process (2.1.1P). This process allows the System Administrator and Technician to select a particular node and update all elements in the Node, Node-Repl, and Replacement data stores except node-number.

(ii) **Change Node#** process (2.1.2P). This process allows the System Administrator to change the Node number of a particular node. Node-number from both the Node and the Node-Repl data stores are changed.

(iii) **Add Node** process (2.1.3P). This process allows the System Administrator to add a new node to the system. Other related information, such as associated UD#s, pertaining to new nodes must be added by other processes.

(iv) **Delete Node** process (2.1.4P). This process allows the System Administrator to delete a node from the system, using node-number to delete the related instances in the Node and Node-Repl data stores.

(v) **Add UD to Node (By Noae)** process (2.1.5P). This process allows the System Administrator and Technician to add UD#s related to a particular node-number by adding Node-Kepl data store instances to :he system. This process checks to see if the UD# exists. If it does not, a message will request the user to add the UD# before creating an instance in the Node-Repl data store.

(vi) **Delete UD from Node (By Node)** process (2.1.6P). This process allows the System Administrator and Technician to delete UD#s related to a particular node-number by deleting instances in the Node-Repl data store from the system.

(b) **Replacement (UD) Maintenance** process (2.2). This process allows the System Administrator and Technician to perform maintenance on the Replacement data store, and related data stores (Part and Node-Repl), by selecting a particular UD#. This process explodes into the following primitive processes (Figure B-11):

(i) **Change Replacement Info By UD#** process (2.2.1P). This process allows the System Administrator and the Technician to change information related to a particular circuit-card-location-ref (UD#) in the Replacement and

Part data stores. If part-number is changed, the corresponding Part data store instance will also have its part-number attribute changed.

(ii) **Change UD#** process (2.2.2P). This process allows the System Administrator and the Technician to change circuit-card-location-ref (UD#) for a particular instance in the Replacement data store and related instances in the Node-Repl data stores.

(iii) **Add UD to Nodes (By UD)** process (2.2.3P). This process would allow the System Administrator to add Replacement data store and corresponding Node-Repl data store instances for a single UD#. This would be an alternative to adding a UD# to a number of nodes one node at a time. For Replacement data store instances not related to an existing Part-number, a new instance in the Part data store would be created.

(iv) **Delete UD from Nodes (By UD)** process (2.2.4P). This process is similar to that of 2.2.3P, but deletes rather than adds UD#s related to Nodes. This is accomplished by deleting Node-Repl instances.

(c) **Part Maintenance** process (2.3). This process allows the System Administrator and Technician to perform maintenance on the Part data store, and related data stores (Replacement and Node-Repl), by selecting a particular Part#. This process explodes into the following primitive level processes (Figure B-12):

(i) **Change Part Info** process (2.3.1P). This process allows the System Administrator and Technician to change information relating to a particular part in the Part data store, by part-number.

(ii) **Change Part#** process (2.3.2P). This process allows the System Administrator and Technician to change the part-number of a particular part, changing related part-numbers in both the Part and Replacement data stores.

(iii) **Delete Parts & UDs** process (2.3.3P). This process allows the System Administrator to delete Part instances from the Part data store. However it checks to see if corresponding instances with the same part-number exist in the Replacement data store. If they do exist, the user will be asked to delete related Replacement instances first.

(iv) **Add Parts** process (2.3.4P). This process allows the System Administrator and Technician to add instances to the Part data store.

(4) System Usage Subsystem. The System Usage Subsystem is exploded from the systems diagram into two levels (Figures B-13 and B-14). There are two processes in the first level, one of which is a primitive level process.

(a) **Enter Usage** process (3.1P). This process allows the Technician to enter system usage data. The identifier attribute will be a usage-number, which will document consecutive entries. Other items of interest will be usage-date, part-replaced (Part#), and notes (UD#, etc.). These attributes are stored in the Usage data store.

(b) **Report Usage Data** process (3.2). This process explodes into the following primitive level processes (Figure B-14):

(i) **Report Usage** process (3.2.1P). This process will retrieve usage details from the Usage data store in a formatted report.

(ii) **Annual Report** process (3.2.2P). This process will retrieve usage details for the current year from the Usage data store, in a formatted report, and (as required) archive data to clear the Usage data store.

C. OUTPUT OF REQUIREMENTS ANALYSIS PHASE

The statement of requirements is the output of the requirements phase. This statement includes the description of the data and process requirements, the entity-relationship diagram, and a leveled set of data flow diagrams. The next chapter discusses the next stage of database development, the design phase.

III. DATABASE SYSTEM DESIGN

The design phase consists of two parts, data design and process design. In data design, also known as logical database design, data requirements specified in the requirements phase are converted into a relational design which may be implemented later in any specific database software. In process design, also known as application design, update, display, and control mechanisms such as forms, menus, and reports, for the application are developed.

A. DATA DESIGN

1. General Procedures

During data design, entities, and the relationships between entities, are described in terms of relational database designs using the relational model. This is accomplished by first defining a relation for each entity. These relations have the same name as the entity and the attributes of the relation are the properties of the entity. The key attribute is the same as the identifier (or unique) property of an entity. (Kroenke, 1992, p. 206)

After initial data design, relations are checked to ensure they are free from modification anomalies. If not, normalization is used to eliminate these anomalies which might result in an improperly designed database. It should be noted, however, that normalization often adds additional relations by breaking entities into smaller units. The best database design is a combination of minimizing modification anomalies while at the same time preventing the design from becoming too contrived or complex. (Kroenke, 1992, pp. 207-208)

Once a relation is constructed for each entity, with all of the entity's properties, the different kinds of relationships in the E-R model are also represented. The representation of one-to-one (1:1) and one-to-many (1:N) relationships is straightforward.

Each entity is represented as a relation and then the key attribute of one of the relations is also stored in the other. In the case of the 1:N relationship, the key attribute of the parent (on the "one" side) is stored in the relation representing the child (on the "many" side). The key attribute stored in the relation, whether 1:1 or 1:N, is called the foreign key since it technically does not belong to that relation. (Kroenke, 1992, pp. 211-214)

To represent many-to-many relationships a new relation is created, called an **intersection relation**. This intersection relation represents the relationship itself, and the key is the combination of the keys of both of its parent relations. (Kroenke, 1992, p.215-217)

2. Specific Database Systems Relations

The entity-relationship diagram shown in Figure A-1, Appendix A is converted into a relational model using the principles described above. This model is shown in Figure 1, below. This system is primarily designed with the expert system interface in mind, so all relations and attributes have been designed to simplify the resulting output. The transformation of the entities in this system are described in detail below.

The NODE entity is represented as the relation **NODE**, with the attributes of Node# and Module Ref. The key attribute is Node#.

The REPLACEMENT entity is represented as the relation **REPLACEMENT**, with the attributes UD#, Alt Loc, and Notes. UD# is the key attribute. This relation is the child of the **PART** relation, therefore the attribute Part# is also included in the **REPLACEMENT** relation as a foreign key.

The PART entity is represented as the relation **PART**, with the attributes Part#, NSN, Price, Allowance, Parts On Hand, and Parts on Order. Part# is the key attribute of this relation.

The NODE-REPL entity is represented as the relation **NODE-REPL**, with the attributes Node#, UD#, and Circuit Ref. The combination of Node# and UD# is the key attribute of this relation. This relation is an intersection relation between the two parent relations, **NODE** and **REPLACEMENT**.

For this thesis, the USAGE entity is not represented in the relational model, as discussed in Chapter II.

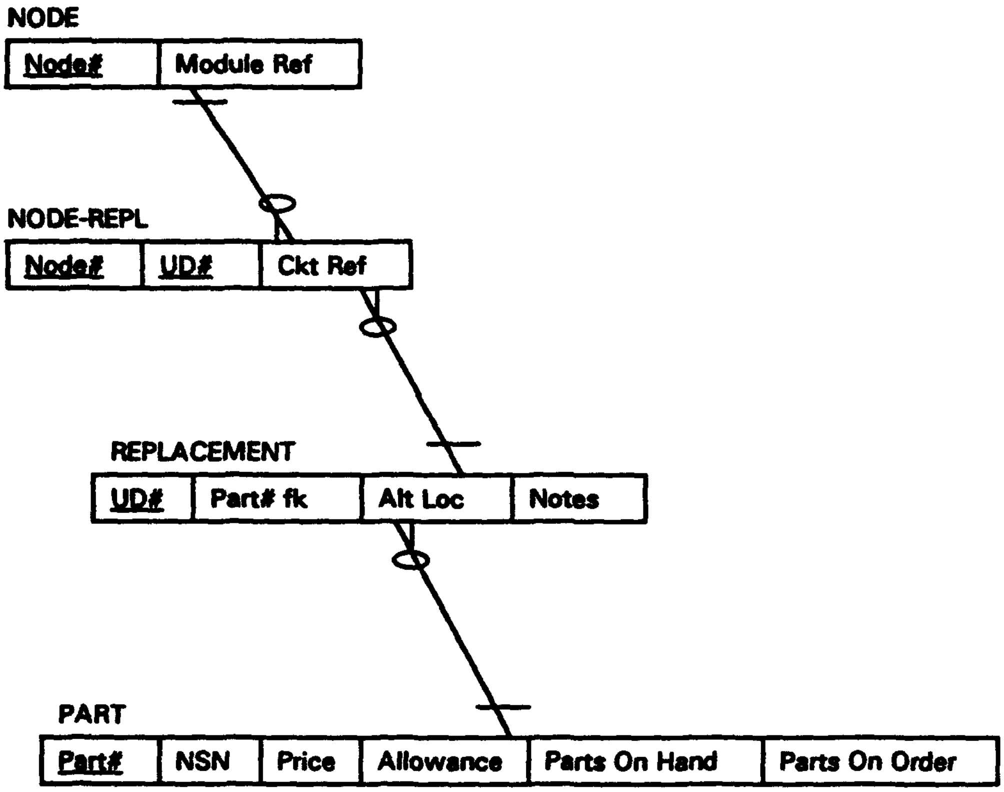


Figure 1- Relation Diagram

B. PROCESS DESIGN

Process design involves the design of menus, screens, forms, reports and the logic associated with these items. In most databases, the primary concern is the output requirements.

1. Menu Design

a. General Design Strategy

There are a number of common strategies for user interface design. The most popular is the use of menu selections where various options are presented to the end-user. The user then can easily selects an appropriate action from those presented on the menu. In some cases, the menuing technique is driven by the database software being used, because the mechanisms for one or the other type are more easily implemented. One common technique available in current database software is the use of pull-down menus, where the user highlights the chosen action using arrow keys, a mouse, or initial letters of the action. If submenus exist, they descend from the pull-down menu choice, presenting more choices to the user. This allows the user to traverse through a hierarchical structure, selecting one of a collection of functions. (Whitten, et. al., 1989, p.585)

Many Windows-based database software programs take an object-oriented design approach, where command buttons invoke macros which perform certain tasks. With event-driven programs, instead of presenting the user a strict hierarchical structure, it may be possible to provide more convenient and natural ways for users to do things. (Jones, 1994, pp. 31-32)

b. Specific Design

This system menu is primarily hierarchical, but some functions will be combined where it makes sense, and will not be presented exactly as found in the decomposition diagrams. The menu hierarchy is provided in Appendix C, Section A. All functions and menu selections are provided via command buttons, which allow the user to either select a button using the Windows pointing device (mouse or trackball) or by typing in the highlighted letter (underlined) on the "button". In addition to the selections shown in the menu hierarchy listing, each menu level contains one or more buttons which return the user to the previous menu and/or the main menu (as appropriate). The menus are discussed in detail below.

There are two subsystems, one for administrators and one for users (technicians). The User menu is separated from the **Administrator** menu, even though many of the functions are the same, so that Database Administrators can have access to a more complete set of data maintenance functions than Users would require. The first menu screen presented to Administrators is the Opening screen, which is shown in Figure C-1, Appendix C. Database administrators (DBAs) have the option of accessing either the User or Administrator version of the system.

(1) User Menu. The main User Menu screen allows the user to select from four command button choices, three which invoke submenus (the Part Information submenu, the Usage History submenu, and the DB Maintenance submenu), and one to exit the system. (Figure C-2) This is the first menu screen presented to users accessing the system.

(a) Part Information Submenu. The Part Information submenu presents four choices to the user: two which invoke part information functions (Browse Part Supply Info and Update Part Supply Status), one which invokes the Part Reports Submenu, and one which returns the user to the main (User) menu. (Figure C-3)

(i) Browse Part Supply Information. This function allows the user to browse supply information concerning a specific part selected from a list of parts. This provides a basic display mechanism for the PART entity. Screens used for this display are discussed in Section 2, below.

(ii) Update Part Supply Information. This function provides the capability for the user to update the supply information for a specific part. This provides the update mechanism for the PART entity. Forms used for this function are discussed in Section 2, below.

(iii) Part Reports Submenu. The Part Reports submenu presents five choices to the user: three which generate reports for the user, one which returns the user to the Previous menu (Part Information), and one which returns the user to the main menu (User). (Figure C-4) Reports are discussed in detail in Section 3, below.

{A} Parts Not On Hand Report. This option generates a report on parts which are not on hand.

{B} Parts On Order Report. This option generates a report on parts which are on order.

{C} Parts Under Stock Report. This option generates a report on parts which are under allowance level.

{D} Previous Menu. This option returns the user to the previous menu screen (Parts Information).

{E} Return to Main User Menu. This option returns the user to the top level user menu.

(iv) Return to Main User Menu. When selected, this option returns the user to the top level user (Main - User) menu.

(b) Usage Submenu. The Usage submenu presents four choices to the user· one to enter usage information, two for generating reports, and one to return to the main (User) menu. (Figure C-5) As discussed in Chapter II, this submenu is not implemented in this thesis.

(c) DB Maintenance Submenu. This submenu presents four choices to the user: one to update circuit card information, one to add UD#s to/delete UD#s from Nodes, one to update Part Information, and one to return to the main (User) menu. (Figure C-6)

(i) Update Circuit Card Information. This option allows the user to update the NODE, NODE-REPL, and REPLACEMENT entities. The function provides a submenu to the user, allowing selection between updating by Node# or by UD#. This menu screen is shown in Figure C-7.

{A} Update By Node#. This option allows the user to update the NODE, NODE-REPL and/or REPLACEMENT entities for the information related to a particular Node#. Screens used for these updates are discussed in Section 2, below.

{B} Update By UD#. This option allows the user to update the REPLACEMENT and NODE-REPL entities for the information related to a particular UD#. Screens used for these updates are discussed in Section 2, below.

(ii) Add UDs to/Delete Nodes from Nodes. This option allows the user to update the NODE and NODE-REPL entities. This function is not implemented.

(iii) Update Part Information. This option allows the user to update the PART and REPLACEMENT entities. This function is not implemented.

(iv) Return to Main User Menu. When selected, this function returns the user to the top level user menu.

(d) Exit. The Exit function closes the database application after saving any changes.

(2) Administrator (Admin) Menu. The main Admin menu allows the same three submenu choices as the User menu (Part Information. Usage History, and DB Maintenance), plus Exit. This menu will be used by the DBA, and is shown in Figure C-8.

(a) Part Information Submenu. This menu presents the same choices as in the User Subsystem, which is discussed above, and is shown in Figure C-9.

(i) Browse Part Supply Information. This is the same function as in the User Subsystem, and it uses the same screens.

(ii) Update Part Supply Information. This is the same function as in the User Subsystem, and it uses the same screens.

(iii) Part Reports Submenu. The Part Reports submenu in this subsystem presents three choices to the DBA: one which generates a report for the DBA, one which returns the DBA to the Previous menu (Part Information - Admin), and one which returns the DBA to the main menu (Admin). The report is discussed in detail in Section 3, below. (Figure C-10)

{A} System Parts List. This option generates a listing of all parts in the expert system, for use by the DBA.

{B} Previous Menu. This option returns the user to the previous menu screen (Parts Information - Admin).

{C} Return to Main Admin Menu. When selected, this option returns the DBA to the top level Admin menu.

(iv) Return to Main Admin Menu. When selected, this option returns the DBA to the top level Admin menu.

(b) Usage Submenu. This submenu is not covered in this thesis.

(c) DB Maintenance Submenu. This submenu provides the same maintenance functions as in the User Subsystem, plus an additional function to maintain the NODE entity (and related entities) within the system. (Figure C-11)

(i) Update Circuit Card Information. This option provides enhanced capabilities to the DBA for maintenance of the NODE, NODE-REPL, and REPLACEMENT entities, in addition to those provided to the user. The function provides a submenu to the user, allowing an update by either Node# or by UD#. This menu screen is shown in Figure C-12.

{A} Update by Node#. This option provides the same basic capabilities as that of the user menu plus it allows the DBA to change Node# within the NODE and related NODE-REPL entities. Screens used by this function are discussed in Section 2, below.

{B} Update by UD#. This option provides the same capabilities as that on the User menu to update the NODE, NODE-REPL, and NODE-REPL entities.

(ii) Add UDs to/Delete UDs from Nodes. This option is the same as on the User menu.

(iii) Update Part Information. This option is the same as on the User menu.

(iv) Add Nodes to/ Delete Nodes from System. This option allows the DBA to add instances to the NODE entity or delete instances frcm the NODE entity. This function would be used in the case of a adding , deleting, or modifying nodes in the expert system.

(v) Return to Main Admin Menu. When selected, this option returns the DBA to the top level Admin menu.

(d) Exit. This option returns the DBA to the Opening menu, instead of exiting the program.

2. General Form/Screen Use and Design

Forms are used for both data entry and display of information. Since this database does not model any existing paper forms, as is the case with many database systems, forms were designed from scratch with simplicity and consistency in mind. Some forms were designed with a form generator while others were put together using other design tools. The form generator and form design tools will be discussed in more detail in Chapter IV.

Forms can be designed based on either entities or a combination of entities. There are two types of forms in this database application, forms based on a single table, and forms based on more than one table. Multi-table forms can be used to display entities with 1:1 relationships, or entities with 1:N relationships. Some forms are based on a table and some forms are based on the results of a query. Examples of these forms are discussed below.

As with menus, command buttons can be used on forms to execute additional functions or tasks. Procedures or tasks executed in this way allow one or more tasks to be grouped together in a single cohesive presentation, with different options available to the user at his or her selection. Other tasks included in forms are updates to related information, viewing related information, and cancellation of changes. The application forms will be discussed below, along with functions and procedures. Specific form design is discussed in Section 4, below.

3. Process Logic

Process logic describes the logic of the different modules of the system. There are two types of process modules used in the system: procedures and functions. Both procedures and functions are designed with software reuse in mind, that is, in many cases functions and procedures contain actions which may be used by more than one process.

Examples of actions which may by either functions or procedures, or both, include: present a form to a user, read input, search for a particular instance or set of instances of an entity or entities, display instances to the user, cancel actions, update entities, delete entities, and others.

Process logic is discussed with respect to specific system forms and reports, and is presented in detail in Appendix D, Section A.

4. Specific System Forms and Associated Logic

a. User Forms.

(1) Browse Part Supply Information. When invoked from the Part Information submenu, this function involves the use of a set of two forms.

The input form (Browse Part) is shown in Figure D-1, Appendix D. This form is based on a query of the PART entity, which is invoked by the user selecting a Part# from a scroll list of all the system part numbers. Information about that part is displayed by selecting or "pushing" the **Locate** command button. A **Cancel** function button is also available. When a Part# and the **Locate** button are selected, the associated procedure (see Appendix D, section A, U1.1L) is executed. The **Cancel** button executes procedure U1.1C, returning the user to the previous screen, Part Information Menu.

The output form (Part Supply Info Browse) is based on the same query and presents the results of the procedure (Figure D-2). This form has one command button, **Return**, which takes the user back to the previous (input) screen using procedure U1.1L-R.

(2) Update Part Supply Status. When invoked from the Part Information submenu, this procedure (U1.2) involves the use of a set of two primary forms and three secondary forms.

The input form (Update Part) is shown in Figure D-3. This form requires input from the user in the form of Part#. By selecting the **Locate** command button, the user invokes a process (Procedure U1.2L) which locates a particular instance of the PART entity and displays it to the user. If there is no matching part, a message is provided to the user. A **Cancel** function button is also available, which executes procedure U1.2C, returning the user to the previous menu form (Part Information Menu).

The output form (Part Supply Info) is based on the PART entity and presents the results of the procedure (Figure D-4). This form has five command buttons: **Return**, which returns the user back to the Part Information menu using procedure U1.2L-R; **More**, which saves the existing information, including any updates, and takes the user back to the previous (input) screen using procedure U1.2L-M; **Issue**, which invokes the process which displays the Issue Parts form (U1.2L-I); **Order**, which invokes the process which displays the Parts Ordered form (U1.2L-O); and **Receive**, which invokes the process which displays the Parts Received form (U1.2L-R).

One secondary form associated with this procedure, Issue Parts, is shown in Figure D-5. This form requires the user to input the amount of parts issued. The user may either **Cancel** this action (U1.2L-IC), or **Update** the number of parts on hand (U1.2L-IU), and then return to the previous form (Part Supply Info).

The next secondary form, Parts Ordered, is shown in Figure D-6. This form requires the user to input the amount of parts ordered. The user may either **Cancel** this action (U1.2L-OC), or **Update** the number of parts on order (U1.2L-OU), and return to the previous form (Part Supply Info).

The third secondary form, Parts Received, is shown in Figure D-7. This form requires the user to input the amount of parts received. The user may either **Cancel** this action (U1.2L-RC), or **Update** the number of parts on order and parts on hand (U1.2L-RU), and return to the previous form (Part Supply Info).

(3) Update Circuit Card Information (By Node#). When invoked from the Part Information submenu using the Select Change submenu, this function involves the use of a set of two forms and two subforms.

The input form (Input Node#) is shown in Figure D-8. This form requires the users to input a Node#. By selecting the **Locate** command button, the user invokes a process (Procedure U2.1.1L) which locates a particular NODE instance, if it exists, and displays that NODE and its related NODE-REPL and REPLACEMENT information. If there is no matching NODE instance, a message is provided to the user. A **Cancel** function button is also available, which executes procedure U2.1.1C.

The output form (Update Node-Replacement - User) is based on the NODE entity and is displayed in Figure D-9. A subform appears within this form, which displays the related NODE-REPLACEMENT entity information. Also a sub(sub) form within the subform displays the REPLACEMENT entity information. In this manner, it is possible to display a NODE and all REPLACEMENT instances associated with that Node#, by using instances in the NODE-REPL entity to link these two other entities. This form has three command buttons: **Clear,** which clears any changes, before they are committed using procedure U2.1.1L-C; **Exit,** which returns the user back to Select Change menu using procedure U2.1.1L-E; **More,** which saves the existing information, including any updates, and takes the user back to the previous (input) screen using procedure U2.1.1L-M. The subform has two command buttons: **Fwd,** which displays the next NODE-REPL instance associated with that Node# using procedure U2.1.1L-F; and **Back** (Procedure U2.1.1L-B), which displays the previous instance of NODE-REPL associated with that Node#.

(4) Update Circuit Card Information (By UD#). When invoked from the Part Information submenu using the Select Change submenu, this function involves the use of a set of two forms. This function also permits another function to be invoked, Update UD# , using a third form.

The input form (Input UD#) is shown in Figure D-10. This form requires the user to input a UD#. By selecting the **Locate** command button, the user invokes a process (Procedure U2.2.1L) which locates a matching UD# in the system, if it exists, and displays that particular Replacement instance to the user. If there is no matching Replacement instance, a message is provided to the user. A **Cancel** function button is also available, which executes procedure U2.2.1C.

The output form (Update UD# - Replacement) is shown in Figure D-11, and is based on the REPLACEMENT entity. This form has four command buttons: **Update UD#,** which invokes the process U2.2.1L-U, and is discussed in more detail below; **Clear,** which clears any changes made, before any updates are made, using procedure U2.2.1L-C; **Exit,** which returns the user back to the Select Change menu using procedure U2.2.1L-E; **More**, which saves the existing information, including any updates, and takes the user back to the previous (input) screen using procedure U2.2.1L-M.

The Update UD# process invokes the third form, Change UD# (Figure D-12). The subform has two command buttons: **Cancel**, which cancels the Update UD# process using process U2.2.1L-UC; and **Change UD#**, which invokes a process (U2.2.1L-UD) which accepts a new UD# and presents a message giving the user an opportunity to confirm the change by selecting **Yes** or **No**. If the user selects "No" the process is cancelled and he/she is returned to the Change UD# form. If the user selects "Yes", this activates the function "Update Related UD" (Appendix D, Section A-3). This function first finds all of the instances of the REPLACEMENT entity with the old UD# and updates them with the new UD#, then it finds all related (with the same UD# as the old UD#) instances of the NODE-REPL entity and updates them with the new UD#. Upon completion of this function, the user is returned to the Update UD# - Replacement form.

b. Administrator (Admin) Forms.

(1) Common User and Admin Forms. Most of the Admin Forms are the same as the User forms and perform the same function. The primary difference in the processes invoking the forms or processes attached to command buttons is the menu screens from which the user starts or to which the user returns to after the completion of the process(es). Process logic is contained in Appendix D, Section A-2.

(2) Unique Admin Form: Update Circuit Information (By Node#) When invoked from the Part Information submenu using the Select Change submenu, this function involves the use of a set of three forms and two subforms.

The primary input form (Input Node#) is shown in Figure D-8, and is the same as the User form of the same name. This form requires the user to input a Node#. By selecting the **Locate** command button, the user invokes a process (Procedure A2.1.1L) which locates a particular Node# in the NODE entity, if it exists, and displays that NODE and its related NODE-REPL and REPLACEMENT instances information. If there is no matching NODE instance, a message is provided to the user. A **Cancel** function button is also available, which executes procedure A2.1.1C.

The output form (Update Node-Replacement - Admin) is based on the NODE entity and is displayed in Figure D-13. A subform within this form, displays related NODE-REPL entity information. A sub(sub) form within the subform is used to display related REPLACEMENT entity information. In this manner, it is possible to display a NODE and all REPLACEMENT instances associated with that Node#, by using instances in the NODE-REPL entity to link these two entities. This form has four command buttons: **Clear**, which clears any changes made, before they are committed, using procedure A2.1.1L-C; **Exit**, which returns the user back to Select Change menu using procedure A2.1.1L-E; **More**, which saves the existing information, including any updates, and takes the user back to the previous (input) screen using procedure A2.1.1L-M; and **Update Node#**, which invokes procedure A2.1.1L-N, that changes the current Node# to another one as specified by the user. This last process is discussed in more detail below. The subform has two command buttons: **Fwd**, which displays the next NODE-REPL instance associated with that Node# using procedure A2.1.1L-F; and **Back** (Procedure A2.1.1L-B), which displays the previous instance of NODE-REPL associated with that Node#.

The Update Node# process (A2.1.1L-N) invokes the third form, Change Node# (Figure D-14). This form has two command buttons: **Cancel**, which cancels the Update Node# process using process A2.2.1L-NC; and **Change Node#**, which invokes a process (A2.2.1L-NA) which accepts a new Node# and presents a message giving the user an opportunity to confirm the change by selecting **Yes** or **No**. If the user selects "No" the process is cancelled and the user is returned to the Change Node# form. If the user selects "Yes", this activates the function "Update Related Node" (Appendix D, Section A-3). First, this function finds all of the instances of the NODE entity with the old Node# and updates them with the new Node#, then it finds all related (with the same Node# as the old Node#) instances of the NODE-REPL entity and updates them with the new Node#. Upon completion of this process, the user is returned to the Update Node-Replacement -Admin form.

5. Report Design and System Reports

There are currently four reports in this system, three for the User Subsystem, and one for the Administrator (Admin) Subsystem. Examples of these reports are provided in Appendix D, Section C, and are discussed below. Process logic for the processes involved in these reports is detailed in Appendix D, Section A.

a. User Reports

(1) Parts Not On Hand Report. This report is invoked from the Report Parts Info Menu - User, using the Report Parts Not On Hand procedure (U1.3.1). A sample report is shown in Figure D-15. This report is based on the results of a search for all instances of the PART entity, where the number of Parts On Hand equals zero. A list of the Part#s for instances matching the search criteria is then used to obtain a sub-listing of all UD#s related to the resultant Part#s from the REPLACEMENT entitiy. Part# and all associated UD#s, for parts which are not in stock, are reported to the user.

(2) Parts On Order Report. This report is invoked from the Report Part Info Menu - User, using the Report Parts On Order procedure (U1.3.2), and is shown in Figure D-16. This report is based on the results of a search for all instances of the PART entity, such that the number of Parts On Order is greater than zero. A list of the Part#s for instances matching the search criteria is then used to obtain a sub-listing of all UD#s related to the resultant Part#s from the REPLACEMENT entity. Part#, Number of Parts On Order, and all associated UD#s, for parts which are on order, are reported to the user.

(3) Parts Under Stock Report. This report is invoked from the Report Parts Info Menu - User, using the Report Parts Under Stock procedure (U1.3.3). A sample of the report is shown in Figure D-17. This report is based on the results of a search for all instances of the PART entity, such that the number of Parts On Hand is less than the parts Allowance. A list of the Part#s for instances matching the search criteria is displayed along with Number Parts On Hand, Number Parts On Order, and the Part Allowance.

b. Admin Reports

There is currently one Admin report, which is invoked from the Report Parts Info Menu - Admin, using the System Parts List procedure (A1.3.1). Part of this listing is shown in Figure D-18. This report is based on the results of a search for all instances of the PART entity which had a Part#. A list of the Part#s for instances matching the search criteria is then used to obtain a sub-listing of all UD#s related to the resultant Part#s from the REPLACEMENT entity . Part# and all associated UD#s for parts in the system are reported to the administrator.

C. OUTPUT OF DESIGN PHASE

The output of the design phase is a document that describes the structure of the database. This structure of the database includes a description of the relations, their attributes, and the relationships between relations, and the related processes. The description of the process design includes the menus, forms, screens, and process logic. The next chapter discusses the next stage of database development, the implementation phase.

IV. DATABASE SYSTEM IMPLEMENTATION

Similar to the requirements and design phases, the implementation phase consists of two parts, data implementation and process implementation. In data implementation the relational model is converted into the database structure of a specific DBMS. Process implementation involves the construction of forms, reports, menus, procedures, and functions developed during the design phase. During this phase a specific DBMS software is used, and implementation of the system becomes dependent on the functionality and design of this software and its language, features, limitations, and structures. This chapter describes system implementation by discussing software selection, the DBMS used, data implementation, and process implementation.

A. SOFTWARE SELECTION

1. Software Requirements

This thesis is not only interested in developing a standalone database application, but also a database application that integrates with an expert system application. Therefore, a primary requirement for DBMS software to be used for this thesis is that it be compatible with Softsell Adept^{TM} expert system shell used to develop the MK92 FCS Maintenance Advisor being developed at the school. Since Adept is a Microsoft Windows-based program, the selected DBMS must have the ability to pass data or information using Windows mechanisms.

Another consideration was whether the software supported the development of a "run-time" version for the user, so that the full database environment is not required to be included with the operational version. This requirement saves both money and storage space, which is generally scarce on laptop computers.

A third consideration was, of course, the functionality and ease of use of both the development environment and the user environment.

2. Available Software Programs: Advantages and Disadvantages

With above selection criteria in mind, the initial software selection was made early in 1993. At that time the major Windows database programs had been just introduced and experience with their use was rather limited.

a. Microsoft Access^{TM} DBMS

Microsoft Access DBMS had the advantage that it is a Microsoft product. Since Windows is also a Microsoft product, conceivably Access would have a better implementation of one of the primary Windows communications mechanisms of interest, Dynamic Data Exchange. The application development environment was considered to be the easiest, but was less capable in developing and running queries than Paradox. (Coffee, 1993, pp. 270-297) One feature of interest was the availability of an application development kit, which could produce run-time executable programs for distribution to the end-users.

b. Borland Paradox^{TM} DBMS for Windows

The Paradox DBMS was also well reviewed, and in some respects considered to be better than Access. Paradox has a report generator which was considered to be the best of these three databases, but required the use of its programming language for most tasks unlike the Access' macro facility which simplifies development (Coffee, 1993, pp. 270-297). Paradox has a programming language which is considered to be a superior, object-oriented, C-like language (Coffee, 1993, p. 285). A run-time engine or application development version was not advertised, but likely to be announced in the future since one is available for the Paradox version for DOS. The major drawback with Paradox was that communications between two different programs is difficult enough, let alone if the database and expert system interface were to involve three vendors (Windows, Paradox, and Adept).

c. *Microsoft FoxPro™ DBMS for Windows*

FoxPro DBMS is considered to be a superior development environment, but challenging for a programmer not already experienced in programming in an XBase programming language, such as dBASE uses (Campbell and Hudnall, 1993, p. 25). With this in mind, there seemed to be too much of a learning curve to overcome if FoxPro were to be used. DDE capabilities were also unknown, although Microsoft was a major player in converting FoxPro, from being initially a program for the Macintosh, to versions for Windows and DOS. It is therefore likely that FoxPro's DDE environment is as capable as that of Access.

3. DBMS Selected

Base on an analysis of application requirements and the characteristics of initial DBMS selection as well as literature reviews, Access was selected as the development environment for this thesis. FoxPro was initially considered after being recommended by NSWC sponsors, but was eliminated from further consideration after it was discovered, as mentioned above, that it has a steep learning curve.

B. MICROSOFT ACCESS™ DBMS OVERVIEW

Microsoft Access DBMS provides a comprehensive development environment for developing database applications. It consists of six main components: tables, forms, queries, reports, macros, and modules. These components are invoked from the Access main database window, shown in Figure 2, to develop all database objects (tables, forms, reports, etc.) for an application. In addition, an extensive help system is available, which includes information based on a search, examples, and "Cue Cards." Cue cards are an on-line form of tutorial which steps the user through the creation/design of a particular object. The following sections discuss each component of Access in some detail..

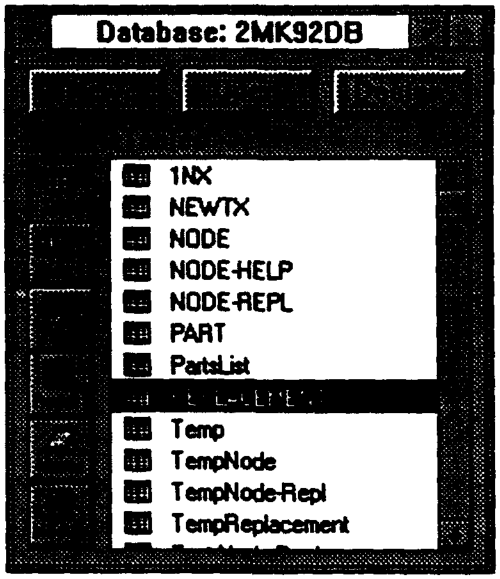


Figure 2 - Database Window

1. Tables

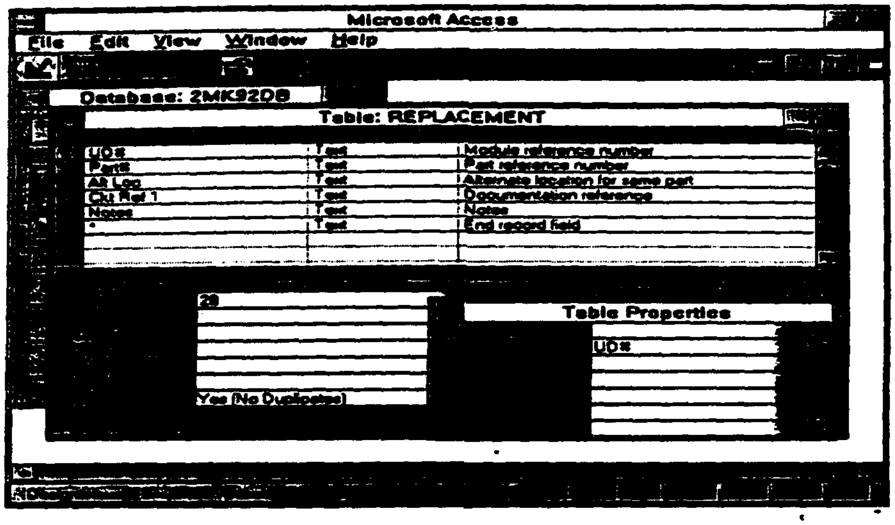
a. Table Creation and Definition

Tables are created by selecting the "Table" and "New" buttons in the database window. This brings up the Table environment or "Design View." In this view the fields are named, and the type and length of data that will be stored in each field as well as any rules which govern data entry into the table are specified. Table properties such as a description, a key, and indexes may also be entered in this view or at a later time. These items will be discussed in more detail below. The design environment for a table is shown below, in Figure 3. (Jones, 1994, pp. 39-40)

There are eight types of fields including text, memo, number, data/time, currency, counter, yes/no, and OLE object. Memo fields allow a great deal of flexibility in the amount of data it can hold, and are useful for storage of a large amount of text since up to 32,000 characters can be stored in a memo field for each record. Descriptions for each field can also be added. (Jones, pp. 40-42)

Once the field type has been defined, there are a number of field properties applicable to each type of field. For example, the properties for a text field are the field size, the caption, the default value (if any), the validation rule (if any), validation text (if any), and whether the field will be indexed. There is a default size for text fields, which may be ch^ged in the properties box. There are other properties associated with the other data types. "Validation rules" are a feature which lets the developer control how data is accepted into the fields of a table, so program code is not needed for validating data on data-entry forms. Other properties may be added or changed, as appropriate, including field sizes. (Jones, 1994, pp. 46-49)

In most tables, there will be a key field or fields, and this relates directly to the key attribute(s) in the relation design. The key field (or combination of fields) is assigned by highlighting the particular field or fields and then selecting the "Key" command button on the toolbar menu, a feature which can be seen in Figure 3. An index can be added to most fields of a table (all fields except fields of data type memo, yes/no, and OLE object). Indexes are used to speed the performance of searches on a given field. (Jones, 1994, p. 60)



Database: 2MK92DB

Table: REPLACEMENT

UD#	Text	Module reference number
Part#	Text	Part reference number
Alt Loc	Text	Alternate location for same part
Ckt Ref 1	Text	Documentation reference
Notes	Text	Notes
*	Text	End record field

20

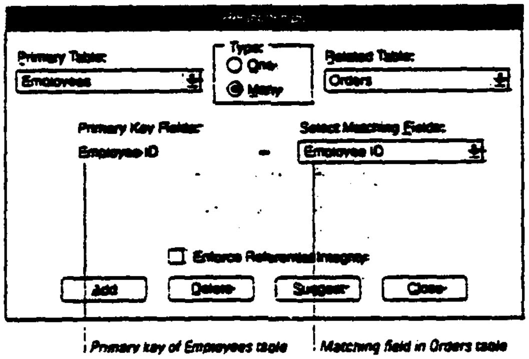
Yes (No Duplicates)

UD#

Figure 3 - Table Design Environment Example

b. Establishing Relationships Between Tables

Once the tables in a database havc oeen created, relationships between these tables can be established. This is performed by selecting "Edit/Relationships" from the menu and then filling in the options in the Relationship dialog box, which is shown in Figure 4.



Primary Table: Employees

Type: ○ One ● Many

Related Table: Orders

Primary Key Fields: Employee ID = Select Matching Fields: Employee ID

□ Enforce Referential Integrity

Primary key of Employees table

Matching field in Orders table

Figure 4 - The Relationships Dialog Box

Defining relationships between tables allows for the automatic definition of the related field during the design of queries, form/subform combinations, and report/subreport combinations. The effect of these defined relationships will be discussed later within the context of the different types of objects in later sections of this chapter. This will also provide enforcement of referential integrity between the data in related tables, if desired. If the referential integrity option is turned on, then Access will maintain referential integrity during operations which involve the editing and deleting of records. This means that records cannot be added to a related table if there is no corresponding entry for the matching field in the primary table, and similarly, a record from the primary table cannot be deleted if that will leave related records in the other table as "orphans." (Jones, 1994, p. 62) This feature may be desirable during the use of a database, especially if specific program code is not used to perform the same functions.

