SANDIA REPORT

SAND89—1619 • UC—705
Unlimited Release
Printed November 1989

Connector Selection Program Implementation Notes

Version 1.0

Nicole E. Sevier

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A06
Microfiche copy: A01

SAND89-1619
Unlimited Release
Printed November 1989

Distribution
Category UC-705

SAND--89-1619

DE90 004983

Connector Selection Program Implementation Notes Version 1.0

Nicole E. Sevier
Interconnections Division
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

The connector selection program is a database application that allows engineers to locate information about connectors that meet their requirements. This document describes the design and implementation of the database, the data input application, and the user interface.

Nijssen's Information Analysis Methodology (NIAM) was used to characterize the connector data requirements which yielded the database design. This design was transformed into database record structures that were implemented in the relational database management software ORACLE. After the database was in place, data input screens were created to capture the connector data, analyze it, and place it in the proper database record structures. Finally, a user interface was designed and developed that displays or prints the information contained in the database, associated drawings, and documentation related to the program and its data.

ACKNOWLEDGMENTS

I would like to thank the many people that helped in creating the connector selection program. Special thanks to:

Everyone in the Interconnections Division, Ed Machin (retired), Ed Ehrman (retired), and Frank Daut (retired) for their help designing the connector database and testing the user interface.

John Sharp, 2825 for informing me about Nijssen's Information Analysis Methodology (NIAM), for providing an opportunity to for me to learn NIAM, and for his continual support and encouragement.

Mike Schaefer, Michele Miley, Daural Hobbs, and Maurice Smith at Allied Signal Aerospace Corporation Kansas City Division for their help in creating the connector NIAM Information Structure Diagram (ISD), the database record structures, and the precedence diagram.

Melissa Myerly, 2857 for initially drawing the NIAM ISD and continually redrawing it in different formats for talks and reports.

John Mareda, 2644, Greg Neugebauer, 2854, and Dal Jensen, 2534 for their assistance regarding the connector graphics.

Kathy Branagan, 2551, for helping me test the user interface, for editing the Connector Selection Program User's Guide, and for her enthusiasm and support.

David Armour, 2551 and John Orman, 2825 for reviewing this document, and Mabel Hurley, 3151 for editing this document.

Table of Contents

Table of Contents

Connector Selection Program
Implementation Notes
Version 1.0

1. Introduction

At Sandia an engineer can choose from over 1200 qualified electrical connectors for a given application. Formally, engineers selected a connector from a small group of familiar connectors. Since this technique did not always result in the selection of the connector that was best suited to the application a user friendly, menu driven connector database was designed and developed to assist engineers in selecting the most suitable connector from the entire stock.

This paper presents the design and development of the connector database and user interface. It describes the objective that created the project and the methodology used throughout the design and development, how the database structure was designed and developed, the data input process and implementation, and finally, the user interface design and development. The appendix, which constitutes most of this document, contains valuable supporting information and examples.

2. Objective

There are two times when an engineer needs information about connectors: (1) when the engineer does not know what connector to use, but has some requirements that the connector must meet and (2) when the engineer knows what connector to use. In the first case, the engineer would want to input the requirements and be provided a list of connectors that meet those requirements. The engineer would then like to select a connector from the list and view more detailed information for that connector. In the second case, the engineer would want to input the connector number and be provided with detailed information about that connector. In this case, the list of connectors would contain only one number, so no selection would be necessary. This concept is illustrated in Figure 1.

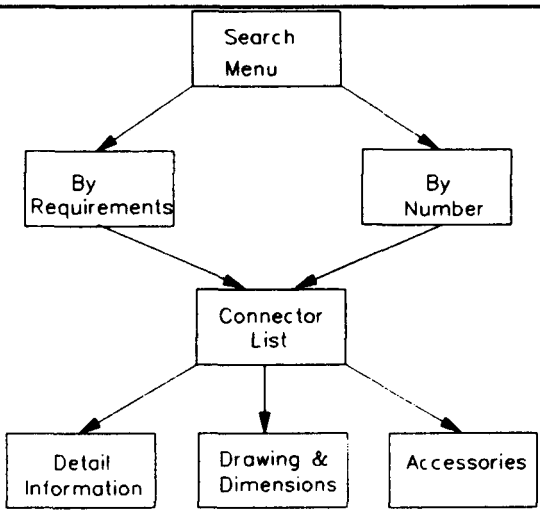


Figure 1 The Concept

The connector selection program was designed and developed specifically to meet these two cases.

3. Development Process

The general approach to designing and developing the connector database was Nijssen's Information Analysis Methodology (NIAM). NIAM is a conceptual information modeling technique that was used to characterize the connector data requirements. It provides the designer techniques for gathering, modeling, and analyzing information. The major benefit of NIAM was the ability to gain acceptance and approval of the conceptual model from management and engineers before an actual database was created. After the conceptual model was approved, it was implemented in the selected database management system. NIAM conceptual models may be implemented in any database management system. Since SAND88-0272 describes NIAM and the connector selection model in detail, this document only briefly addresses the area.

While the connector information gathering and modeling was progressing so was database management system (DBMS) research. Several DBMSs were evaluated to determine which would be the best for the connector selection database. The ORACLE relational database management system was selected. The reasons for this choice will be described later.

When the requirements were defined and the DBMS was selected, the next step was to create the ORACLE database record structures. These structures were created using a process called grouping. Grouping allows a designer to systematically transform a NIAM model into database record structures. These record structures are then evaluated and code is created for a particular DBMS.

After the database was defined, data could be input into the database and a user interface could be designed and developed. Once the basic user interface was in place the help, graphics, and report facilities were implemented. The user interface testing and correction phase completed the development life cycle.

The system was then delivered by giving several tutorials to interested individuals and groups. Feedback from these groups was evaluated and the user interface was modified to make it as user friendly as possible. The system is now available to any interested personnel at Sandia.

The formal documentation for the connector selection program consists of:

1. SAN88-0272 Developing a Connector Selection DBMS Using NIAM
2. SAN89-1286 Connector Selection Program User's Guide
3. SAND89-1619 Connector Selection Program Implementation Notes

The remainder of this document describes the design, development, and implementation issues addressed at each step of the life cycle.

Applicable information not incorporated in this report or other reports associated with the connector selection program include the PC-IAST output, which describes the database record structure; the ORACLE SQL*Forms code for the two input forms and the three output forms; and the graphic drawings. This data was not included because of its volume, but can be obtained from Nicole E. Sevier, 2551.

4. Conceptual Model

Nijssen's Information Analysis Methodology begins by gathering information about the subject at hand. In this case, that task consisted of gathering information about connectors, by attending connectors seminars were attended, interviewing engineers and reading connector books, and evaluating the hardcopy data for connectors. This research provided a vast amount of information. To trim the data down to a reasonable size, a survey was sent to the people within Sandia who needed connectors. The survey asked the engineers to select the parameters that they knew when they were looking for a connector and to select the data that would assist them with their final selection. A copy of the survey can be found in Appendix A.

About two-thirds of the surveys came back and the information was summarized as the initial connector selection requirements. An initial NIAM model was created for these requirements with the help of NIAM modeling experts R. M. Schaefer, M. Miley, and D. L. Hobbs from Allied-Signal Corporation, Kansas City Division (KCD).

A NIAM model is called an Information Structure Diagram (ISD). This model allows an individual to represent data and its relationship with other data in a format that is easily understood and evaluated. One of the benefits of the ISD is that it is a drawing and engineers are accustomed to reviewing drawings. So, with a minimum amount of training, they are comfortable with reviewing the ISD.

The connector ISD was reviewed by the connector engineers in the Interconnections Division and two retired connector experts to make sure that the information was complete and correctly represented. One of the most important lessons learned from the interviews with the experts was that the connector family grouping would not be valid for most of the connector data. Initially, it appeared that connectors were grouped by family names (e.g., JT, LJT, PT, ...) that characterized the basic shape and style of the connector. It was believed that certain data would be the same for all the connectors within a given family. The connector experts agreed that this was the original intention and it may have worked for a period of time; however, the engineers did not believe that it currently held true nor that it would hold true in the future. Based on the engineers advice and a review of some of the actual data, the model was redone to eliminate the assumptions described by the family relationship.

A relationship between the connector families and series was considered as a possible subtype association, too. It turned out that connectors within a given family-series combination have the same basic shape and dimensions for the multicontact circular connectors. This relationship was implemented in the design.

The WhiteStar working group, the KCD modeling experts, and two outside modeling consultants evaluated the ISD to confirm that the relationships were represented properly in the ISD. These reviews were very useful because the reviewers had no previous ideas about connectors and therefore were more likely to question every piece of data, every relationship, and every assumption made.

The connector ISD was translated into English sentences to further review the information contained in the ISD. Additional comments and errors were corrected by reviewing these sentences with the connector engineers.

After several iterations of ISD reviews and the English sentence reviews the ISD was frozen for the implementation of the connector selection database and user interface. The model in the SAND88-0272 contains the ISD as it was frozen for the implementation.