To define a relationship, the primary table is selected first. Then the type of relationship is defined as either "One," which defines a one-to-one relationship, or "Many," which defines a one-to-many relationship. Next, the related table is selected in that list box. The matching field used to link the tables is then selected. Finally, if the "Enforce Referential Integrity" feature is desired, that check box is selected.

c. Data Entry

Data entry may be performed by using a form or directly in the Table "Datasheet" view. If a Table is selected and the "Open" command button is used, it will open that table's datasheet view showing all existing records. This view allows the entry of new records and existing records can be changed. If it is not necessary to view existing records, but merely add new ones, once a table is opened, "Records/Data Entry" may be selected from the menu to facilitate the entry of new data records.

2. Queries

Queries are one of the most important components of a database system, because they provide the capability to display and/or report data to the user. Queries are used to find data within the Tables of a database. Once a query is created, it becomes the basis for forms, reports, graphs, and/or other queries.

a. Types of Queries

There are several types of queries: *select* queries, which retrieve data to be viewed or updated; *crosstab* queries, which present data in a spreadsheet format; and *action* queries, which can be used to update existing tables, delete records, and make a new table from other tables. (Microsoft Corporation, Access User's Guide, 1992)

b. Query Development

Access has a feature called *graphical query by example (QBE)*, which allows queries to be created quickly by selecting the tables to be used and then selecting the desired fields of those tables. Tables can also be joined in several ways, such as with an *outer join*, so that records from one table which do not match records from the second table can still be displayed to the user. Criteria for particular fields can be specified, so that records which have data matching the criteria will be selected.

To design a new query, the "Query" button on the Access database menu is selected followed by the "New" button. At this time, an interface is opened which allows the user to select the object(s) on which the query will operate on (Tables or Queries) in the "Add Table" window. When objects are added, they appear in the query window, such as the table PART, shown in Figure 5 below. After all of the desired objects are added to the QBE window, the relationships between them are established by connecting the related fields of the two tables. If the relationship is obvious, i.e. the two fields have the same name or relationships have been established earlier, relationships in the QBE environment appear automatically. If relationships are not previously established, they can be created or "drawn" by clicking on the first related field and holding the left mouse button down and dragging until it touches the related field on the second table. These relationships are shown in the diagrams of the completed queries. (Figures 12 and 13, below)

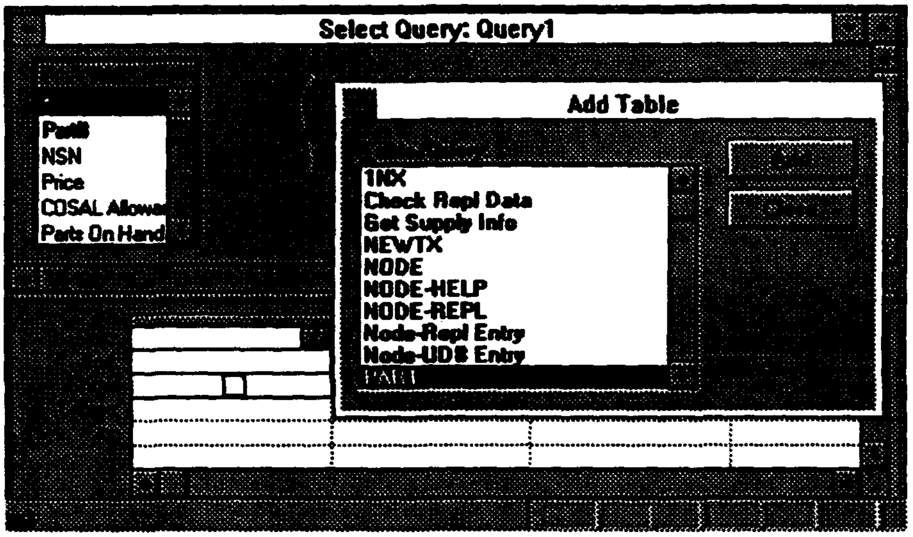


Figure 5 - Adding a Table to a Query

After drawing relationships, particular fields of interest are selected from the tables and placed into the "Field" box of the QBE grid. The easiest way to perform this task is to use the mouse and click on the particular field in a table and "drag" it into the "Field" box of the grid. The developer can choose whether or not to display a field; whether to use an ascending, or descending, or no order; and on what criteria the selection will be based, if any.

3. Forms

Menus, screens, and forms are all developed in Access as Forms, with the term screen and form being generally synonymous.

a. Form Development Environment

Forms can be created using an easy to use generator called a "Form Wizard" or from scratch. The generator requires the programmer to select a Table or

Query on which the form will be based, and then it proceeds to ask what type of form, which fields to include, and what "look" the form will have from several options presented. The basic form is designed by the "wizard." Subsequently, the developer could make changes to that form using other form development tools, which are described below.

The form design "toolbox" is used to add a number of different kinds of controls to particular form. These controls include text boxes, labels, command buttons, check boxes, list boxes, subreport/subforms, lines, option buttons, and toggle buttons. Text boxes are used for input or output of information and are either bound to a field, so they display the information from that field, or unbound. Unbound text boxes are often used for user input of information. Command buttons are used to "activate" processes relating to a form. List boxes are related to a particular list of choices, usually the particular data items stored in a particular field of a table. The choices in a list box, or combo box, can also be enumerated lists which are not from another object.

Various properties, such as control name, control source, status bar text, data format, default values, and validation rules can be set for each control. For text boxes, there are other properties such as whether the box can shrink or grow, when it will be displayed, etc.

A palette feature can be used to quickly apply colors to text, backgrounds, and borders within forms. The palette can also be used to change the "look" of controls giving them a sunken or raised 3-D effect.

The toolbox, palette, and properties box are all windows which may be activated by command buttons on the screen toolbar. The toolbar can also be used to toggle between the design view and the form view of the form. The size of the form "window" can easily be resized by dragging the borders in or out using a mouse.

In addition, once a form or menu is created, it is simple to copy it to another form name, and change the control sources and properties for a particular control. This "copy and paste" capability provides consistency within various levels of menus and across an application.

The form design environment is shown in Figure 6.

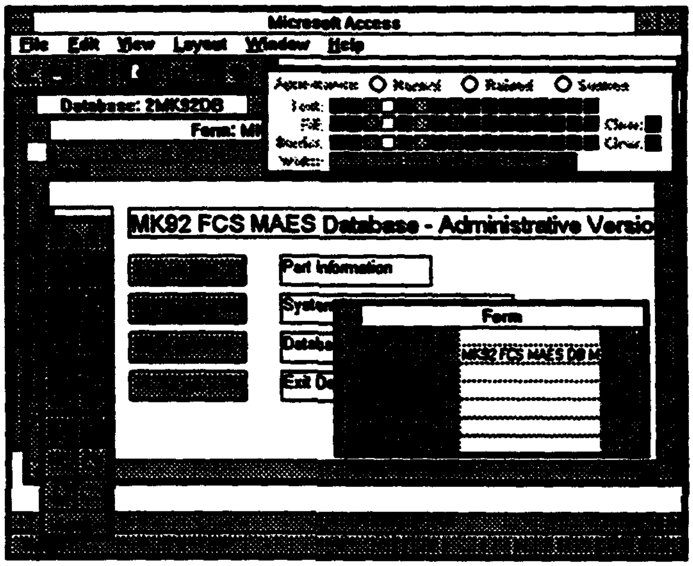


Figure 6 - Form Design Environment

b. Menu Screens

The design of menu forms involves three tasks: the design of the form itself, the design of a mechanism to perform process actions, and a method for connecting the form and the actions mechanism. Menu forms are designed using blank forms, and can not be designed using the form wizard since menu forms are not bound to an object. Labels can be added for different headings, and command buttons with labels can be added for invoking various processes.

Macros are used as the mechanism by which process actions are performed by a form (both menu and other types of forms). Forms are invoked by macros, and use macros to invoke related processes. Macros can be invoked upon the opening of a form, by a control such as a command button, or upon the exiting from a form. The use of macros with forms will be described further in Section 5.

c. *Input/Output Screens*

Input and output forms are more complex than menu forms. Where menu forms included only one action type of control, the command button, input/output screens usually include additional controls. Most output forms are bound to a source such as a table or query. If the data to be displayed comes from a single table, the form is bound to that table, and each text box or list box on the form is bound to a field in that table. If the data in a single form comes from more than one table, but does not involve a 1:M relationship on the form itself, a query can be performed on those tables, and the form's object source can be the query. As with a single table, one or more fields in the query may be bound to a particular display control.

It is possible to display related information in a form/subform combination. Generally the subform is created first, or created at the same time with the Form Wizard. When the subform is created first, the main form is created, and then the toolbox is used to add the subform control. Forms are linked via MASTER and CHILD entries in the subform control properties listing, using the common field in the related tables. These MASTER/CHILD fields relate directly to the Key/Foreign key relationship in the underlying relations. Neither of these fields have to be displayed on either form, but must exist in the sources on which the forms are based. An example of the creation of the subform is contained in Section D, below.

Forms which accept input from the user may be bound to a particular source or may be unbound. An unbound form uses text boxes and/or possibly other controls, to accept user input. This form will then use a process invoked by a command button to connect the information in the text box to another object. An example of this will be provided in Section D. Display or output forms become input forms when they accept input in the form of updates from the user.

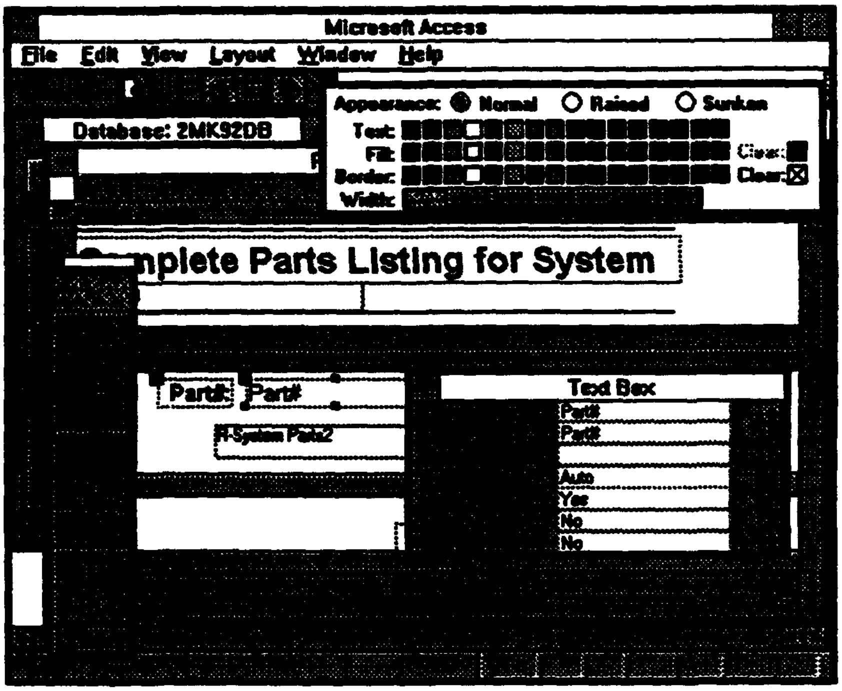
4. Reports

Reports can be created from scratch or using an easy to use generator called a "Report Wizard". As with the Form Wizard, the report generator requires the programmer to select a Table or Query on which the report will be based. The generator then asks questions such as what type of report, which fields to include, and what "look" the users want for their report from several options presented. Once the basic report is designed by the generator, it is easy to make changes after afterwards. The palette and toolbox can be used to add various design elements to reports.

Reports can be one of several types: Single column, with all fields lined up vertically; Groups/Totals, with groupings up to ten levels; or Mailing Labels. Text boxes can be added to reports that display data based on a calculation such as the sum (total) of the values. Multi-table reports can be created using a report/sub-report combination, with multiple instances in the table on which the sub-report is based related to a particular instance of the table on which the master (or main) report is based. Default reports come with controls which provide the current date each time the report is printed or viewed, and page numbers, but these features can be modified or deleted as desired.

The design of reports involves both the design of the report itself and the design of the mechanism used to invoke the report from the menu. In this application, many of the reports are report/subreport combinations, which means that subreports were designed first. When the main report is designed, a subreport control is added to the report. The properties of this control include the source of the control and which fields are used to link the report and subreport in a MASTER/CHILD relationship like that used with multi-table forms. In general, the linking field is only displayed on one of the forms, but need not be displayed on either.

Examples of the implementation of several reports are provided Section D. below. The report design environment is shown in Figure 7.


Microsoft Access
File Edit View Layout Window Help
Appearance: ● Normal ○ Raised ○ Sunken
Text:
Fill:
Border:
Width:
Clear:
Clear: ☒

Database: 2MK92DB

Complete Parts Listing for System

Part#: Part#

R-System Parts2

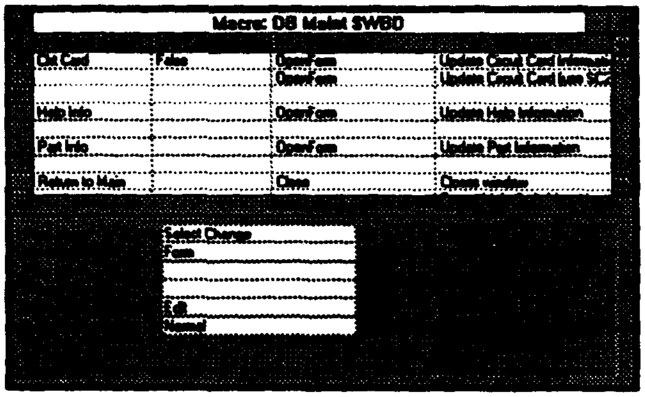
Text Box
Part#
Part#
Auto
Yes
No
No

Figure 7 - Report Design Environment

5. Macros

a. Macro Design

Macros contain one or more instructions which are grouped together to perform various procedures or actions. These instructions or actions model the process logic in the design of a database application. In Access, Macros are used to replace program code in most instances, thus simplifying application development. An example of a macro design environment is provided in Figure 8.



Macro: DB Maint SWBD

Ckt Card	False	OpenForm	Update Circuit Card Informati
		OpenForm	Update Circuit Card (use SC2
Help Info		OpenForm	Update Help Information
Part Info		OpenForm	Update Part Information
Return to Main		Close	Close window

Select Change Form

Edit

Normal

Figure 8 - Macro Design Environment

Macros can be used to perform routine tasks, such as the following (Jones, 1994, p.178):

1. Automatically opening or displaying frequently used objects, such as tables, forms, and reports.
2. Validating data entered into a form with greater flexibility than is provided within the validation mechanism in a Table.
3. Automating transfer of data (import or export) between an Access application and other software packages.

Macros consist of a series of "actions" and each action has one or more argument which applies to that action. Macro actions include the following: Close (window), GoToControl, FindRecord, Maximize, OpenForm, MsgBox, OpenQuery, OpenReport, Quit, RunCode, RunMacro, SetWarnings, and others. These actions will not be discussed, except in the context of specific application requirements later in this chapter. Each one of these actions has its own set of arguments, for example the arguments for the OpenForm action are the name of the form which is to be opened and several other items which apply.

The macro design environment allows the user to select an action from a list of permitted actions, and then the appropriate arguments are displayed in the window below. The mechanism for entry of arguments is also shown in Figure 7. Macros can be designed as a single set of actions, or as macro groups. Storing related macros together as a group simplifies the design environment for the programmer.

b. *Using Macros With Forms*

On forms, command buttons are used to invoke these macros, through the use of the "On Push" property of the command buttons. The macro name is entered into this property, and when that button is "pushed" or selected, the macro actions are invoked in sequence.

Menu forms are displayed using the macro "OpenForm" command, which will display a particular form to the user. Some menus invoke submenus, and in this case, a macro is used by the menu to open the submenu form. Generally, each menu command button is linked to a macro which opens a form of one type or another. What is not apparent to the user, is that there are other actions which are performed by the same macros, either before or after the form is displayed. An example of this might be a macro action which turns "Echo" off, so that the user does not see messages which Access normally provides while macros are run. Another example is a macro action which would "Maximize" a form, thereby filling the entire screen with a form, after it is invoked.

c. *Linking Input and Output Forms*

One useful technique for linking input forms and output forms is through the macro action OpenForm and its "Where Condition" argument. The initial form displayed may request an item of interest. This form is not bound to a particular source, and the control into which the user types the requested information is also unbound. A command button on the form can then invoke a macro which retrieves the necessary information based on user input, and then displays it. The simplest form of the macro works in the following manner: the OpenForm action contains a number of arguments, the first of which is the name of the form which is to be opened. The next important argument is "Where Condition," which allows the programmer to specify which field in the source of the display form is based on the control in the form receiving the user input. An example of the syntax used is as follows: [Field Name] = Forms! [Input Form Name]![Control Name]. This presents the user with the record having a match in that particular source field with the information provided into the Input Form in the control of the name specified. Another important argument is whether the user will be allowed to edit the data presented or whether it will be "read only."

6. Modules

a. *Use of Access Basic*

Access Basic is not a traditional programming language and cannot really be treated as such, since the language is tightly integrated with Access-specific *objects* (such as tables, forms, queries, etc.) such that the language can directly manipulate these objects. Access Basic is primarily used to perform tasks that cannot be performed with Access objects though the use of macros, or is used to shield the user from the inner workings of the application in a controlled run-time environment. Access Basic is particularly useful for transaction processing, error handling and trapping, performing DDE, and the creation of reusable code libraries. (Perschke and Liczbanski, 1993, p. 170)

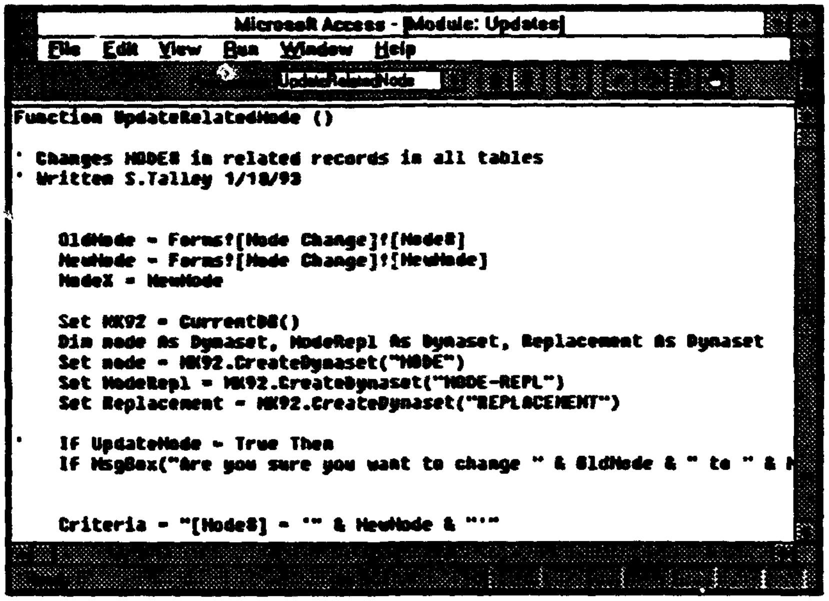
Access Basic code is stored in objects called *modules*. Modules are divided into *procedures* or functions, which are best used to perform specific tasks in order to modularize program code and maximize reuse. (Perschke and Liczbanski, 1993, p. 181)

Modules are used to define functions specified in the design phase of Chapter III. Some actions specified in logical procedures are actually implemented in modules vice macros, usually because they perform specific tasks which the macro environment is not capable of performing. Modules can invoke macros, using the "RunMacro" command, and macros can invoke functions in modules, using the "RunCode" action. This allows for flexibility and the maximum reuse of macros and functions once they are designed.

b. Access Basic Language

Access Basic provides the capability for a number of different types of data and the manipulation of those data types. It also contains several flow control structures such as decision structures (If... Then... Else..., Select Case), loop structures (Do Until... Loop, Do While... Loop, Do ... Loop While, For... Next). "Do Loops," for example, are used to execute a block of code while a condition is True. (Perschke and Liczbanski, 1993, pp. 187-188)

Within the Access Basic language, there are different types of elements, and some of these are described as follows: **Actions**, such as those used with macros, can be executed directly from Access Basic procedures using the "DoCmd" statement; **Functions** are preprogrammed language elements which return a value which can be returned in a statement, such as DDEInitiate, DDERequest, and many others; **Objects**, such as Forms, Reports, Tables, Database, and others; **Operators**, like "+", "And", "Or", "Not", and others; **Properties**, which are the same as the properties attached to objects, and which can be controlled directly from the code if necessary; and several others. An example of the Module Design Environment and the Access Basic Language is shown in Figure 9. (Microsoft Corporation, 1992, Access Language Reference)


Microsoft Access - [Module: Updates]
File Edit View Run Window Help
UpdateRelatedNode

```
Function UpdateRelatedNode ()

' Changes NODE# in related records in all tables
' Written S.Talley 1/18/93


    OldNode = Forms![Node Change]![Node#]
    NewNode = Forms![Node Change]![NewNode]
    NodeX = NewNode

    Set MK92 = CurrentDB()
    Dim node As Dynaset, NodeRepl As Dynaset, Replacement As Dynaset
    Set node = MK92.CreateDynaset("NODE")
    Set NodeRepl = MK92.CreateDynaset("NODE-REPL")
    Set Replacement = MK92.CreateDynaset("REPLACEMENT")

'   If UpdateNode = True Then
    If MsgBox("Are you sure you want to change " & OldNode & " to " & N


    Criteria = "[Node#] = '" & NewNode & "'"
```

Figure 9 - Module Design Environment / Access Basic Language

C. DATA IMPLEMENTATION

Using the Access Tables facility, the relational design developed previously was converted into an Access database. This process is fairly straightforward, in that relations and their properties are converted respectively into database tables and fields. Tables and their fields are listed in Appendix E. Most fields used in this program were of a Text nature, since there were very few data items which were strictly numerical or "fixed" formats.

The tables in this application contain long fields, due to the nature of the data they contain. The length of fields, and therefore the length of a record, makes viewing records in a "datasheet" or spreadsheet-type of format impractical.

Validation rules are entered in the form of expressions inside the Table design function of Access, and are consistent with the attribute definition "masks" stated in Appendix A, Section B. (Jones, 1994, pp. 48 - 49)

D. PROCESS IMPLEMENTATION

Process implementation consists of linking the required forms and reports, developed initially during the design phase of Chapter III and shown in Appendix D, with their underlying processes. The processes designed in Chapter III are implemented through the use of queries, macros, and modules which connect the forms and reports to the appropriate data in the tables. The methods and tools used to implement processes are discussed below.

1. Process Development and Examples

Process logic developed in the design phase of Chapter III can be roughly translated into a series of tasks which are implemented in a number of different ways. Some tasks may be implemented in more than one way, and not all possible methods of performing various tasks will be covered in this thesis, rather only those actually used.

a. Implementation of Specific Process Tasks

Process tasks which are part of the process logic developed in the Design Phase, such as update, activate, retrieve, and others, are implemented in fairly standard ways within this application. Appendix E contains a listing of these tasks and the way(s) in which they are implemented (modules, macros, etc.).

b. Process Example: Locate Supply Parts

The process which is listed in Procedure U1.2L LOCATE(U) in Appendix D is implemented using a combination of macros and an Access Basic procedure. First, the INPUT PART# [UPDATE PART] form is open. The user provides input in the form of a Part# and then "pushes" the **Locate** command button which activates this procedure.

The "On Push" property activates a macro called Part Macros.Locate Parts. This macro performs actions described in Figure 10:

1. **Set Value** - Description: Hides form
   Arguments: Visible: No

2. **OpenForm** - Description: Opens the Part Supply Info form, retrieves the record which has the value of Part# contained in the Part# field in the Part# Input Form, and displays it in the form.
   Arguments: Form: Part Supply Info Description: Opens the Part Supply
   View: Form
   Where Condition: [Part#]=Forms![Part# Input]![Part#]
   Data Mode: Edit
   Window Mode: Normal

3. **RunCode** Description: Checks to see if the Part# in the Part Supply Info form is a null value, if so, then the Part# does not exist or no value was entered. A message box, with a message to that effect is provided to the user and the user returns to the Part# Input form.
   Arguments: Function Name: CheckEmptyPart()

Figure 10 - Locate Part Macro

The function referred to in the **RunCode** action, CheckEmptyPart() is shown in Figure 11. Note, this function calls another macro, with the **DoCmd RunMacro** statement. This macro is the one which returns the user to the Input Part# screen.

```
Function CheckEmptyPart ()

  PartX = Forms![Part Supply Info]![Part#]
  If IsNull(PartX) Then
    wID = MsgBox("No matching Part # found (or No Part # entered)", 32,
      "No Match Found")
    DoCmd RunMacro "Part Macros.More Parts"
  End If

End Function
```

Figure 11 - CheckEmptyPart() Function

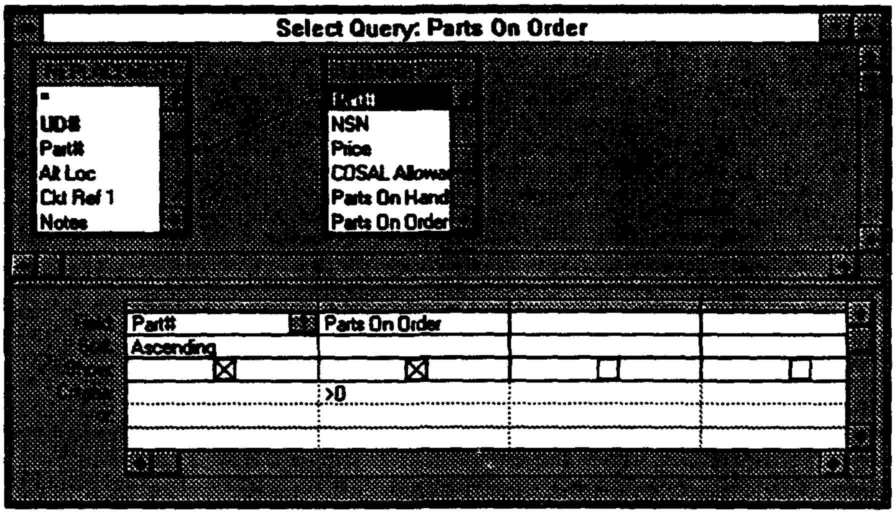
2. Query Development and Examples

Queries are important to this application because they provide the basis for several important functions, which will be discussed in this and later sections. Queries using outer joins and/or specified criteria are used in this application. Both select queries and action queries are used in this application, and will be discussed in the examples provided. A list of the queries in this application is provided in Appendix E.

This section will contain discussion concerning the implementation of several of the queries in this database application. Note, there are a number of different ways in which these queries may be accomplished, in order to achieve the same results.

a. Parts On Order Query

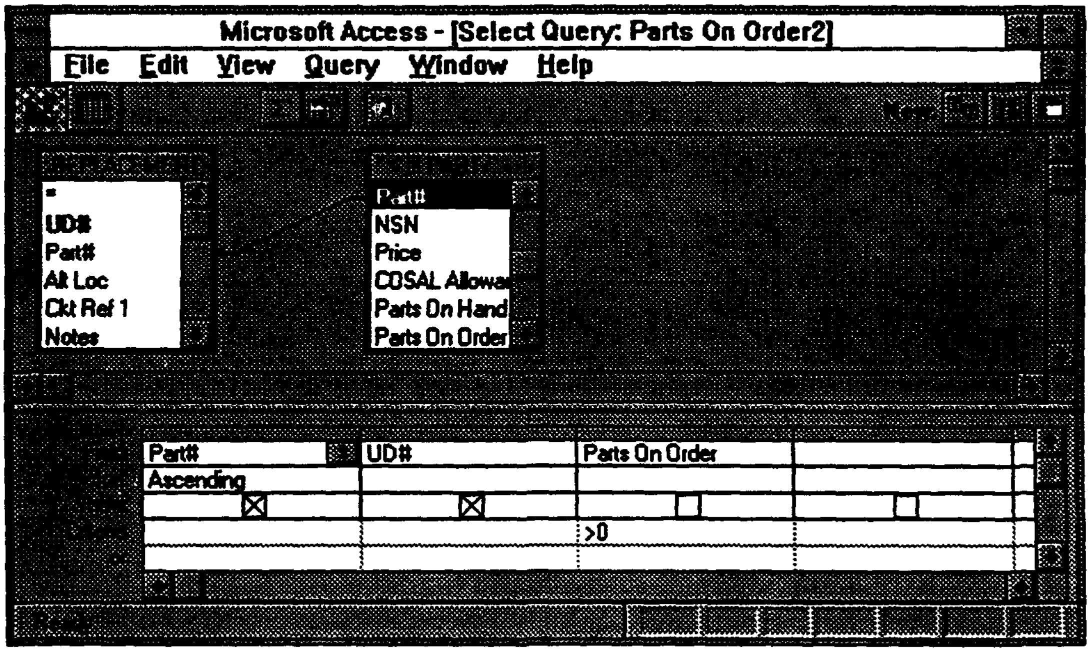
The Parts on Order query used for the Parts on Order report was created in the QBE environment. The development of the report is discussed in Section 5, below, but the implementation of the actual query is discussed here. In this query, two tables are used in the query: PART and REPLACEMENT. These tables are linked through the common attribute Part#. Because it is possible to have PART instances which do not have corresponding REPLACEMENT instances, an outer join from PART to REPLACEMENT is used to retrieve all instances of PART, including those that do not have REPLACEMENT instances associated with them. Note, this outer join is designated by the arrow on the relationship link between the tables in Figure 12. The selection criteria used is that the value of the field Parts On Order is greater than zero, and the fields Part# and Parts On Order are displayed. This query is shown in Figure 12, below.



Part#	Parts On Order		
Ascending			
☒	☒	□	□
	>0		

Figure 12 - Parts on Order Query

A second query is also used for the same report, Parts On Order2. This query is used to link the REPLACEMENT table and the PART table to get all UD#s associated with the selected Part#., This query is shown in Figure 13.


Microsoft Access - [Select Query: Parts On Order2]
File Edit View Query Window Help
*
UD#
Part#
Alt Loc
Ckt Ref 1
Notes
Part#
NSN
Price
COSAL Allowa
Parts On Hand
Parts On Order

Part#	UD#	Parts On Order	
Ascending			
☒	☒	□	□
		>0	

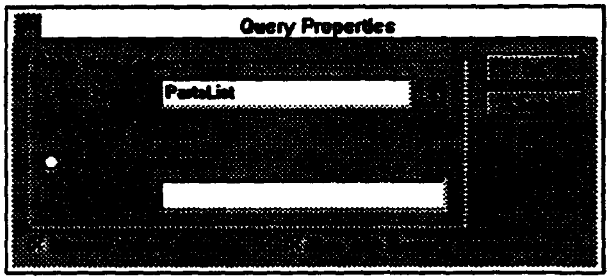
Figure 13 - Parts on Order2 Query

b. System Parts List Query

There are three queries used in creating the System Parts List report, which is a list of all the parts which are used by the expert system, i.e. parts that currently have replacements. Two are created in a similar manner as in the Parts On Order query above, but instead of being based on one of the main tables in the database, they are based on a temporary table which is created through the use of a "Make Table" action query. The use of the action query will be discussed here, with respect to the creation of the System Parts List report.

The first step in creating a "Make Table" action query, is to create a select query, using the QBE environment as discussed above. The purpose of this query is to select a list of unique parts from the REPLACEMENT table to be used as the basis for other queries. The criteria Part# Is Not Null is also used, to eliminate records in this table which have no value for Part#.

Once the select query is designed and tested, it is converted into a "Make Table" action query by selecting "Query: Make Table" from the menus at the top of the Access window. The name of the table to be created, when the query is performed, is entered into a form as shown in Figure 14, below. The table created by this query, can be considered a temporary table, and is named **PartsList**. Using the "Check Box" for "Unique Values Only" at the bottom of this form ensures there are no duplicate values in this new table, which could occur when several UD#s have the same Part#, as Part# is not the original key field of the REPLACEMENT table.

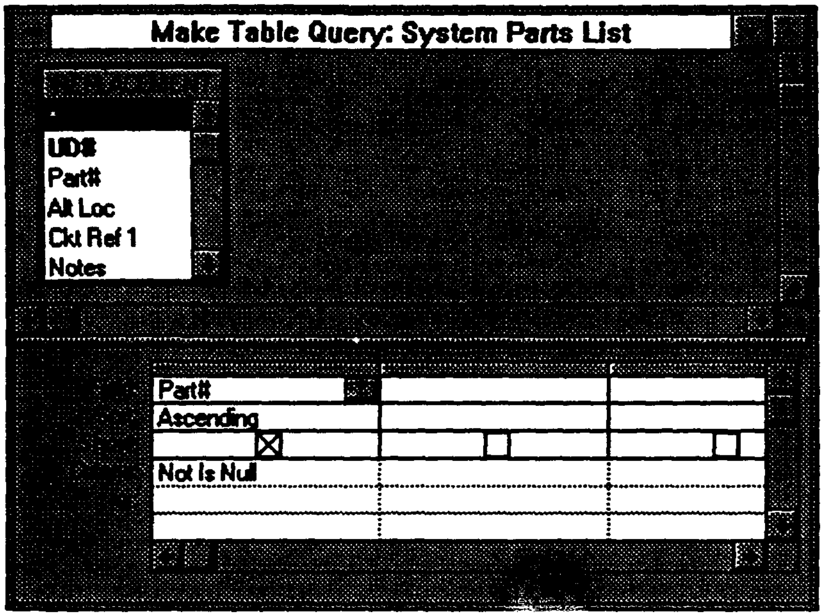


Query Properties

PartsList

Figure 14 - Make Table Query Properties

Each time this query is run, the PartsList table will be overwritten with new data. Unless warnings are suppressed, this action will tell the user that this query will overwrite existing data and verify whether or not the query should continue. The Systems Part List query is used as the basis of another query, which are used in the System Parts List report, and is shown in Figure 15. The other queries for this report are implemented in the same manner as those for the Parts On Order report, in the previous example, except they are based on the temporary PartsList table instead on one of the application's base tables.



Make Table Query: System Parts List

*
UD#
Part#
Alt Loc
Ckt Ref 1
Notes

Part#		
Ascending		
☒	□	□
Not Is Null		

Figure 15 - System Parts List Query

3. Menu/Screen/Form Development Examples

Examples of the implementation of several forms are provided in the section below. A list of the application forms is provided in Appendix E. This list also contains documentation concerning each form's source, its controls and the processes invoked by these controls, and any special characteristics or properties of the form.

a. *Update Node-Replacement - Admin Form*

This form is one of the most complex forms in this application. It is comprised of three separate forms, which are linked together in a form/subform relationship. The "Form Wizard" is used to design the form/subform combination. With two layers of subforms, only the inner two forms could be designed using the Form Wizard method. All forms may also be created with the Form Wizard, using the "single-column" selection, and adding a subform control from the Toolbox as required.

(1) Form: Replacement Info Sub(sub)form. The first form created was the form based on the REPLACEMENT table. The first step after selecting the "Form: New" button, was to choose the correct table, and then select the "Form Wizard" button. After that, the **Embossed** "look" was selected from the choices presented. Next, the fields in the table were listed, and the ones to be included in the table were selected, in the particular order desired. The final step is to select a title (form heading) for the form.

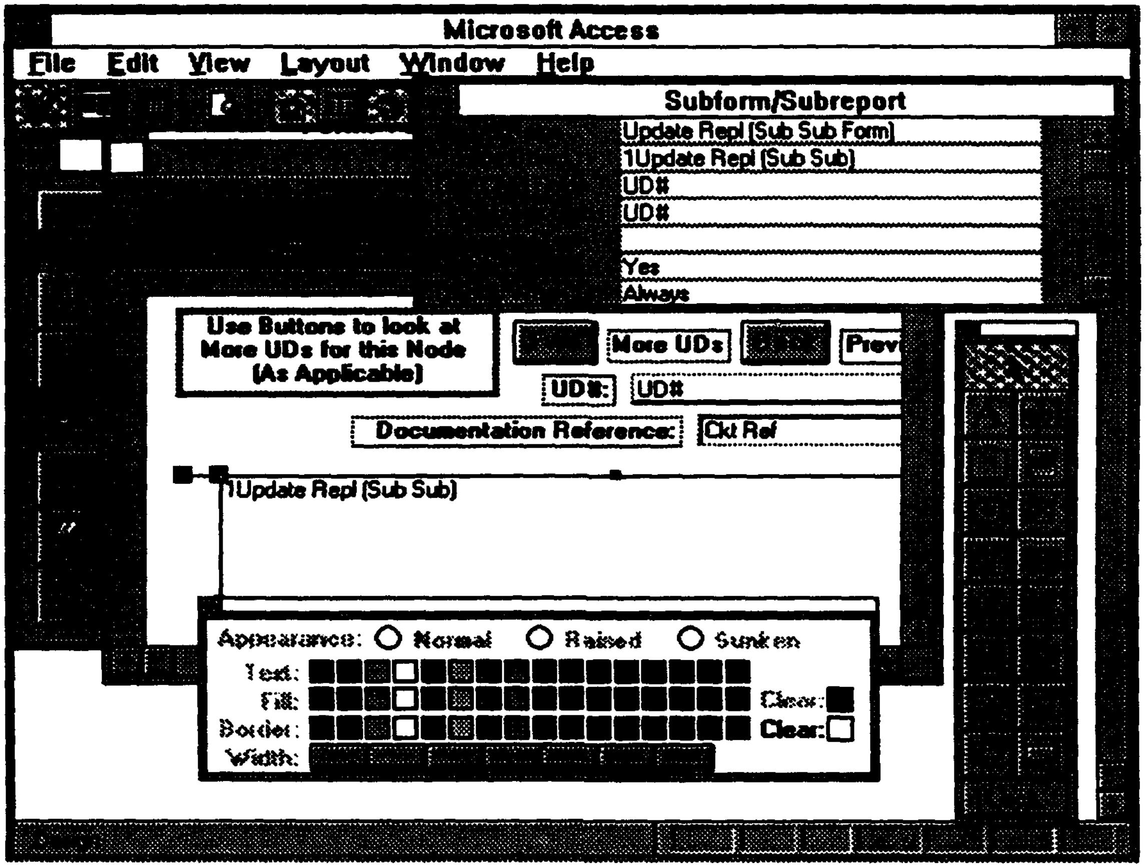
In this form's properties box, the scroll bars and the record selectors were checked off to prevent the user from trying inadvertently to select records using these tools. A property in the Form Header (section) properties box was used to "hide" the header section. Finally the form was resized and the text boxes moved so that the field labels display in their entirety.

(2) Form: Node-Replacement Info Subform. The Form Wizard was used again to create a form, based on the NODE-REPL table. In addition, to changes made to the previous form, the "look" of the UD# field was changed to "raised" versus "sunken", to distinguish a field which is not directly updatable from ones which are not.

Several controls were added to this form: two command buttons, three labels, and the subform. The "command button" button on the toolbox was used to add these controls to the form. First, the properties were changed as follows: the caption was changed, so the words on the button indicated the function of the button; and the Update Macros: Scroll Up macro was added to the "On Push" property of the "Back" button and another macro to the "Fwd" button, to provide the correct functionality. Three labels were added to provide information to the user concerning the buttons. The palette was used to add a frame and color to one of the labels, to draw the users attention to the information contained therein.

The third type of control, the subform/subreport control, was then added to the form. Through trial and error and switching between form views (Design and Open), the subform was sized so that it displayed correctly on this form when this form is opened. The Source Object control must contain the name of the subform, which in this case is **Replacement Info Sub(sub)**, and the Link Master/Child Fields property is set to UD#.

Figure 16 shows the design environment of this form. The properties box for the subform control is shown in the upper right-hand corner, the toolbox is shown in the lower right-hand corner, and the palette is shown in the lower center of the figure.


Microsoft Access
File Edit View Layout Window Help
Subform/Subreport
Use Buttons to look at More UDs for this Node (As Applicable)
More UDs
Previ
UD#:
Documentation Reference:
Appearance: ○ Normal ○ Raised ○ Sunken
Text:
Fill:
Border:
Width:
Clear:
Clear:□

Update Repl (Sub Sub Form)
1Update Repl (Sub Sub)
UD#
UD#

Yes
Always

UD#
Ckt Ref
1Update Repl (Sub Sub)

Figure 16 - Node-Repl Info Form Design Environment

(3) Form: Node-Replacement Info. This is the main form for this set of subforms, and is the one opened by the command button "By Node#" on the "Select Change" Submenu of the DB Maintenance Submenu. The form and its controls are designed in the same manner as the previous form. There are two differences to this form: one is that the header is left so it is visible and the second is that the "*" button invokes a macro which in turn invokes a series of screen displays and associated logic.

b. Input Node# Form

This form was not designed using the Form Wizard since it is not based on a source object . Form properties were set so that there were no record selectors or scroll bars visible, as in the previous forms. When a blank form is opened, there is no automatic creation of a form header or footer, so a Label control was used to add a form heading. Two command buttons were added, with each button invoking a different macro using the On Push property.

4. Report Development Examples

The reports implemented in this application are listed in Appendix E. These reports were all implemented with the Report Wizard, using the "Executive Look."

a. Parts On Order Report

The Parts On Order report is comprised of two reports, **Parts On Order** and **Parts On Order2**, which are combined in a report/subreport combination. A sample report is provided in Figure D-16, Appendix D.

(1) Report: Parts On Order Subreport. The Parts On Order report is a single-column type report based on the Parts On Order2 query, which was discussed in Section 2.b.(1)(b) above. While both Part# and UD# are included in the report source query, only UD# is included on this subreport since Part# will be shown on the main report. The field to be used for a sort order was also selected. The default settings for Report Header and Footer were changed so that no Headers or Footers, either Report or Page, were visible. The design environment of this report is shown in Figure 17.

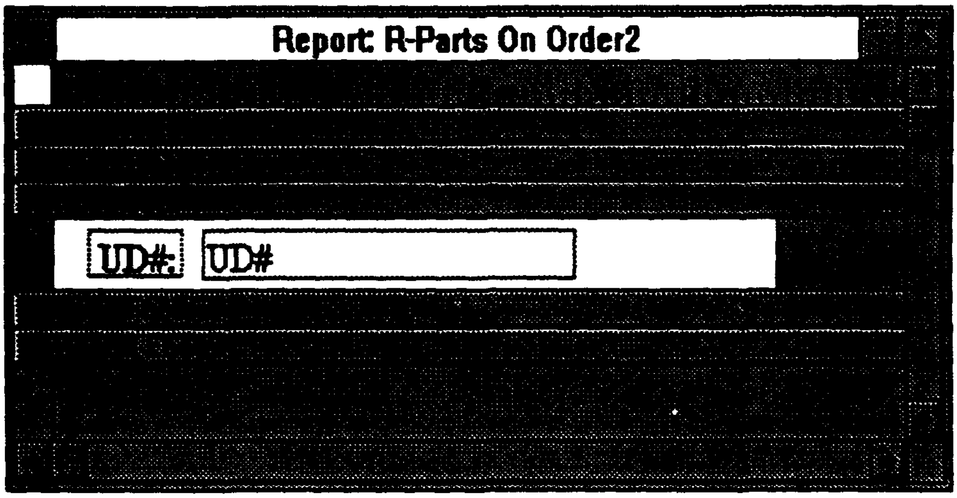


Figure 17 - Parts On Order Subreport Design Environment

(2) Report: Parts On Order. This report was based on the Parts On Order query (See Section 2.b.(1)(a), above). Both Part# and Parts On Order Fields were placed on the report, sorted in order of Part#.

A subreport control was created through the use of the toolbox, with its source being the name of the subreport (Parts On Order2). The size of the subreport must be manipulated within its design environment so that the detail section of the subreport fits within the size of the subreport control. This is performed largely through trial and error, and by noting the size of both items on the ruler. The "Link Child Fields" and "Link Master Fields" are set to Part#, since this is the attribute which links the report and subreport such that all UD#s for a particular Part# are listed in the subreport section of the report. In addition, the default is set so that one record of the main report is displayed on a single page. If, as in this case, it is desirable to have multiple records per page, it is necessary to resize the detail section of the form so that the Page Footer section break is right after the last item in the report design, the Parts On Order field and its label.

The design environment of this report is shown in Figure 18. An example of the report itself is shown in Figure D-16, Appendix D.

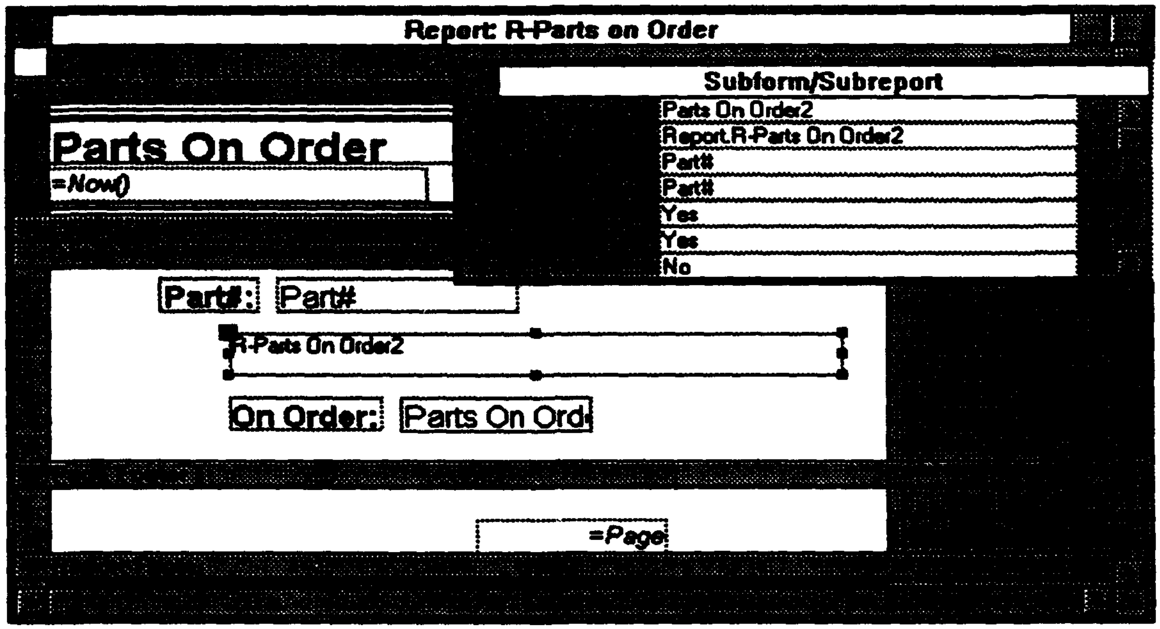


Figure 18 - Parts On Order Report Design Environment

b. System Parts List Report The System Parts List report is comprised of two reports, **System Parts** and **System Parts2**, which are combined in a report/subreport combination. The implementation of these reports is very similar to the implementation of the Parts On Order Report, and only the differences between the two reports will be discussed. An example of the printout of this report is provided in Figure D-18, Appendix D.

(1) Report: System Parts List Subreport. The System Parts List report was created using the "Report Wizard," based on the System Parts query, which was discussed in Section 2.b.(2)(b), above. This report was created by copying the Parts On Order report and making two changes. First, the Report property Record Source was

changed to System Parts2 and the detail section of the report was enlarged slightly to fit into the space of the main report.

(2) Report: System Parts List. This report was designed using the "Report Wizard," based on the System Parts query (See Section 2.b.(2)(a) above). Part# is the only field on this report and is sorted in ascending order. The subreport control was added, its label removed, and System Parts2 was entered into its Source Object property. The Child Link Fields and Master Link Fields properties were also UD#. Again, this provides all UD#s for each Part#. In order to report as many parts (and their UD#s) per page as possible, the detail section was compressed to the smallest practical size. The design environment of this report is shown below in Figure 19.

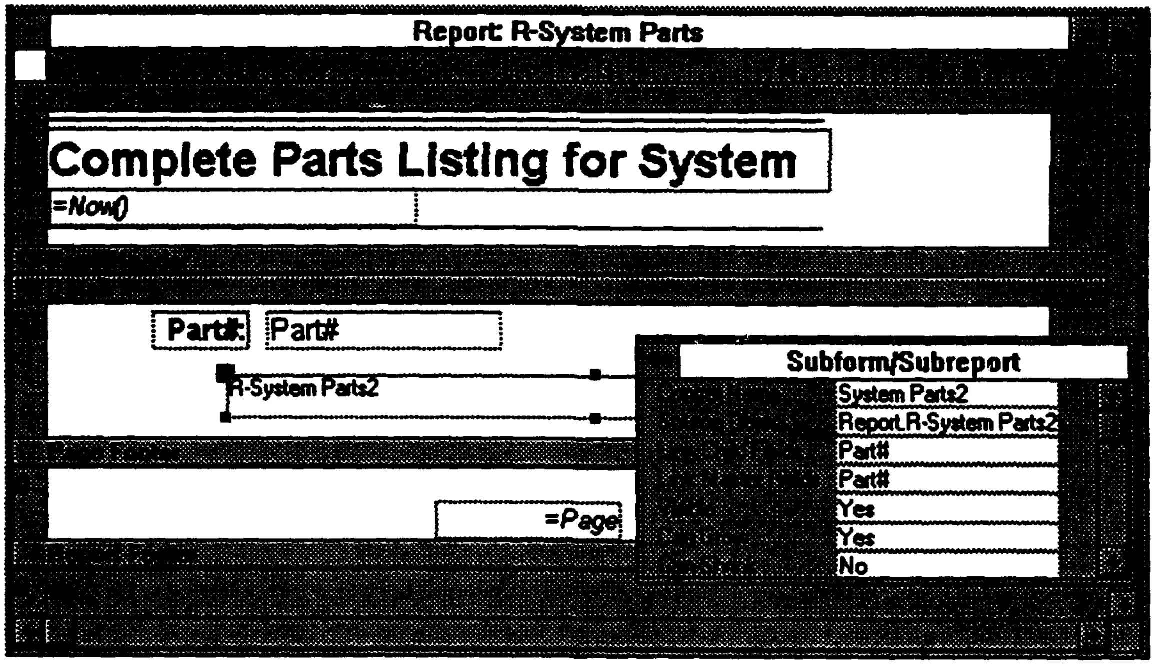


Figure 19 - System Parts List Report Design Environment

E. OUTPUT OF THE IMPLEMENTATION PHASE

At the conclusion of the implementation phase, the database application is completed, and operational, as designed. This includes the development of the tables, forms, reports, and their underlying processes and queries. If a prototyping methodology was not being employed, the database application would be independently tested and delivered to the user. Instead, the prototype application is used as a device to elicit feedback for future enhancements. Once the users experiment with the prototype, they are able to better define their requirements and comment on the preliminary design, and its structure and processes could be changed accordingly. The final chapter contains some of the feedback received from the program managers at NSWC, which will guide the direction of follow-on work.

The next chapter contains a discussion of the separate issue in this thesis, that of the interface between the MK92 FCS MAES and this database application.

V. INTERFACE BETWEEN EXPERT AND DATABASE SYSTEM

In general, the purpose of creating an interface between the expert system and a database is to provide a powerful mechanism for storing and managing information required by the expert system user, which is more efficient than mechanisms provided by the expert system itself. Efficiency can be defined both in terms of flexibility to retrieve different kinds of data for different purposes and in terms of maintainability and modifiability. This chapter discusses Windows interprogram communications mechanisms and the efforts made towards using these mechanism to link the MK92 FCS MAES and the Access database application which stores and manages the information the expert system is requires.

A. WINDOWS INTERPROGRAM COMMUNICATION

1. Dynamic Data Exchange (DDE)

DDE is one method of interprogram communications between two Windows applications. The program that initiates the communication and requests data or services is called the client, and the program that responds to the client's request is called the server. Some applications can be both client and server.

DDE can be used to establish links between programs in several different ways. As previously mentioned, one method involves requesting data (or services) from the server. Another method involves the server notifying the client that an item has changed value, after which time, the client could make a request in order to obtain the new data. A third method involves a "hot link," which means that the server application sends the new value to the client any time the data value changes. (Perschke and Liczbanski, 1993, p. 243)

In general, the application initiating a DDE link opens a DDE channel with the other application. The client can then use a number of DDE functions or statements to perform different tasks. Since the syntax used for DDE by different programs is not standard, functions available to Access and Adept are discussed separately.

a. DDE in Access

Access is capable of handling a number of different DDE functions or commands, and these are listed in Appendix F. Access has the capability of being both a DDE client or a DDE server. **DDEInitiate()** is used to initiate a conversation between Access and another application. Included in the argument for this function is the name of the *application* which can respond to DDE, such as the name of "Adept," and the name of the *topic*, which in the case of Adept is the specific "application name" being executed. The topic name must be recognized by the "called" application. This function is used to establish a channel between two applications, which can be used later with other DDE functions. The use of this and other functions is shown in the example provided in Appendix F, Section C-2. (Microsoft Corporation, Access Language Reference, 1992, p.118)

Another important function is **DDERequest()**, which is used to request an item of information from another application. Arguments for this function include the *channel* (as previously mentioned) and the *item* of interest. The name of the item must be something which is recognized by the other application, such as variable or spreadsheet name. (Microsoft Corporation, Access Language Reference, 1992, p.121)

DDETerminate() is used to close a channel which has been opened, with the argument being the *channel.* (Microsoft Corporation, Access Language Reference, 1992, p. 124)

Additional commands are available, as shown in Appendix F, Section A-1, but they are not used in this thesis.

b. DDE in Adept

DDE in Adept is very similar to DDE in Access, except that different syntax is used for similar functions. Like Access, many functions involve a *channel*, an *application*, a *topic*, and/or an *item*. Theoretically, Adept can carry on a number of conversations with different client programs at the same time. In a similar manner, another application can request data from more than one Adept application, or two different applications can request information from two different Adept applications at the same time. (Symbologic Corporation, 1991, p. 33)

The first step in initiating a conversation with another application is to use the **OpenChannel** function to open a channel. To use this function, the name of the *channel*, the other *application*, and the *topic* are required (Symbologic Corporation, 1991, p. 36). In the case of an application running under Access, the application name is "MSAccess," and in the case of an Access run-time application, the application name is the name of the run-time executable file[1]. The syntax of the commands actually used by the Adept application for the purpose of this thesis is provided in Appendix F, Section C-1.

Adept can request information from another application, using the **Request** function. In this case, the *channel*, *item*, and *data* arguments are used. *Item* is a variable which identifies the data (in the other application) and *data* is a variable where the data is to be stored in Adept. (Symbologic Corporation, 1991, p. 37)

Execute can be used to execute a function, run a program, perform a task, or a number of other things in the other application. This function uses the arguments *channel* and *command*. The command is the most important part of this

[1] A file with the ".MDB" extension.

function, as it must be something that can be understood by the other application. In general, the syntax of the other program must be used to correctly use this function. (Symbologic Corporation, 1991, p. 39) A good example of this is the use of Execute with respect to an Access application. Only commands which Access understands can be used with the Adept Execute command.

One other function which might be useful is **Poke**. This function uses arguments *channel, item,* and *data* to specify what data item in Adept should be "sent" to the other application, and where in the (client) application it should go. (Symbologic Corporation, 1991, p. 39)

Other commands are available, as shown in Appendix F, Section B, but they are not used in this thesis.

2. Object Linking and Embedding (OLE)

Object linking and embedding, or OLE, is a Windows mechanism which allows objects created in one application to be linked to or embedded in another application. This method of interprogram communication is closely related to DDE, and like DDE there are both clients and servers. As of Version 1.1, Access can only act as an OLE client, in that it can only accept OLE objects from the server application.[2] An enhanced version of OLE, OLE 2.0 is beginning to be used in applications, however Access 1.1 is not OLE 2.0 compliant. (Jennings, 1993, pp. 504-505)

OLE capabilities were first used with Microsoft Excel^{TM} and PowerPoint^{TM}, and were officially introduced with Windows 3.1. In general, an OLE server provides a source "document" to an OLE client "document." Once an OLE object is embedded or linked to the destination document, this document becomes a "compound document." In the case of Access, the destination document can be tables or forms. A

[2] Access 2.X is expected to be OLE 2.0 compliant.

source document can be a file from a word processor, a spreadsheet from an application like Excel, a slide from a graphics program like PowerPoint, a graphics image from a variety of programs, or even multimedia objects such as music. (Jennings, 1993, pp. 505-506)

When an OLE object is embedded, a copy of the OLE object's data is included in the destination document. Embedding an OLE object ensures that the object's data is available regardless of what happens to the source. Linking a document is more appropriate when the source document is likely to be changed periodically. However, if the source document gets moved to another location or deleted, it no longer exists at the destination either. (Jennings, 1993 pp. 510-511) A linked object will also update in the client application whenever it is edited or updated in the server application which created it. (Perschke and Liczbanski, 1993, p. 238) In some cases linking will save disk space, since the object is only stored in one location. Graphics images, however, will require as much or more disk space to link to an Access table or form, as to embed it. (Jennings, 1993, pp. 510-511)

One of the primary advantages to storing documents as OLE objects within Access, rather than as "pictures," is that by using OLE the object can be edited through the original server program which was used to embed the program. A source document which has been embedded as a picture, rather than as an OLE object, can no longer be edited. (Jennings, 1993, p. 511)

Other applications which can act as servers include Lotus Corporation Ami Pro^{TM}, WordPerfect^{TM} for Windows, CorelDRAW!^{TM}, and Windows Paintbrush^{TM}. There are also a number of commercial OLE-compliant drawing and image editing applications which can be used to create and manipulate photos and other images. (Jennings, 1993, pp. 511-517)

Adept is not OLE compliant at this time, but may be in future versions.

B. IMPLEMENTATION OF DDE INTERFACE BETWEEN ADEPT AND ACCESS APPLICATIONS

1. General Requirements

Logically, the basic functionality of the interprogram communication between Adept and Access is as follows:

1. When a conclusion is reached in Adept and part information is needed from the database, Adept must initiate a conversation with Access, asking if Access is ready to communicate.
2. If Access is ready to communicate, a communication channel is opened between Access and Adept.
3. Adept sends a message to Access asking for data, or asking Access to perform a query and supplies the appropriate parameters.
4. Access acknowledges the request and carries it out (or denies it).
5. After the data and commands are exchanged, Adept sends a message to the Access program notifying it that the conversation is about to be terminated and then closes the communication channel.

2. Possible Solutions

To perform the above basic functionality, an obvious solution is for the Adept application to send the Node Number into the Access Application, where a query would be run, and then the Adept application could request (or the Access application send) the results of query, so it can be displayed to the user.

Another obvious solution would be for an SQL-type query to be made directly from Adept, eliminating the need to send query parameters to Access.

OLE was obviously not an appropriate interface between the Adept expert system and the Access database because Adept does not currently support OLE.

a. Establishing Communications: First Approach

(1) Methodology. The following steps were taken in the first attempt in establishing communications between the two programs:

1. **Open** a channel between Adept and Access, with Adept as the client and Access as the server.
2. Use **Poke** from Adept to Access to place the query parameter into a temporary table in the Access application.
3. **Execute** a query, whose parameter is the value received from Adept through the poke.
4. **Request** the results of the query and store it in a variable in Adept to be displayed to the user.
5. **Close** channel.

(2) Results. The problem with this approach is that **Poke** cannot be used to send data into an Access table. Since the initial attempt at establishing a link between the expert system application and the database application using the **Poke** command to send information to the Access application failed to work, a second approach was used.

b. Second Approocch

(1) Methodology

1. **Open** a channel between Adept and Access, with Adept as the client and Access as the server.
2. **Execute** an SQL-type query directly from Adept, which locates the correct records based on a parameter related to a particular expert system result node.

4. **Request** the information into an Adept variable.

3. **Close** channel.

(2) Results. SQL queries of any sort did not seem to work from Adept to Access. While the set of statements required for the type of query in requirement 2, above, is complex, even simple SQL-type statements did not get any response from Access. Adept may not support the kind of statements Access needs or the correct syntax may not have been used.

The first attempt to use of SQL-type statements was to perform the complete query as shown in Figure 20, below. The second attempt to use of SQL-type statements was to try to solve the problem of how to get information into an Access table as shown in Figure 20.

```
**Select particular records from one table to start with:
AccCall = OpenChannel("MSAccess", "2MK92DB.MDB;SQL");
Execute (AccCall, "RunSQL SELECT * FROM Node WHERE [Node#] =
        ""N006"";");

**Update 1NX table with Node Number vice Poke command:
AccCalls = OpenChannel("MSAccess","C:\ACCESS\2MK92DB.MDB;SQL");
Execute (AccCalls, "RunSQL UPDATE 1NX SET [Node#] = ""N006"";");
```

Figure 20 - Adept DDE Statements Using SQL

c. *Third Approach*

(1) Methodology

1. **Open** a channel between Adept and Access, with Adept as the client and Access as the server.

2. Use the **Execute** command to execute an Access Query, with the appropriate parameters.

3. Use **Request** to get the results of the query and store it in a variable in Adept to be displayed to the user.

4. **Close** channel

(2) Results. Problem: While Adept DDE syntax supports the use of an argument (i.e. a query parameter) in the Execute command, Access does not support the use of arguments with its queries or macros. Since Access functions do support arguments, passing parameters between the Node variable in Adept and the Node_number variable in an Access function was attempted, also without success.

In the Execute statement in Adept, as previously discussed, the first item is the channel, and the second is the command. The command has to be something understood by Access. First, the command "RunCode PartsQuery()" (or any number of syntax variations) could not be used in an Execute statement, as the **RunCode** function can not be used directly by a DDE Execute. Instead, the RunCode action can be used in a macro, which then may be executed using the syntax "Execute(Chan, "[1TableOnly]". Unfortunately, there does not appear to be any way to pass arguments to a macro.

There appears to be problems with the compatibility of the Adept SQL syntax with that of Access in that arguments outside of the Access statements cannot be passed with them. Even if an argument could be used, it might not be in the form of a variable, thus eliminating some of the benefits of using a database if each Node number had to be "hard-coded" inside scripts.

d. Fourth Approach

(1) Methodology This methodology was suggested by Microsoft Product Support personnel, and involves having Access request the query parameter from Adept, instead of having Adept send it to Access.

1. **Open** a channel between Adept and Access, with Adept as the client and Access as the server.
2. Use **Execute** to run an Access Basic function which would:
   a. **Open** a (second) channel between Access and Adept, with Access as the client and Adept as the server.
   b. Perform a **DDERequest** from Access to retrieve the query parameter, node number.
   c. Store the variable returned in a temporary table.
   d. **Close** Access to Adept (second) channel.
3. Use **Execute** to run a query, whose parameter is the value received from Adept through Access' DDERequest. Several different types of queries were attempted, and these will be discussed below.
4. Use **Request** to get the results of the query and store it in a variable in Adept to be displayed to the user.
5. **Close** channel.

(2) Phased Testing. Because this was a complex series of actions, each step was tested separately before running the entire sequence. One problem with interprogram communication is that there is no easy way to debug code, and only through trial and error and modular testing, can problems be tracked to any particular statement.

The query (step 4) was tested first. After that, the DDE request by Adept (step 5) was tested. Next, the request from Access to Adept (step 2b) was tested. Each step, and the problems encountered, is discussed in the following sections.

(a) Query Solutions. Once it was determined that Adept could easily retrieve the contents of an entire table from Access, the goal became to place all of the required information, the results of query, in a single temporary table. This table has a structure which matches the data items required by the expert system. Because this is an extremely complicated query in Access Basic, a better solution is to implement a "Make Table" Action query using QBE which is invoked from a macro.

If the Node# was placed in a temporary table, the macro containing the **OpenQuery** action could be executed by the Adept application, and the temporary "results" table would contain the correct information. Therefore step 4 was working correctly, given the query argument was correct in the table on which the query was based. This query is discussed in detail below, in Section 3.

(b) Adept Request. Given the table created in Step 4, above, the next step is to retrieve it. As was predicted, retrieval of whole table was simple, and each record could also be retrieved separately. This process is discussed in detail below in Section 3.

(c) Access Request. If the Adept application assigns the item of interest to a variable, an Access basic function was able to perform tasks which requested the value of that variable from Adept, placed it into an Access variable, and then stored that variable into another temporary table. This is also discussed in detail below, in Section 3.

(3) Results. While each one of these processes worked independently, they did not work correctly in sequence. First, Adept calls Access and

executes a macro. This macro then executes a function containing the DDE commands inside, among others, which in turn open a channel with Adept and run the same DDERequest command which had already been tested. This time, when this function was executed from Access, it did not perform correctly and Access could not get Adept to respond. This problem is also covered in more detail in Section 3 below.

3. Implementation Specifics

The Adept side of the interprogram communications is contained in a "combination node," which is the combination of two custom nodes and one display node. This combination node is shown in Figure F-1. The Access side is contained in several different structures: macros, functions, tables, and a query. The implementation of specifics on both sides, with respect to the methodology in the fourth approach discussed above, are provided below.

a. Node Number from Adept to Access

The Adept node contains a script in which the item of interest, the node number, is placed into the variable **Node**. The node number is directly related to the expert system conclusions, which are shown in the knowledge representation provided by the experts. This script also contains statements which open a DDE channel to the Access application and then execute a macro, as discussed above. The script which performs these functions is shown in Appendix F, Section C-1 (Part I).

The Access macro, executed through the DDE command by Adept, uses the command **RunCode** to execute the Access Basic function **GetNode()**. This function first opens a channel to Adept, and then requests the value of the Adept variable Node, from the Adept application using the DDERequest function. As mentioned in Section A.1.(a), above, the *item* in this request must be using syntax recognized by the server application, in this case Adept. The **GetNode()** function and DDERequest syntax is shown in Appendix F.

The temporary Table **1NX** is used to store the value of Node. Access Basic commands in **GetNode()** delete the previous value and add the new value to the 1NX table, close the table, and then terminate the channel.

b. Part Information Records from Access to Adept

First, a "Make Table" action query is used to retrieve the required information into a temporary table, and then Adept requests the information contained in that table. Ideally, the macro which performed the **RunCode** action in the previous section would then perform the **OpenQuery** action which executes the query. These tests were not performed in this manner since the program halted in the **GetNode()** function and the query would never have been executed.

(1) Part Information Query by Access. Initial tests of this function were performed by manual input of a value of Node# into the 1NX table. The function which performs this task, **MakeTable()**, executes three macros: **WarningOff**, **1NewTxQuery**, and **WarningOn**. The WarningOff/ WarningOn combination is used to suppress warning messages which occur when the query overwrites existing records. In a procedure such as this, the warning messages would either hang up communication at worst, or annoy the user unnecessarily at best. Since the user does not need to know what is happening, it is best to suppress messages after the functions generating them are well-tested.

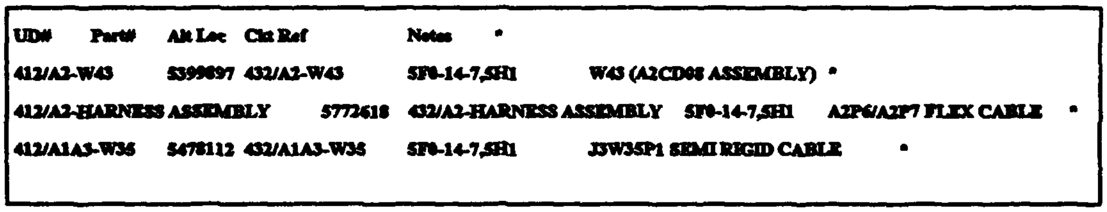
The **1NewTxQuery** macro performs one task; it performs an "OpenQuery" action on the Query **1XQ**, thus executing it. This query is an action query which is performed on the 1NX, NODE-REPL, and REPLACEMENT tables. Since 1NX contains only one value, this query finds all matching instances of REPLACEMENT by the relationship between NODE# in the first two tables and UD# in the second and third tables. In addition, since this is a "Make Table" type of action query, it places the selected records into a table which is named **NEWTX**.

(2) Information Request by Adept. The second part of the custom node retrieves the information from the table created with the macro used in the previous node. After a channel was opened, each record in the temporary table was retrieved using the DDE command **Request**. With respect to a table, Request can be used with arguments such as "FirstRow," which requests the first record in a table; "NextRow," which requests the next row in the table; "FieldNames," which returns the names of the fields in the table; and "All," which returns the entire table including field names. In this case, after the first record was requested, up to three other records were also requested. This script works for up to four replacement parts, a number selected because currently the data only shows nodes which require one, two, or three related records. Currently, expert system nodes only required three records.

This request and display are awkward at best, because programming techniques used with arrays could not be used by Adept scripts. If records could be counted, a loop could be used with a counter, vice the fixed values. Since there was no simple way to count the number of records, "flag" values were used, such that if a variable still contained the flag value, a record had not been assigned to that variable. These flag variables were used in a series of "If...then" statements to determine how many records or "Rows" had been retrieved from Access.

c. *Display of Part Information to User*

The purpose of then second Adept node and its script is to format the information retrieved from the database. If Access records stored data in fixed sized fields, it would not be necessary to change the format before it is displayed. In fact, the entire table, field headings and all, would have provided an effective and relatively quick way to display the information. Access however, stores each field in only the amount of space required by the data it contains; that is, if a field does not contain any information it is stored in the minimum amount of space, and if a field contains a long data item, then it takes up more storage space. This creates a problem when several records are listed one after another since the fields will not line up vertically, unless the data for a field just happens to be the same size. Each field is also separated by <CTRL><TAB>. An example of what this would look like is provided in Figure 21, below. In a word processor, it might be possible to get the fields to line up, if tabs were correctly placed, but the Adept display did not have that capability so a different method had to be used.



```
UD#       Part#     Alt Loc  Ckt Ref              Notes      *
412/A2-W43          5399897  432/A2-W43           5F0-14-7,SH1         W43 (A2CD08 ASSEMBLY)  *
412/A2-HARNESS ASSEMBLY        5772618  432/A2-HARNESS ASSEMBLY  5F0-14-7,SH1   A2P6/A2P7 FLEX CABLE  *
412/A1A3-W36        5478112  432/A1A3-W35         5F0-14-7,SH1         J3W35P1 SEMI RIGID CABLE     *
```

Figure 21 - Records Retrieved[3]

First, the "*" character was added in a field at the end of each record as a record end marker. This allowed the length of each record to be determined by finding the "*" character and using the "Length" function. The record was also put into a text variable using the "GetSubText" function. Next, the value of each field is determined by finding all characters before the first <CTRL><TAB> combination. "FindText" finds the length of this field, and then "GetSubText" gets the contents of the field by using the arguments (record variable, start (offset), and ending (offset)). After the first field in the record variable is determined, "GetSubText" is used to get everything remaining, and

[3] This display is an example of what the display would look like if the entire table were retrieved, except the * field would not be required.

assigns this to the record variable. At this time, the process repeats, since now the second field in the record is the first field in this variable. This continues until the values of all five fields for each record is placed into a separate variable. Since these fields are still of various lengths the best way to display them is in a vertical (left justified) manner. This is shown in Figure F-2, Appendix F. The script used for this is shown in Section C-1 (Part II), Appendix F.

This above methodology is extremely contrived and extremely inflexible. In addition, the performance is very slow when there are several records.

C. FINAL RESULTS

This section primarily applies to the fourth approach, but some of the "lessons learned" came from other approaches as well. After abortive attempts to execute an Access function, it was discovered that DDE can not directly use several commands, such as *RunCode. Macros can not* be *used directly,* because *the* DDE*Request function can* only be executed from Access Basic procedures. Therefore, Adept can execute a macro, which uses the RunCode action to execute an Access Basic procedure. In the fourth attempt, this procedure contains statements which perform a DDERequest to Adept, requesting the value of the variable Node, which contains the node number about which the user requires information.

If this procedure is executed from Access, there was no problem; Access could indeed retrieve the correct Node value (assigned by an Adept node) and place it into an Access variable, which could then be put into a table. The code required to perform this task is provided in Appendix F, Section C. It could not, however, be executed from Adept. The Adept application performed a DDE execution of the 1TestDDENodeReq macro, which instructed the Access application to make a request. When that request was made, the Adept application did not respond back to Access, and the process "timed out."

It appears that once an Adept application makes a DDE request from an application, it ceases to listen for other requests. At this time there is no explanation for this problem, and Softsell Product Support personnel are looking into a solution or a way to work around this anomaly.

One other possible solution was briefly explored, by using an alternate Adept procedure as a third-party. There were two ways to approach this, and both were tried. One was to have the MK92 FCS MAES (or the test application) call a second application running in the background. The main application would tell the second application to "call" the Access application, which would then request the value of Node from the main application. This met with similar results as the first trial, that is, either one application did not recognize the other or the process did not execute properly and "timed-out." One other possible solution would be for the Main Adept application to send the Node number to the secondary Adept application, and have Access request the information from the secondary application, which should be "listening" for requests since there were no outstanding requests of its own. Initial trials with this methodology did not produce successful results since attempts to put Node into a static (one not being operated on by a user) procedure did not work.

Part of this research was successful, since establishing certain types of one-way communications was fairly effective. Possible uses for DDE should be explored in future applications, however DDE can not be relied on to solve all interface problems, since there appears to be no consistent implementation of DDE even within Microsoft's own programs let alone with third-party vendors. OLE 2.0 may be more capable and combine some of the benefits of both DDE and OLE to provide an interface between two programs, not just for display of objects, but for actual use and manipulation of them by the client application in addition to allowing changes from the server application. Until

OLE objects can be recognized as text strings, vice as something else, it will be difficult to use them in databases for query purposes when the user input will generally still be in the form of text.

Other possible solutions to the Adept - Access application interface problems are discusse in Chapter VI.

VI. LESSONS LEARNED AND CONCLUSIONS

This project established the viability of developing a separate database that interfaces with the MK92 FCS MAES and can be used as a stand alone application for tracking and maintaining Parts and other information. In this chapter, the feedback provided by the project sponsors concerning the functionality of the prototype database application is discussed. In addition, requirements for follow-on work for both the interface between the MK92 FCS MAES and the database application, and the stand-alone run-time and administration versions of the application are presented.

A. SPONSOR FEEDBACK ABOUT PROTOTYPE

Feedback from sponsors concerning the prototype database application fall under four categories: Expert System Interface, Parts Supply Information, Usage, and Database Maintenance. Each or these categories are discussed below.

1. Expert System Interface

a. Acknowledged Communication Problems

As previously mentioned, there are problems with the using DDE for interprogram communications with the current versions of Adept and Access. The sponsors acknowledged these problems, as well as the probability that other possible solutions may not provide a seamless interface as was hoped. The main problem with alternative solutions is that it is unlikely that the interface between the two programs will be transparent to the user, and may actually require user input for information retrieval instead of being totally "automated."

b. Performance Issue

Due to the slow performance of Adept in displaying an Access table in an effective manner, further pursuit of the Adept-Access DDE interface may not be of any value. If another DBMS software program were known to be more capable of performing the required DDE functions and could store data in fixed length fields, the overall performance would probably be acceptable. Since, however, the DDE performance of other DBMS software is still unknown, it is more beneficial to look at other interface methods.

2. Parts Supply Information Subsystem

While both sponsors and shipboard technicians indicated that parts information is a highly desired feature, some record format changes would be more beneficial. Technicians do not necessarily have a supply of spare parts, therefore keeping track of inventory is not really required. The sponsors however, felt they could use an inventory system at their site. This system would be used to keep track of parts they receive from decommissioned ships and would be distributing to other ships at no cost, since the parts are not in the actual supply system. Slight changes to existing forms and tables could be made to provide this capability, as well as perhaps an additional table to track requests received from ships Changes to the forms, reports, and processes available to the technicians will not be significant and will merely simplify those which already exist.

3. DB Maintenance Subsystem

While there is still discussion concerning the functionality of this system, and what items of data users would be able to change, the need for users to maintain at least some of the data is a desirable feature. When this feature is implemented, password protection should provide security within the application to prevent inadvertent damage.

4. Usage Subsystem

While there is value in obtaining usage data from the users for future use by management, this may be outside the scope of the application's actual requirements.

5. Administrator and User Interfaces

The separation of Administrator and User menus and functions was recognized as being a good idea. Non-programmers may be doing the bulk of data store maintenance with respect to this system, and may be responsible for updating data stores and providing updates to the user. Providing extra capabilities for the administrator will facilitate overall system maintenance while keeping the user interface as simple as possible.

B. DIRECTION OF FOLLOW-ON WORK

Follow-on work should continue the investigation of a viable method for accessing the database from the expert system. It would also implement changes suggested by the program manager and complete the implementation of several maintenance related functions which were not completed in this thesis.

1. Expert System - Database Interface

One possible way to provide an expert system - database interface is to provide the user with the capability to use the database directly to retrieve the required information. This might be achieved by using an icon activated program which first opens Access to the required screen through the use of a macro, and then begins the Adept program. In this approach, both Adept and Access and applications will be running simultaneously. The user, however, will only be seeing the Adept screens, since they generally will be in the foreground. In order to retrieve information about a required replacement or part, the user could invoke a process to provide direct manipulation of the database.

If this process could "minimize" the Adept window, the Access wiidow would then be visible to the user. At this time, the user would have to make an entry in order to retrieve the correct information. This would accomplish "manually" what this thesis tried to automate through the use of DDE. Perhaps Node# would still be a good argument for the query even though it is not particularly meaningful to the technician, because it is short and fairly easy to remember (vice UD#s which are difficult to remember). Using Node#s would allow the current database structure to be used in retrieving all applicable information about Parts and Replacements, instead of about one UD# at a time.

The display form would use a command button to invoke a procedure to maximize Adept and put Access back in the background until needed.

2. Database Application Functionality

As mentioned above, the functionality of each of the different subsystems requires a number of changes. Follow-on work will complete the planned functionality, which still remains a requirement, particularly in the area of maintenance, and will make other changes as requested by the sponsor. The sponsors have expressed an interest in adding the capability to track excess parts which are distributed to ships. This may require additional tables tc track the requests from particular ships for particular parts.

3. Database Application Environment

Once the application is complete, the database environment has to be made secure. Generally, the run-time environment does not allow the user access to objects; however, a user could inadvertently halt processes or cause problems if care is not taken to fix some of the current loopholes. Error-trapping routines need to be added to ensure there are no abrupt terminations of the application. Customized help files may be added, and would prove useful. (Perschke and Liczbanski, pp. 295-296)

4. Documentation

Documentation for both the user and the administrator should be completed. This thesis provides the foundation for some of the system documentation, but updates will be required as the system changes. In addition, this thesis does not provide a User's Manual, since this is only the first generation prototype of this system.

C. LESSONS LEARNED

1. Database Selection

Using a database which supported fixed fields would have solved some of the problems and performance issues associated with the display of information retrieved from the database by Adept. This may be a moot point, however, if no database can accept a query parameter passed from Adept.

2. Application Development

More interface with the user would have been helpful. A great deal of time in developing this system was spent in researching the interface between the expert system and the database and in attempts to get it working, rather than specifying the functionality of the system precisely. It should be pointed out that even if the nature of some of the prototype's processes and forms change, most of the existing objects and processes could be used with little modification.