A few changes were necessary during the development process. Test parameters that were specified as mandatory were changed to non-mandatory to allow connectors to be added to the database without complete descriptions because it was believed that some data was more useful then none. Also, additional information was necessary for the accessories to properly link some of them to connectors in the database. These changes can be seen by comparing the model in Appendix B of this document with the model illustrated in the SAND88-0272.

5. Database Management System

A significant amount of research was devoted to selecting a database management system for the connector selection program. This research was primarily in the areas of network, hierarchical, and relational DBMSs.

The relational database structure appeared to be the most promising for the connector selection program. This was primarily because a relational database structures can be modified and added to much more easily than a hierarchical or network databases and very little, if any, information is duplicated. Both of these features were desirable because it was presumed that, over time, the connector database structure would have to be modified and added to. Duplicate data in a database is difficult to maintain because changing an item in the database requires that every occurrence of the data be found and changed. Therefore, if the data occurs in only one place, it is easier to find and needs to be changed only once. NIAM was structured after relational databases and was already being used for the design and development.

Many relational database management systems were available on the market. ORACLE, INGRES, BCSRim, RBase, IDMS/R, and Goldengate were evaluated as potential database management systems. These evaluations are summarized in Appendix C.

ORACLE was chosen as the database management software. ORACLE provides easy-to-use facilities for creating a database, manipulating the database, inputting information into the database via a batch job or interactively, and getting information out of the database. INGRES provided basically the same facilities; however, at the time it was not fully compatible with the standard query language SQL and did not have a personal computer version compatible with its micro or mainframe version.

6. Database Tables and Indexes

After the ISD had been stabilized and the database management system had been selected, the database record structures could be generated. The record structures could be obtained by applying a technique called grouping to the ISD. The grouping process is described in SAND88-0272.

The process has been automated by the Control Data Corporation into a software product, Information Analysis Support Tools (IAST). The product runs on a PC or on a Cyber mainframe. The connector ISD was input into the IAST software on both the PC and a Cyber. The record structures generated by the Cyber are the records that were used for the connector database implementation. They were selected because they were generated using a less-restricted grouping process and produced fewer tables. The PC used a more-restrictive grouping process that generated four times as many tables, most of which were felt to be unnecessary.

The record structures created by the IAST software can be found in Appendix D. A few additional check lists and temporary tables were needed during the implementation. The check lists are used by the input screens to make sure that the information being input is valid. The temporary responsible engineer table is being used until a list of Sandia Employees and their employee numbers can be obtained from the personnel department.

Indexes were created at this time as part of the database structure. An index for a database is similar to an index for a book. It provides a means for the database to quickly find information in a given table.

To create an index for a book, the author picks the words in the book that s/he wants to find easily and locates all the references to those words. An index for a database table is created in the same way. The implementer picks columns in a table that are commonly referenced and creates an index for those columns. The database management software will create references for the columns chosen.

Unique indexes are used to specify the columns in a table that must be unique. For example, two parts should never have the same part number. This can be enforced by creating a unique index on the part number column. An error message will be generated if someone attempts to duplicate a part number. The indexes and unique indexes created for the connector tables are included in Appendix D.

7. Data Input Screens

ORACLE provides a facility for creating input screens, SQL*FORMS. This facility was used to implement input screens for capturing the connector data and placing it in the proper tables. The input screens provide a standard method for inputting the initial connector data, as well as adding new connectors in the future. They also check the data to make sure that the parameter data is within valid ranges, that the materials specified are valid materials, and that data relationships are maintained. They do not require any data to be input repetitiously and do not require the person entering the information to know anything about the database structure.

The input screens were implemented before the user interface was so that data could be input into the database while the user interface was being developed. Data could have been input into flat files, but it was felt the time and effort put into the design and the development of the input screens would be worthwhile.

The input screens were developed in two stages. The first set was developed for collecting data for connectors and the second set was for incorporating the accessory data. These screens are described in the next two sections. However, before the implementation of these screens can be discussed, some of the basic SQL*FORMS language and format must be reviewed.

7.1 SQL*FORMS Language and Format

SQL*FORMS is used to create forms. To an engineer a form is a set of screens where information can be displayed, entered, or edited. To a developer, a form is a sequence of **blocks**. A block corresponds to a table in the database or a set of variable fields. Within a block corresponding to a table, a **record** is one row of data from the table. One or more records can be displayed on a screen at one time.

Within each block there are **fields**. Fields are like blank spaces on a form, where information appears. The fields are where the data is displayed, entered, or edited. Each field has a fixed size, position, and type of data that can be entered there. A block that corresponds to a table will have fields associated with the columns in the table. A block that is not linked to a table will have fields that are used as variables.

A screen is the engineer's view of the fields. The fields from many different blocks may occur on one screen. Text and graphics may also be displayed on a screen.

SQL*FORMS is controlled by **triggers**. Triggers are sets of commands that are executed when a certain event occurs while a form is being run. Trigger commands can be SQL*FORMS commands or SQL commands. Many triggers are built into SQL*FORMS. For example, there are triggers for saving data in the database, for searching for data in the database, and for movement within the blocks and fields. Triggers are typically invoked when the engineer presses a key on the keyboard.

Triggers are defined at three levels: the field level, the block level, or the form level. The form-level triggers can be used throughout the form. Block-level triggers operate with the block they are defined and they apply to every field in the block. In contrast, a field-level trigger applies only when executed in the defined field.

More information about SQL*FORMS can be found in the ORACLE SQL*FORMS documentation set.

The descriptions of the input forms and the user interface forms created for the connector selection program using SQL*FORMS will describe the basic flow of the screens as they would be seen by the engineer and the internal block structure. Any assumptions made about the engineers and the designers will be described, as well as any special attributes about the triggers that are unique to the connector selection program. Items that are common to any SQL*FORM will not be discussed.

7.2 Connector Input Screens

The connector selection input forms (connin) allow an individual to input, change, and delete the information associated with a connector.

7.2.1 Flow

The input blocks are arranged in the manner illustrated in Figure 2. The arrows show where there is a screen change. The names of the screens are shown to the right of the diagram. The solid lines depict where the blocks and tables change within a screen. The dashed line between the connector to contact block and the contact block is an optional block or table change. The or's imply that a decision is made based on information that was input on a previous screen.