D. CONCLUSION

This thesis proved the viability of using interprogram communicationbetween a database and an expert system to enhance the functionality of the expert system and to help the users save time spent looking up related information manually. Database vendors are working to standardize interprogram communications as users demand the capability. New releases of both Adept and Access are expected to provide capabilities which may solve many of the existing problems. Insights gained in this thesis can be used as a model to develop approaches for other attempts at interprogram communications.

This thesis is a typical example of the problems and issues faced today in the software development environment when trying to get different vendor's software to communicate with one another. This sort of interprogram communications has been one of the most difficult problems to solve, and will continue to be so, as long as adherence to any sort of communication protocol standards is not demanded by the users and/or application developers.

APPENDIX A - ENTITIES, ATTRIBUTES, AND ENTITY-RELATIONSHIP DIAGRAM

A. ENTITY DEFINITIONS AND ATTRIBUTES

NODE Entity

NODE#;	node-number
MODULE REF;	module-reference

REPLACEMENT Entity

UD#;	circuit-card-location-ref
PART#;	part-number
ALT LOC;	alternate-location
NOTES;	replacement-notes

PART Entity

PART#;	part-number
NSN;	stock-number
PRICE;	part-price
ALLOWANCE;	part-allowance
PARTS ON HAND;	parts-on-hand
PARTS ON ORDER;	parts-on-order

NODE-REPL Entity

NODE#;	node-number
UD#;	circuit-card-location-ref
CKT REF;	circuit-reference

USAGE Entity

USE#;	usage-number
DATE;	usage-date
PART REPLACED;	part-replaced
NOTES;	usage-notes
(Others TBD)	

__ denotes key attribute of entity

B. ATTRIBUTE DEFINITIONS

alternate-location
Text 25
Location (UD#) where same part may be found in system

circuit-card-location-ref
Text 28
UD# or other reference information for replacement part

circuit-reference
Text 27
Documentation reference

module-reference
Text 20
Reference number on knowledge diagrams

node-number
Text 6, Mask N####, # any digit
Artificially generated number used to track nodes

part-allowance
Integer
Number of a part allowed on board (COSAL or other requirement)

part-number
Text 20
Reference number for part

part-replaced
Boolean
Set if part replaced

parts-on-hand
Integer
Number of a part on board

parts-on-order
Integer
Number of a part on order

price

Currency

Price of part

replacement-notes

Text 32

Information for user

stock-number

Text 15

National Stock Number (NSN) for part

usage-date

Date

Date of usage

usage-notes

Memo

Information about usage

usage-number

Text 6, Mask U######, # any digit

Artificial number for usage occurrence

C. ENTITY-RELATIONSHIP (E-R) DIAGRAM

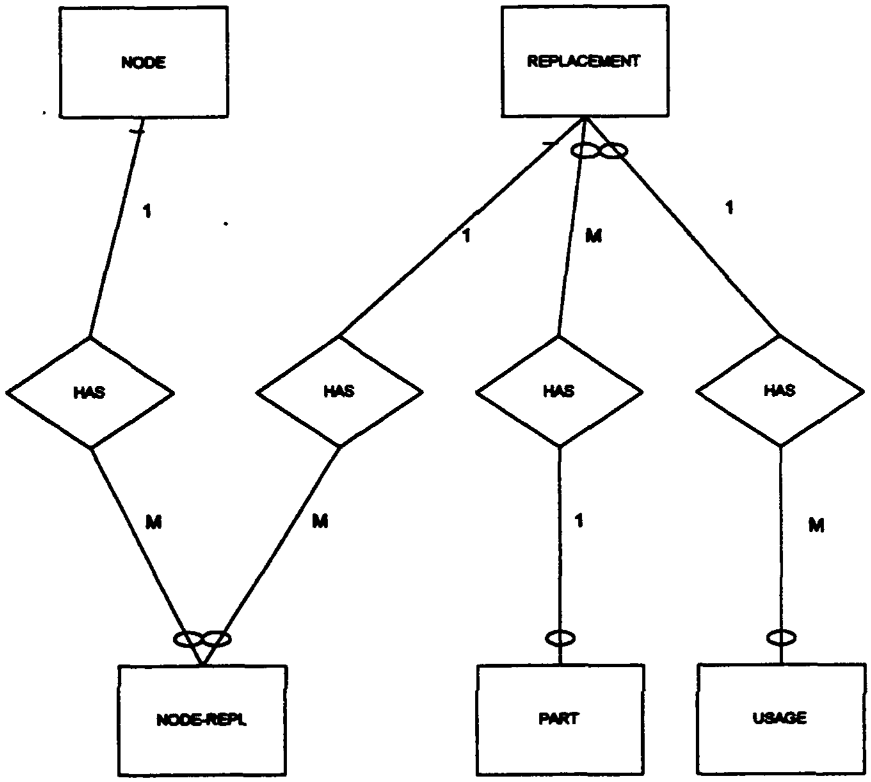


FIGURE A-1

APPENDIX B. DECOMPOSITION AND DATA FLOW DIAGRAMS

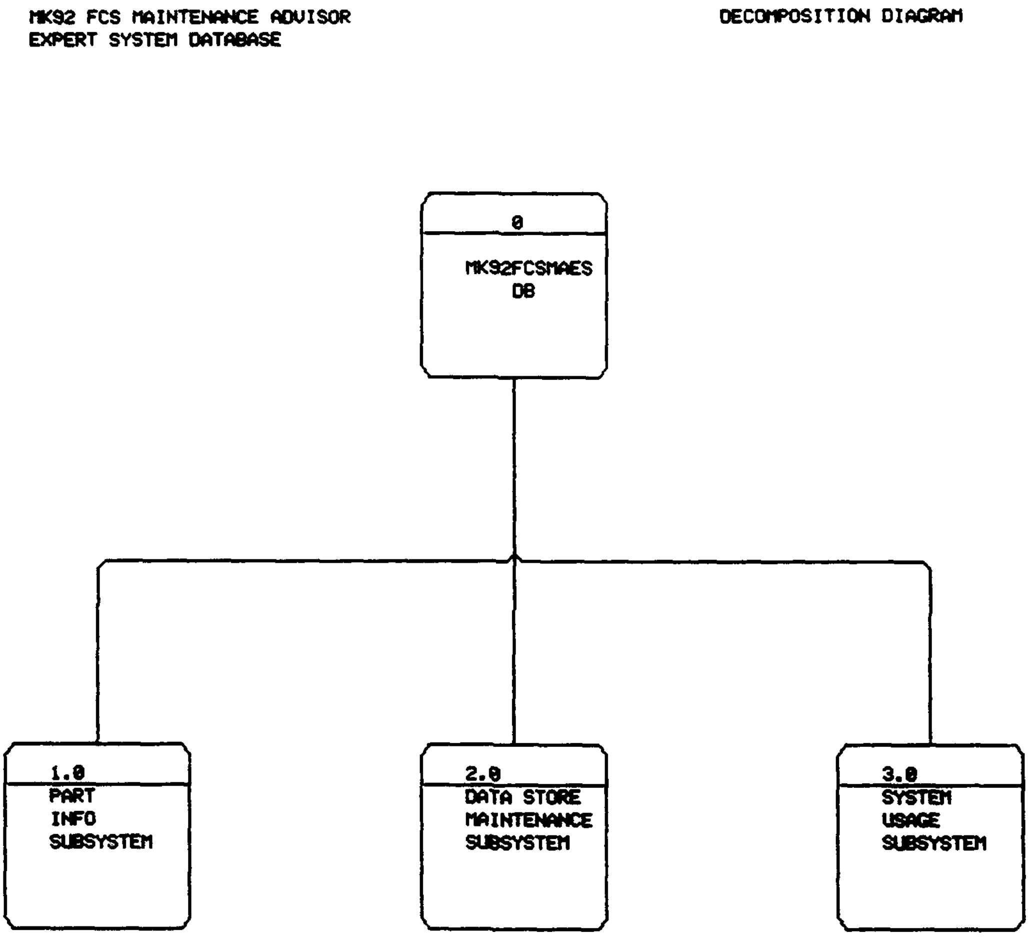


Figure B-1 - MK92 FCS MAES DB Decomposition Diagram

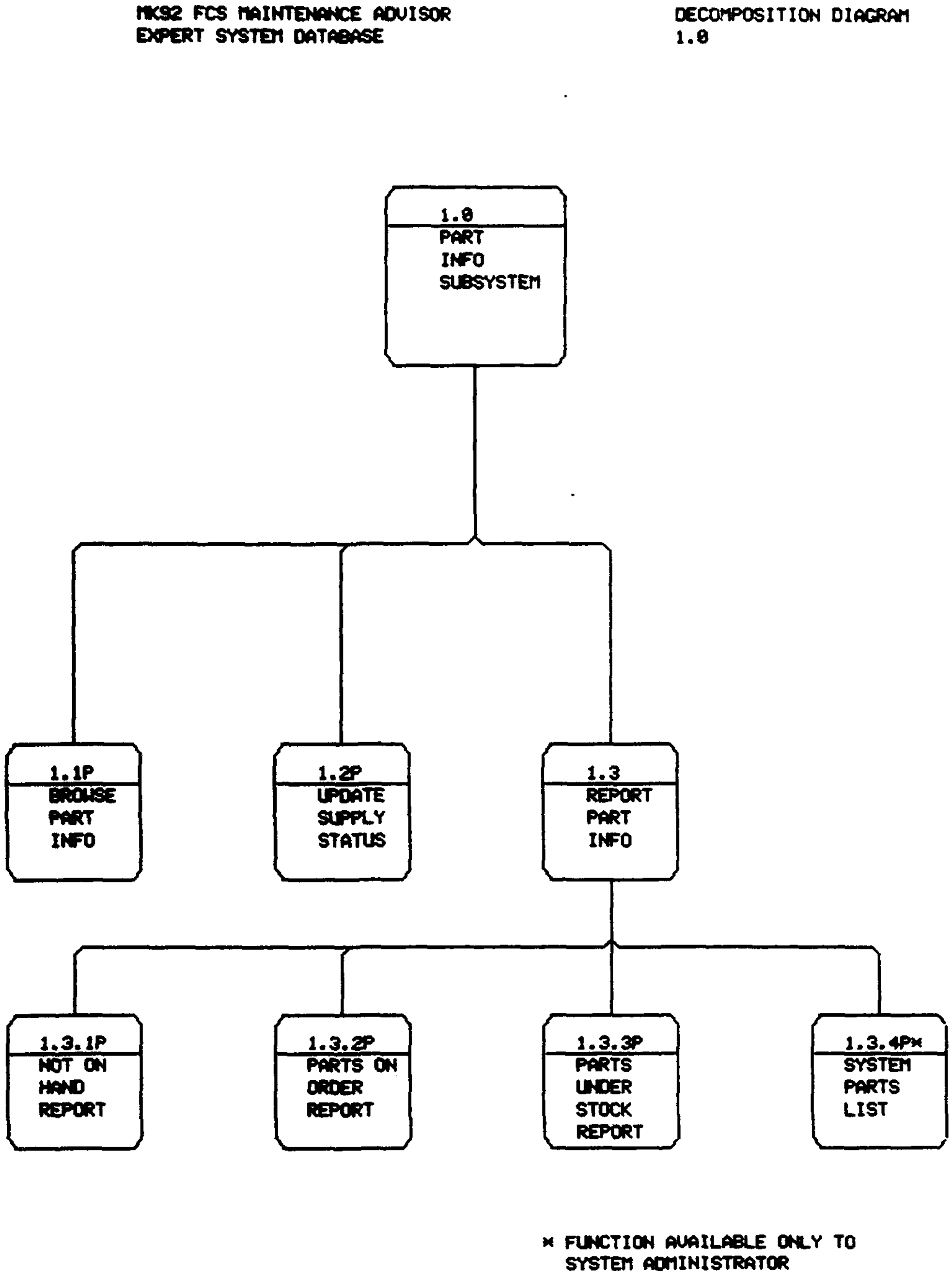

MK92 FCS MAINTENANCE ADVISOR
EXPERT SYSTEM DATABASE
DECOMPOSITION DIAGRAM
1.0
1.0 PART INFO SUBSYSTEM
1.1P BROWSE PART INFO
1.2P UPDATE SUPPLY STATUS
1.3 REPORT PART INFO
1.3.1P NOT ON HAND REPORT
1.3.2P PARTS ON ORDER REPORT
1.3.3P PARTS UNDER STOCK REPORT
1.3.4P* SYSTEM PARTS LIST
* FUNCTION AVAILABLE ONLY TO SYSTEM ADMINISTRATOR

Figure B-2 - MK92 FCS MAES DB Decomposition Diagram [1.0]

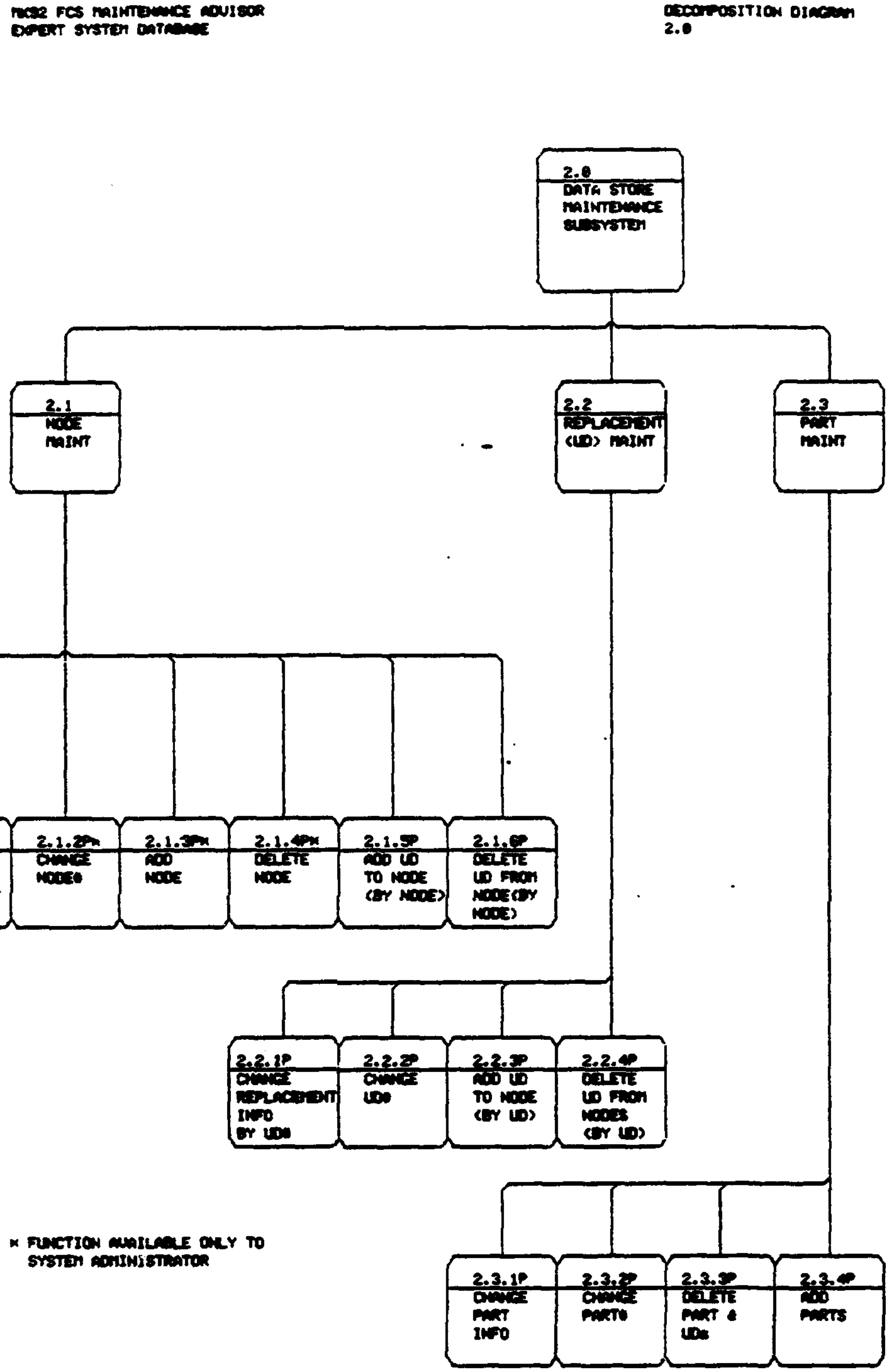


Figure B-3 - MK92 FCS MAES DB Decomposition Diagram [2.0]

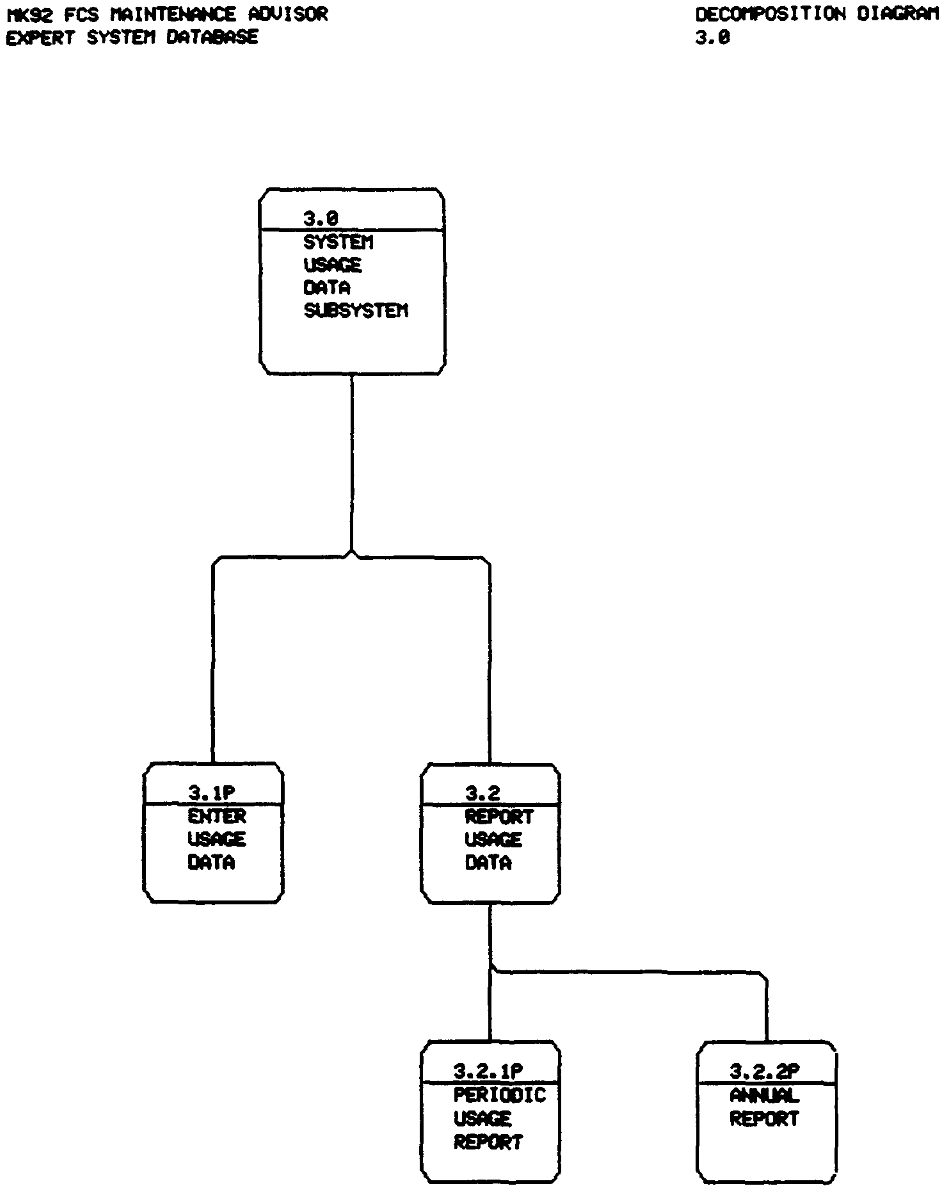


Figure B-4 - MK92 FCS MAES DB Decomposition Diagram [3.0]

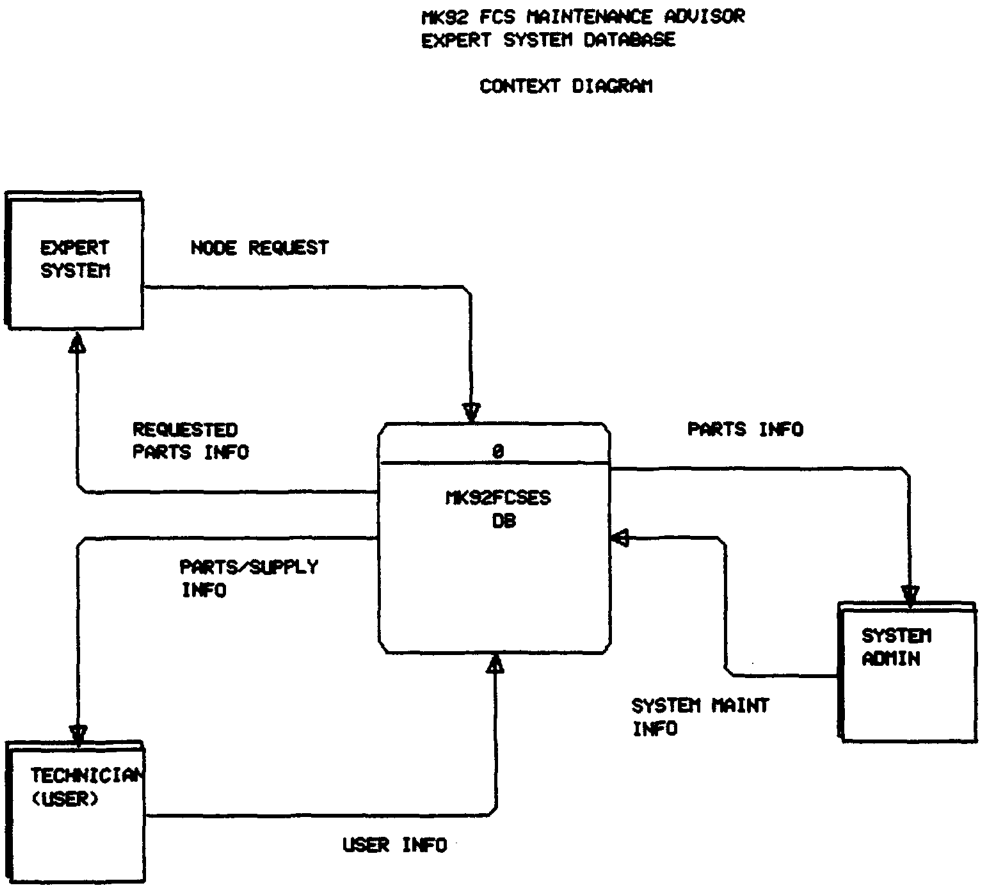


Figure B-5 - MK92 FCS MAES DB Context Diagram

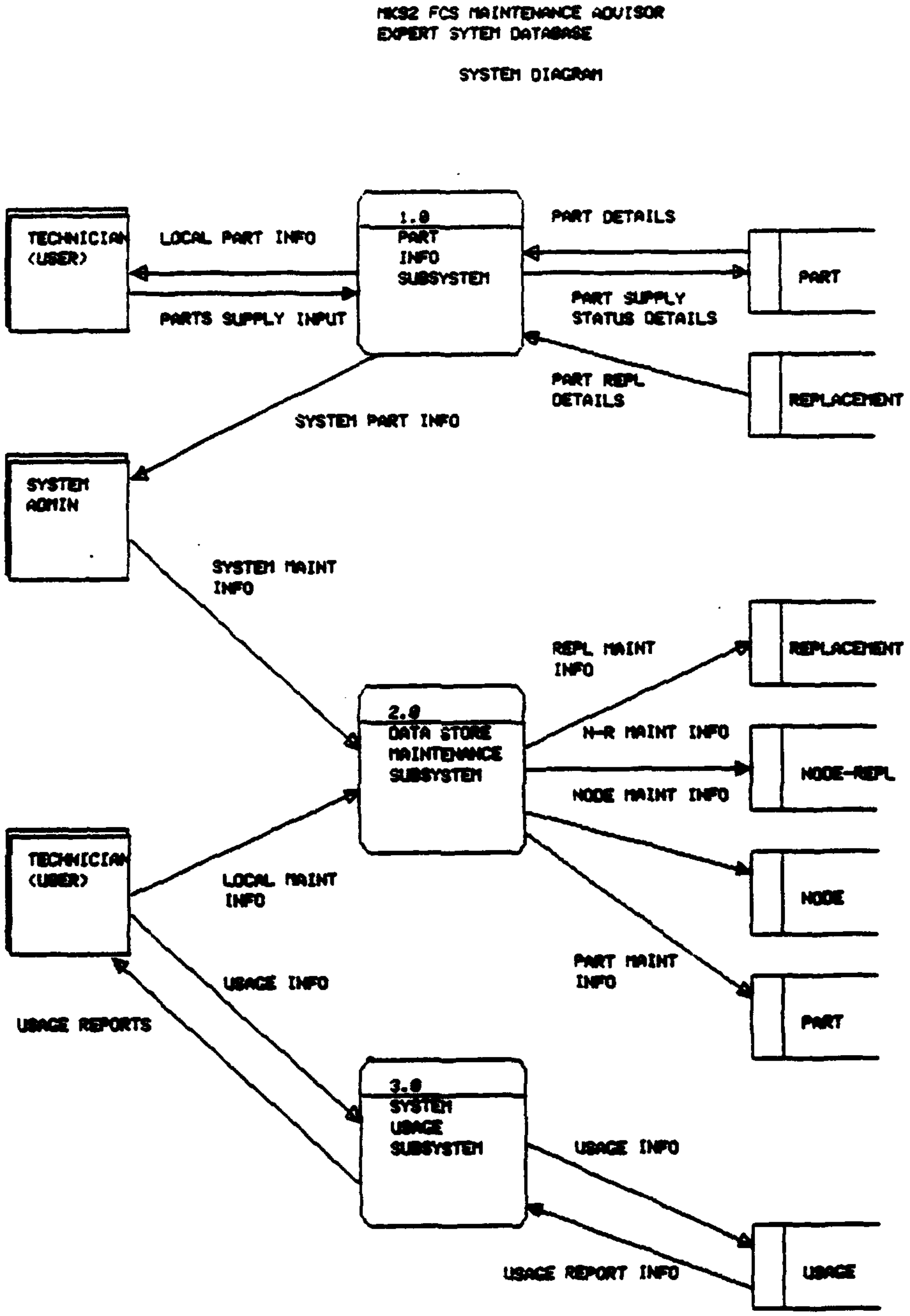


Figure B-6 - MK92 FCS MAES DB System Diagram

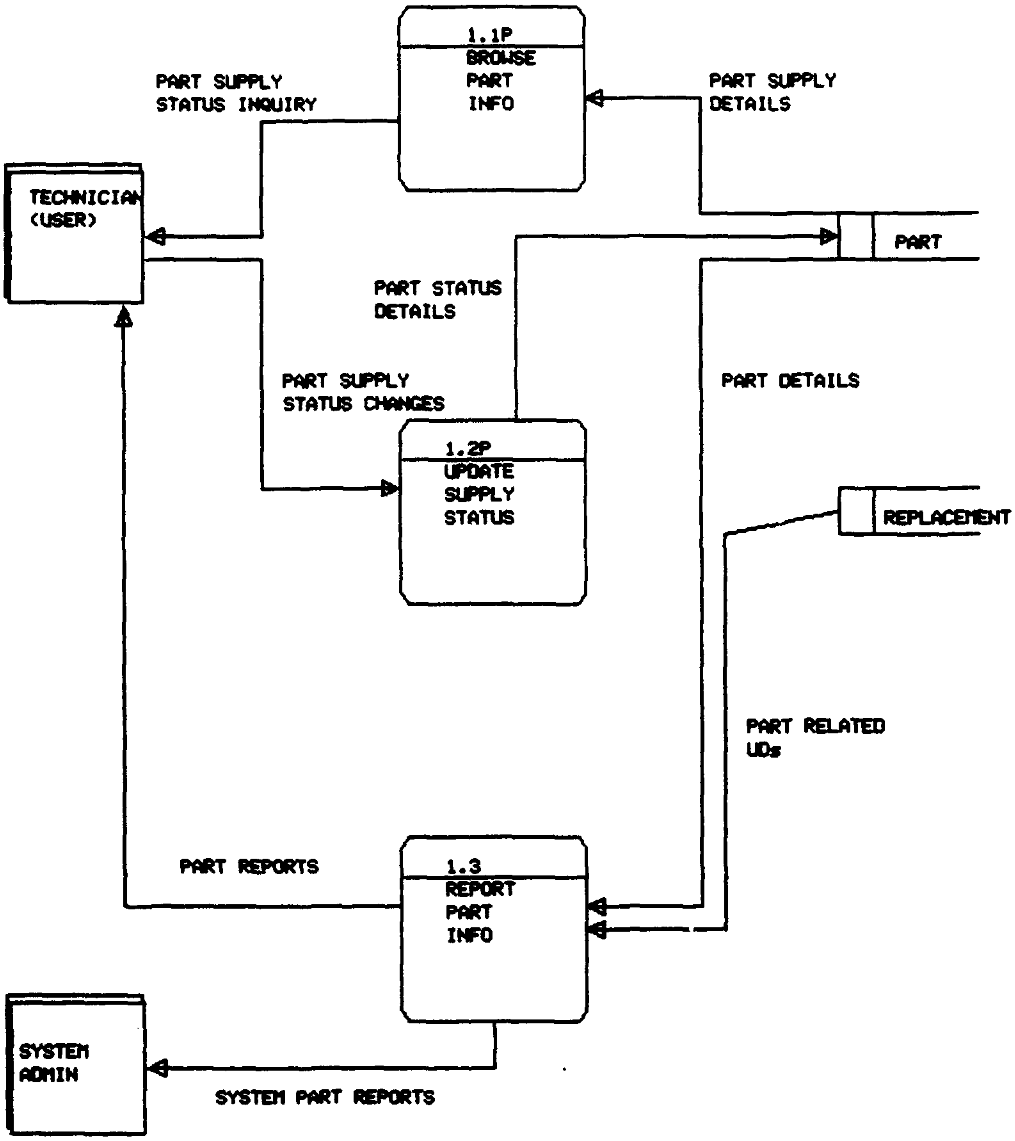


Figure B-7 - DFD: Part Info Subsystem [1.0]

DFD - REPORT PART INFO [1.3]

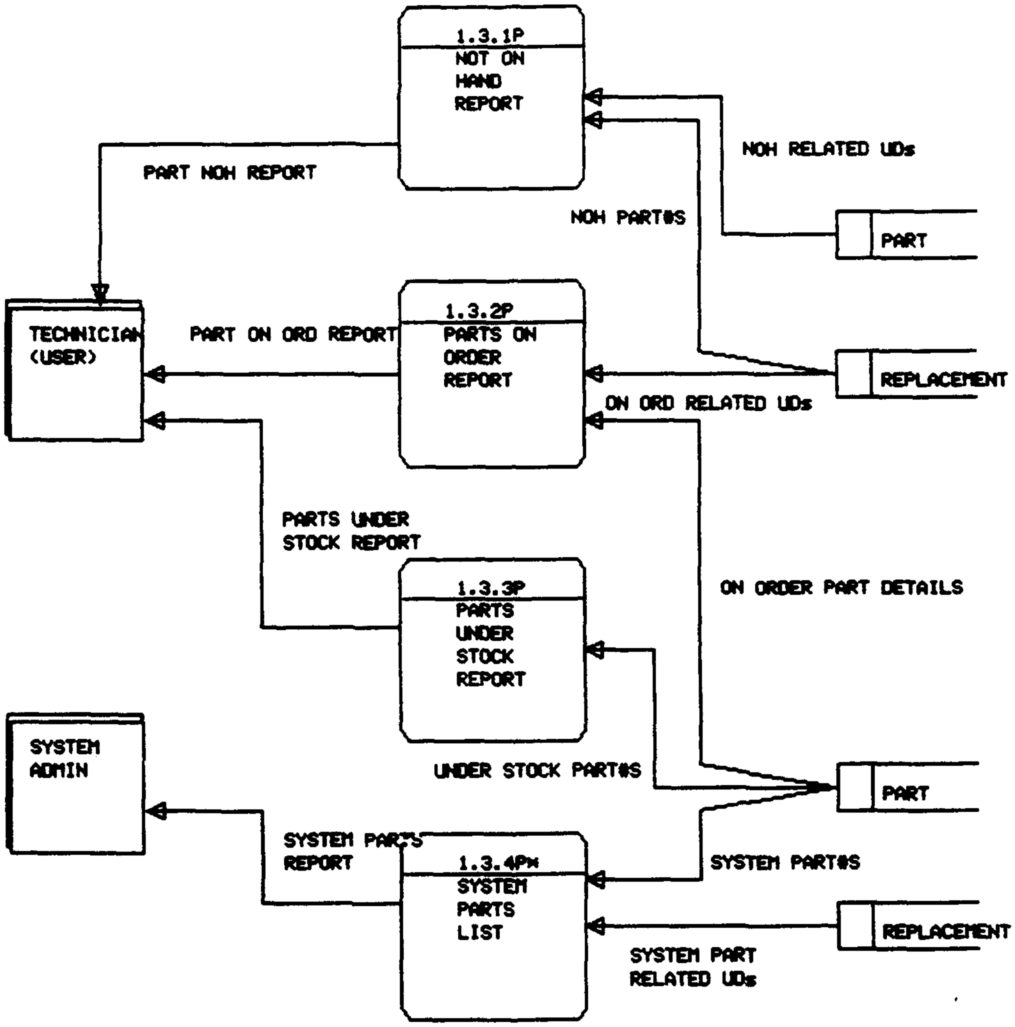


Figure B-8 - DFD: Report Part Info [1.3]

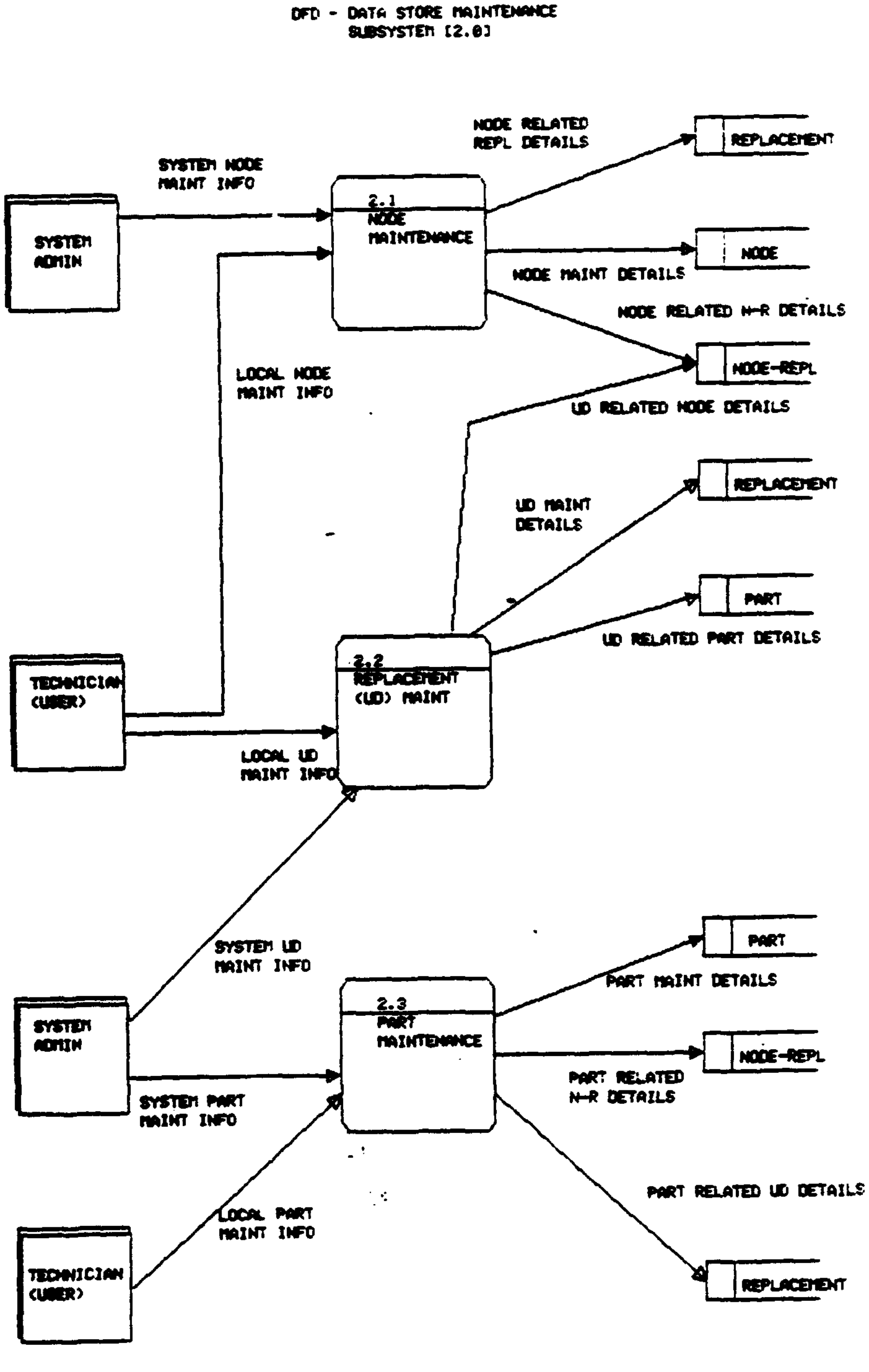


Figure B-9 - DFD: Data Store Maintenance Subsystem [2.0]

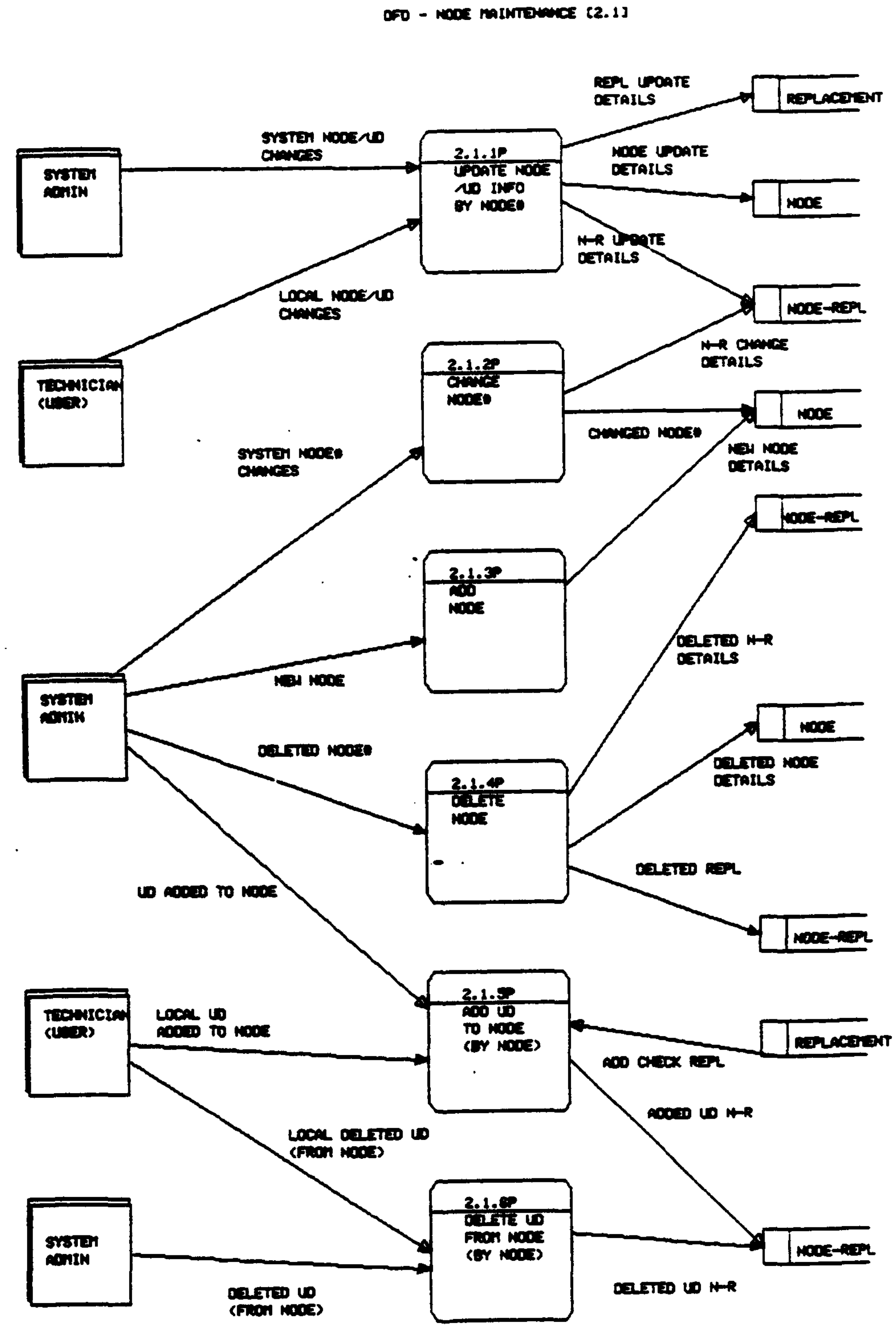


Figure B-10 - DFD: Node Maintenance [2.1]

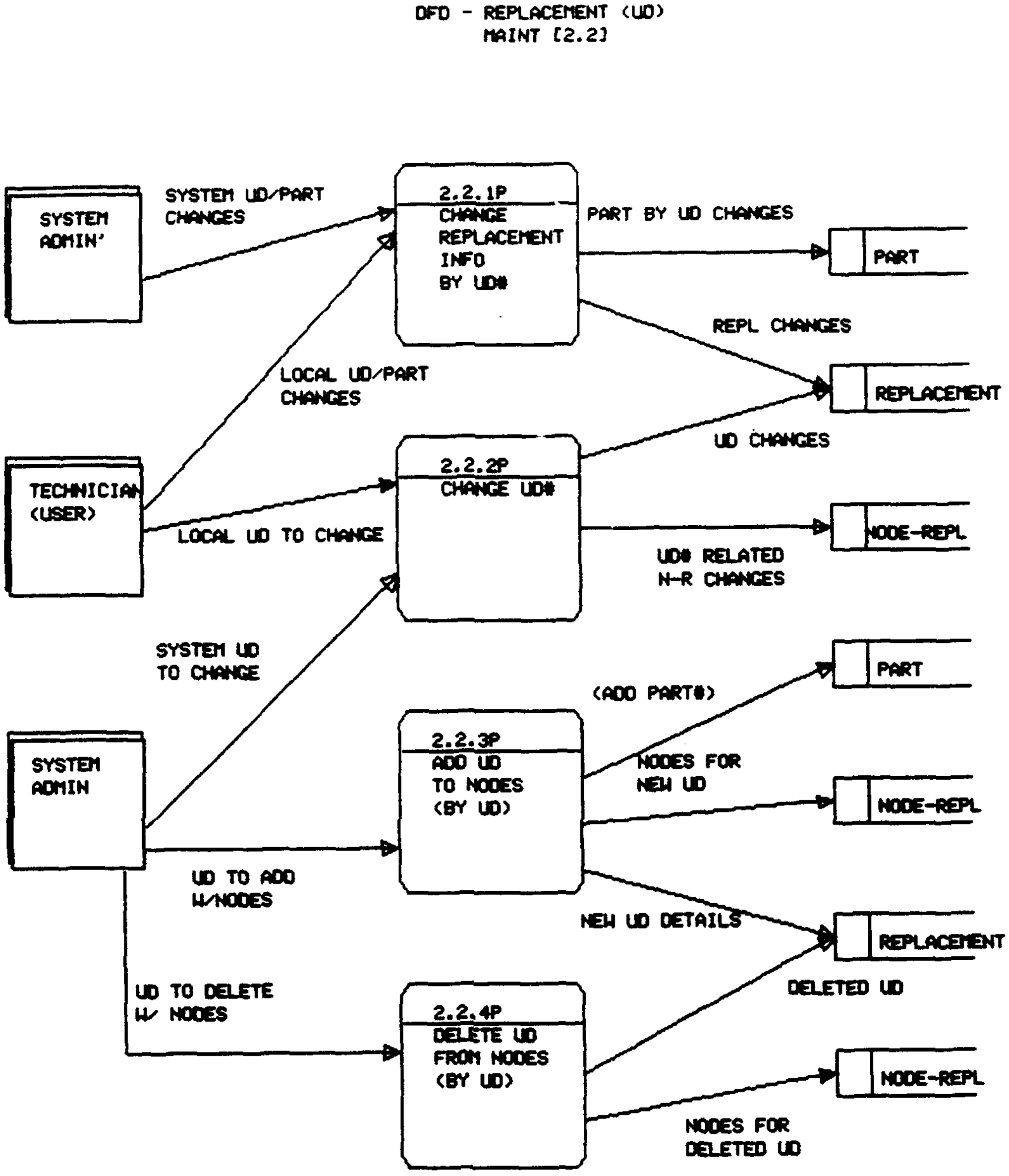


Figure B-11 - DFD: Replacement (UD) Maintenance [2.2]

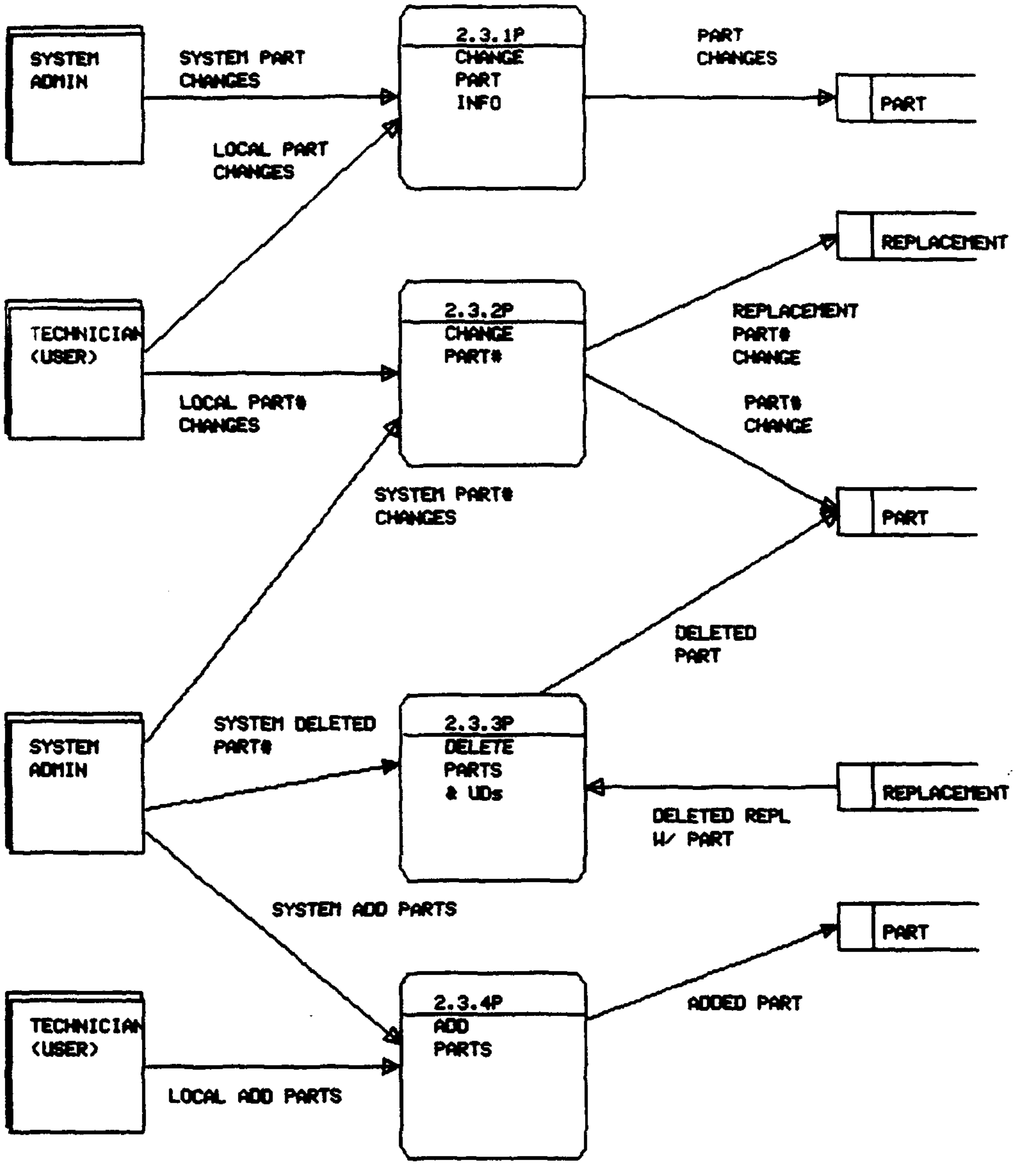


Figure B-12 - DFD: Part Maintenance [2.3]

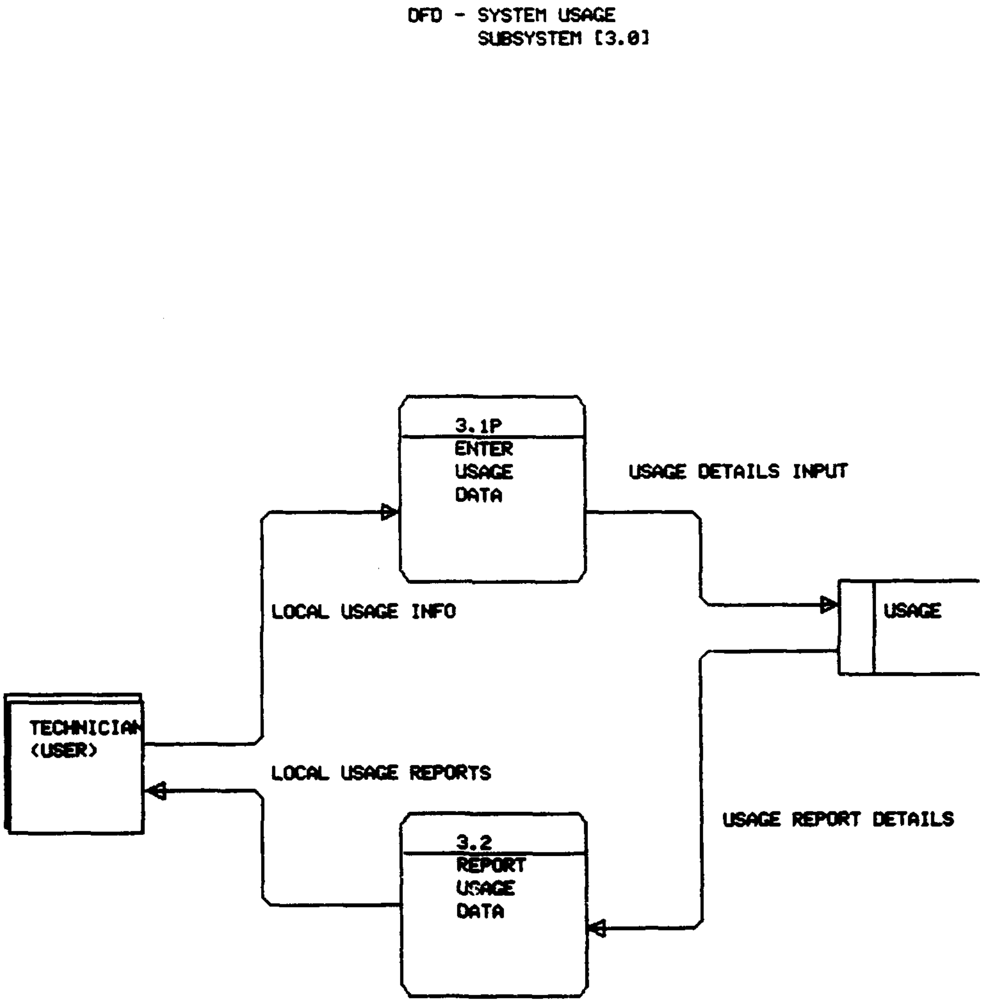


Figure B-13 - DFD: System Usage Subsystem [3.0]

DFD - REPORT USAGE DATA [3.2]

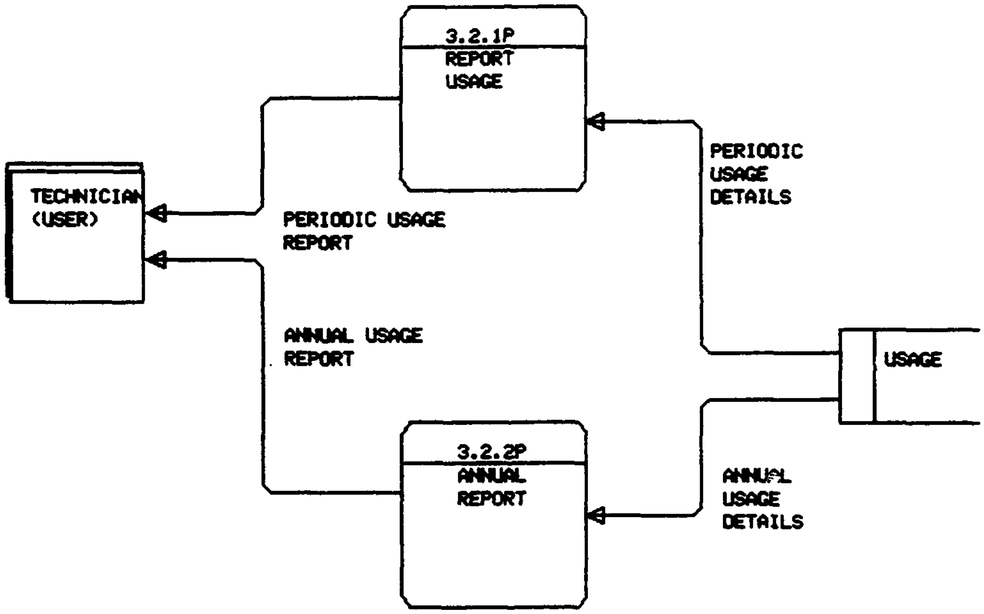


Figure B-14 - DFD: Report Usage Data [3.2]

APPENDIX C - MENU HIERARCHY AND SCREENS

A. MENU HIERARCHY

References to Decomposition Diagram are provided in { }

USER MENU

- Part Information (Part Info) Submenu {1.0}
  - Browse Part Supply Info (Supply) {1.1P}
  - Update Part Supply Status (Update) {1.2P}
  - Part Information Reports (Report) Submenu {1.3}
    - Report Parts Not On Hand (Not on Hand) {1.3.1P}
    - Report Parts On Order (On Order) {1.3.2P}
    - Report Parts Under Stock (Under Stock) {1.3.3P}
- Usage History (Usage) Submenu {3.0}
  - Enter Usage Data (Enter) {3.1P}
  - Periodic Usage Report (Periodic) {3.2.1P}
  - Annual Report (Annual) {3.2.2P}
- DB Maintenance (DB) Submenu {2.0}
  - Update Circuit Card Info (Ckt Card) Submenu <2A>
    - Select By Node# (By Node#) {2.1.1P}
    - Select By UD# (By UD#) {2.2.1P}
      - [Change UD#] {2.2.2P}
  - Update UDs (Submenu) <2B>
    - Add UDs to Nodes (Add) Submenu <2B-1>
      - Select by UD# [Add New UD]{2.2.3P}
      - Select by Node# [Add New UD]{2.1.5P}
    - Delete UDs from Nodes (Delete) Submenu <2B-2>
      - Select by UD# {2.2.4P}
      - Select by Node# {2.1.6P}
  - Update Part Info (Part) Submenu <2C>
    - Update Info by Part# {2.3.1P}
      - [Change Part#] {2.3.2P}
    - Add Parts {2.3.4P}
    - Delete Parts {2.3.3P}

ADMINISTRATION MENU

- Part Information (Part Info) Submenu {1.0}
  - Browse Part Supply Info (Supply) {1.1P}
  - Update Part Supply Status (Update) {1.2P}

    - Part Information Reports (Report) Submenu {1.3}
        - Systems Part List (Parts List) {1.3.4P)
- DB Maintenance (DB) Submenu {2.0}
    - Update Circuit Card Info (Ckt Card) Submenu [2A]
        - Select By Node# (By Node#) {2.1.1P}
            - [Change Node#] {2.1.2P}
        - Select By UD# (By UD#) {2.2.1P}
            - [Change UD#] {2.2.2P}
    - Update UDs (Submenu) [2B]
        - Add UDs to Nodes (Add) Submenu [2B-1]
            - Select by UD# [Add New UD] {2.2.3P}
            - Select by Node# [Add New UD] {2.1.5P}
        - Delete UDs from Nodes (Delete) Submenu [2B-2]
            - Select by UD# {2.2.4P}
            - Select by Node# {2.1.6P}
    - Update Part Info (Part) Submenu [2C]
        - Update Info by Part# {2.3.1P}
            - [Change Part#] {2.3.2P}
        - Add Parts {2.3.4P}
        - Delete Parts {2.3.3P}
    - Node Info [2D]
        - Add Nodes {2.1.3P}
        - Delete Nodes {2.1.4P}
- Usage History (Usage) Submenu {3.0}
    - To Be Determined

[] Additional functions provided on screen, vice on menu
<> Combination menus referenced in Process Logic section (Appendix D, Section A)

B. MENU SCREENS

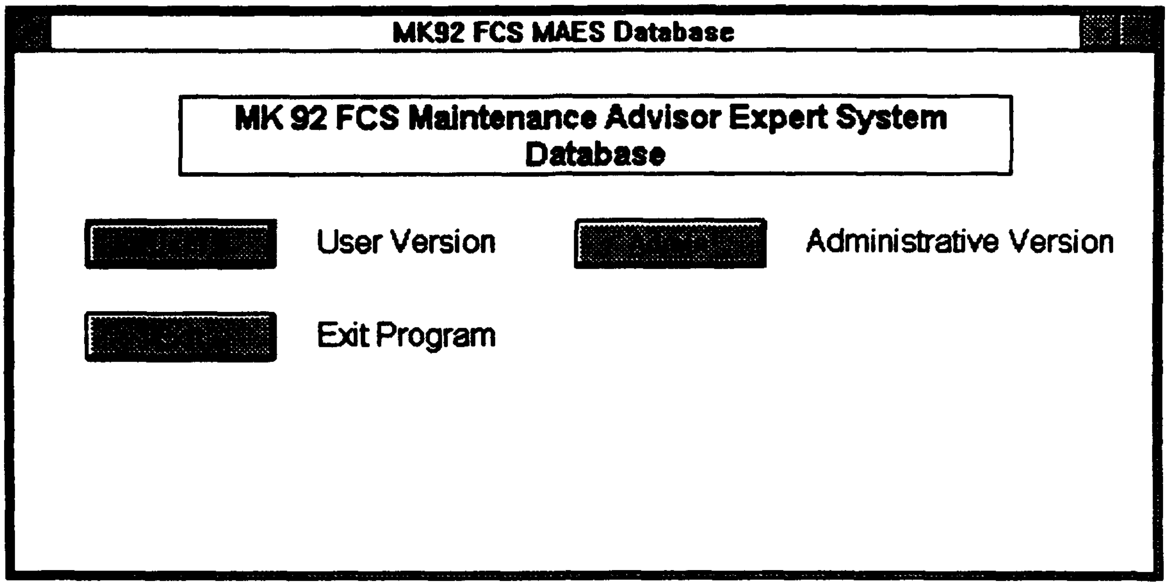


Figure C-1 - Opening Menu

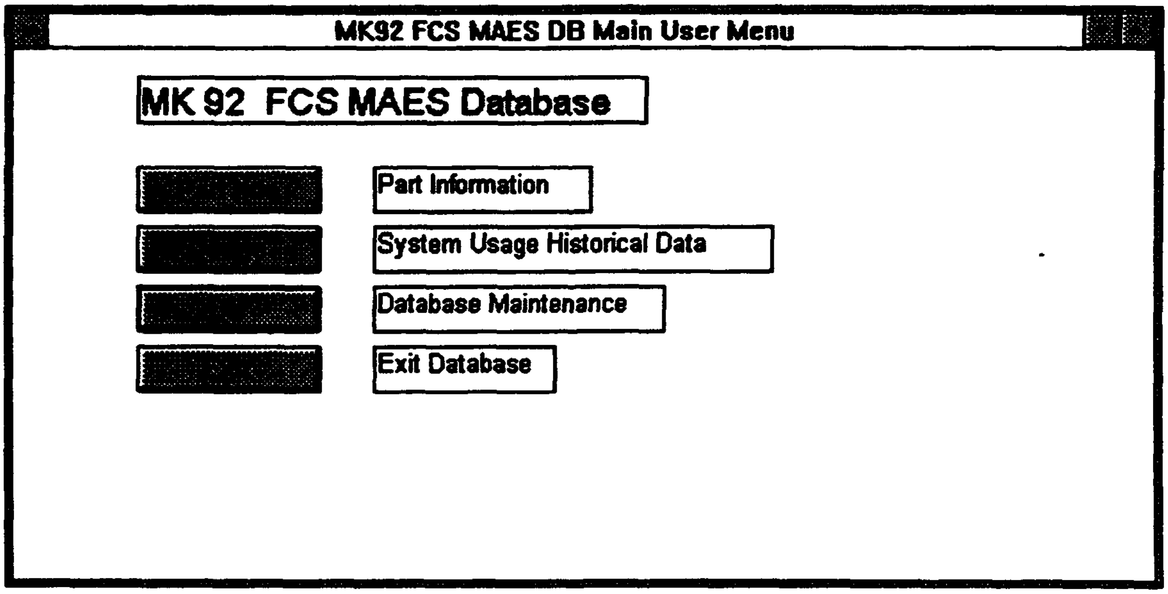


Figure C-2 - User Menu Screen

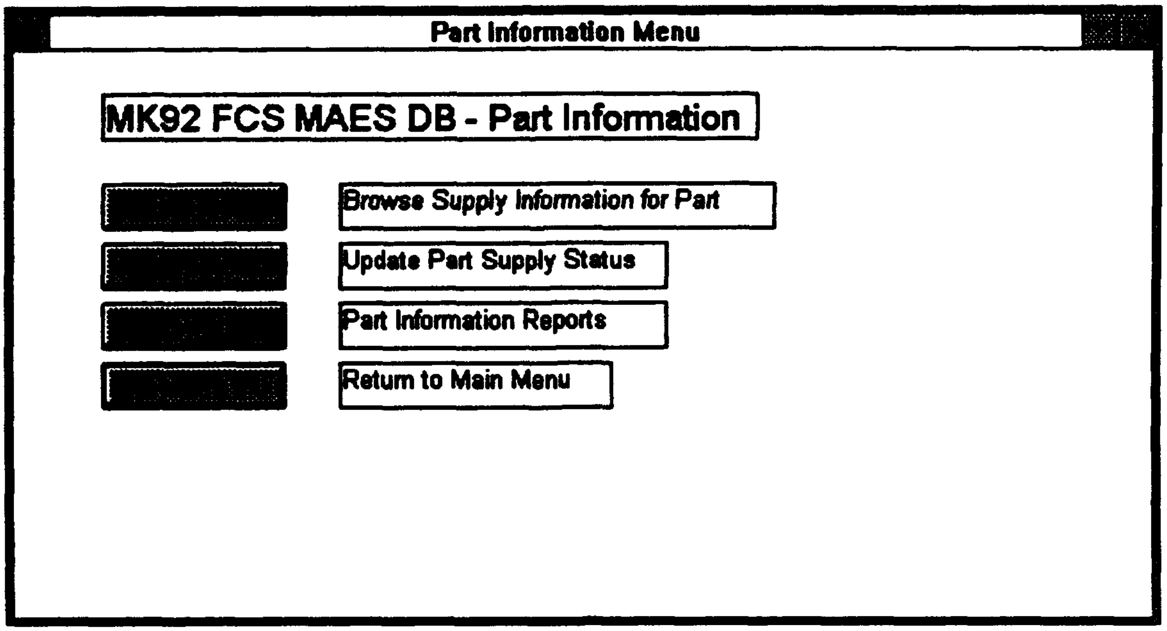

Part Information Menu
MK92 FCS MAES DB - Part Information
Browse Supply Information for Part
Update Part Supply Status
Part Information Reports
Return to Main Menu

Figure C-3 - Part Information Submenu

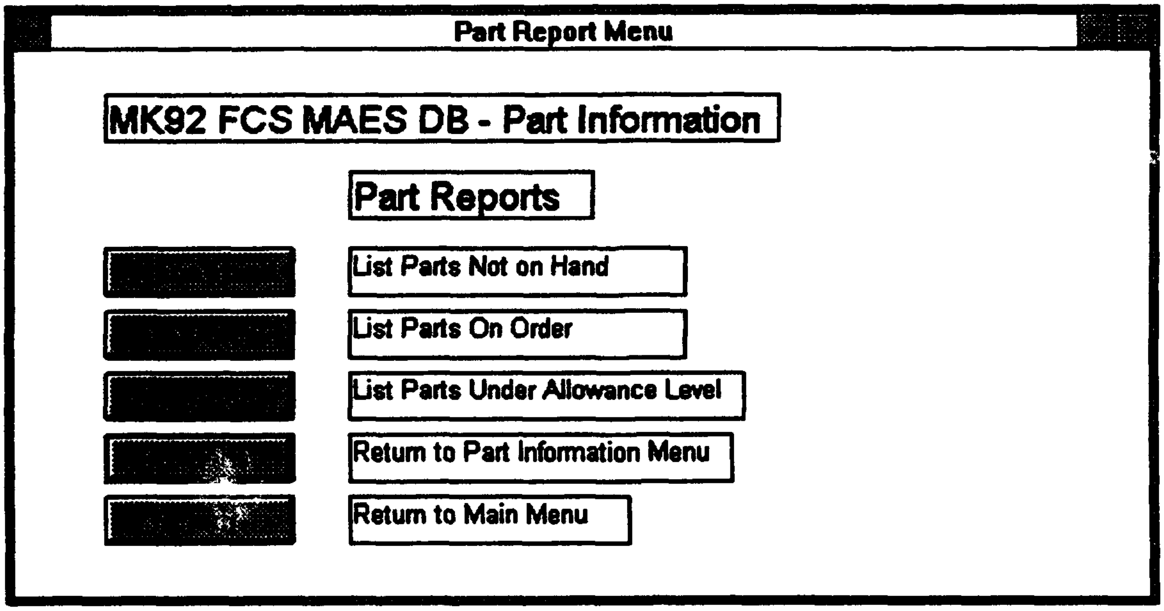

Part Report Menu
MK92 FCS MAES DB - Part Information
Part Reports
List Parts Not on Hand
List Parts On Order
List Parts Under Allowance Level
Return to Part Information Menu
Return to Main Menu

Figure C-4 - Part Reports Submenu

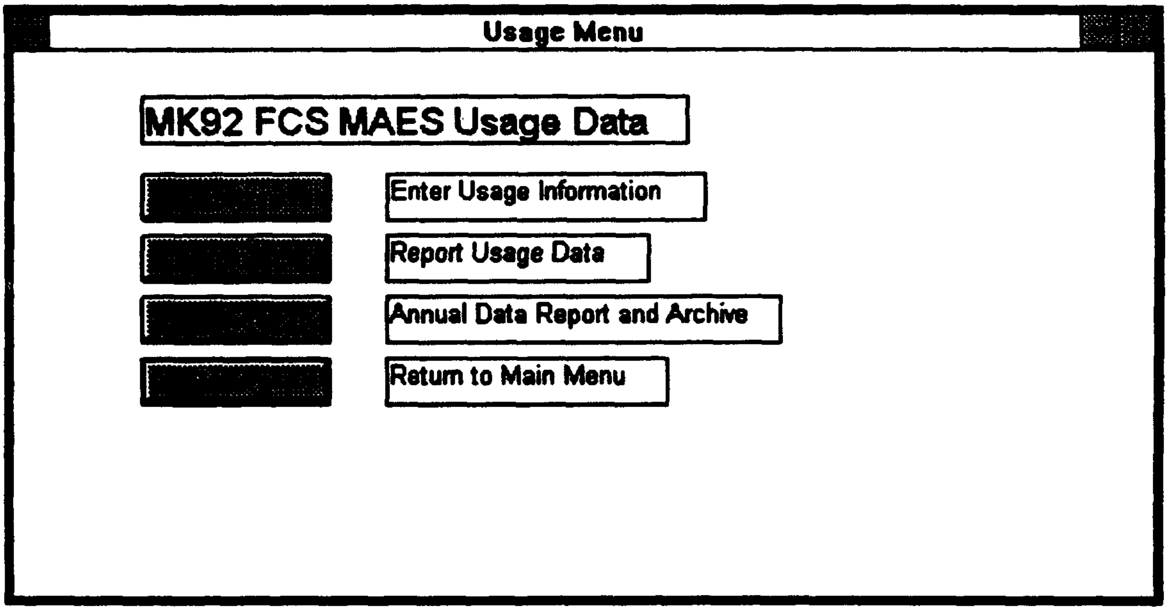


Figure C-5 - Usage Submenu

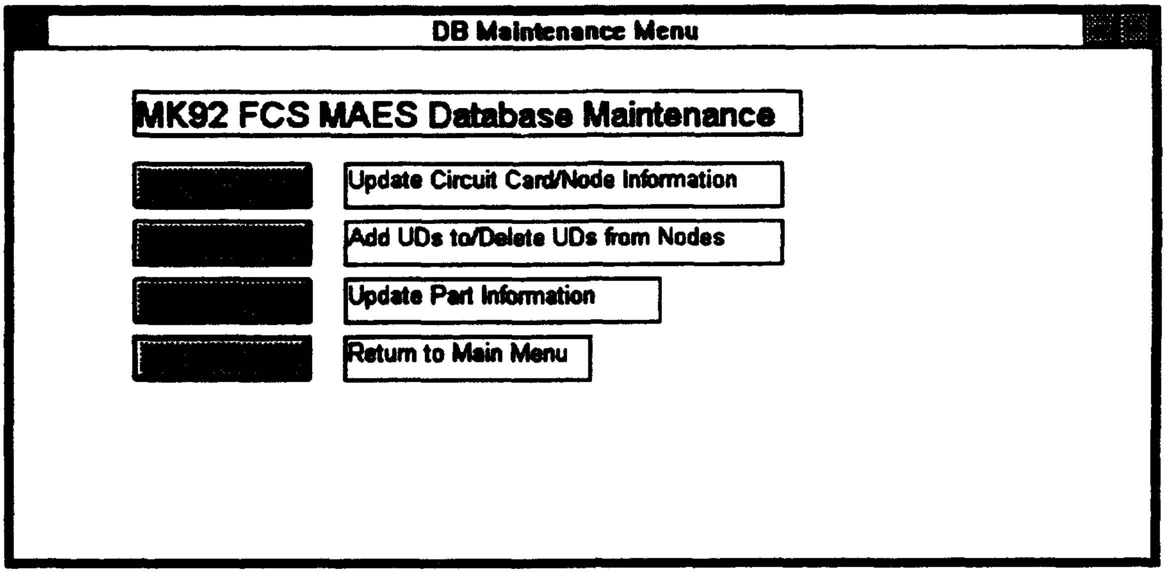

DB Maintenance Menu
MK92 FCS MAES Database Maintenance
Update Circuit Card/Node Information
Add UDs to/Delete UDs from Nodes
Update Part Information
Return to Main Menu

Figure C-6 - DB Maintenance Submenu

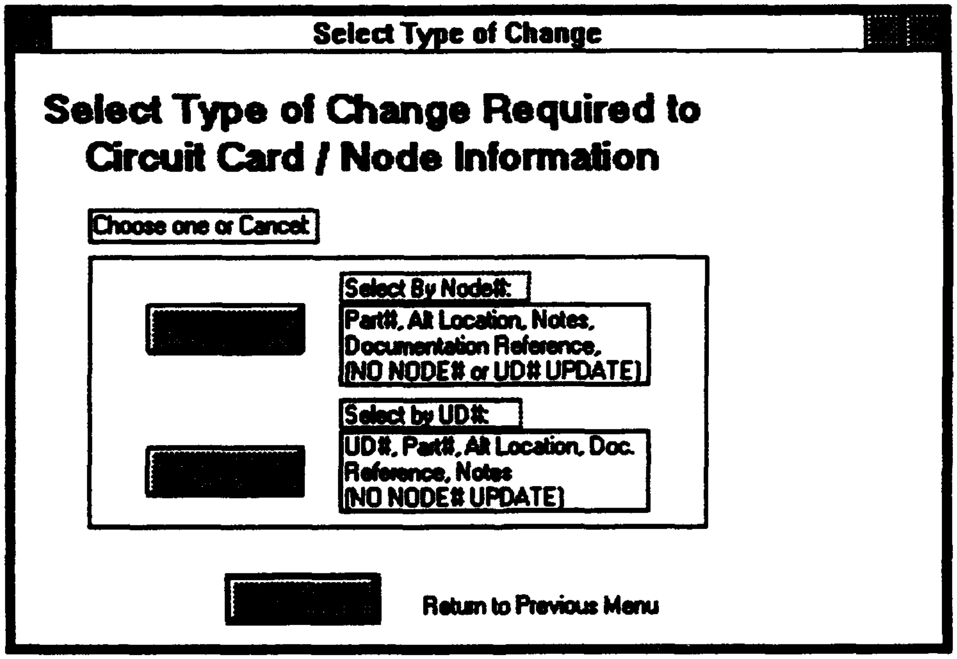

Select Type of Change
Select Type of Change Required to Circuit Card / Node Information
Choose one or Cancel
Select By Node#:
Part#, Alt Location, Notes, Documentation Reference, (NO NODE# or UD# UPDATE)
Select by UD#:
UD#, Part#, Alt Location, Doc. Reference, Notes (NO NODE# UPDATE)
Return to Previous Menu

Figure C-7 - Update Selection Screen

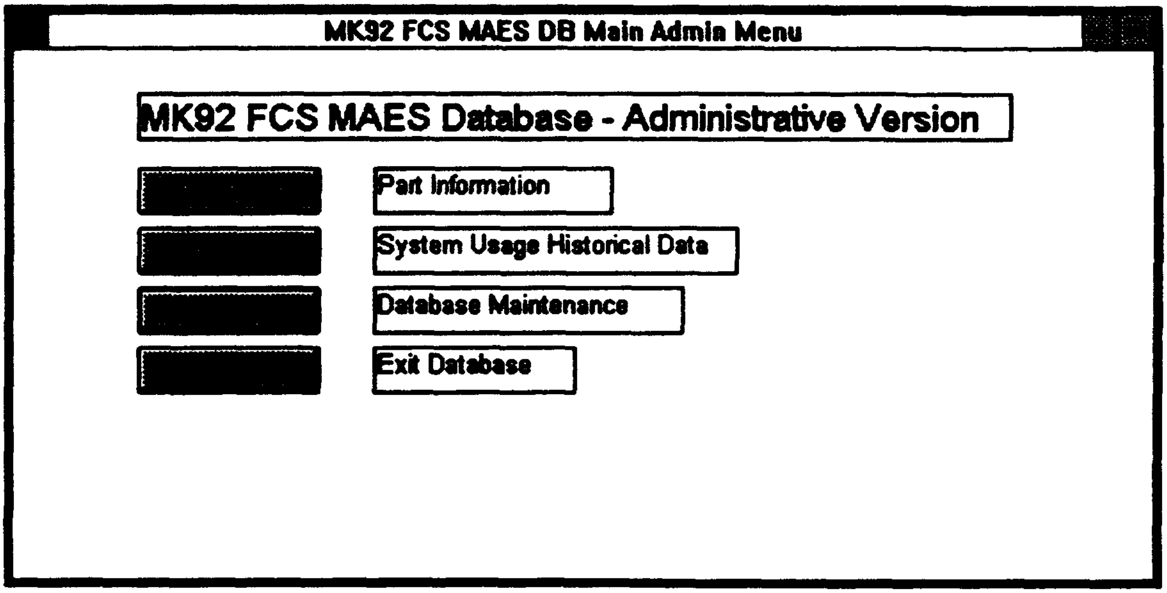

MK92 FCS MAES DB Main Admin Menu
MK92 FCS MAES Database - Administrative Version
Part Information
System Usage Historical Data
Database Maintenance
Exit Database

Figure C-8 - Administrator (Admin) Menu

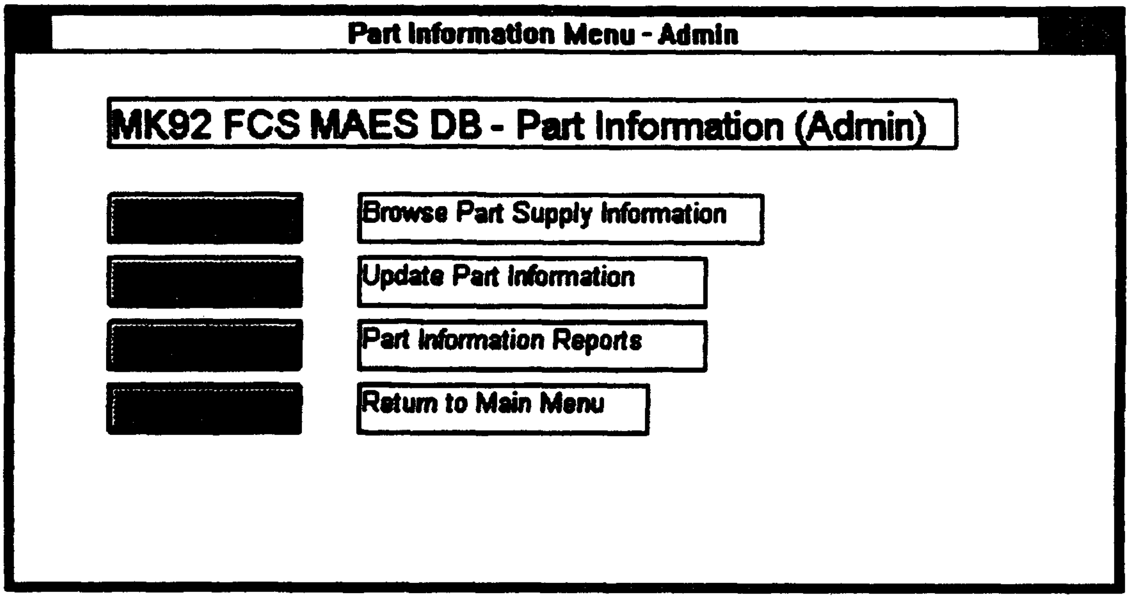

Part Information Menu - Admin
MK92 FCS MAES DB - Part Information (Admin)
Browse Part Supply Information
Update Part Information
Part Information Reports
Return to Main Menu