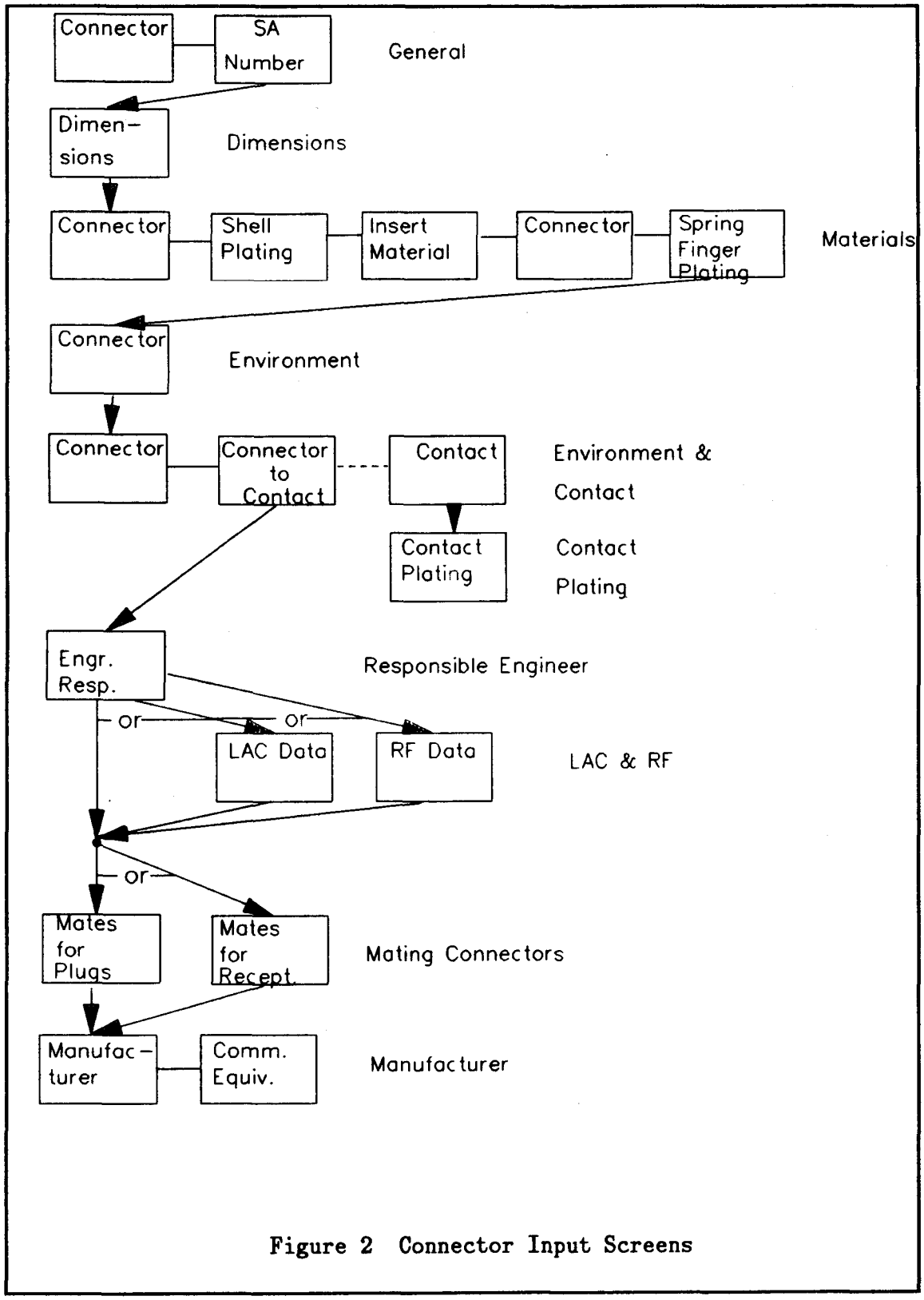


Figure 2 Connector Input Screens

The arrangement of the connector input screens corresponds to external data and the sequence it was found in for the first connectors input into the database. The general screen corresponds to the data commonly found in the MC/SA Special Design Connector Catalog. The dimensions screen data came from the catalog or from the AY Drawing of the connector. The material screen information was commonly found in the connectors' Automated Materials List (AML). The environment screens are for inputting environmental data which was found in the connectors' Product Specification (PS). The contact information on the environment and contact screen was in the AML and the PS. The responsible engineer's name and special LAC and RF data were in the PS. The mating connector information, manufacturer, and commercial equivalent are typically known only by the design engineer. Thus, the design allowed the person gathering or inputting the information to go through the documents in the following order: the catalog, the AY, the AML, the PS, and any additional information.

7.2.2 Assumptions

The input screens are arranged so the input, change, and delete procedures may be done without detailed knowledge of the database structure. The input screens assume that the individual performing these tasks knows and understands the SQL*FORMS operator functions. The implementation of these screens assumes that the individual understands the tables and indexes associated with connectors, as well as the SQL*FORMS design functions.

7.2.3 Special Attributes

The main difference between these screens and typical SQL*FORMS screens is that the actions are performed when the engineer is on a field in the connector block. The clear record, create record, delete record, duplicate record, next record, and previous record keys act on all the tables associate with the given connector. For example, if the engineer is located at the connector screens and s/he presses the delete record key, all the records associated with that connector in all the connector tables will be deleted. This includes the connector data as well as the SA number, all the dimensions, shell plating materials, insert materials, spring finger materials, LAC or RF data, responsible engineer, mating connectors, manufacturers, and commercial equivalents. This is true for all the "record" functions.

The "record" functions work as prescribed by SQL*FORMS for the other blocks. This must be the case, since these tables may have multiple records for one connector and the engineer may need to clear, create, delete, duplicate, and traverse through the records in these tables without affecting the rest of the connectors data.

The connector "record" functions are established with form triggers. The "record" functions for the other blocks are set up using block triggers. Thus, each block has "record" triggers associated with it.

The connector block and the contact block allow the engineer to use the enter and execute query keys. These keys are useless for the other

blocks, since the engineer does not have access to the part number fields on these blocks. However, the engineer can search for connectors and contacts by part numbers, or by characteristics. SQL*PLUS may be used to query other database entries.

SQL*FORMS provides a trigger for entering and executing queries for each block that has a table associated with it. The cursor identifies the block that is affected when the engineer presses the enter query key. The enter query trigger asks the engineer to input values in the block fields, to describe the search criteria, and to press the execute query key when they are finished. This activates the execute query trigger, that searches for all the records in the block's table that meet the criteria described by the engineer. These triggers have been modified for the connector input screens.

The enter query form trigger will automatically place the engineer in the connector block and perform the query for this block. The engineer is told to press the execute query key a second time to invoke the execute query form trigger. This trigger executes the query for the remaining tables, unless engineer is already in the contact block. The contact block has a block trigger that executes the query on the contact block and the contact plating block.

Block triggers and field triggers are combined to make the transition between the blocks and screens transparent to the engineer.

The first connectors to be input into the connector selection program were the connectors depicted in the MC/SA Special Design Connector Catalog. The information required by the first few screens was input directly from the catalog. The rest of the information was copied onto hardcopies of the connector selection input screens from the connector AYs, PSs, and AMLs. The information was then input from the hardcopies.

This process worked very well for gathering and inputting data for the connectors in the catalog. To gather and input data for connectors that were not in the catalog and for newly developed connectors, a input/change form was created. A copy of this form can be found in Appendix E. This form is given to the connector engineers when they find an error for an existing connector or have information about a new connector. Thus, the connector input screens were used for the first listing in the database and are being used for the maintenance of the data.

7.3 Accessory Input Screens

The accessory input forms allow an individual to input, change, and delete the information associated with connector accessories, including the EMR hardware, connector covers, pin protectors, potting mold adapters, and potting molds. The blocks are arranged as depicted by Figure 3. Each screen is made from several blocks. The screen names are shown below their respective blocks. The arrows indicate where the screens change, and the solid lines show where the blocks/tables change.

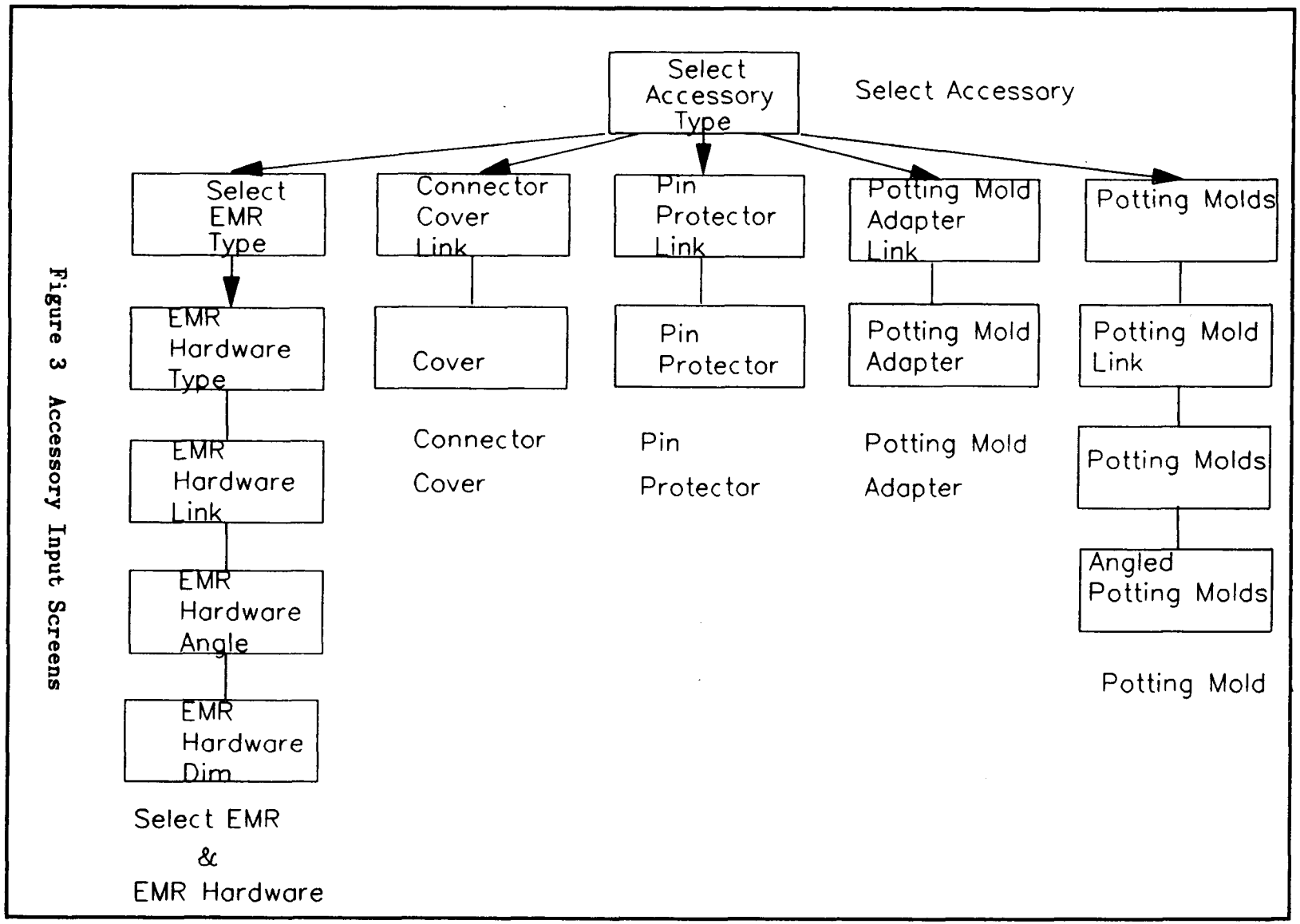


Figure 3 Accessory Input Screens

7.3.1 Flow

The engineer begins by selecting the type of accessory s/he wants to input information about. If the engineer selected EMR hardware, then the engineer must specify the type of EMR hardware s/he wants to input information about. The engineer then inputs the data associated with the chosen accessory.

7.3.2 Assumptions

The screens assume that the person inputting the information is familiar with SQL*FORMS operator functions; however, the user does not need to know or understand the internal database structure to input accessory information. The person working on these forms should know about the database structure and SQL*FORMS development functions.