Figure C-9 - Part Information (Admin) Submenu

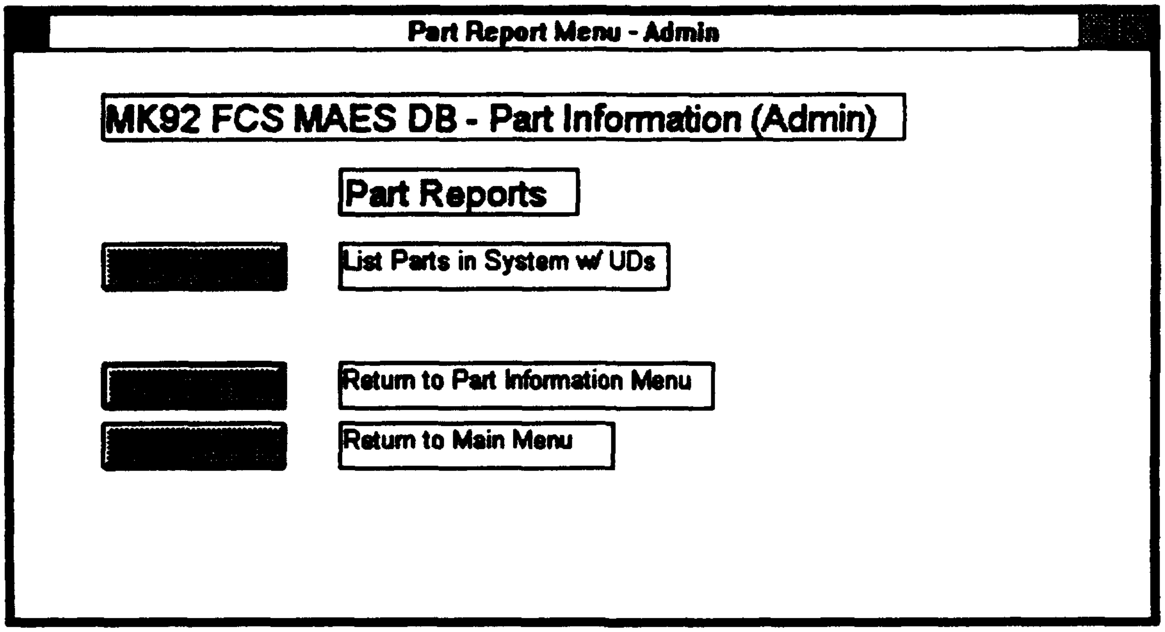

Part Report Menu - Admin
MK92 FCS MAES DB - Part Information (Admin)
Part Reports
List Parts in System w/ UDs
Return to Part Information Menu
Return to Main Menu

Figure C-10 - Part Report (Admin) Submenu

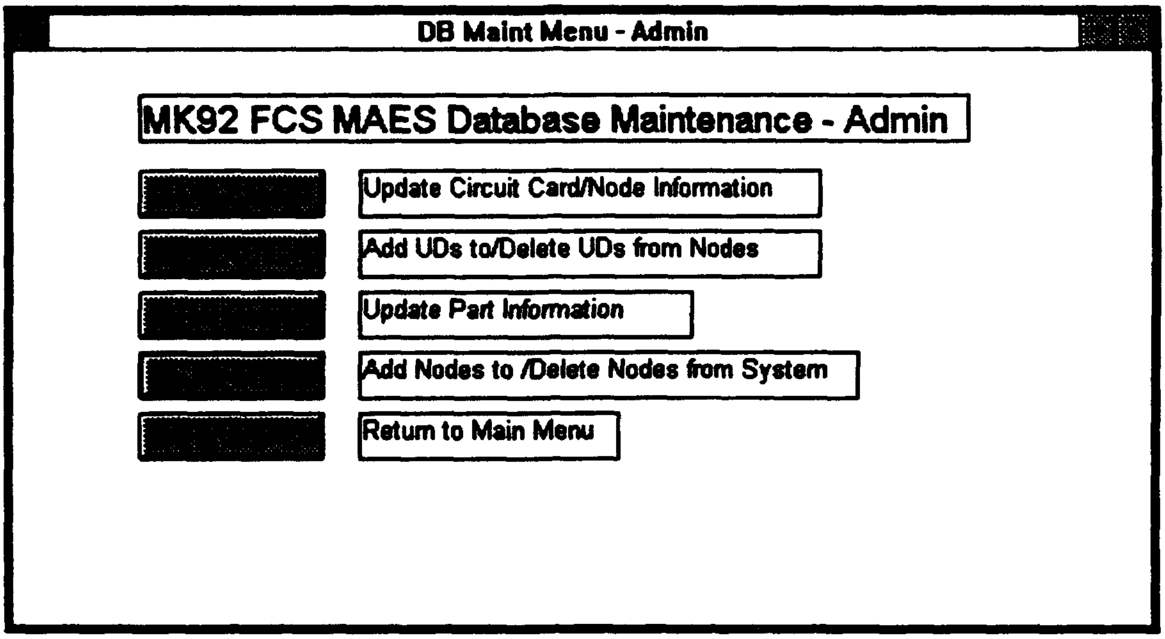

DB Maint Menu - Admin
MK92 FCS MAES Database Maintenance - Admin
Update Circuit Card/Node Information
Add UDs to/Delete UDs from Nodes
Update Part Information
Add Nodes to /Delete Nodes from System
Return to Main Menu

Figure C-11 - DB Maintenance (Admin) Submenu

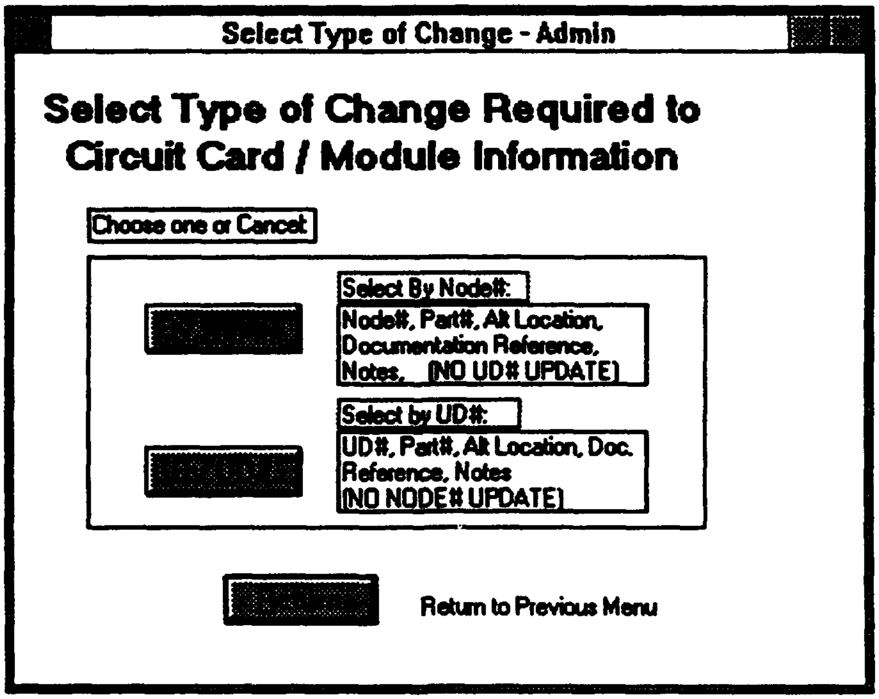

Select Type of Change - Admin
Select Type of Change Required to Circuit Card / Module Information
Choose one or Cancel
Select By Node#:
Node#, Part#, Alt Location, Documentation Reference, Notes. (NO UD# UPDATE)
Select by UD#:
UD#, Part#, Alt Location, Doc. Reference, Notes (NO NODE# UPDATE)
Return to Previous Menu

Figure C-12 - Update Selection (Admin) Screen

APPENDIX D - PROCESS LOGIC, SYSTEM FORMS, AND SYSTEM REPORTS

A. PROCESS LOGIC

```
Program START
    Activate OPENING MENU procedure

OM OPENING MENU: (procedure)
Activate OPENING Menu - Display Menu
    On Command Button Push, Run procedures
        (USER MENU, ADMIN MENU, EXIT)
```

1. User Procedures

```
UM USER MENU: (procedure)
Activate USER Menu - Display Menu
    On Command Button Push, Activate procedures
        (PART INFO MENU - USER, USAGE HISTORY MENU, DB MAINT
               MENU -USER, EXIT)

U1.0 PART INFO MENU- USER: (procedure)
Activate PART INFO - USER Menu - Display Menu
    On Command Button Push, Activate procedures
        (SUPPLY, UPDATE SUPPLY INFO, REPORT PARTS INFO MENU -
             USER, RETURN)

    U1.1 SUPPLY: (procedure)
    Activate BROWSE PART form
    Input PART# from list selection
        On Command Button Push, Activate subprocedures
            (LOCATE, CANCEL)

        U1.1L LOCATE: (procedure)
        Activate PART SUPPLY INFO form
            Retrieve PART instance WHERE PART# = Form[BROWSE
               PART][PART#]
        Display PART instance
            On Command Button Push, Activate subprocedures
               (RETURN)
        U1.1L-R RETURN: (procedure)
        Close PART SUPPLY INFO form
        Activate USER MENU procedure

    U1.1C CANCEL: (procedure)
    Close PART SUPPLY INFO form
    Activate PART INFO MENU - USER procedure

U1.2 UPDATE SUPPLY INFO: (procedure)
Activate UPDATE PART form
    Accept PART#
        On Command Button Push, Activate subprocedures
           (LOCATE(U), CANCEL(U))

    U1.2L LOCATE(U): (procedure)
    Check to see if Part# input provided and it exists
        if not, display message and Activate UPDATE SUPPLY INFO
                procedure
    Retrieve PART instance WHERE PART# = Form[UPDATE PART][PART#]
    Activate PART SUPPLY INFO UPDATE form
    Display PART instance
    Accept changes to Part information
        On Command Button Push, Activate subprocedures
           (MORE, RETURN(L), ISSUE, ORDER, RECEIVE)

           U1.2L-M MORE: (procedure)
           Update PART instance WHERE PART# = Form[PART SUPPLY
                INFO UPDATE][PART#]
           Close PART SUPPLY INFO UPDATE form
           Activate UPDATE SUPPLY INFO subprocedure

           U1.2L-R RETURN(L) (procedure)
           Update PART instance WHERE PART# = Form[PART SUPPLY
                INFO UPDATE][PART#]
           Close PART SUPPLY INFO UPDATE form
           Activate PART INFO MENU - USER procedure

           U1.2L-I ISSUE (procedure)
           Activate ISSUE PARTS form
           Set initial number of parts issued to 0
           Accept update to number of parts issued
           On Command Button Push, Activate subprocedures
                (UPDATE(I), CANCEL(I))
        U1.2L-IU UPDATE(I): (procedure)
        Calculate number of parts on hand: Subtract number of
                parts issued from number of parts on hand
        Close ISSUE PARTS form

        U1.2L-IC CANCEL(I): (procedure)
        Close ISSUE PARTS form

    U1.2L-O ORDER (procedure)
    Activate PARTS ORDERED form
    Set initial number of parts ordered to 0
    Accept update to number of parts ordered
    On Command Button Push, Activate subprocedures
        (UPDATE(O), CANCEL(O))

        U1.2L-OU UPDATE(O): (procedure)
        Calculate number of parts on order: Add number of parts
                ordered to number of parts on order
        Close PARTS ORDERED form

        U1.2L-OC CANCEL(O): (procedure)
        Close PARTS ORDERED form

    U1.2L-R RECEIVE (procedure)
    Activate PARTS RECEIVED form
    Set initial number of parts received to 0
    Accept update to number of parts received
    On Command Button Push, Activate subprocedures
        (UPDATE(R), CANCEL(R))

        U1.2L-RU UPDATE(R): (procedure)
        Calculate number of parts on order: Subtract number of
                parts received from number of parts on order
        Calculate number of parts on hand: Add number of parts
                received to number of parts on hand
        Close PARTS RECEIVED form

        U1.2L-RC CANCEL(R): (procedure)
        Close PARTS RECEIVED form

U1.2C CANCEL(U)
Close UPDATE PART form
Activate PART INFO MENU -USER procedure
```

U1.3 <u>REPORT PARTS INFO MENU - USER</u>: (procedure)

```
Activate PARTS REPORT MENU - USER form
    On Command Button Push, Activate subprocedures
        (REPORT PARTS NOT ON HAND, REPORT PARTS ON ORDER,
            REPORT PARTS UNDER STOCK, PREVIOUS (RPU),
            RETURN TO MAIN(RRU))
```

U1.3.1 <u>REPORT PARTS NOT ON HAND</u> (procedure)

```
Query PART for PARTS NOT ON HAND:
    Select Part#, Number Parts On Hand WHERE
        Number Parts On Hand = 0
Query PART and REPLACEMENT for UDs FOR PARTS NOT ON HAND:
    Select Part#, UD# WHERE Number Parts On Hand = 0
Display PARTS NOT ON HAND report from PARTS NOT ON HAND
        query
Display PARTS NOT ON HAND subreport from UDs FOR PARTS NOT
        ON HAND query
    WHERE PART# for PARTS NOT ON HAND subreport = PART# for
        PARTS NOT ON HAND report
On Command Button Push, Activate procedures
        (CANCEL, PRINT, ZOOM) **
    ** Note: these are Access procedures, and are not covered further
```

U1.3.2 <u>REPORT PARTS ON ORDER</u> (procedure)

```
Query PART for PARTS ON ORDER:
    select Part#, Number Parts On Order WHERE
        Number Parts On Order > 0
Query PART and REPLACEMENT for UDs FOR PARTS ON ORDER:
    select Part#, UD# WHERE Number Parts On Order > 0
Display PARTS ON ORDER report from PARTS ON ORDER query
Display PARTS ON ORDER subreport from UDs FOR PARTS ON ORDER
       query
    WHERE PART# for PARTS ON ORDER subreport = PART# for
        PARTS ON ORDER report
On Command Button Push, Activate procedures
        (CANCEL, PRINT, ZOOM) **
    ** Note: these are Access procedures, and are not covered further
```

```
          U1.3.3 REPORT PARTS UNDER STOCK (procedure)
          Query PART for  PARTS UNDER STOCK:
                select Part#, Allowance, Parts On Hand, Parts On Order WHERE
                            (Number Parts On Hand < Allowance)
          Display PARTS UNDER STOCK report from PARTS UNDER STOCK
                query
          On Command Button Push, Activate procedures
                   (CANCEL, PRINT, ZOOM) **
                ** Note: these are Access procedures, and are not covered

          U1.3P PREVIOUS(RPU) (procedure)
          Close PARTS REPORT MENU - USER form
          Activate PART INFO MENU- USER procedure

          U1.3R RETURN TO MAIN (RRU) (procedure)
          Close PARTS REPORT MENU - USER form
          Activate USER MENU procedure

     U1.0R RETURN
     Close PART INFO - USER menu
     Activate USER MENU procedure

U2.0 DB MAINT MENU - USER (procedure)
Activate DB MAINT - USER Menu - Display Menu
     On Command Button Push, Activate procedures
          (CKT CARD(U), UD(U), PART INFO(U), RETURN(DU))

     U2.A CKT CARD(U) (procedure)
     Activate SELECT CHANGE Menu - Display Menu
          On Command Button Push, Activate procedures
               (BY NODE#(U), BY UD#, RETURN)

          U2.1.1 BY NODE#(U) (procedure)
          Activate INPUT NODE# form
          Accept Node# input
               On Command Button Push, Activate procedures
                   (CANCEL(NU), LOCATE(NU))
```

U2.1.1L LOCATE(NU) (procedure)

```
Retrieve NODE, NODE-REPL, and REPLACEMENT instances
    WHERE Node# = Form[INPUT NODE#][NODE#]
Check to see if Node# input provided and it exists
    if not, display message and Activate BY NODE# procedure
Activate UPDATE NODE-REPLACEMENT - USER forms
Display NODE instance and first NODE-REPL and REPLACEMENT
           instances
Accept changes to Node and Replacement information
    On Command Button Push, Activate procedures
           (CLEAR(NR), MORE(NRU), EXIT(NRU), FWD(NR),
                  BACK(NR))
```

U2.1.1L-C CLEAR(NR) (procedure)

```
Cancel changes to Node and Replacement information
```

U2.1.1L-M MORE(NRU) (procedure)

```
Update NODE, NODE-REPL, REPLACEMENT instances where
       Node# = Form[UPDATE NODE-REPLACEMENT-
       USER][NODE#] AND UD# = Form[UPDATE NODE -
       REPLACEMENT(sub))][UD#]
Close UPDATE NODE-REPLACEMENT forms
Activate BY NODE#(U) procedure
```

U2.1.1L-E EXIT(NRU) (procedure)

```
Update NODE, NODE-REPL, REPLACEMENT instances where
       Node# = Form[UPDATE NODE-REPLACEMENT]
       [NODE#] AND UD# = Form[UPDATE NODE -
       REPLACEMENT-(sub))][UD#]
Close UPDATE NODE-REPLACEMENT - USER forms
Activate CKT CARD procedure
```

U2.1.1L-F FWD(NR) (procedure)

```
Display next NODE-REPL and REPLACEMENT instances for
       NODE#
```

U2.1.1L-B BACK(NR) (procedure)

```
Display previous NODE-REPL and REPLACEMENT instances for
       NODE#
```

U2.1.1C CANCEL(NU) (procedure)

```
Close INPUT NODE# form
Activate CKT CARD(U) procedure
```

```
U2.2.1 BY UD#(U) (procedure)
Activate INPUT UD# form
Accept UD# input
    On Command Button Push, Activate procedures
        (CANCEL(UDU), LOCATE(UU))

    U2.2.1C CANCEL(UDU)
    Close INPUT UD# form
    Activate CKT CARD(U) procedure

    U2.2.1L LOCATE(UD)
    Retrieve REPLACEMENT instances
        WHERE UD# = Form[INPUT UD#][UD#]
    Check to see if UD# input provided and it exists
        if not, display message and Activate BY UD# procedure
    Activate UPDATE UD# - REPLACEMENT form
    Display REPLACEMENT instance
    Accept changes to Replacement information
        On Command Button Push, Activate procedures
             (UPDATE UD#, CLEAR(UD), EXIT(UDU), MORE(UD))

        U2.2.1L-U UPDATE UD# (procedure)
        Activate CHANGE UD# form
        Accept UD# input
        Check to see if UD# input provided and it does not already
             exist, if not correct display message and Activate
             UPDATE UD# procedure
        On Command Button Push, Activate procedures
             (CANCEL(UD), CHANGE UD#(UD))

             U2.2.1L-UC CANCEL(UD) (procedure)
             Close CHANGE UD# form

             U2.2.1L-UD CHANGE UD#(UD) (procedure)
             Accept NEW UD# input
             Validate change (Yes/No)
                    if NO, Close CHANGE UD# form
             Activate UPDATE RELATED UD function
             Display UPDATE UD# - REPLACEMENT form where
                  UD# = NEW UD#

        U2.2.1L-C CLEAR(UU) (procedure)
        Cancel changes to Replacement information
```

```
          U2.2.1L-E EXIT(UDU) (procedure)
          Update REPLACEMENT instance where UD# = Form[UPDATE
               UD# - REPLACEMENT][UD#]
          Close UPDATE UD# - REPLACEMENT form
          Activate CKT CARD(U) procedure

          U2.2.1L-M MORE(UU) (procedure)
          Update REPLACEMENT instance where UD# = Form[UPDATE
               UD# - REPLACEMENT][UD#]
          Close UPDATE UD# - REPLACEMENT form
          Activate BY UD#(U) procedure

     U2.B UPDATE UD(U) (procedure)
     Not Yet Designed

     U2.C UPDATE PART INFO(U) (procedure)
     Not Yet Designed

     U2.0-R RETURN(DU) (procedure)
     Close DB MAINT - USER MENU
     Activate USER MENU

U3.0 USAGE HISTORY MENU (procedure)
Activate USAGE MENU
     On Command Button Push, Activate procedures
          (ENTER USAGE DATA(U), PERIODIC USAGE REPORT(U), ANNUAL
                    USAGE REPORT(U), RETURN(HU))

     U3.1 ENTER USAGE DATA(U) (procedure)
     Not Yet Designed

     U3.2 PERIODIC USAGE REPORT(U) (procedure)
     Not Yet Designed

     U3.3 ANNUAL USAGE REPORT(U) (procedure)
     Not Yet Designed

     U3.3-R RETURN(HU) (procedure)
     Close USAGE MENU
     Activate USER MENU procedure
```

2. Administrator Procedures

```
AM ADMIN MENU (procedure)
Activate ADMIN Menu - Display Menu
          On Command Button Push, Activate procedures
                    (PART INFO MENU - ADMIN, USAGE MENU, DB MAINT MENU -
                              ADMIN, EXIT ADMIN)

A1.0 PART INFO MENU- ADMIN: (procedure)
Activate PART INFO - ADMIN Menu - Display Menu
          On Command Button Push, Activate procedures
                    (SUPPLY, UPDATE SUPPLY INFO, REPORT PARTS INFO MENU -
                           ADMIN, RETURN(AR))

          A1.1 SUPPLY: (procedure)  [SAME AS U1.1]
          Activate BROWSE PART form
          Input PART# from list selection
                    On Command Button Push, Activate subprocedures
                              (LOCATE(AS), CANCEL(CS))

                    A1.1L LOCATE(AS): (procedure) [SAME AS U1.1L]
                    Activate PART SUPPLY INFO form
                              Retrieve PART instance WHERE PART# = Form[BROWSE
                                     PART][PART#]
                    Display PART instance
                              On Command Button Push, Activate subprocedures
                                     (RETURN(AS))

                              A1.1L-R RETURN(AS): (procedure)
                              Close PART SUPPLY INFO form
                              Activate ADMIN MENU procedure

                    A1.1C CANCEL(CS): (procedure)
                    Close PART SUPPLY INFO form
                    Activate PART INFO MENU - ADMIN procedure

          A1.2 UPDATE SUPPLY INFO: (procedure) [SAME AS U1.2]
          Activate UPDATE PART form
                    Accept PART#
                              On Command Button Push, Activate subprocedures
                                     (LOCATE(UA), CANCEL(UA))
```

```
A1.2L LOCATE(UA): (procedure)
Check to see if Part# input provided and it exists
    if not, display message and Activate UPDATE SUPPLY INFO
            procedure
Retrieve PART instance WHERE PART# = Form[UPDATE PART][PART#]
Activate PART SUPPLY INFO UPDATE form
Display PART instance
Accept changes to Part information
    On Command Button Push, Activate subprocedures
        (MORE, RETURN(LA), ISSUE, ORDER, RECEIVE)

        A1.2L-M MORE: (procedure) [SAME AS U1.2L-M]
        Update PART instance WHERE PART# = Form[PART SUPPLY
                INFO UPDATE][PART#]
        Close PART SUPPLY INFO UPDATE form
        Activate UPDATE SUPPLY INFO subprocedure

        A1.2L-R RETURN(LA) (procedure)
        Update PART instance WHERE PART# = Form[PART SUPPLY
                INFO UPDATE][PART#]
        Close PART SUPPLY INFO UPDATE form
        Activate PART INFO MENU - ADMIN procedure

        A1.2L-I ISSUE (procedure) [SAME AS U1.2L-I]
        Activate ISSUE PARTS form
        Set initial number of parts issued to 0
        Accept update to number of parts issued
        On Command Button Push, Activate subprocedures
                (UPDATE(I), CANCEL(I))

              A1.2L-IU UPDATE(I): (procedure) [SAME AS U1.2L-IU]
              Subtract number of parts issued from number of parts on hand
              Close ISSUE PARTS form
              A1.2L-IC CANCEL(I): (procedure) [SAME AS U1.2L-IC]
              Close ISSUE PARTS form

        A1.2L-O ORDER (procedure) [SAME AS U1.2L-O]
        Activate PARTS ORDERED form
        Set initial number of parts ordered to 0
        Accept update to number of parts ordered
        On Command Button Push, Activate subprocedures
                (UPDATE(O), CANCEL(O))
                A1.2L-OU UPDATE(O): (procedure) [SAME AS U1.2L-
                  OU]
                Add number of parts ordered to number of parts on order
                Close PARTS ORDERED form

                A1.2L-OC CANCEL(O): (procedure) [SAME AS U1.2L-
                  OC
                Close PARTS ORDERED form

            A1.2L-R RECEIVE (procedure) [SAME AS U1.2L-R]
            Activate PARTS RECEIVED form
            Set initial number of parts received to 0
            Accept update to number of parts received
            On Command Button Push, Activate subprocedures
                 (UPDATE(R), CANCEL(R))

                A1.2L-RU UPDATE(R): (procedure) [SAME AS U1.2L-
                  RU]
                Subtract number of parts received from number of parts on
                       order
                Add number of parts received to number of parts on hand
                Close PARTS RECEIVED form

                A1.2L-RC CANCEL(R): (procedure)[SAME AS U1.2L-
                  RC]
                Close PARTS RECEIVED form

    A1.2C CANCEL(UA)
    Close UPDATE PART form
    Activate PART INFO MENU -ADMIN procedure

A1.3 REPORT PARTS INFO MENU - ADMIN: (procedure)
Activate PARTS REPORT MENU - ADMIN form
    On Command Button Push, Activate subprocedures
        (SYSTEM PARTS LIST, PREVIOUS(RPA), RETURN TO
           MAIN(RRA))

    A1.3.1 SYSTEM PARTS LIST (procedure)
    Query PART for SYSTEM PARTS:
        select Part# WHERE Part# <> Null
    Query PART and REPLACEMENT for UDs FOR SYSTEM PARTS:
        select Part#, UD# WHERE Part# = Parts list[Part#]
    Display SYSTEM PARTS report from SYSTEM PARTS query
```

```
        Display UDs FOR SYSTEM PARTS subreport from UDs for SYSTEM
               PARTS query
            WHERE PART# for UDs FOR SYSTEM PARTS subreport = PART#
                for SYSTEM PARTS report
        On Command Button Push, Activate procedures
                (CANCEL, PRINT, ZOOM) **
            ** Note: these are Access procedures, and are not covered further

        A1.3P PREVIOUS(RPA) (procedure)
        Close PARTS REPORT MENU - ADMIN form
        Activate PART INFO MENU- ADMIN procedure

        A1.3R RETURN TO MAIN (RRA) (procedure)
        Close PARTS REPORT MENU - ADMIN form
        Activate ADMIN MENU procedure

    A1.0R RETURN(AR)
    Close PART INFO - ADMIN menu
    Activate ADMIN MENU procedure

A2.0 DB MAINT MENU - ADMIN (procedure)
Activate DB MAINT - ADMIN Menu - Display Menu
    On Command Button Push, Activate procedures
        (CKT CARD(A), UD(A), PART INFO(A), RETURN(DA))

    A2.A CKT CARD(A) (procedure)
    Activate SELECT CHANGE Menu - Display Menu
        On Command Button Push, Activate procedures
            (BY NODE#(A), BY UD#, RETURN)

        A2.1.1 BY NODE#(A) (procedure)
        Activate INPUT NODE# form
        Accept Node# input
            On Command Button Push, Activate procedures
                (CANCEL(NA), LOCATE(NA))


            A2.1.1L LOCATE(NA) (procedure)
            Retrieve NODE, NODE-REPL, and REPLACEMENT instances
                WHERE Node# = Form[INPUT NODE#][NODE#]
            Check to see if Node# input provided and it exists
                if not, display message and Activate BY NODE# procedure
            Activate UPDATE NODE-REPLACEMENT - ADMIN forms
```

```
Display NODE instance and first NODE-REPL and REPLACEMENT
        instances
Accept changes to Node and Replacement information
    On Command Button Push, Activate procedures
        (UPDATE NODE#, CLEAR(NRA), MORE(NRA),
            EXIT(NRA), FWD(NRA), BACK(NRA))

    A2.1.1L-N UPDATE NODE#  (procedure)
    Activate CHANGE NODE# form
    Accept NODE# input
    Check to see if NODE# input provided and it does not already
        exist, if not correct display message and Activate
        UPDATE NODE# procedure
    On Command Button Push, Activate procedures
        (CANCEL(UN), CHANGE NODE#)

        A2.2.1L-NC CANCEL(UN) (procedure)
        Close CHANGE NODE# form

        A2.2.1L-ND CHANGE NODE# (procedure)
        Accept NEW NODE# input
        Validate change (Yes/No)
            if NO, Close CHANGE NODE# form
        Activate UPDATE RELATED NODE function
        Display UPDATE NODE-REPLACEMENT- ADMIN form
          where NODE# = NEW NODE#

    A2.1.1L-C CLEAR(NRA) (procedure)
    Cancel changes to Node and Replacement information

    A2.1.1L-M MORE(NRA) (procedure)
    Update NODE, NODE-REPL, REPLACEMENT instances where
        Node# = Form[UPDATE NODE-REPLACEMENT-
        ADMIN][NODE#] AND UD# = Form[UPDATE NODE -
        REPLACEMENT-ADMIN(sub)][UD#]
    Close UPDATE NODE-REPLACEMENT- ADMIN forms
    Activate BY NODE# (A) procedure
```

```
        A2.1.1L-E EXIT(NRA) (procedure)
        Update NODE, NODE-REPL, REPLACEMENT instances where
                Node# = Form[UPDATE NODE-REPLACEMENT-
                ADMIN][NODE#] AND UD# = Form[UPDATE NODE -
                REPLACEMENT-ADMIN(sub)][UD#]
        Close UPDATE NODE-REPLACEMENT- ADMIN forms
        Activate CKT CARD(A) procedure

        A2.1.1L-F FWD(NR) (procedure) [SAME AS U2.1.1L-F]
        Display next NODE-REPL and REPLACEMENT instances for
                NODE#

        U2.1.1L-B BACK(NRU) (procedure) [SAME AS U2.1.1L-B]
        Display previous NODE-REPL and REPLACEMENT instances for
                NODE#

    A2.1.1C CANCEL(NA) (procedure)
    Close INPUT NODE# form
    Activate CKT CARD(A) procedure

A2.2.1 BY UD#(A) (procedure)
Activate INPUT UD# form
Accept UD# input
    On Command Button Push, Activate procedures
        (CANCEL(UDA), LOCATE(UDA))

    A2.2.1C CANCEL(UDA)
    Close INPUT UD# form
    Activate CKT CARD(A) procedure

    A2.2.1L LOCATE(UDA)
    Retrieve REPLACEMENT instances
        WHERE UD# = Form[INPUT UD#][UD#]
    Check to see if UD# input provided and it exists
        if not, display message and Activate BY UD# procedure
    Activate UPDATE UD# - REPLACEMENT form
    Display REPLACEMENT instance
    Accept changes to Replacement information
        On Command Button Push, Activate procedures
                (UPDATE UD#, CLEAR(UD), EXIT(UDA),
                        MORE(UDA))
    A2.2.1L-U UPDATE UD# (procedure)
    Activate CHANGE UD# form
    Accept UD# input
    Check to see if UD# input provided and it does not already
        exist, if not correct display message and Activate
        UPDATE UD# procedure
    On Command Button Push, Activate procedures
        (CANCEL(UU), CHANGE UD#)

        A2.2.1L-UC CANCEL(UU) (procedure)
        Close CHANGE UD# form

        A2.2.1L-UD CHANGE UD# (procedure)
        Accept NEW UD# input
        Validate change (Yes/No)
            if NO, Close CHANGE UD# form
        Activate UPDATE RELATED UD function
        Display UPDATE UD# - REPLACEMENT form where
        UD# = NEW UD#

    A2.2.1L-C CLEAR(UD) (procedure)
    Cancel changes to Replacement information

    A2.2.1L-E EXIT(UDA) (procedure)
    Update REPLACEMENT instance where UD# = Form[UPDATE
        UD# - REPLACEMENT][UD#]
    Close UPDATE UD# - REPLACEMENT form
    Activate CKT CARD(A) procedure

    A2.2.1L-M MORE(UDA) (procedure)
    Update REPLACEMENT instance where UD# = Form[UPDATE
        UD# - REPLACEMENT][UD#]
    Close UPDATE UD# - REPLACEMENT form
    Activate BY UD#(A) procedure

A2.B UPDATE UD(A) (procedure)
Not Yet Designed

A2.C UPDATE PART INFO(A) (procedure)
Not Yet Designed
```

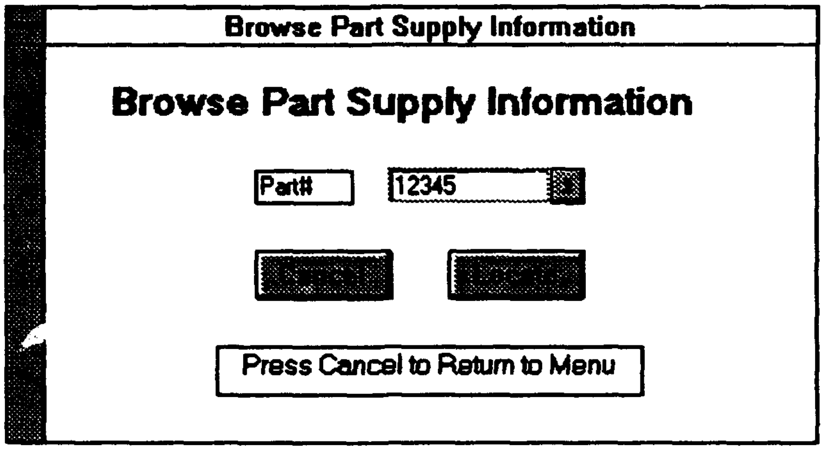
A2.0-R **<u>RETURN(DA)</u>** **(procedure)**
Close DB MAINT - ADMIN MENU
Activate ADMIN MENU

A3.0 **<u>USAGE HISTORY MENU(A)</u>** **(procedure)**
Activate USAGE MENU
Not Yet Designed

3. System Functions

```
Function UPDATE RELATED UD
          SELECT all NODE instances
          Old UD# = Form[Change UD#][UD#]
          SELECT all REPLACEMENT instances WHERE UD# = Old UD#
                    store in TEMP entity
          UPDATE all instances in TEMP entity
                    UD# = New UD#
          INSERT into REPLACEMENT all instances in TEMP entity
          DELETE all instances in TEMP entity

          Loop Until Done
                    FIND NODE-REPL instances WHERE UD# = Old UD#
                    UPDATE UD# to New UD#
          End
          Form[UPDATE UD# - REPLACEMENT][UD#] = New UD#

Function UPDATE RELATED NODE
          Old Node# = Form[Change Node#][Node#]
          SELECT all NODE instances WHERE Node# = Old Node#
                    store in TEMP entity
          UPDATE all instances in TEMP entity
                    Node# = New Node #
          INSERT into NODE all instances in TEMP entity
          DELETE all instances in TEMP entity

          Loop Until Done
                    FIND NODE-REPL instances WHERE Node# = Old Node#
                    UPDATE Node# to New Node#
          End
          Form[Update Node-Replacement -Admin][Node#] = New Node#
```

B. SYSTEM FORMS



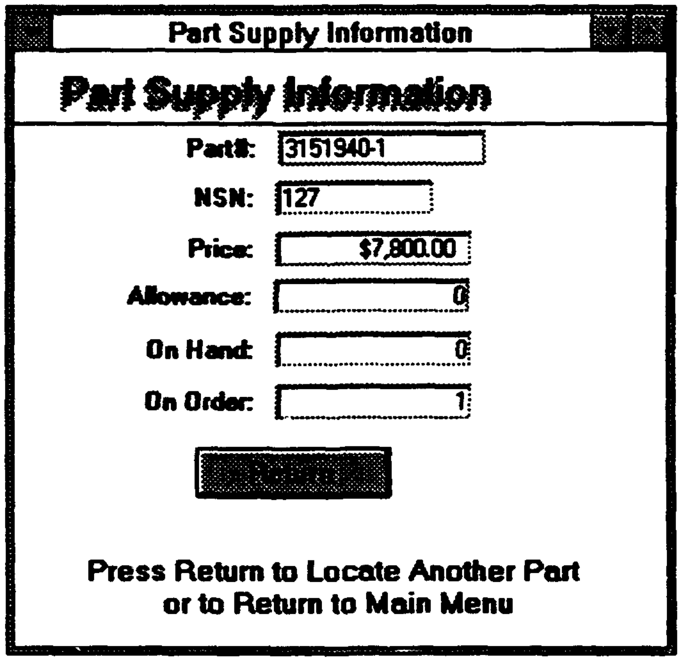
Browse Part Supply Information

Browse Part Supply Information

Part# 12345

Press Cancel to Return to Menu

Figure D-1 - Browse Part Supply Information Form



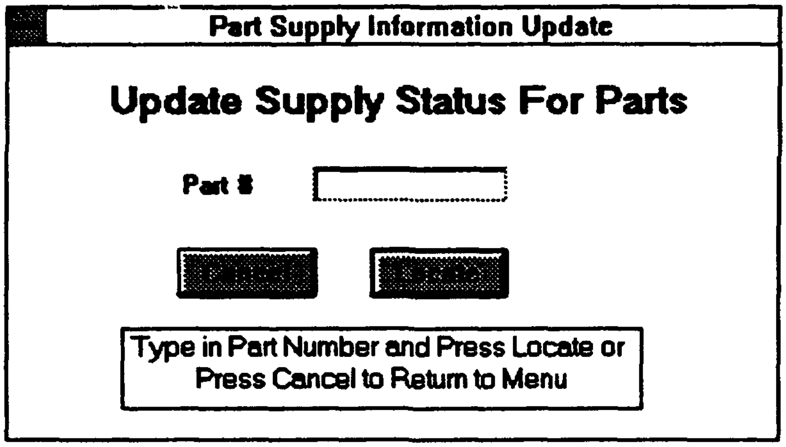
Part Supply Information

Part Supply Information

Part#:	3151940-1
NSN:	127
Price:	$7,800.00
Allowance:	0
On Hand:	0
On Order:	1

Press Return to Locate Another Part
or to Return to Main Menu

Figure D-2 - Part Supply Information Form



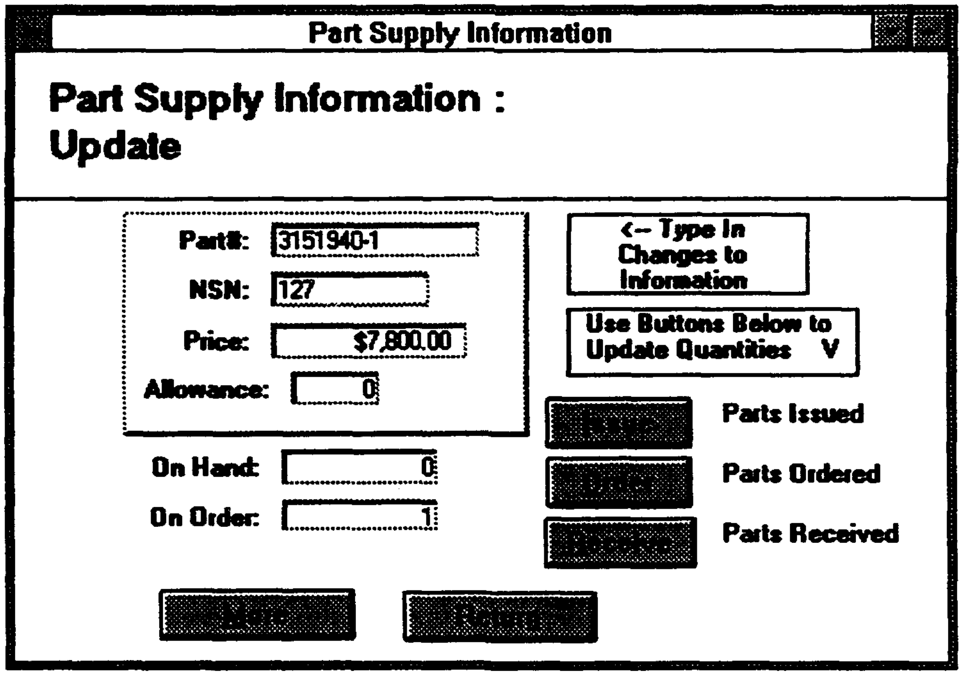
Part Supply Information Update

Update Supply Status For Parts

Part #

Type in Part Number and Press Locate or Press Cancel to Return to Menu

Figure D-3 - Part Supply Information Update Form



Part Supply Information

Part Supply Information : Update

Part#: 3151940-1

NSN: 127

Price: $7,800.00

Allowance: 0

On Hand: 0

On Order: 1

<-- Type In Changes to Information

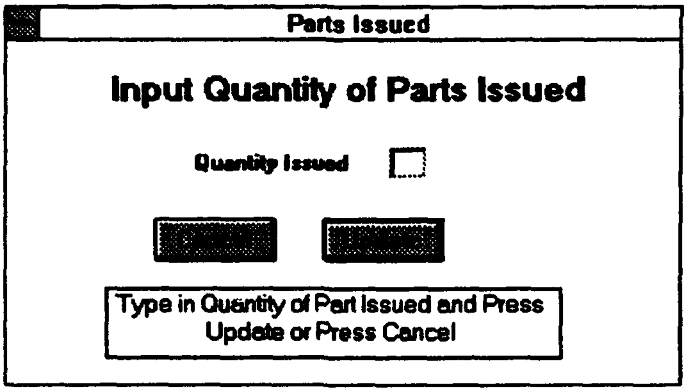
Use Buttons Below to Update Quantities V