7.3.3 Special Attributes

The EMR hardware, potting mold adapters, and potting molds are linked to the connectors based on connector families and series or families and shell sizes. Additional information is necessary for some of the connector covers. Pin protectors require the family and the insert arrangement to be associated with a connector. The tables and blocks that link the accessories with connectors have names that end with the word "link." The remaining tables include additional information about the accessories.

The form triggers commit the information to the database and clear the blocks for the next item. No queries are allowed at the form level, but are available for the appropriate accessory blocks containing the part number.

The part number for an accessory is entered only in the accessory blocks and is copied to the link, dim, and angle blocks for the engineer.

To delete all the information associated with a given accessory the engineer must delete all the records in all the blocks for that accessory. No automatic global delete is provided.

The transition between screens and blocks is implemented in block and field triggers.

8. Output Screens

The connector selection program output screens are displayed in Figure 4 as they would be seen by a engineer. Engineers begin at the search selection menu where they decide what type of search they would like to perform. If they know the connector part number or SA number, then the next screen would ask them to enter that number. If they know general or specific requirements that the connector they are looking for must have, then they would be asked to input this information. Once they have input the number or requirements, the system would search the database for the connectors with the corresponding number or requirements. In either case, a list of one or more connectors would be

displayed. The engineer could then select a connector from this list and view more detailed information about that connector. The detailed information screens include: General information, contact information, environment or product specification information, material information, mating connectors, a drawing with dimensions, and accessory information.

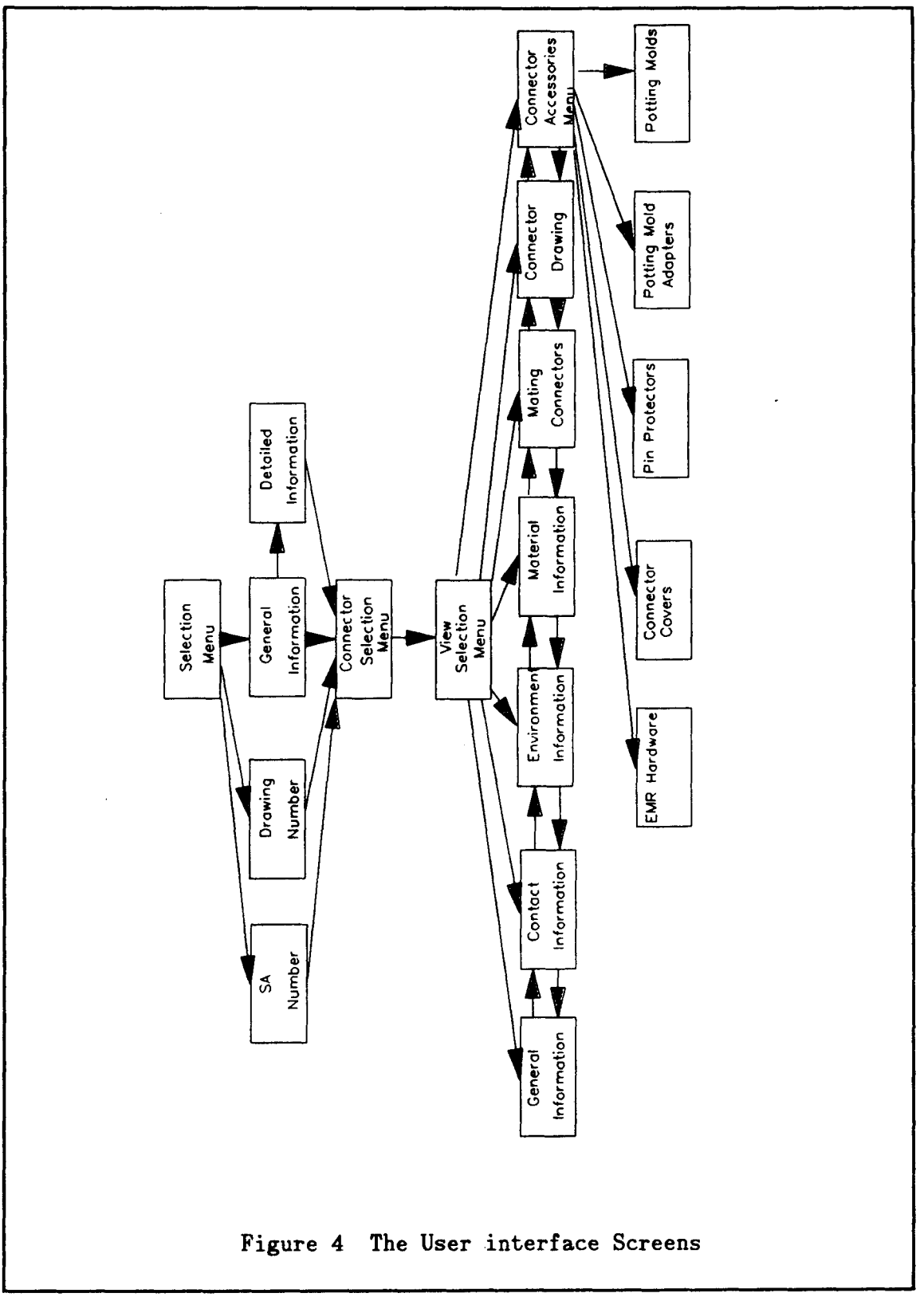


Figure 4 The User interface Screens

The connector selection output screens were originally developed using a PC version of ORACLE. Because of space constraints and speed conditions on the PC, the output screens were developed in three parts. The first part contains the connector search screens. These are the first set of screens from the search selection menu to the connector selection menu. The second part encompasses the output screens, which include all the output screens, except for the accessory output screens. The accessory output screens make up the third part.

The separation of these three sets of screens will be noticeable to the engineer only by the presence of a blank screen displayed while the transfer of control is taking place.

The connector selection program was later moved to the WhiteStar MicroVAX. Appendix F describes how to transfer an ORACLE database from a PC to a VAX. The WhiteStar MicroVAX provides a central location where many engineers in different areas can easily access the program.

All the screens in the connector selection program have the same basic appearance. Figure 5 illustrates the screen format used. The first line, at the top of the screen displays the name of the current program. The title line displays the name of the current screen. The area in the middle of the screen is where the menu options, requirement input fields, lists of components, and information about the current component are displayed and where most of the activity occurs. The key definition area is where the buttons on the engineer's keyboard are mapped to the program functions. The message line displays status information, and the ORACLE line tells the engineer whether s/he is in insert/replace mode when inputting information, the current screen page number, and the number of records displayed or found.

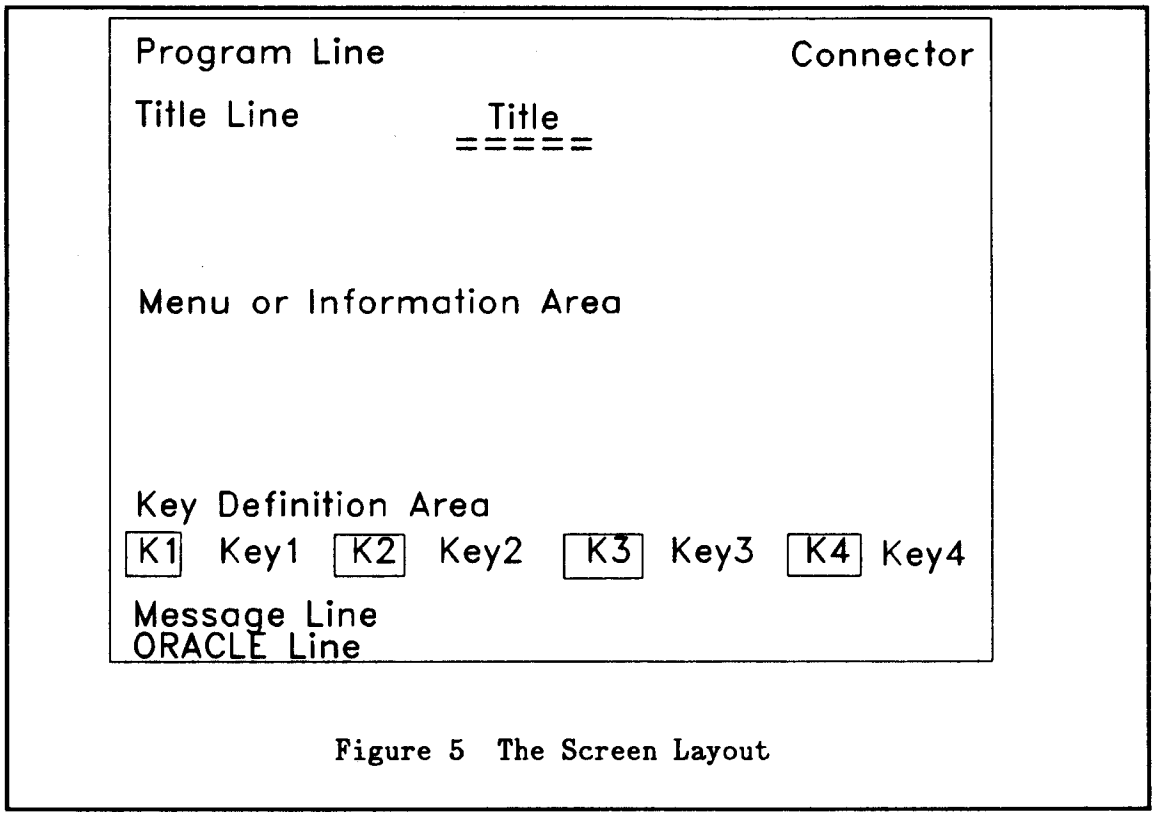


Figure 5 The Screen Layout

The keyboard buttons displayed in the key definition are dependent upon the engineers terminal type or terminal emulator. The keys displayed will differ depending on whether the engineer has a VT100 or a VT200 series terminal or emulator.