Parts Issued

Parts Ordered

Parts Received

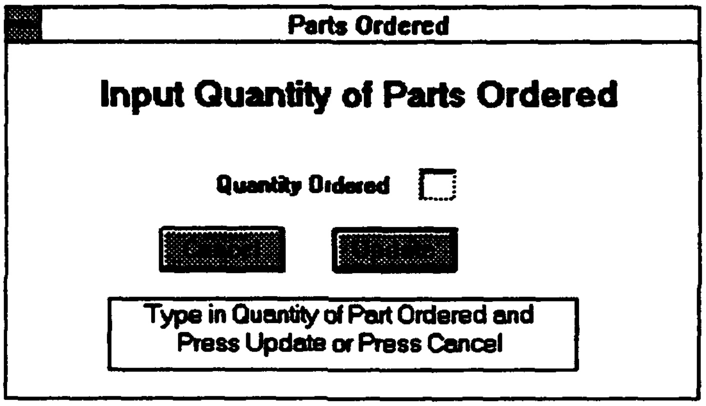
Figure D-4 - Part Supply Information Update Form



Input Quantity of Parts Issued

Type in Quantity of Part Issued and Press Update or Press Cancel

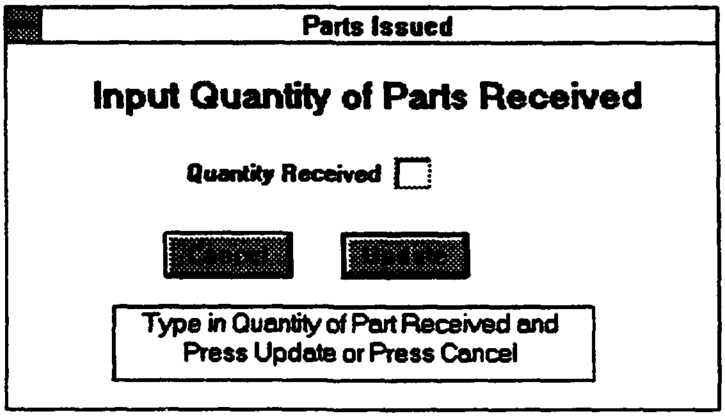
Figure D-5 - Parts Issued Form



Input Quantity of Parts Ordered

Type in Quantity of Part Ordered and Press Update or Press Cancel

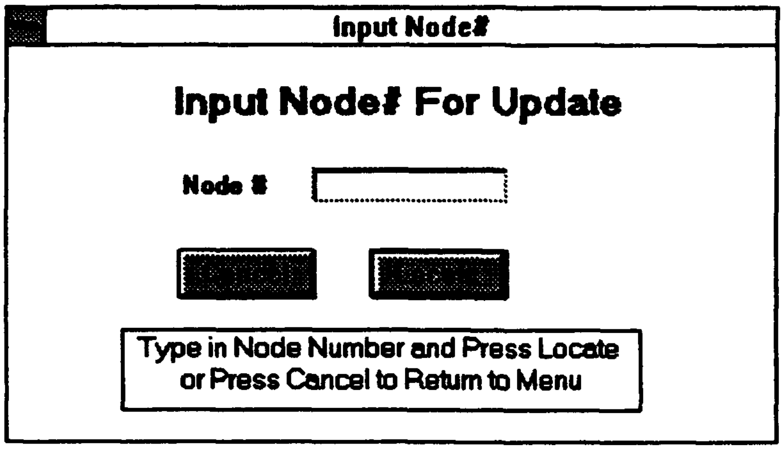
Figure D-6 - Parts Ordered Form



Input Quantity of Parts Received

Type in Quantity of Part Received and Press Update or Press Cancel

Figure D-7 - Parts Received Form



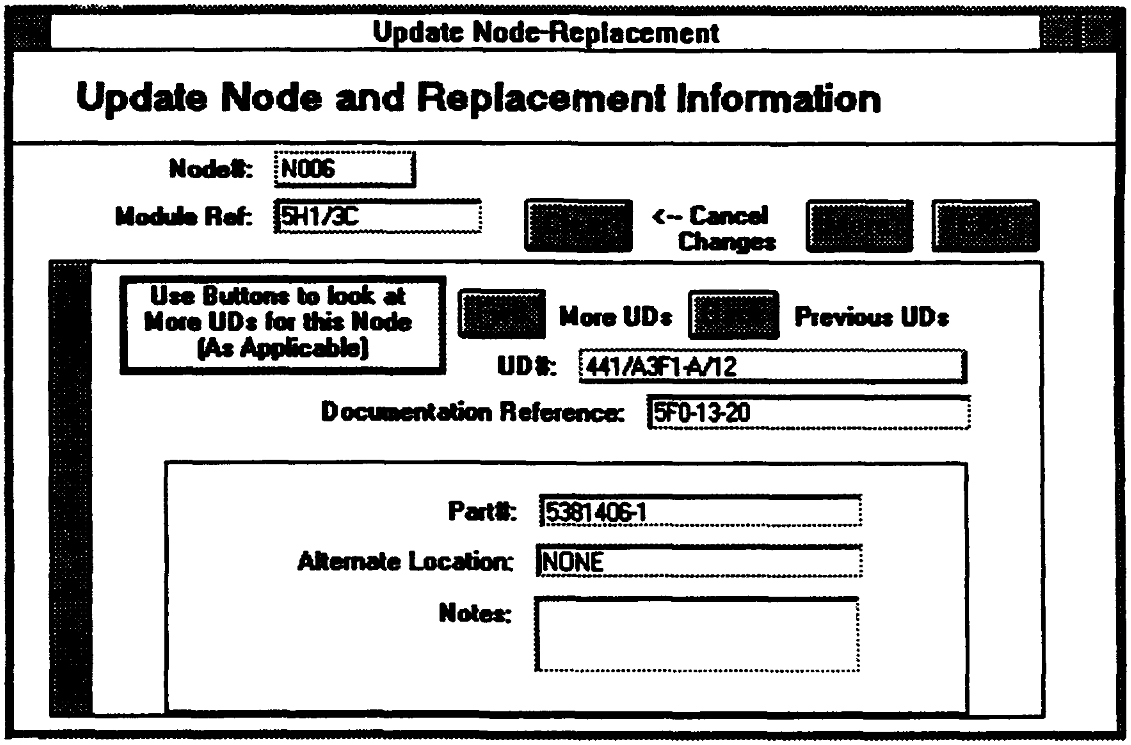
Input Node#

Input Node# For Update

Node #

Type in Node Number and Press Locate
or Press Cancel to Return to Menu

Figure D-8 - Input Node# Form



Update Node-Replacement

Update Node and Replacement Information

Node#: N006

Module Ref: 5H1/3C

<-- Cancel Changes

Use Buttons to look at More UDs for this Node (As Applicable)

More UDs

Previous UDs

UD#: 441/A3F1-A/12

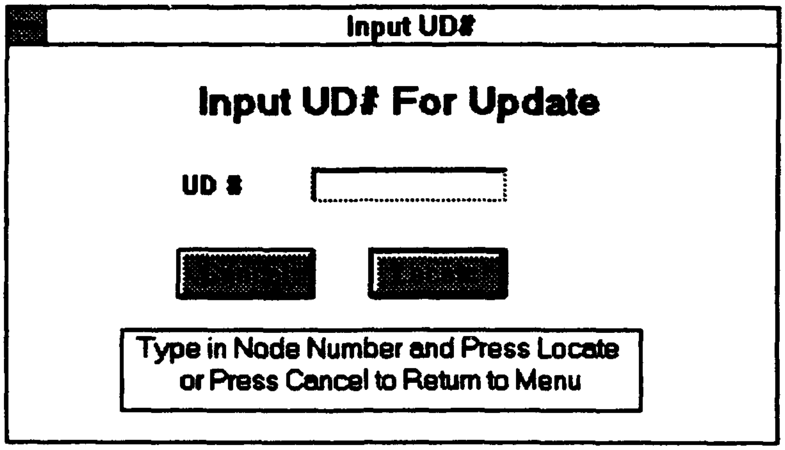
Documentation Reference: 5F0-13-20

Part#: 5381406-1

Alternate Location: NONE

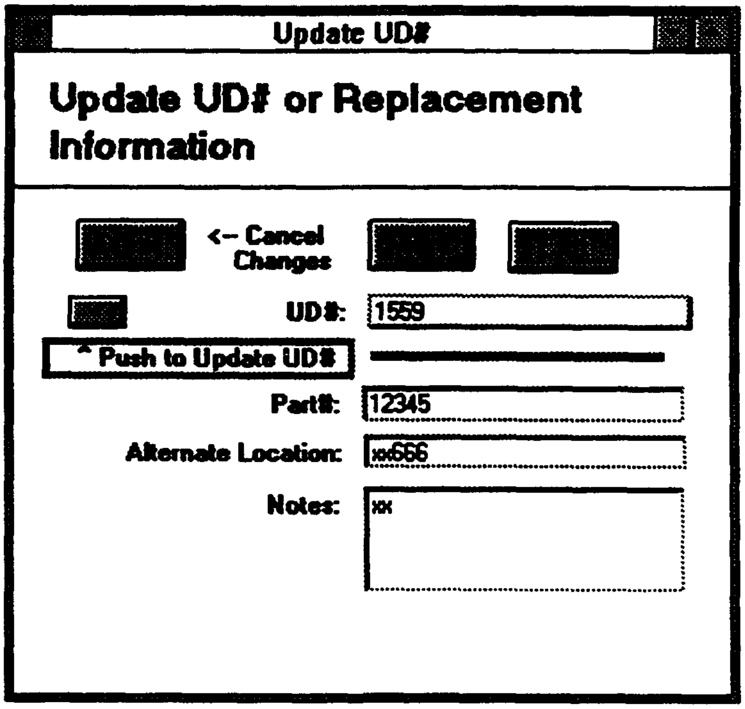
Notes:

Figure D-9 - Update Node-Replacement Information



Type in Node Number and Press Locate or Press Cancel to Return to Menu

Figure D-10 - Input UD# Form



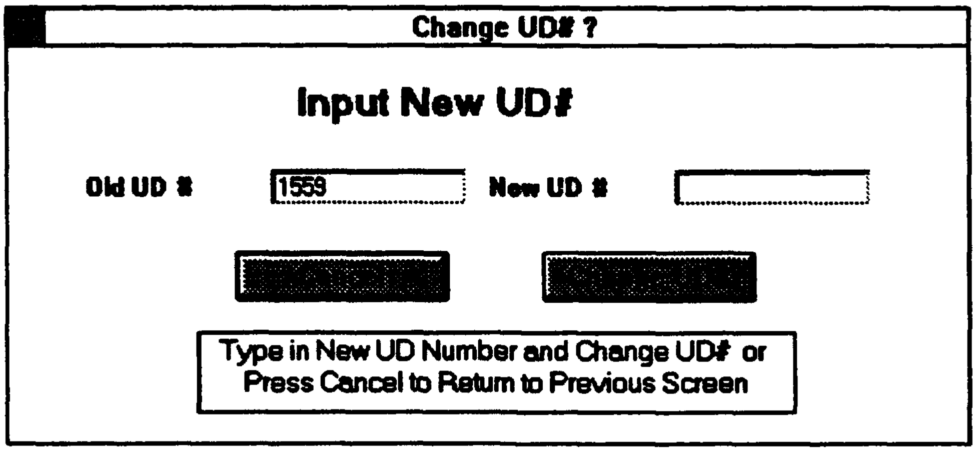
UD#: 1559

Part#: 12345

Alternate Location: xx666

Notes: xx

Figure D-11 - Update UD# - Replacement Form



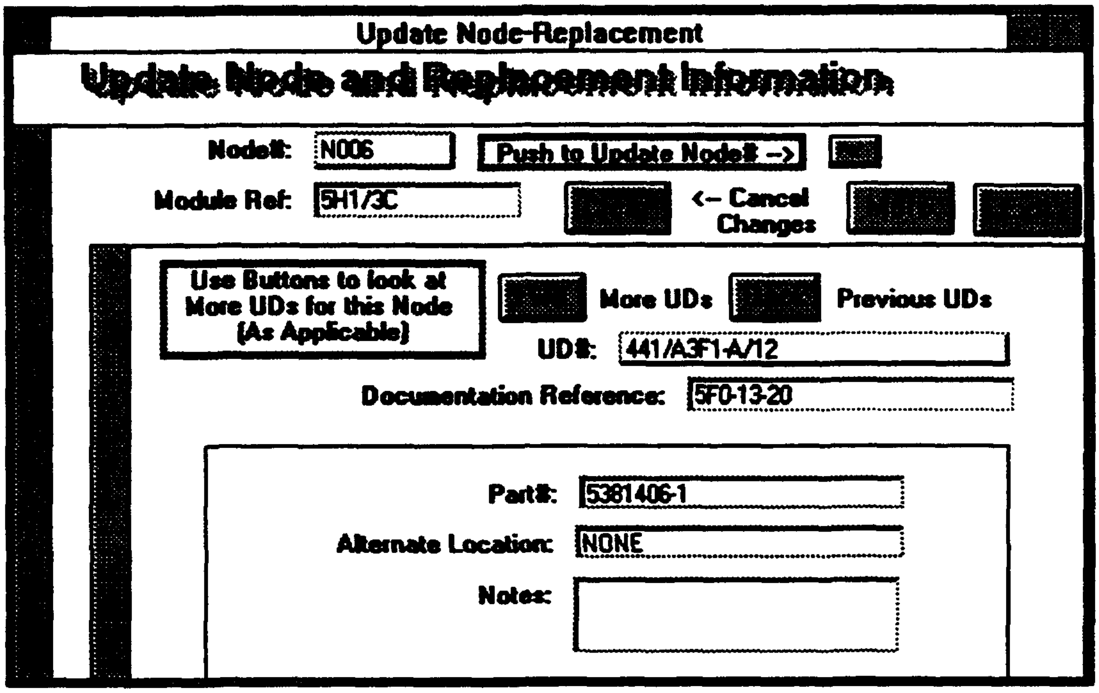
Change UD# ?

Input New UD#

Old UD # 1559 New UD #

Type in New UD Number and Change UD# or Press Cancel to Return to Previous Screen

Figure D-12 - Change UD# Form



Update Node-Replacement

Update Node and Replacement Information

Node#: N006

Module Ref: 5H1/3C

Use Buttons to look at More UDs for this Node (As Applicable)

UD#: 441/A3F1-A/12

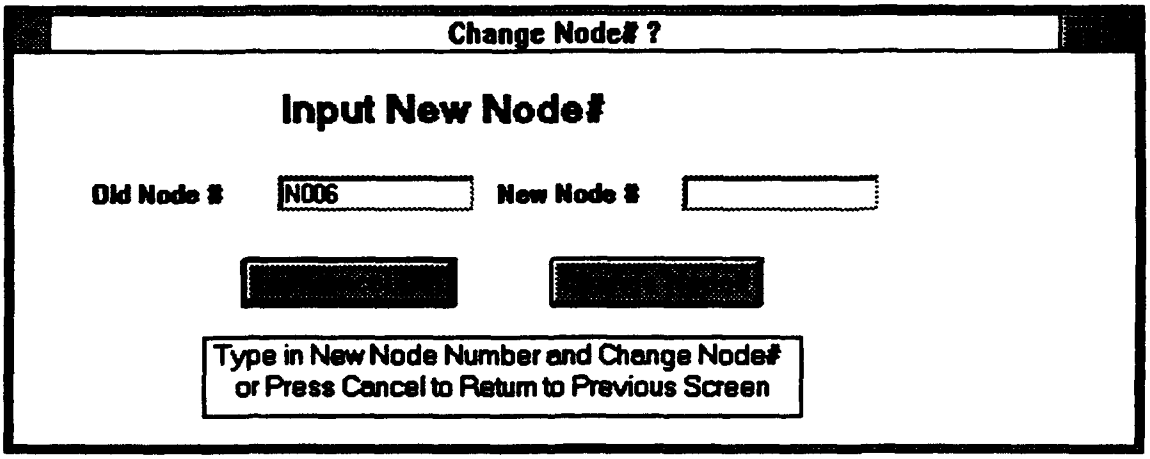
Documentation Reference: 5F0-13-20

Part#: 5381406-1

Alternate Location: NONE

Notes:

Figure D-13 - Update Node-Replacement (Admin) Form



Change Node# ?

Input New Node#

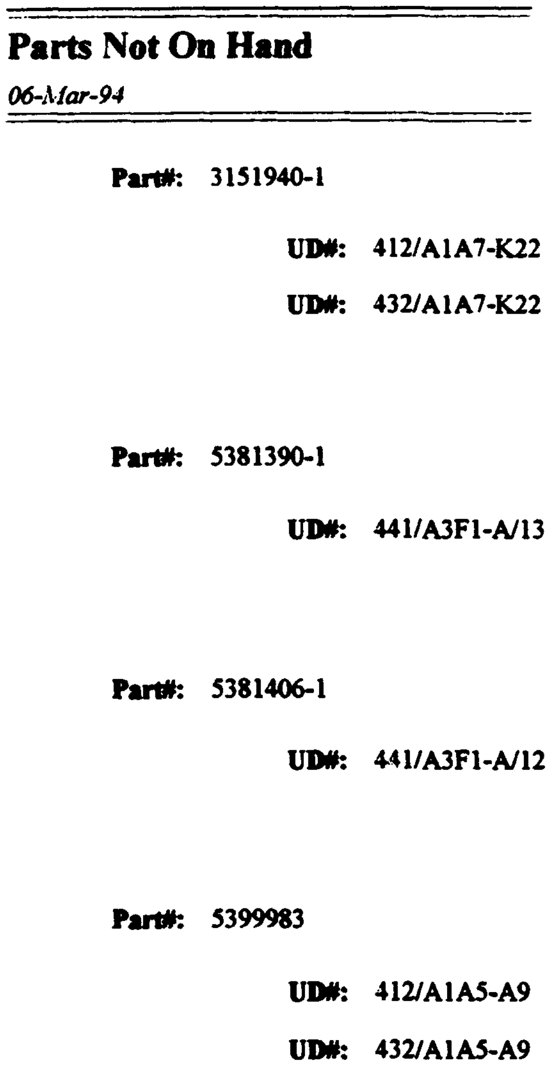
Old Node # N006

New Node #

Type in New Node Number and Change Node# or Press Cancel to Return to Previous Screen

Figure D-14 - Change Node# Form

C. SYSTEM REPORTS



Parts Not On Hand

06-Mar-94

Part#: 3151940-1

UD#: 412/A1A7-K22

UD#: 432/A1A7-K22

Part#: 5381390-1

UD#: 441/A3F1-A/13

Part#: 5381406-1

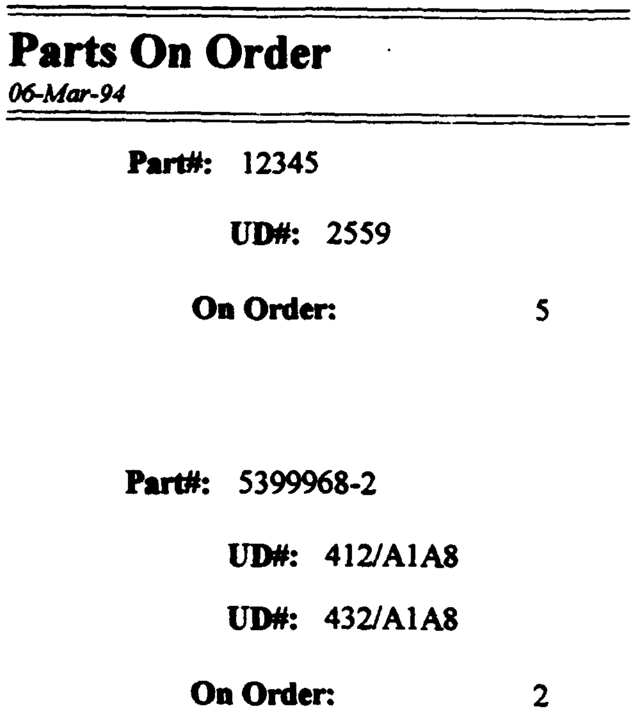
UD#: 441/A3F1-A/12

Part#: 5399983

UD#: 412/A1A5-A9

UD#: 432/A1A5-A9

Figure D-15 - Parts Not On Hand Report



Parts On Order

06-Mar-94

Part#: 12345

UD#: 2559

On Order: 5

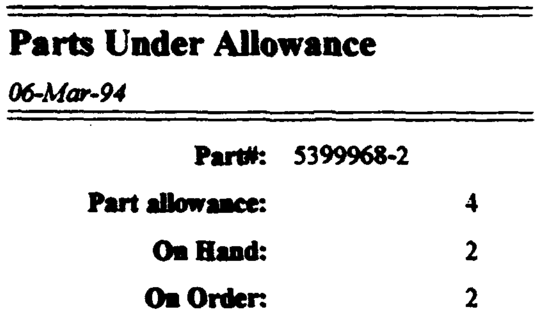
Part#: 5399968-2

UD#: 412/A1A8

UD#: 432/A1A8

On Order: 2

Figure D-16 - Parts On Order Report

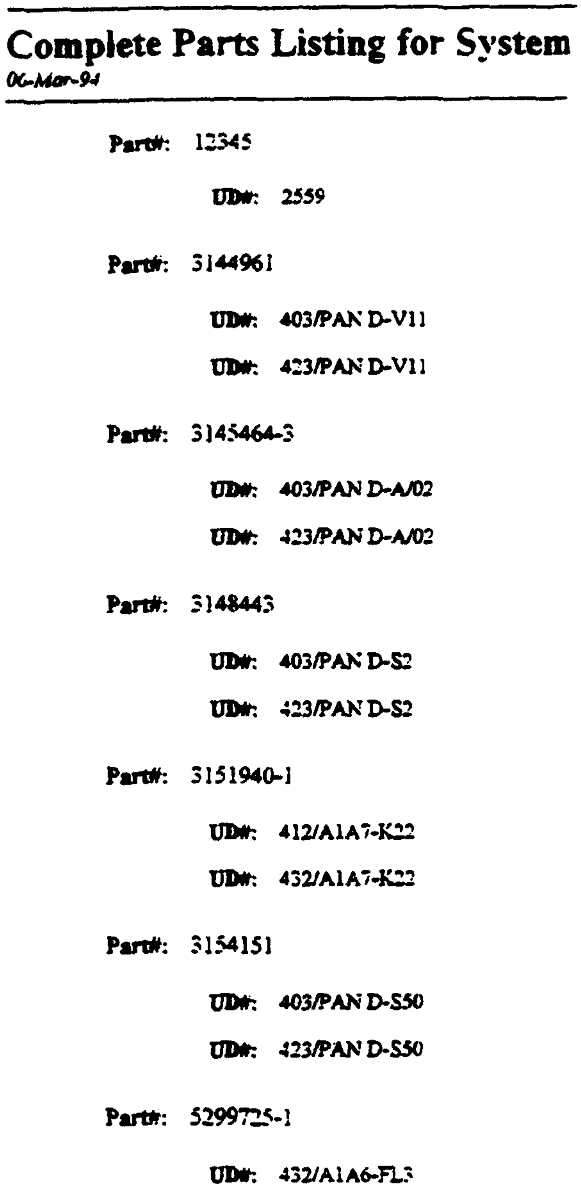


Parts Under Allowance

06-Mar-94

Part#:	5399968-2
Part allowance:	4
On Hand:	2
On Order:	2

Figure D-17 - Parts Under Stock Report



Complete Parts Listing for System

06-Mar-94

Part#: 12345

UD#: 2559

Part#: 3144961

UD#: 403/PAN D-V11

UD#: 423/PAN D-V11

Part#: 3145464-3

UD#: 403/PAN D-A/02

UD#: 423/PAN D-A/02

Part#: 3148443

UD#: 403/PAN D-S2

UD#: 423/PAN D-S2

Part#: 3151940-1

UD#: 412/A1A7-K22

UD#: 432/A1A7-K22

Part#: 3154151

UD#: 403/PAN D-S50

UD#: 423/PAN D-S50

Part#: 5299725-1

UD#: 432/A1A6-FL3

Figure D-17 - System Parts List Report

APPENDIX E. IMPLEMENTATION OF PROCESS LOGIC

A. STANDARD METHODS OF PROCESS LOGIC IMPLEMENTATION

"Accept ..." refers to a form with an unbound text box which will accept user input.

"Accept changes to..." updates the record when changes are made in a bound text box, in a form. This update is actually automatically performed, unless actions are taken to prevent it from happening.

The **"Activate ... menu"** or **"Activate... form"** is what happens when a command button is "pushed" and macro opens a form. The "On Push" property is used to invoke a macro which performs the **OpenForm** action. The name of the form must be included in the arguments, and other arguments may also be used. The **"Activate ... form WHERE [Item] = Form![FormName]! [Form Item]"** is used in "Locate" actions to locate a particular record of interest, where the form(input) item is compared against the items in the table on which the activated form is based.

"Calculate number of ..." uses a macro with the **RunCode** action to perform the required calculations. The procedure executed by this action first checks to see if the number entered by the user is a positive number, and if it is not, a message box is displayed and the user is returned to the entry screen. If the entry is a positive number then calculations are performed on the values in the appropriate fields.

"Check ... input provided and it exists" process checks to see if input is actually provided and it exists as data in the field and table specified, and then displays a message if it doesn't. This process uses both a macro and a Access Basic procedure. The "Locate" button first performs the **OpenForm** action, attempting to locate the key value the user has placed into the text box. If no match is found, Access actually brings up a record with a null value in its key field. The next action in the macro is **RunCode** which runs the procedure **CheckEmpty...()** which compares the value in the record displayed to see if the key is null, and if it is, a message box is created with a message and a command button. This command button only allows the function to continue to the next statement therein, which is to invoke a macro. This macro then performs an **OpenForm** action on the original input screen so the user may try another input value if an error was made or cancel the action.

"**Close...form**" uses the **Close** action in a macro with the argument being the name of the form which will be closed.

"**Display ... instance**" is actually part of the same implementation process as "Retrieve" or "Activate WHERE," since when Access retrieves a record using the OpenForm command with the WHERE CONDITION, it displays that instance in the form which has been opened.

"**Display ... report from ... query**" uses the macro action **OpenReport** with the arguments of the Report Name the "Print Preview" view. The report's source is the specified query. "**Display ... subreport...**" uses similar actions, however this report is displayed within another report and is invoked by the property of the subreport control on the main report. In addition, the statement "**(Field) for ... report**" designates the field to be used for the Link Master /Link Child Fields properties.

"**Display next... instances for ...**" uses the macro action **ScrollDown** to move to the next record in a group of records.

"**Display previous ... instances for ...**" uses the macro action **ScrollUp** to move to the previous record in a group of records.

"Input" generally refers to the selection of an item from a list or direct user input into the Text Box of an unbound form.

"Query... for (Query Name), select... WHERE ..." uses a query designed in the QBE environment to implement a query which looks at specified tables to produce a query of name Query Name. Specified fields are displayed and WHERE designates the criteria used for the selection of records.

"Retrieve... instance WHERE..." is implemented in the same manner as "Activate...form WHERE ..." process covered above.

"Retrieve...(related tables) instances WHERE..." is implemented using a subform/form combination which retrieves records related to the main record through the Link Fields in subform control. If

"Set initial value to..." uses the SetValue action in a macro, with the arguments of the field, and the specific value of this field to which it will be set. .

B. APPLICATION OBJECTS

1. APPLICATION TABLES AND FIELDS

PRIMARY TABLES W/ FIELDS (THOSE IMPLEMENTED)

NODE - used to store Node# and a drawing reference
FIELDS: NODE#, MODULE REF

NODE-REPL - used as intersection between NODE and REPLACEMENT also contains a documentation reference figure for that combination of Node# and UD#
FIELDS: NODE#, UD#, CKT REF

REPLACEMENT - used to store UD# related info such as Part# and an Alternate Location for that same part
FIELDS: UD#, PART#, ALT LOC, NOTES

PART - stores supply information about a part
FIELDS: PART#, NSN, PRICE, ALLOWANCE, PARTS ON HAND PARTS ON ORDER

TEMPORARY TABLES

1NX - Used for DDE Query - stores Node number
FIELDS: NODE#

NEWTX - Used for DDE Query - stores results of Make Table query
FIELDS: UD# , PART#, ALT LOC, CKT REF, NOTES

TempNode (same structure as NODE, empty, used by SQL Update)

TempNode-Repl (same structure as NODE-REPL)

TempReplacement (same structure as REPLACEMENT)

PartsList - used for System Parts Report
FIELDS: PART#

2. APPLICATION QUERIES

QUERIES

Get Supply Info - Gets information for Browse Parts Supply Information process
TABLES: PART

Parts Not On Hand - Finds parts for which Parts On Hand = 0 for Parts Not On Hand Report
TABLES: REPLACEMENT —– PART

Parts Not On Hand2 - Used for subreport of Parts Not On Hand Report, lists UD#s
TABLES: REPLACEMENT —– PART

Parts On Order - Finds parts for which Parts On Order > 0 for Parts On Order Report
TABLES: REPLACEMENT <—– PART

Parts On Order2 - Used for subreport of Parts On Order Report, lists UD#s
TABLES: REPLACEMENT <—– PART

Parts Under Stock - Finds parts for which On Hand amountt is less than allowance for Parts Under Stock Report
TABLES: PART

System Parts - Gets list of parts for System Parts List report
TABLES: PartsList

System Parts List - ACTION QUERY - Makes table of Part#s
TABLES: REPLACEMENT

System Parts - Gets list of UD#s for Parts for System Parts List subreport
TABLES: PartsList—>REPLACEMENT

1XQ - ACTION QUERY - Makes table of Parts Info for expert system request
TABLES: 1NX —NODE-REPL —– REPLACEMENT

3. APPLICATION REPORTS

R- Parts Not On Hand - Reports Parts which are not on hand and may require ordering with their respective UD#s

SOURCE: Parts Not On Hand (query)

Parts Not On Hand2 - subreport for Parts Not On Hand report

SOURCE: Parts Not On Hand2 (query)

R-Parts on Order - Reports Parts which are on order with their respective UD#s

SOURCE: : Parts On Order (query)

R-Parts On Order2 - subreport for Parts On Order report

SOURCE: Parts On Order2 (query)

R-Parts Under Stock - Reports Parts which are under allowance level

SOURCE: Parts Under Stock (query)

R-System Parts - Reports all Part#s in System, and their respective UD#s

SOURCE: System Parts (query)

R-System Parts2 - subreport for System Parts report

SOURCE: System Parts2 (query)

4. APPLICATION FORMS: CONTROLS, PROCESSES, AND PROPERTIES

1Update Node-Repl - Update Node-Repl Form from NODE - No Node# Update

RECORD SOURCE: NODE

BUTTONS:	On Push	
[Clear]	Update Macros.	Clear - Cancel Changes
[Exit]		Exit2 - Exit Form and Save
[More]		More - Save and request another Node-Repl

SPECIAL CHARACTERISTICS:

SUBFORM - Source: 1Update Node-Repl(sub)
Link Master/Child Field = Node#

1Update Node-Repl M - Update Node-Repl Form from NODE incl. Update Node button

RECORD SOURCE: NODE

BUTTONS:	On Push	
[Clear]	Update Macros.	Clear - Cancel Changes
[Exit]		Exit2 - Exit Form and Save
[More]		More - Save and request another Node-Repl
[*]		Change Node -Opens form to input new Node#

SPECIAL CHARACTERISTICS:

SUBFORM - Source: 1Update Node-Repl(sub)
Link Master/Child Field = Node#

1Update Node-Repl (sub) - View UD#s related to node, update Doc Ref info

RECORD SOURCE: NODE-REPL

BUTTONS: On Push

[Fwd] Update Macros. Scroll Down - Move to Next UD#

[Back] Scroll Up - Move to Previous UD#

SPECIAL CHARACTERISTICS:

UD# - Locked to prevent inadvertent change must use Change UD#

Node# not on form (on master form)

SUBFORM - Source: 1Update Repl (Sub Sub)

Link Master/Child Field = UD#

1Update NR-UD - Used to update UD#, Replacement info

RECORD SOURCE: REPLACEMENT

BUTTONS: On Push

[Clear] Update Macros. Clear - Cancel Changes

[Exit] Exit UD - Save and Exit Form

[More] More UD - Save and request another UD#

[*] Change UD - Use to change UD to maintain referential integrity across related tables

1Update Repl (Sub Sub) - Subform - REPLACEMENT info

RECORD SOURCE: REPLACEMENT

BUTTONS: On Push

None

DB Maint SWBD M - Admin DB Maintenance Menu Form

RECORD SOURCE: None

BUTTONS: On Push

[Ckt Card] DB-Maint SWBD - M.Ckt Card - Opens Select Change Menu

[Update UDs] Update UDs - Not implemented

[Part Info] Part Info - Not implemented

[Return] Return to Main - Return to Main Menu

DB Maint SWBD U - User DB Maintenance Menu Form

RECORD SOURCE: None

BUTTONS: On Push

[Ckt Card] DB-Maint SWBD - U. Ckt Card - Opens Select Change Menu

[Update UDs] Update UDs - Not implemented

[Part Info] Part Info - Not implemented

[Return] Return to Main - Return to Main Menu

MK92 Main Switchboard - Maint - Admin Main Menu
RECORD SOURCE: None
BUTTONS: On Push
[Part Info] MK92 Main - M. Open Parts M - Opens Part Info Menu
[Usage] Open Usage M - Opens Usage Menu
[DB] Open DB Maint-M - Opens DB Maint Menu
[Exit] Exit - Exits to Access/Opening

MK92 Main Switchboard - User - User Main Menu
RECORD SOURCE: None
BUTTONS: On Push
[Part Info] MK92 Main - U. Open Parts - Opens Part Info Menu
[Usage] Open Usage - Opens Usage Menu
[DB] Open DB-Maint U - Opens DB Maint Menu
[Exit] Exit - Exits to Access/Opening or Quits in Runtime

Node Change - used in Node-Repl update
RECORD SOURCE: NODE
BUTTONS: On Push
[Cancel] Update Macros. Cancel Node Change - Cancels Change
[Change Node#] Change Node3 - Changes Node# in NODE and NODE-REPL records

Node# Input2 - used in Node-Repl update
RECORD SOURCE: None
BUTTONS: On Push
[Cancel] Update Macros. Exit - Cancels and returns to previous screen
[Locate] Find Node2 - Used to Locate correct Node, Node-Repl, and Replacement records

Node# Input2 M - used in Node-Repl update Admin
RECORD SOURCE: None
BUTTONS: On Push
[Cancel] Update Macros. Exit M - Cancels and returns to prev screen
[Locate] Find Node2 M - Used to Locate correct Node, Node-Repl, and Replacement records

Not Implemented - used by SWBD / buttons not implemented
RECORD SOURCE: None
BUTTON: On Push
[Return] Close NI - Closes this form and returns to previous form

Opening SWBD - **Opening Administrator Menu**

RECORD SOURCE: None

BUTTONS:	On Push
[User]	Opening SWBD. Open User - Opens Main User Menu
[Admin]	Open Maint - Opens Main Admin Menu
[Exit]	Close - Exits to Access Database

Part Info SWBD - **for user to get Part Info screens/reports**

RECORD SOURCE: None

BUTTONS:	On Push
[Supply]	Part Info SWBD. Supply - Browse Part Supply Info
[Update]	Update - Update Part Supply Info
[Report]	Report - Part Supply Info Reports
[Return]	Return to Main - Returns to Main Menu

Part Info SWBD M - to get Part Info screens/reports for Administrator

RECORD SOURCE: None

BUTTONS:	On Push
[Supply]	Part Info SWBD. Supply - Browse Part Supply Info
[Update]	Update - Update Part Supply Info
[Report]	Report M - Part Supply Info Reports Admin
[Return]	Return to Main M - Returns to Main Admin Menu

Part Report SWBD - Submenu for Part Info Menu

RECORD SOURCE: None

BUTTONS:	On Push
[Not on Hand]	Part Report SWBD. Not On Hand - Lists Parts Not in Stock
[On Order]	On Order - Lists Parts/UDs on order
[Under Stock]	Under Stock - Lists Parts/UDs under allowance
[Previous]	Return to Prev - Returns to Part Info Menu
[Return]	Return to Main - Returns to Main Menu

Part Report SWBD M - Submenu for Part Info Menu Admin

RECORD SOURCE: None

BUTTONS:	On Push
[System]	Part Report SWBD. System M - Lists Parts/UDs in system
[Previous]	Return to Prev - Returns to Admin Part Info Menu
[Return]	Return to Main - Returns to Main Admin Menu

Part Supply Info - Allows update to Part Supply info except Part#

RECORD SOURCE: PART

BUTTONS:	On Push
[More]	Part Macros. More Parts - Brings up another Part record
[Cancel]	Cancel2 - Cancels and exits
[Issue]	Open Issue - Opens the Parts Issued form
[Order]	Open Order - Opens the Parts Ordered form
[Receive]	Open Receive - Opens the Parts Received form

Part Supply Info Browse - Allows browse of part supply info for a part# from list

RECORD SOURCE: Get Supply Info (query)

BUTTONS:	On Push
[Return]	Part Macros. More Parts Test - returns to previous screen

Part Supply Info Test - Allows browse of part supply info for a part# from list

RECORD SOURCE: Get Supply Info (query)

BUTTONS:	On Push
[Return]	Part Macros. More Parts Test - returns to previous screen

Parts Issued

RECORD SOURCE: None

BUTTONS:	On Push
[Update]	Part Macros. Issue Update - Updates the number of parts on hand
[Cancel]	Issue Cancel - Cancel and return to previous form

Parts Ordered

RECORD SOURCE: None

BUTTONS:	On Push
[Update]	Part Macros. Order Update - Updates number parts on order
[Cancel]	Order Cancel - Cancel and return to previous form

Parts Received

RECORD SOURCE: None

BUTTONS:	On Push
[Update]	Part Macros. Receive Update - Updates number parts on order and parts on hand
[Cancel]	Receive Cancel

Part Supply Info Test

RECORD SOURCE: Get Supply Info (query)

BUTTONS:	On Push
[Return]	Part Macros. More Parts Test - Returns to prev screen

SPECIAL CHARACTERISTICS:

Before Update: Clear Record
On Update: Clear Record
On Close: Clear Record

Part# Input

RECORD SOURCE: None
BUTTONS: On Push
[Cancel] Part Macros. Cancel - cancel and return to previous screen
[Locate] Locate Parts - Gets another Part#

Part# Input Browse

RECORD SOURCE: Get Supply Info (query)
BUTTONS: On Push
[Cancel] Part Macros. Cancel Test - Cancel and returns to prev screen
[Locate] Locate Parts Test - Gets another Part#
SPECIAL CHARACTERISTICS:

COMBO BOX:
Control Source: Part#
Row Source Type: Table/Query
Row Source: Get Supply Info (query)

Part# Input Test

RECORD SOURCE: Get Supply Info (query)
BUTTONS: On Push
[Cancel] Part Macros. Cancel Test - Cancel and return to prev screen
[Locate] Locate Parts Test - Locates another Part#
SPECIAL CHARACTERISTICS:
On Close: Clear Record

Select Change2 - Select Change Type Menu

RECORD SOURCE: None
BUTTONS: On Push
[By Node#] Update Macros. Select Node U - Allow changes by Node#
[By UD#] Select UD - allows changes by UD#
[Return] Select Cancel2 - Cancel and return to prev menu

Select Change2 M - Select Change Type Menu Admin

RECORD SOURCE: None
BUTTONS: On Push
[By Node#] Update Macros. Select Node M - Allow changes by Node#
[By UD#] Select UD - allows changes by UD#
[Return] Select Cancel2 - Cancel and return to prev menu

UD Change

RECORD SOURCE: REPLACEMENT
BUTTONS: On Push
[Cancel] Update Macros. Cancel UD Change - Cancels change
[Change UD#] Change UD3 - Process for Changing UD#

UD# Input - Allows UD# for update of Replacement info by UD#

RECORD SOURCE: None

BUTTONS:	On Push	
[Cancel]	Update Macros. Exit UD - Cancel change return to prev screen	
[Locate]		Find UD2 - Locate Replacement record for a UD#

Usage SWBD - Allows user to record and report on usage history (functions not implemented)

RECORD SOURCE: None

BUTTONS:	On Push	
[Enter]	Usage SWBD. Enter - Enter usage info	
[Retrieve]		Retrieve - retrieve usage info ad hoc
[Annual]		Annual Report - Produce formatted report
[Return]		Return to Main - Returns to Main User Menu

5. MACROS AND ARGUMENTS

AutoExec	OpenForm: MK92 Main Switchboard - User	
MK92 Main - U.	ARGUMENTS	
Open Parts	OpenForm:	Form: Part Info SWBD
Open Usage	OpenForm:	Form: Usage SWBD
Open DB Maint - U	OpenForm:	Form: DB Maint SWBD - U
Exit	Close:	Form: MK92 Main Switchboard - User
Part Info SWBD.		
Supply	OpenForm:	Form:
Update	OpenForm:	Form:
Report	OpenForm	Form:
Return to Main	Close:	Part Info SWBD
	Close:	DB Maint SWBD - U
	OpenForm:	MK92 Main - U
Part Report SWBD.		
Not on Hand	OpenReport:	Report: R-Parts Not On Hand
On Order	OpenForm:	Not Implemented
Under Stock	OpenForm:	Not Implemented
Return to Prev	Close:	Part Report SWBD
	OpenForm.	Part Info SWBD
Return to Main	Close:	Part Report SWBD
	Close:	Part Info SWBD

Usage SWBD		
Enter	OpenForm:	Form: Not Implemented
Retrieve	OpenForm:	Form: Not Implemented
Annual Report	OpenForm:	Form: Not Implemented
Return to Main	Close:	Form: Usage SWBD
DB Maint SWBD - U		
Ckt Card	OpenForm:	Form: Select Change (False)
	OpenForm:	Form: Select Change2
Help Info	OpenForm:	Form: Not Implemented
Part Info	OpenForm:	Form: Not Implemented
Return to Main	Close:	Form: DB Maint SWBD - U
Close NI	Close:	Form: Not Implemented
Warning Off	SetWarning	off
Warning On	SetWarning	on
Part Macros		
Find Parts	OpenForm:	Form: Part Supply Info Test
	SetValue:	Visible: No
Locate Parts	OpenForm:	Form: Part Supply Info Where Condition: Part# = Forms! [Part# Input]![Part#]
More Parts	Close:	Form: Part Supply Info
	SetValue:	Forms![Part# Input]![Part#] / Null
	Close:	Form: Part# Input
	OpenForm:	Form: Part# Input
Cancel	SetValue:	Forms![Part# Input]![Part#] / Null
	Close:	Form: Part# Input
Cancel2	SetValue:	Forms![Part# Input]![Part#] / Null
	Close:	Form: Part Supply Info
	Close:	Form: Part# Input
More Parts Test	Close:	Form: Part Supply Info Test
	OpenForm:	Form: Part# Input Test (Read Only)
Locate Parts Test	SetValue:	Visible/No
	OpenForm:	Form: Part Supply Info Test Where Condition: Part# = Forms! [Part Input Test]![Part#]
Cancel Test	Close:	Form: Part# Input Test
Cancel2 Test	Close:	Form: Part Supply Info Test
	Close:	Form: Part# Input Test

Finish	Close:	Form: Part Supply Info Test
	OpenForm:	Form: Part# Input Test (Read Only)
	Close:	Form: Part# Input Test
Open Order	OpenForm:	Form: Parts Ordered
	SetValue:	Forms![Parts Ordered]![OrderAmt] / 0
Open Issue	OpenForm:	Form: Parts Issued
	SetValue:	Forms![Parts Issued]![IssueAmt] / 0
Open Receive	OpenForm:	Form: Parts Received
	SetValue:	Forms![Parts Received]![ReceiveAmt]/ 0
Issue Update	RunCode:	IssueParts()
Issue Cancel	Close:	Form: Parts Issued
Order Update	RunCode:	OrderParts()
Order Cancel	Close:	Form: Parts Ordered
Receive Update	RunCode:	ReceiveParts()
Receive Cancel	Close:	Form: Parts Received

Update Macro

Find Node	SetValue:	Visible/NO
(uses Query)	OpenForm:	Form: 1Update Node-Repl (Test) Where Condition: Node#=Forms! [Node# Input]![Node#]
Find Node2	SetValue:	Visible/NO
	OpenForm:	Form: 1Update Node-Repl Where Condition: Node#=Forms! [Node# Input2]![Node#]
	RunCode:	CheckEmptyNode()
More	Close:	Form: 1Update Node-Repl
	SetValue:	Forms![Node# Input2]![Node#] / Null
	Close:	Form: Node# Input2
	OpenForm:	Form: Node# Input2
Exit	SetValue:	Forms![Node# Input2]![Node#] / Null
	Close:	Form: Node# Input2
	OpenForm:	Form: Select Change (False)
	OpenForm:	Form: Select Change2
Save	SetValue:	Forms![Node# Input2]![Node#] / Null
	Close:	Form: 1Update Node-Repl
	Close:	Form: Node# Input2
Exit2	SetValue:	Forms![Node# Input2]![Node#] / Null
	Close:	Form: 1Update Node-Repl
	Close:	Form: Node# Input2
	Close:	Form: Node# Input Test
	Close:	Form: Select Change (False)
	Close:	Form: Select Change2

Clear	SendKeys:	{Esc} / Yes
Change Node	OpenForm:	Form: Node Change Where Condition: [Node#]=Forms! [1Update Node-Repl]![Node#]
Change Node2	Echo -	Off "Getting Node Informatiom"
	RunCode:	AskUpdateNode ()
	RunCode:	UpdateRelatedFields ()
	Close:	Form: Node Change
Change Node3	Echo -	Off "Getting Circuit Card Information"
	RunCode:	AskUpdateNode () (False)
	RunCode:	UpdateRelatedNode ()
	Close:	Form: 1Update Node-Repl
	OpenForm:	Form: 1Update Node-Repl Where Condition: [Node#] = Forms! [Node Change]![NewNode]
	Close:	Form: Node Change
Reset Node#	SetValue:	(doesn't work)
Not Implemented	OpenForm:	Form: Not Implemented
Change UD	Close:	Form: 1Update Node-Repl (False)
	OpenForm:	Form: UD Change Where Condition: [UD#]=Forms! [1Update NR-UD]![UD#]
Change UD3	Echo	/ Off "Getting Replacement Information..."
	RunCode:	AskUpdateUD () (False)
	RunCode:	UpdateRelatedUD ()
	Close:	Form: 1Update UD-NR
	OpenForm:	Form: 1Update Node-Repl! Where Condition: [Node#]=Forms! [UD Change]![NewUD]
	Close:	Form: UD Change
Find UD2	SetValue:	Visible / NO (False)
	OpenForm:	Form: 1Update NR-UD Where Condition: [UD#]=Forms! [UD# Input]![UD#]
	RunCode:	CheckEmptyUD ()
Exit UD	Set Value:	Forms![UD# Input]![UD#] / Null
	Close:	Form: 1Update NR-UD
	Close:	Form: UD# Input
	OpenForm:	Form: Select Change (False)
	OpenForm:	Form: Select Change2

Select Node	Close:	Form: Select Change (False)
	SetValue:	Visible / No
	OpenForm:	Form: Node# Input2
Select UD	Close:	Form: Select Change (False)
	SetValue:	Visible / No
	OpenForm:	Form: UD# Input
Select Std	Close:	Form: Select Change (False)
	SetValue:	Visible / No
	OpenForm:	Form: Node# Input2
Select Cancel	Close:	Form: Select Change
Cancel Node Change	Close:	Form: Node Change
	SendKeys:	{Esc} / Yes
Cancel UD Change	Close:	Form: UD Change
	SendKeys:	{Esc} / Yes
Select Cancel2	Close:	Form: Select Change2
More UD	Close:	Form: 1Update NR-UD
	SetValue:	Forms![UD# Input]![UD#] / Null
	Close:	Form: UD# Input
	OpenForm:	Form: UD# Input

6. APPLICATION ACCESS BASIC MODULES

UPDATES

(Declarations)

```
Option Compare Database   'Use database order for string comparison

   Dim UpdateUD, UpdateNode
   Dim OldUD, OldNode, NewUD, NewNode, NodeX
   Dim MK92 As Database
   Dim wID%
```

Function AskUpdateNode ()

```
   OldNode = Forms![Node Change]![Node#]
   NewNode = Forms![Node Change]![NewNode]
   UpdateNode = False

   If MsgBox("Are you sure you want to change " & OldNode & " to " & NewNode &
"?", 292) = 6 Then
      UpdateNode = True
   End If
   If UpdateNode = False Then
```

```
      NewNode = Null
   End If
End Function

Function AskUpdateUD ()

   OldUD = Forms![UD Change]![UD#]
   NewUD = Forms![UD Change]![NewUD]
   UpdateUD = False

   If MsgBox("Are you sure you want to change " & OldUD & " to " & NewUD & "?",
292) = 6 Then
      UpdateUD = True
   End If

End Function

Function UpdateRelatedNode ()
' Changes UD# in related records in all tables
' Changes NODE# in related records in all tables
' Written S.Talley 1/10/93

   OldNode = Forms![Node Change]![Node#]
   NewNode = Forms![Node Change]![NewNode]
   NodeX = NewNode

   Set MK92 = CurrentDB()
   Dim Node As Dynaset, NodeRepl As Dynaset, Replacement As Dynaset
   Set Node = MK92.CreateDynaset("NODE")
   Set NodeRepl = MK92.CreateDynaset("NODE-REPL")
   Set Replacement = MK92.CreateDynaset("REPLACEMENT")

If MsgBox("Are you sure you want to change " & OldNode & " to " & NewNode & "?",
292) = 6 Then
   Criteria = "[Node#] = '" & NewNode & "'"
   Node.FindFirst Criteria
   If Not Node.Nomatch Then
      MsgBox (NewNode & " is already in use as an Node#. Please enter another Node#")
      Exit Function

   End If

   DoCmd RunMacro "Warning Off"
```

```
      DoCmd RunSQL "SELECT  * INTO TempNode FROM NODE WHERE [Node#]
= Forms![Node Change]![Node#];"
      DoCmd RunSQL "UPDATE TempNode SET [Node#] = Forms![Node
Change]![NewNode];"
      DoCmd RunSQL "INSERT Into Node SELECT * FROM TempNode;"
      DoCmd RunSQL "DELETE * FROM TempNode WHERE [Node#] = Forms![Node
Change]![Node#];"

   Criteria = "[Node#] = '" & OldNode & "'"

   NodeRepl.FindFirst Criteria
   If Not NodeRepl.Nomatch Then
      Do While NodeRepl.[Node#] = OldNode
         NodeRepl.Edit
         NodeRepl.[Node#] = NewNode
         NodeRepl.Update
         NodeRepl.FindNext Criteria
      Loop
   End If

   Node.FindFirst Criteria
   Node.Delete
   Criteria = "[Node#] = '" & NewNode & "'"
   Node.FindFirst Criteria
Else
   Forms![Node Change]![NewNode] = Forms![Node Change]![Node#]
End If

   DoCmd RunMacro "Warning On"

End Function

Function CheckEmptyNode ()

   NodeX = Forms![1Update Node-Repl]![Node#]
   If IsNull(NodeX) Then
      MsgBox ("No matching node found (or No Node# entered)")
      DoCmd RunMacro "Update Macros.More"
   End If

End Function

Function CheckEmptyUD ()
```

```
   UDX = Forms![1Update NR-UD]![UD#]
   If IsNull(UDX) Then
      MsgBox ("No matching UD# found (or No UD# entered)")
      DoCmd RunMacro "Update Macros.More UD"
   End If

End Function

Function UpdateRelatedUD ()
' Changes UD# in related records in all tables
' Written S.Talley 1/10/93

   OldUD = Forms![UD Change]![UD#]
   NewUD = Forms![UD Change]![NewUD]

   Set MK92 = CurrentDB()
   Dim Node As Dynaset, NodeRepl As Dynaset, Replacement As Dynaset
   Set Node = MK92.CreateDynaset("NODE")
   Set NodeRepl = MK92.CreateDynaset("NODE-REPL")
   Set Replacement = MK92.CreateDynaset("REPLACEMENT")

'  If UpdateNode = True Then
If MsgBox("Are you sure you want to change " & OldUD & " to " & NewUD & "?", 292)
= 6 Then
   Criteria = "[UD#] = '" & NewUD & "'"
   Replacement.FindFirst Criteria
   If Not Replacement.Nomatch Then
      MsgBox (NewUD & " is already in use as an UD#. Please enter another UD#")
      Exit Function
   End If

   DoCmd RunMacro "Warning Off"
      DoCmd RunSQL "SELECT  * INTO TempReplacement FROM REPLACEMENT
WHERE [UD#] = Forms![UD Change]![UD#];"
      DoCmd RunSQL "UPDATE TempReplacement SET [UD#] = Forms![UD
Change]![NewUD];"
      DoCmd RunSQL "INSERT Into Replacement SELECT * FROM
TempReplacement;"
      DoCmd RunSQL "DELETE * FROM TempReplacement WHERE [UD#] =
Forms![UD Change]![UD#];"

   Criteria = "[UD#] = '" & OldUD & "'"
   NodeRepl.FindFirst Criteria
   If Not NodeRepl.Nomatch Then
      Do While NodeRepl.[UD#] = OldUD
         NodeRepl.Edit
         NodeRepl.[UD#] = NewUD
         NodeRepl.Update
         NodeRepl.FindNext Criteria
      Loop
   End If
   Replacement.FindFirst Criteria
   Replacement.Delete
   Criteria = "[UD#] = '" & NewUD & "'"
   Replacement.FindFirst Criteria
Else
   Forms![UD Change]![NewUD] = Forms![UD Change]![UD#]
End If

   DoCmd RunMacro "Warning On"

End Function
```

SUPPLY

(Declarations)

```
Option Compare Database   'Use database order for string comparisons

Dim NewOrder, OldOrder As Integer
Dim OrderAmt
Dim MK92 As Database
Dim NewIssue, OnHand, OldIssue, NewOnHand As Integer
Dim IssueAmt
Dim NewReceive, OldReceive As Integer
Dim ReceiveAmt
```

Function IssueParts ()

```
OnHand = Forms![Part Supply Info]![Parts on Hand]
IssueAmt = Forms![Parts Issued]![IssueAmt]
CheckNumber = True
CheckPositive = True

'Check to see if IssueAmt is an integer
If (Not IsNumeric(IssueAmt)) Then
```

```
   MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel.")
   CheckNumber = False
   DoCmd GoToControl "IssueAmt"
End If

'If IssueAmt is a number, then check to see if it's positive
If CheckNumber = True Then
   If IssueAmt < 0 Then
      MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel. (2)")
      CheckPositive = False
      DoCmd GoToControl "IssueAmt"
   End If
End If
If (CheckPositive = True) And (CheckNumber = True) Then

   NewOnHand = OnHand - ReceiveAmt
'  Check to see if amt on hand (NewOnHand) < 0, if so, sent message and don't update
   If NewOnHand < 0 Then
      MsgBox ("Amount issued is more than that on hand.  Check part supplies and
correct entry.")
   Else
      Forms![Part Supply Info]![Parts on Hand] = NewOnHand
   End If
   DoCmd Close

Else
   Forms![Parts Issued]!IssueAmt = 0

End If

End Function
```

Function OrderParts ()

```
OldOrder = Forms![Part Supply Info]![Parts on Order]
OrderAmt = Forms![Parts Ordered]![OrderAmt]
CheckNumber = True
CheckPositive = True

'Check to see if OrderAmt is an integer
If (Not IsNumeric(OrderAmt)) Then
```

```
   MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel.")
   CheckNumber = False
   DoCmd GoToControl "OrderAmt"
End If

'If OrderAmt is a number, then check to see if it's positive
If CheckNumber = True Then
   If OrderAmt < 0 Then
      MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel. (2)")
      CheckPositive = False
      DoCmd GoToControl "OrderAmt"
   End If
End If
If (CheckPositive = True) And (CheckNumber = True) Then

   NewOrder = OldOrder + OrderAmt
   Forms![Part Supply Info]![Parts on Order] = NewOrder
   DoCmd Close

Else
   Forms![Parts Ordered]!OrderAmt = 0
End If
End Function
```

Function ReceiveParts ()

```
OnHand = Forms![Part Supply Info]![Parts on Hand]
OldOrder = Forms![Part Supply Info]![Parts on Order]
ReceiveAmt = Forms![Parts Received]![ReceiveAmt]
CheckNumber = True
CheckPositive = True
'Check to see if ReceiveAmt is an integer
If (Not IsNumeric(ReceiveAmt)) Then
   MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel.")
   CheckNumber = False
   DoCmd GoToControl "ReceiveAmt"
End If

'If ReceiveAmt is a number, then check to see if it's positive
If CheckNumber = True Then
```

```
  If ReceiveAmt < 0 Then
    MsgBox ("Amount entered must be a positive number.  Delete entry and try again or
cancel. (2)")
    CheckPositive = False
    DoCmd GoToControl "ReceiveAmt"
  End If
End If
If (CheckPositive = True) And (CheckNumber = True) Then
'  Update Amt on Hand (Add Recived to On Hand)
  NewOnHand = OnHand + ReceiveAmt
'  Check to see if amt on hand (NewOnHand) < 0, if so, sent message and don't update
  If NewOnHand < 0 Then
    MsgBox ("Amount on hand is less than zero.  Check part supplies and correct
entry.")
  Else
'    Update Amt on Order (Subtract Received to On Hand)
    NewOrder = OldOrder - ReceiveAmt
    ' Check to see if this is < 0, send message if true
    If NewOrder < 0 Then
      MsgBox ("Amount on order has been calculated as less than zero.  Check
outstanding orders and correct entry. (Value has been reset to 0).")
      NewOrder = 0
    End If
    Forms![Part Supply Info]![Parts on Hand] = NewOnHand
    Forms![Part Supply Info]![Parts on Order] = NewOrder
  End If
  DoCmd Close
Else
  Forms![Parts Received]!ReceiveAmt = 0
End If
End Function
```

Function CheckEmptyPart ()

```
  PartX = Forms![Part Supply Info]![Part#]
  If IsNull(PartX) Then
    wId = MsgBox("No matching Part # found (or No Part # entered)", 64, "No Match
Found")

    DoCmd RunMacro "Part Macros.More Parts"
  End If
End Function
```

APPENDIX F. DDE INTERFACE DOCUMENTATION

A. ACCESS BASIC LANGUAGE ELEMENTS FOR DDE

Access Basic Language Element	Element Type	Purpose	Available From
DDE()	Function	Initiates a DDE process with another application and returns the requested information	
DDEInitiate()	Function	Initiates a DDE and conversation with another application	Access Basic form controls
DDERequest()	Function	Requests an item from a DDE server application	Access Basic form controls
DDESend()	Function	Initiates a DDE process with another application and sends data to the specified item in that application	Record Source property of certain controls only
DDEExecute()	Statement	Sends a command to another application over an established DDE channel	Access Basic and form controls
DDEPoke()	Statement	Sends data to another application over an established channel	Access Basic and form controls
DDETerminate()	Statement	Closes a specified DDE conversation channel	Access Basic and form controls
DDETerminateAll()	Statement	Closes all open DDE conversation channels	Access Basic and form controls

(Perschke, 1992, p. 244)

B. ADEPT FUNCTIONS FOR DDE

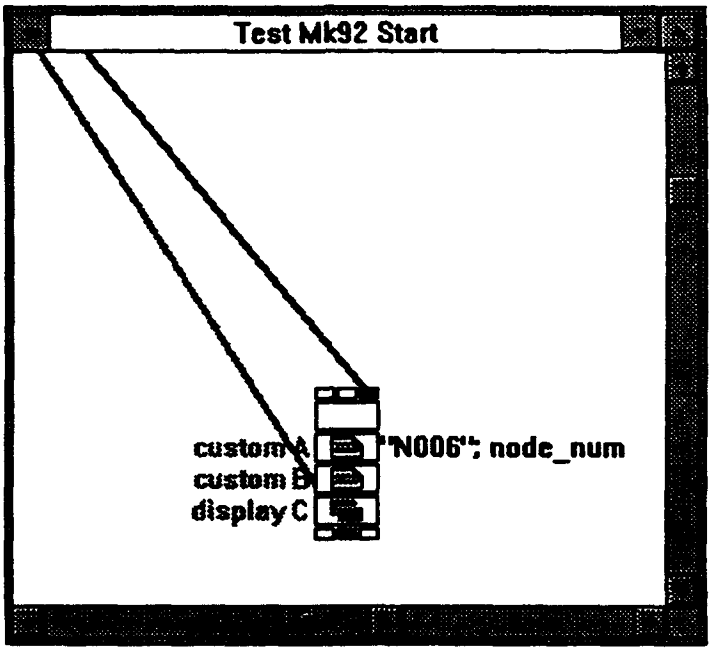
Function	Description
Advise	Asks a server program to send a data item to Adept each time it changes
CloseAllChannels	Closes all open channels
CloseChannels	Closes an open channel
Execute	Sends an executable command to a server program
Notify	Asks a server program to notify Adept each time a data item changes
OpenChannel	Opens a channel to a server program
Poke	Sends a data item to a server program
Request	Asks a server program to send a data item to Adept
Unadvise	Asks a server program to stop sending a data item to Adept each time it changes

(Symbologic Corporation, 1991,

C. ADEPT AND ACCESS APPLICATION INTERFACE PROCESSES

1. Adept Expert System Interface Node And Scripts

a. Interface Node



Custom Inteface Node

Figure F-1

b. Interface Node Scripts

Part 1:

```
%2 = %1;
Node=%2;

// This opens a channel to Access and runs a Macro - current one is called "1TableOnly",
//     which takes the node number (already stored for testing) and runs a "make table
//     query", which gets the correct data for that node from the database - variable Node
//     is not used for anything at this time

AccCall = OpenChannel("MK92T2","D:\mk92db2\MK92T2.MDB");
Execute (AccCall, "[1TableOnly]");
CloseChannel(AccCall)
```

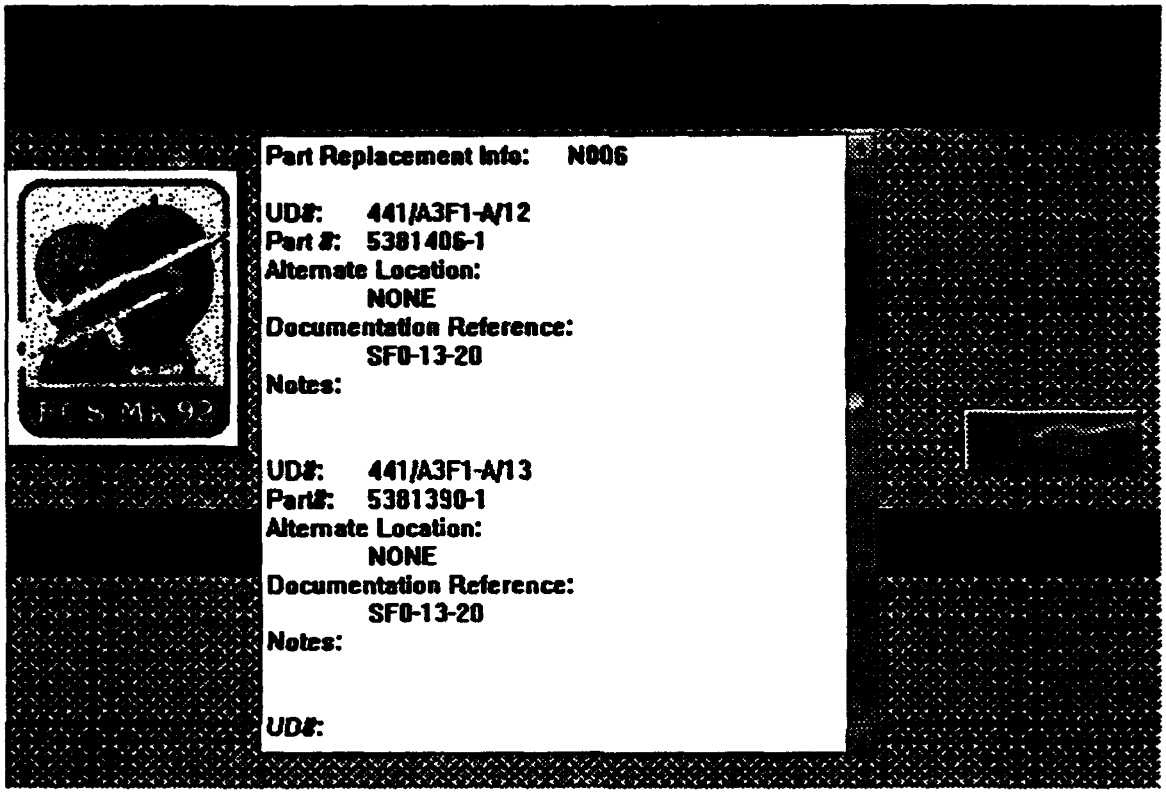
Part II:

```
// Once the table is created, this code opens a channel and pulls it from Access
//     The Open Channel statement is dependent on the path and must be correct for
//     the database being used

AccTalk2=OpenChannel("MK92T2","d:\Mk92db2\mk92t2.mdb;TABLE NEWTX");
// These variables are used to determine how many rows (records) exist
Rows=1;
Row2="999";
Row3="999";
Row4="999";
row=0;

// These statements get the value of the records and check to see how many
//    records exist
Request(AccTalk2,"FirstRow",Row1);
Request(AccTalk2, "NextRow",Row2);
Request(AccTalk2, "NextRow",Row3);
Request(AccTalk2, "NextRow",Row4);
If Row4 == "999" then Rows=3 else Rows=4;
If Row3 == "999" then Rows=2;
If Row2 == "999" then Rows=1;

// These statements assign values of Records to variables to break into fields
//     For empty records this program substitutes the following statement
//     RowX="<CTRL><TAB> <CTRL><TAB> <CTRL><TAB> <CTRL><TAB>
//     <CTRL><TAB>*" -- this eliminates the <NO VALUE> in the display, which
//     otherwise occurs if a RowX (or any other row) does not really exist (5 combinations
//     of <CTRL><TAB> and space fill the empty fields correcly with blank fields

if (Rows==4) then
{
   RowA=Row1;
   RowB=Row2;
   RowC=Row3;
   RowD=Row4;
}
else
if (Rows==3) then
{
   RowA=Row1;
   RowB=Row2;
   RowC=Row3;
   RowD="                                   *";
}
else
if (Rows==2) then
{
   RowA=Row1;
   RowB=Row2;
   RowC="                                   *";
   RowD="                                   *";
}
else
if (Rows==1) then
{
   RowA=Row1;
   RowB="                                   *";
   RowC="                                   *";
   RowD="                                   *";
};
// these statements take the value of each field and assign to variables for 4 rows
until (row == 4) do
{
col=0;
row=row + 1;
nx=6;
LX=0;
If (row == 1) then
{
```

```
Length=FindText(RowA,"*");
Row1Info=GetSubText(RowA,1,Length);
R0=GetSubText(RowA,1,Length);
};

If (row == 2) then
{
Length=FindText(RowB,"*");
Row1Info=GetSubText(RowB,1,Length);
R0=GetSubText(RowB,1,Length);
};

If (row == 3) then
{
Length=FindText(RowC,"*");
Row1Info=GetSubText(RowC,1,Length);
R0=GetSubText(RowC,1,Length);
};
If (row == 4) then
{
Length=FindText(RowD,"*");
Row1Info=GetSubText(RowD,1,Length);
R0=GetSubText(RowD,1,Length);
};

// This starts the loop looking at fields
do
{
   col=col+1;

L0=FindText(R0,"      ");
F0=GetSubText(R0,1,L0-1);
LX=L0+LX;
R0=GetSubText(R0,L0+1,Length-LX);


if col==1 then if (row==1) then (F11=F0) else if (row==2) then (F21=F0) else if (row==3)
then (F31=F0) else (F41=F0);

if (col==2) then if (row==1) then (F12=F0) else if (row==2) then (F22=F0) else if
(row==3) then (F32=F0) else (F42=F0);
```

```
if (col==3) then if (row==1) then (F13=F0) else if (row==2) then (F23=F0) else if
(row==3) then (F33=F0) else (F43=F0);

if (col==4) then if (row==1) then (F14=F0) else if (row==2) then (F24=F0) else if
(row==3) then (F34=F0) else (F44=F0);

if (col==5) then if (row==1) then (F15=F0) else if (row==2) then (F25=F0) else if
(row==3) then (F35=F0) else (F45=F0)

}
while col<nx;

};
CloseChannel(AccTalk2)
```

c. Adept Display of Part Information



```
Part Replacement Info:    N006

UD#:     441/A3F1-A/12
Part #:  5381406-1
Alternate Location:
         NONE
Documentation Reference:
         SF0-13-20
Notes:


UD#:     441/A3F1-A/13
Part#:   5381390-1
Alternate Location:
         NONE
Documentation Reference:
         SF0-13-20
Notes:


UD#:
```

Adept Display of Part Information from Database

Figure F-2

2. Database Application Procedures and Macros

Macro:

1NewTxQuery

Actions: OpenQuery: Query: 1XQ

1TestDDENodeReq

Actions: RunCode: Function: GetNode()

1MakeTable

```
     Actions:   RunCode:   Function:  GetNode()

                RunCode:   Function:  MakeTable()
```

1TableOnly

```
     Actions:   RunCode:   Function:  MakeTable()
```

WarningOn

```
     Actions:   SetWarning: On
```

WarningOff

```
     Actions:   SetWarning: Off
```

Modules:
Procedure DDE
Declarations

```
Option Compare Database   'Use database order for string comparisons

  Dim NodeNum
  Dim MK92 As Database
```

Function GetInfo ()

```
  DoCmd RunMacro "1MakeTable"

End Function
```

Function GetNode ()

```
  ChannelNum = DDEInitiate("Adept", "Test92")

 NodeNum = DDERequest(ChannelNum, "VARIABLE Node")
 Dim db As Database, T As Table

  Set db = CurrentDB()
  Set T = db.OpenTable("1NX")
  T.MoveFirst
  T.Delete
```

```
    T.AddNew
    T![Node#] = NodeNum
    T.Update
    T.Close
    DDETerminate ChannelNum

End Function

Function MakeTable ()

    DoCmd RunMacro "Warning Off"

     DoCmd RunMacro "1NewTx Query"
    DoCmd RunMacro "Warning On"
End Function
```

NOTE: in this function, statements which begin with ' are not operational (comment lines)

```
Function MakeTableXX ()

    DoCmd RunMacro "Warning Off"
'   Set MK92 = CurrentDB()

'   Dim NewT As Table

'   Set NewT = MK92.OpenTable("NEWTX")
'   NewT.MoveFirst
'   Do Until NewT.EOF
'       NewT.Delete
'       NewT.MoveNext
'   Loop
'   NewT.Close

' THIS DIDN'T WORK
'   Dim NX As Dynaset, NewTable As Dynaset, NodeRepl As Dynaset, REPLACEMENT
        As Dynaset
'   Set NX = MK92.CreateDynaset("1NX")
'   Set NodeRepl = MK92.CreateDynaset("NODE-REPL")
'   Set REPLACEMENT = MK92.CreateDynaset("REPLACEMENT")
'   Set NewTable = MK92.CreateDynaset("NEWTX")
' THESE DIDN"T WORK
'   DoCmd RunSQL "SELECT  * INTO TempTable FROM [NODE-REPL],
        REPLACEMENT, 1NX, REPLACEMENT INNER JOIN [NODE-REPL] ON
        REPLACEMENT.[UD#] = [NODE-REPL].[UD#],1NX INNER JOIN [NODE-
        REPL] ON [1NX].[Node#] = [NODE-REPL],[Node#];"
'   DoCmd RunSQL "UPDATE TempTable SET [Node#] = Forms![Node
        Change]![NewNode];"
'   DoCmd RunSQL "INSERT Into NEWTX SELECT * FROM TempTable;"
'      DoCmd RunSQL "DELETE * FROM TempNode WHERE [Node#] =
        Forms![Node Change]![Node#];"
'  THIS DIDNT WORK EITHER
'   DoCmd RunSQL "SELECT  DISTINCTROW REPLACEMENT.* INTO NEWTX
        FROM [NODE-REPL], REPLACEMENT, 1NX, REPLACEMENT INNER
        JOIN [NODE-REPL] ON REPLACEMENT.[UD#] = [NODE-
        REPL].[UD#],1NX INNER JOIN [NODE-REPL] ON [1NX].[Node#] = [NODE-
        REPL],[Node#];"
'   DoCmd RunSQL "SELECT DISTINCTROW * INTO NEWTX FROM [NODE-
        REPL], REPLACEMENT, 1NX, REPLACEMENT INNER JOIN [NODE-
        REPL] ON REPLACEMENT.[UD#] = [NODE-REPL].[UD#],1NX INNER
        JOIN [NODE-REPL] ON [1NX].[Node#] = [NODE-REPL],[Node#];"
'   DoCmd RunSQL "SELECT * INTO NEWTX FROM [NODE-REPL],
        REPLACEMENT, 1NX, REPLACEMENT INNER JOIN [NODE-REPL] ON
        REPLACEMENT.[UD#] = [NODE-REPL].[UD#],1NX INNER JOIN [NODE-
        REPL] ON [1NX].[Node#] = [NODE-REPL],[Node#];"

DoCmd RunMacro "Warning On"
End Function
```

LIST OF REFERENCES

Campbell, T. and Hudnall, M., Ed., "Test Lab," *Compute*, pp. 16-36, August 1993.

Coffee, P, "Super Databases," *PC Computing*, pp. 270-297, October 1993.

Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989.

Jennings, R., *Access^{TM} Developer's Guide*, SAMS Publishing, 1993.

Jones, E., *Ready-Made Access Applications*, Windcrest/McGraw-Hill, 1994.

Kroenke, D. M., *Database Processing: fundamentals, design, implementation*, Macmillan Publishing Company, 1992.

Microsoft Corporation, *Microsoft Access Language Reference*, Microsoft Press, 1992.

Microsoft Corporation, *Microsoft Access User's Guide*, Microsoft Press, 1992.

Perschke, S. and Liczbanski, M., *Access for Windows Power Programming*, Que Corporation, 1993.

Smith, D. C., *Development of a Maintenance advisor Expert System for the MK 92 MOD 2 Fire Control System: FC-1 Designation - Time, Range, Bearing FC-1 Acquisition, FC-1 Track - Range, Bearing, and FC-2 Designation - Time, Range, Bearing, FC-2 Acquisition, FC-2 Track - Range, Bearing, and FC-4 and FC-5*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

Symbologic Corporation, *Symbologic Adept^{TM} Reference*, Symbologic Corporation, 1991.

Whitten, J. L., Bentley, L. D., and Barlow, V. M., *Systems Analysis and Design Methods*, Richard D. Irwin, Inc., 1989.

BIBLIOGRAPHY

Campbell, T. and Hudnall, M., Ed., "Test Lab," *Compute*, pp. 16-36, August 1993.

Coffee, P, "Super Databases," *PC Computing*, pp. 270-297, October 1993.

Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989.

Jennings, R., *Access^{TM} Developer's Guide*, SAMS Publishing, 1993.

Jones, E., *Ready-Made Access Applications*, Windcrest/McGraw-Hill, 1994.

Kroenke, D. M., *Database Processing: fundamentals, design, implementation*, Macmillan Publishing Company, 1992.

Liskin, M., *HELP! Microsoft Access*, Ziff-Davis Press, 1993.

Microsoft Corporation, *Microsoft Access Language Reference*, Microsoft Press, 1992.

Microsoft Corporation, *Microsoft Access User's Guide*, Microsoft Press, 1992.

Perschke, S. and Liczbanski, M., *Access for Windows Power Programming*, Que Corporation, 1993.

Simpson, A., *Understanding Microsoft Access*, Sybex, Inc., 1993.

Smith, D. C., *Development of a Maintenance advisor Expert System for the MK 92 MOD 2 Fire Control System: FC-1 Designation - Time, Range, Bearing FC-1 Acquisition, FC-1 Track - Range, Bearing, and FC-2 Designation - Time, Range, Bearing, FC-2 Acquisition, FC-2 Track - Range, Bearing, and FC-4 and FC-5*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

St. Valentine, C., *Access Basic Cookbook*, Addison-Wesley Publishing Company, 1993.

Symbologic Corporation, *Symbologic Adept^{TM} Reference*, Symbologic Corporation, 1991.

Viescas, J. L., *Running Microsoft Access^{TM}*, Microsoft Press, 1993.

Whitten, J. L., Bentley, L. D., and Barlow, V. M., *Systems Analysis and Design Methods*, Richard D. Irwin, Inc., 1989.

INITIAL DISTRIBUTION LIST

1. Defense Tecnical Information Center 2
Cameron Station
Alexandria, VA 22304-6154

2. Library, Code 052 2
Naval Postgraduate School
Monterey, CA 93943-5000

3. Capt. O. H. Perry III 1
Naval Sea Systems Command
Code 62Z, NC3, Room 8W06
2531 Jefferson Davis Highway
Washington, DC 22243-5160

4. Mr. Ed McGill 1
Naval Sea Systems Command
Code 62ZP, NC3, Room 8W06
2531 Jefferson Davis Highway
Washington, DC 22243-5160

5. FCC Stein 1
Naval Sea Systems Command
Code 62ZP, NC3, Room 8W06
2531 Jefferson Davis Highway
Washington, DC 22243-5160

6. CMDR A.M. Joseph 1
Port Hueneme Division
Naval Surface Warfare Center
Code 4A00
Port, Hueneme, CA 93043

7. Mr. Bill Campbell 1
Port Hueneme Division
Naval Surface Warfare Center
Code 4A32
Port, Hueneme, CA 93043

8. Mr. Henry Seto 1
Port Hueneme Division
Naval Surface Warfare Center
Code 4A32
Port Hueneme, CA 93043

9. Professor Magdi Kamel, Code AS/Ka 2
Naval Postgraduate School
Monterey, CA 94043

10. Professor Martin McCaffrey, Code AS/Mf 1
Naval Postgraduate School
Monterey, CA 93943-5000

11. LCDR Susan G. Talley 1
COMNAVFORKOREA
UNIT # 15250
APO AP 96205-0023

12. LT Janie Crawford, Code 37 1
Naval Postgraduate School
Monterey, CA 93943-5000