If the engineer has a PC with VTERM or an actual VT100 terminal, then the keyboard buttons displayed in the key definition will correspond to the keys on a VT100 or PC keyboard. If the engineer has a VT200 series or a VT300 series terminal, then the keyboard buttons displayed will correspond to the keys on a VT220 keyboard.

The intention of the different keys is not to confuse the engineer, but to take advantage of the keys provided on a VT220. VT220 keyboards have separate keys for page up, page down, and home. It was felt that someone used to working on a VT220 keyboard would be more comfortable using these keys for paging up, paging down, and going home instead of the numeric keypad. The process for implementing the variable key definitions is described in Appendix G.

In order for the variable keys definitions to work properly, it was necessary to determine the engineer's terminal type and pass this information to the output screens. This was done with the command procedure, Runconsel (for RUN CONnector SELection program). Logical names were necessary for the different output screens so that the system and SQL*FORMS could locate them. These logical names were defined in

the command procedure. Finally, the system had to be invoked, also by the command procedure (see Appendix H).

The next three sections describe the output screens, beginning with a general description, the sequence in which the screens are invoked internally is illustrated, any assumptions that were made regarding the engineer and the developer are mentioned, and any special characteristics are described.

8.1 Connector Search Screens

The first part of the connector selection output screens is referred to as Consel1 (for CONnector SELection screen part 1). Consel1 includes the screens where the engineer indicates the type of search s/he wants to perform, inputs the search data, and is given a list of connectors which meet the search criteria.

8.1.1 Flow

The flow of these screens as seen by an engineer is shown in Figure 6. The internal flow of these screens is illustrated by the Figure 7. In the internal flow figure, the arrows indicate where the screens change and the straight lines where the blocks change.

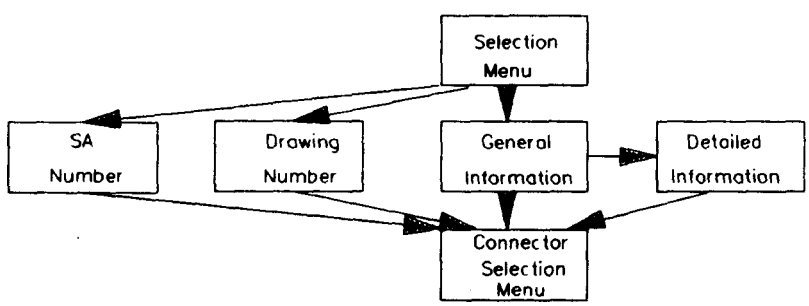


Figure 6 Search Screens (External)

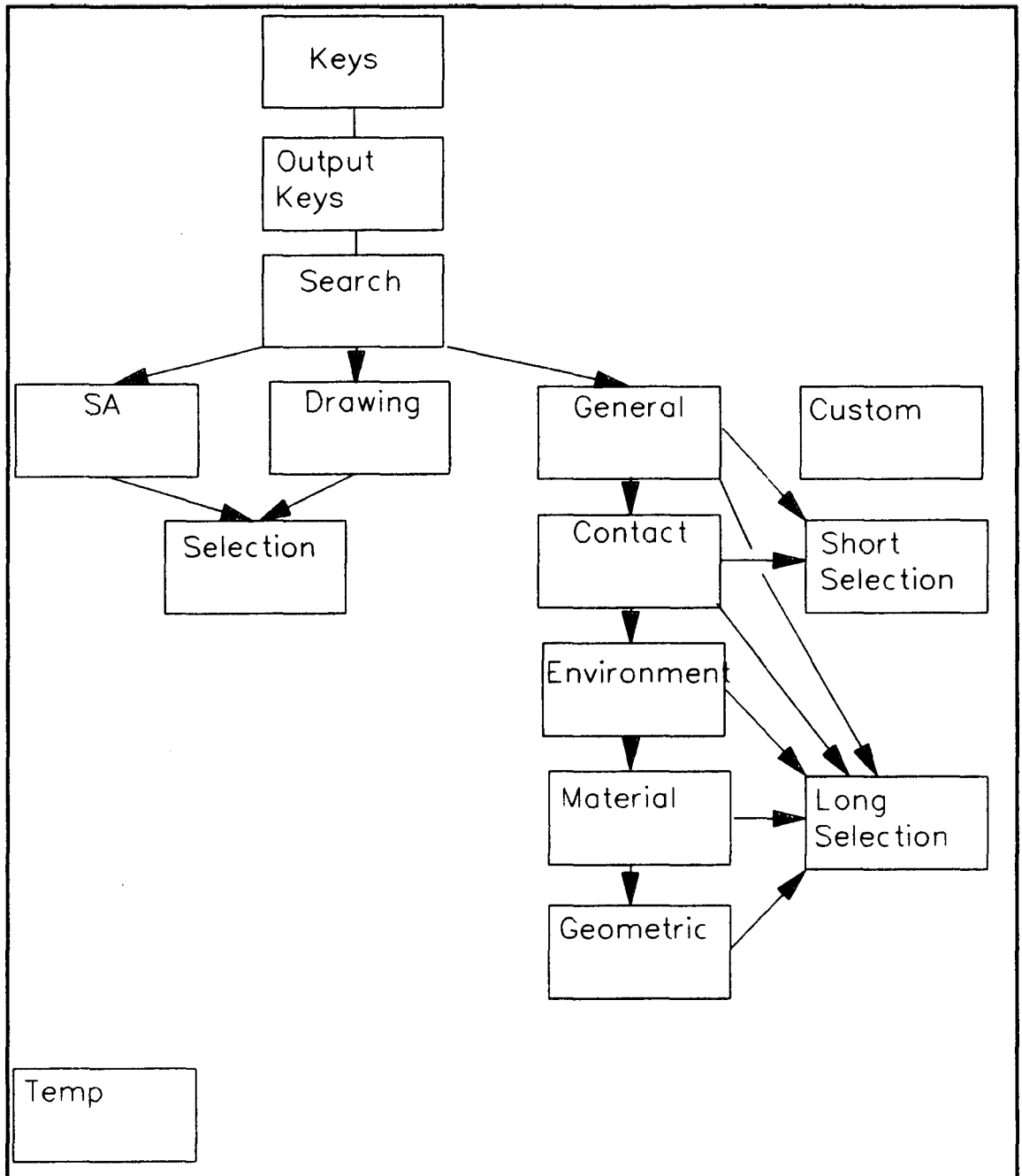


Figure 7 Search Screens (Internal)

Initially, the system determines which set of keys to display at the bottom of each screen, depending upon the type of terminal the engineer has. This information was placed in a table by the Runconsel procedure and the action is by the key-startup trigger and the keys block. Once the keys have been determined, they are displayed on the bottom of each screen by the Output Keys block.

The program then begins at the search Selection screen. This is where the engineer indicates the type of search s/he wants. Once the type of search has been selected, the system goes to the appropriate screen to get the specified search data.

If the engineer searches by SA number or by drawing number, then the search is performed by and output on the Selection block. If the engineer searches by general information, then the search is performed by and output on the Short Selection block. If the engineer searches by specific information, then the search is performed by and output on the Long Selection block. Different selection blocks are necessary as a result of different queries performed.

The engineer may select any connector on any of the selection blocks and view the information known about that connector. This information is displayed by the second connector selection output screen, Consel2.

8.1.2 Assumptions

The use of these forms assumes that the engineer understands the menu options and key definitions. The implementation of these screens assumes that the individual has a detailed knowledge of the tables, views, and indexes associated with connectors, and also understands SQL*FORMS development functions.

8.1.3 Special Attributes

The program must be invoked by the Runconsel command procedure, which creates a table and a record that tell the connector selection program the type of terminal the engineer has. The terminal type is mapped to the appropriate keys by Consel1.

Logical names are necessary to make the transition between screen sets and VMS command procedures. These logical names are created by the Runconsel command procedure.

The Help command procedure is called when the engineer presses the Help key. Thus, this procedure must be in place as well as the Help text in order for Consel1 to function properly.

The key-others form trigger is active, so all keys must be explicitly defined. Form triggers are available for the Help key (key-entqry), the exit key (key-exit), the previous menu or home key (key-clrfrm), and the list of values key (key-listval). The remaining keys are defined by block triggers or field triggers, depending on what is most appropriate.

There are form triggers for checking, wild carding, and clearing the information on the general and specific search screens. Before a search is performed, each field is checked to see if it is blank or not. The wild card character is copied to the fields that are blank to complete the search query. The wild card character must be cleared if the engineer must go back to these screens after any search.

There are form triggers for converting megohms to ohms and degrees F to degrees C. In this way, the engineers may enter insulation resistances and temperatures in the units they are most familiar with.

The copy selection form trigger copies the part number and SA number into global variables so this information can be passed to Consel2. Home and exit flags are set by this trigger to indicate that if these keys were pressed while running Consel2, appropriate action can be taken when control is returned to Consel1.

8.2 Connector Output Screens

The connector selection part 2 output screens are referred to as Consel2 (for CONnector SELection screens part 2). These screens are passed a connector part number and a SA number from the connector selection part 1 output screens, Consel1. The information known about the connector described by the part number is illustrated by the Consel2 screens.

8.2.1 Flow

The Consel2 screens seen by the engineer are shown in Figure 8. The internal screen flow is depicted in Figure 9. The arrows in internal flow figure indicate where there is a screen change and the lines show block changes.

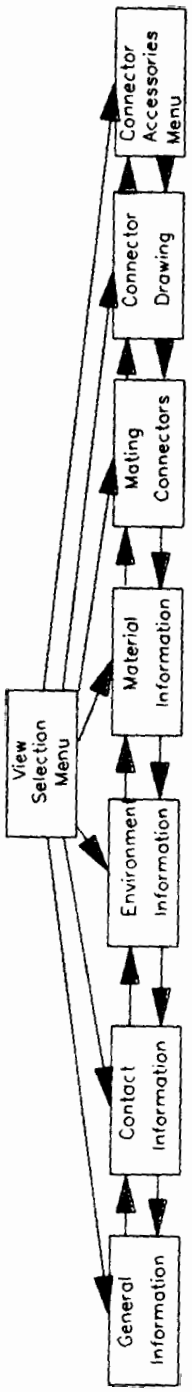


Figure 8 Output Screens (External)

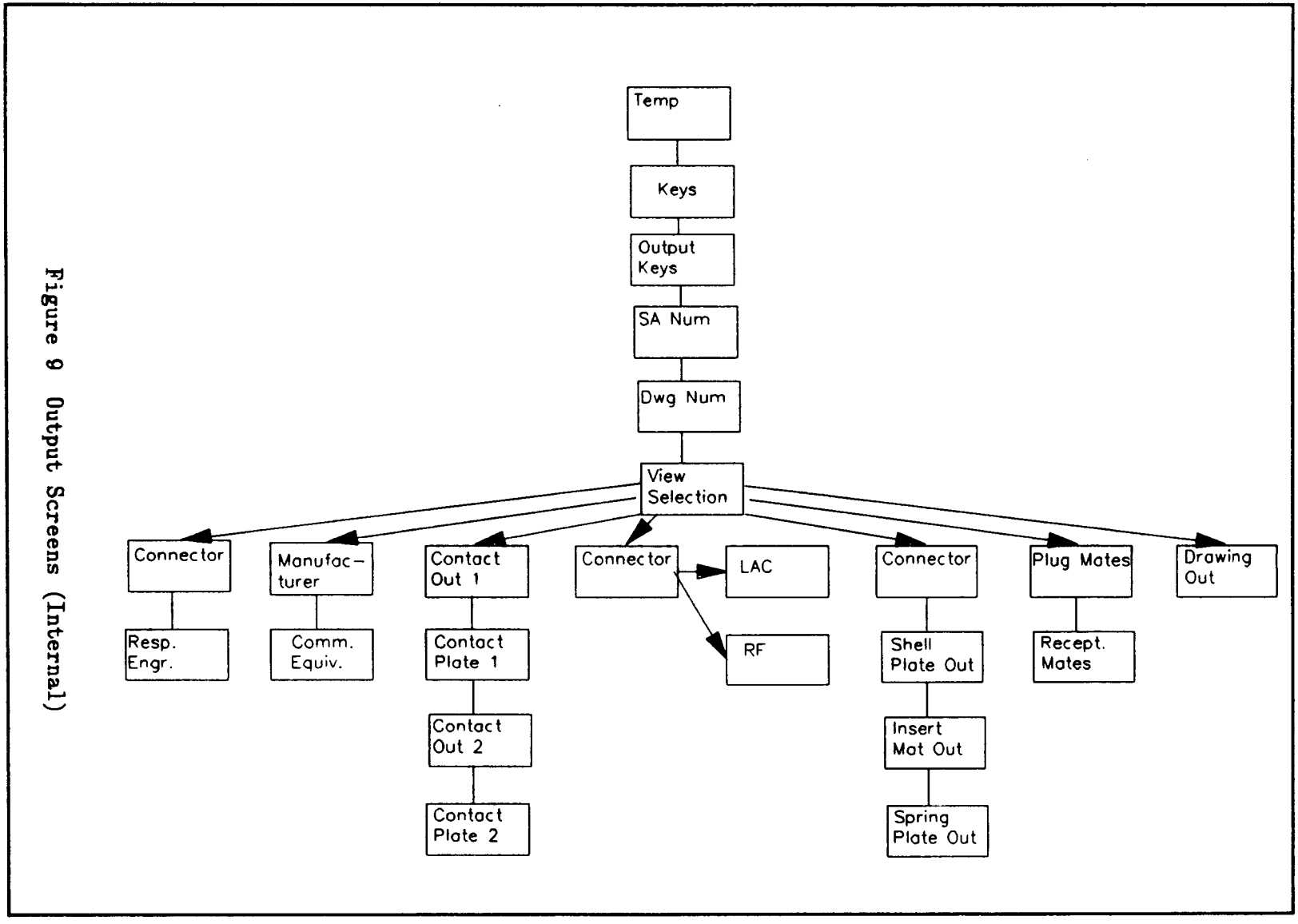


Figure 9 Output Screens (Internal)

The program begins by executing the key-startup form trigger. Key-startup copies the part number and the SA number passed from Consel1 into temporary fields in the Temp block. The type of keys to be displayed on the bottom of each screen is determined by the Key block. Control is passed to the Output Key block to display the keys on the bottom of each screen. The SA Num block displays the SA number at the top of every screen and the Dwg Num block displays the Sandia Drawing number at the top of every screen. Control is then given to the engineer at the View Selection block.

At the View Selection block, the engineer may pick the type of information s/he wants to see about the specified connector. The engineer is then transferred to the proper screen. If the engineer selects connector accessories, then the connector part number and SA number are passed to the Accessory Output screens, Acc_output, the third and final part of the connector selection output screens.

8.2.2 Assumptions

It is assumed that a user of these screens understands the menu options and key definitions. The implementation of these screens assumes that the individual understands the tables, views, and indexes associated with connectors and understands SQL*FORMS development functions.

8.2.3 Special Attributes

Consel1 must be invoked by the Runconsel command procedure so that the terminal type can be determined and the logical names can be established. The logical names are necessary so that Consel2 can be located by the system.

The Help command procedure and Help text must be in place for the Help key to function properly. The Graphics command procedure, programs, and files must be in place for the graphics to function properly. The Report command procedure and programs must be in place for the print report option to function properly.

The program begins by executing the Key-Startup trigger (see flow section above). Flags are set by this trigger to indicate whether a query has been performed or not.

The key-others trigger is active, so all keys must be explicitly defined. Form triggers are available for the help key (key-entqry), the exit key (key-exit), and the previous menu or home key (key-clrfrm). The remaining keys are defined by block triggers or field triggers, depending on what is most appropriate.

Only one field on each of the output screens is accessible by the engineer. The program was established in this way, because there was no reason for the engineer to move the cursor from field to field. The only reason a engineer would have to move to another field on a screen would be to input information in the field. In the output screens, the engineer is allowed only to look at the information.

There are form triggers to traverse through the information screens. The triggers are named as go_X, X_to, and X_from. The go_X triggers cause the program to go to the block X and make sure that the query for block X is executed only once for this connector. The X_to triggers are executed when the next block key is pressed while at the X block. That is, they tell the program where X should go to. The X_from triggers are executed when the previous block key is pressed while at the X block. X_from triggers indicate where the program was before it came to the X block.

The View Choice trigger causes the program to go to the appropriate screen from the View Selection block. The View Msg trigger is the error message for an invalid view selection.

The Print File trigger passes the connector part number and series to the print command procedure. This procedure displays all the information on the screen in a format that is suitable for a hardcopy report.

The Drawing Out block displays a drawing of the connector by passing the connectors part number and series to the graphics command procedure. The command procedure displays a picture of the connector along with its dimensions if the engineer has a terminal capable of displaying graphics.

8.3 Accessory Output Screens

The connector selection part 3 output screens are referred to as Acc_output. To these screens are passed a connector part number and an SA number from the connector selection part 2 output screens, Consel2. The information known about the accessories that can be used with the connector described by the part number is illustrated by the Acc_output screens.

8.3.1 Flow

The Acc_output screens seen by the engineer are shown in Figure 10. The internal screen flow is depicted in Figure 11. The arrows in internal flow figure indicate where there is a screen change, and the lines show block changes.

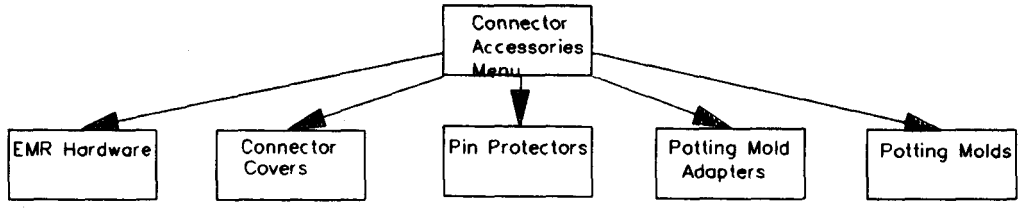


Figure 10 Accessory Output Screens (External)

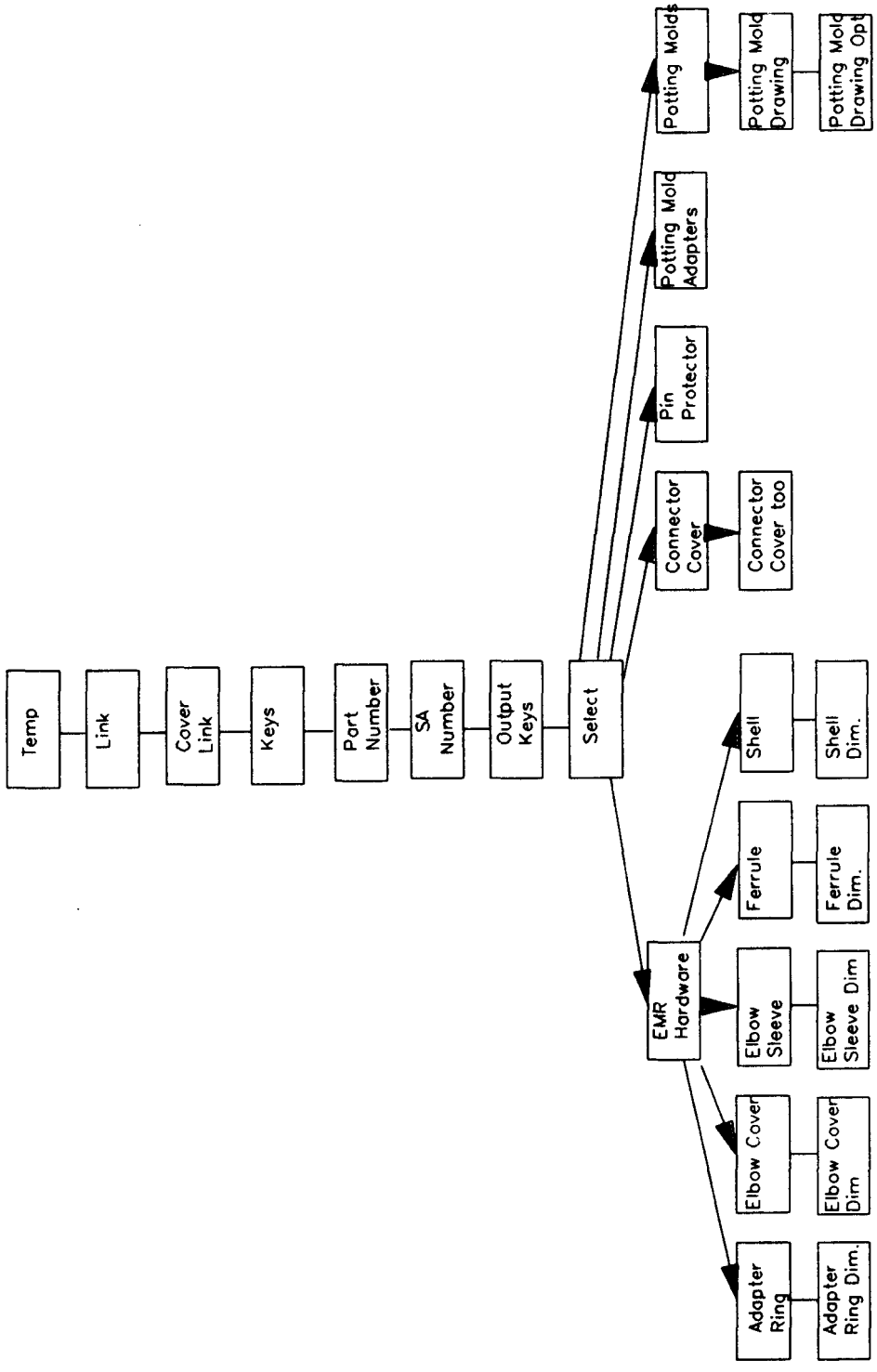


Figure 11 Accessory Output Screens (Internal)

The program begins by executing the key-startup form trigger. Key-startup copies the part number and the SA number passed from Consel2 into temporary fields in the Temp block.

The Link block performs a query to gather information about the specified connector. This information is used by the accessory blocks to determine what accessories can be used with this connector.

The Cover Link block gathers information about the specified connector that may be necessary to determine what connector covers will function with this connector.

The type of keys to be displayed on the bottom of each screen are determined by the Key block. The Part Number block displays the connector part number or Sandia Drawing number at the top of every screen. The SA Number block displays the connector SA number at the top of every screen. The Output Keys block displays the keys found by the Keys block on the bottom of every screen. Control is then given to the engineer at the Select block.

At the Select block the engineer may pick the accessory s/he wants information about for the specified connector. If the engineer selects EMR Hardware, then a list of all the EMR hardware that is available for the specified connector is displayed by the EMR hardware block. The engineer may select any of the items listed and view a drawing of the item and its dimensions. The EMR hardware drawing and part numbers are displayed by the corresponding EMR hardware type blocks: adapter ring, elbow cover, elbow sleeve, ferrule, and shell. The dimensions are displayed on the same screen by the EMR hardware-type blocks with the block name extension, Dim.

If the engineer selects Potting Molds, then all the potting molds available for the specified connector are listed by the Potting molds block. The engineer may select any of the potting molds listed and view a drawing of the specified potting mold and its dimensions. The drawing and part number are displayed by the Potting Mold Drawing, and the dimensions are displayed by the Potting Mold Drawing and the Potting Mold Drawing Opt blocks.

The information known about connector covers spans two screens. The data for these screens is captured by the connector cover and the connector cover too blocks.

The pin protector and potting mold adapter information is displayed on one screen by the corresponding block.

8.3.2 Assumptions

It is assumed that a user of these screens understands the menu options and key definitions. The implementation of these screens assumes that the individual understands the tables, views, and indexes associated with connectors and their accessories.

8.3.3 Special Attributes

Consel1 must be invoked by the Runconsel command procedure so that the terminal type can be determined and the logical names can be established. The logical names are necessary for Acc_output to be located by the system. The Help command procedure and Help text must be in place for the Help key to function properly.

The program begins by executing the key-startup trigger (see the flow section above). Flags are set by this trigger to indicate whether a query has been performed or not.

The key-others trigger is active, so all keys must be explicitly defined. Form triggers are available for the Help key (key-entqry), the exit key (key-exit), and the previous menu or home key (key-clrfrm). The remaining keys are defined by block triggers or field triggers, depending on what is most appropriate.

There are form triggers to get to each of the output screens. The triggers are named as go_X. The go_X triggers cause the program to go to the block X and make sure that the queries associated with block X are executed only once for this connector.

8.4 Summary

The connector selection output screens are divided into three sets: the search screens, the output screens, and the accessory screens. All the screens have the same screen format and are invoked by the Runconsel command procedure. The Help command procedure is necessary for the engineer to access the Help text. The graphics command procedure is necessary for the engineer to see the connector drawings and their dimensions. And finally, the print command procedure is necessary for the engineer to obtain a report of the information provided. Together these screens and command procedures provide a complete user interface for the connector selection database.

The next sections describe the research, design, and development of the help facility, the graphics facility, and the print facility.

9. Help Facility

The Help facility available from the ORACLE SQL*FORMS allows for only 80 characters of information. This is inadequate to describe the fields and features provided by the connector selection program. Several approaches were looked at to expand the Help text to an infinite amount of text. The pros and cons of these approaches are described briefly, a method is selected, and the selected method is characterized in detail.

9.1 Evaluations

From the ORACLE SQL*FORMS an operating system command can be executed by using the SQL*FORMS HOST command. The basic steps for using the HOST command are to create a character string of the command you want to execute, place it in a block variable (global and system variables don't

work), and then send the host computer the block variable. An example is given below of a trigger that would execute a command to type a file. The filename to be typed is contained in the database table, filenames.

```
SELECT "type " || filename
INTO :display.filename
FROM filenames
WHERE attributes = :display.attributes

#HOST :display.filename
```

If the filename selected was "sample.txt" then the command "type sample.txt" would be executed. After the file was typed, control would return to the invoking trigger.

This was the method for accessing procedures and programs external to SQL*FORMS used by some of the approaches investigated.

9.1.1 Database & SQL*FORMS Approach

The first idea was to create a database table that contained the block name, field name, and description for each field in the connector selection program. The process would then be to create a Help block where this information could be displayed. The block name and field name would be passed to the help block to determine the description to display.

This mode of action has some problems that begin with the database table and its description field. This field needs to be variable-length. Frank Ezell, 2812, who investigated this approach found it difficult to display a variable-length field neatly with SQL*FORMS and recommended use of multiple 80-column fields instead.

The descriptions in the database would have to be input and maintained. However, this would have to be done for any method selected. The most logical way would be to create some SQL*Form for inputting, changing, and deleting this information.

A method for outputting this information would be useful for maintaining a User's guide for the program. Most of the information that would go into a User's guide should be contained in these field descriptions. The SQL*Report software could be used to output this information in a User's guide format.

The connector selection program would have to be modified to implement this method because, the engineer does not now traverse through the fields of the output screens, but only one field on each screen.

This method contains information about each field, but no information is supplied for the operating procedures. So, additional work would have to done to describe this information.

Thus, to implement this method:

- a table would have to be defined,
- an input form would have to be designed and developed,
- the information would have to be input,
- an output form would have to designed and developed,
- the connector selection database would have to be modified,
- a method for displaying a description of the operating procedures would have to be developed,
- and an SQL*Report code would have to be written.

This is a significant amount of work to do initially as well as to maintain.

The key advantage of this method would be that all the information about the database would be self-contained.

9.1.2 Files

The second plan was discovered after the Connector Selection Program User's Guide was written. The User's Guide has a reference chapter that contains detailed information about each screen in the connector selection program. The information provided for each screen consists of a general description, a detailed description of every field, and a description of the action(s) that will be performed by all the keyboard buttons defined. This arrangement was designed so that not only could the answer to a specific question be found quickly and easily, but it would be extensive and complete.

This information could be separated out of the User's Guide into files, and the files could be used as the connector selection Help text. The information could be displayed in a manner similar to the Host command example at the beginning of this document. This could be done at the SQL*FORMS form trigger level, so only one trigger would be added to the connector selection program.

The invoking trigger would have to know what screen the engineer was on and determine from this what file to display. The current form, block, and field are SQL*FORMS system variables, so this information is readily available. A table containing the form, block, field, and filename could determine the file to be plotted.

This information would have to be input in the database, but because the amount of data would be very small, no input forms would be needed.

No method would be needed for outputting the information to include it as part of the User's Guide. The opposite would be true: a procedure for breaking the information out of the User's

guide would be needed. The User's Guide was written using Mass11 and Mass11 is capable of merging files before printing. The User's Guide could have pointers to merge the separate screen reference pages. An ASCII file would have to be created for each file and these files would have to be copied to the Connector Selection directory, implying that there would be two copies of the information. There may be a way in the future to eliminate one copy. Until then, the Mass11 files could be considered the official copy.

Maintenance would follow the same procedure as defined above. A change could be made to the Mass11 file, then a new ASCII file could be created and copied to the connector selection directory.

The operating procedure information is included as part of the definition of each screen, so no additional work is necessary to include this information.

Thus, to implement this method:

- a table would have to be defined,
- the form, block, field, and file information would have to be input,
- the trigger would have to be designed and implemented in the connector selection program,
- the User's Guide reference pages would have to separated,
- the ASCII files would have to be created and
- the ASCII files would have to be copied to the connector selection directory.

In comparison to the first method, this method would require less work to implement and less to maintain since it does not have any input forms or output forms. This method separates the information about the database from the database itself and moves it to the User's Guide. They both have to be maintained, so this would not make a big difference.

The descriptions capture both the fields and the operating procedures, so no additional work is needed to implement this type of Help. The information is broken up into fewer pieces because it is divided by screen instead of by field and the interface to the connector selection program is simple. These items reduce the complexity of the process. But because this method provides input forms via a word processor, the person maintaining the system would have to know Mass11 or some other word processor with similar capabilities.

9.1.3 VMS Help

As an extension of the file method came the VMS Help scheme. The files used by the file procedure could easily be modified to include the commands necessary to create a VMS Help library with the connector selection information. The Help command could be

used to start the typical Help scenario, rather than the type command.

If this approach were used, it would only work while the system was running under VMS. However, it would allow the engineer to go more directly to the information of interest. The file technique requires the engineer to traverse through all the information to find the data pertaining to any question.

9.2 Decision

The files method is the approach selected for the implementation of the connector selection Help facility. The primary reasons for selecting this approach were that it was less complex than the SQL*FORMS plan, it could easily yield the VMS Help method if the engineers were unable to quickly find the answers to their questions, and the data was contained in the Connector Selection Program User's Guide already. Why not maintain it there, too?

9.3 Implementation Details

Most of the implementation details are described above and are illustrated in Figure 12. A command procedure is called to issue the "type" command rather than issuing the command directly from the connector selection form trigger. This is not necessary now, but allows for future flexibility.

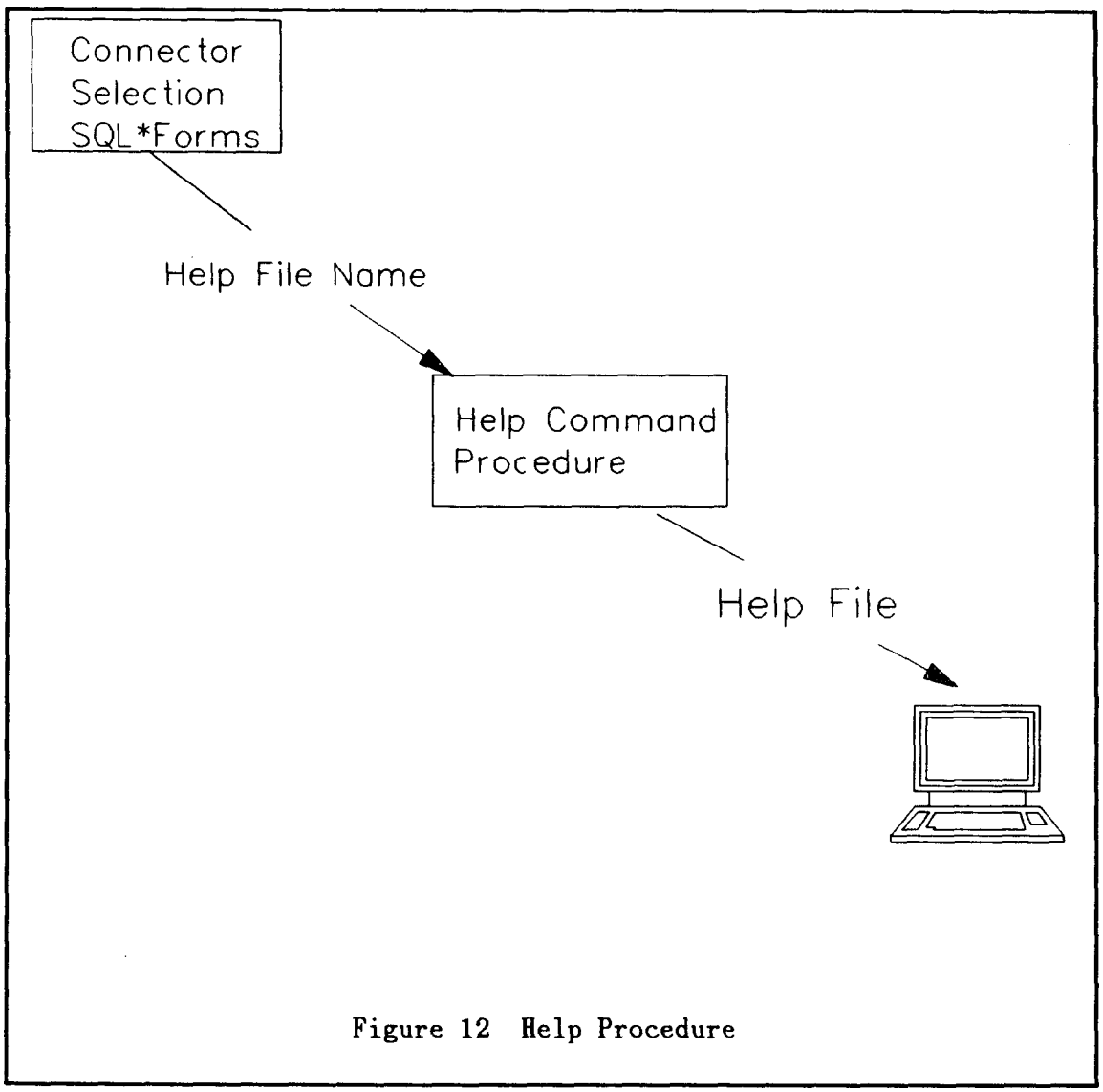


Figure 12 Help Procedure

The command procedure can be found in Appendix I. The form trigger, KEY-ENTQRY, can be found in any of the connector selection program output SQL*FORMS documents. The files displayed can be viewed by running the connector selection program or by reading Chapter 3 of the SAND89-1286. Information about MASS11 can be found in the MASS11 Word Processing User's guide.

10. Graphics Facility

Some components can be described by simple dimensions such as length, width, and height. Connectors are much more complex than this (see the connector shown in Figure 13). It's not easy to associate an understandable English description with each of the dimensions shown. The notion that a picture tells a thousand words makes it relevant to include a drawing.

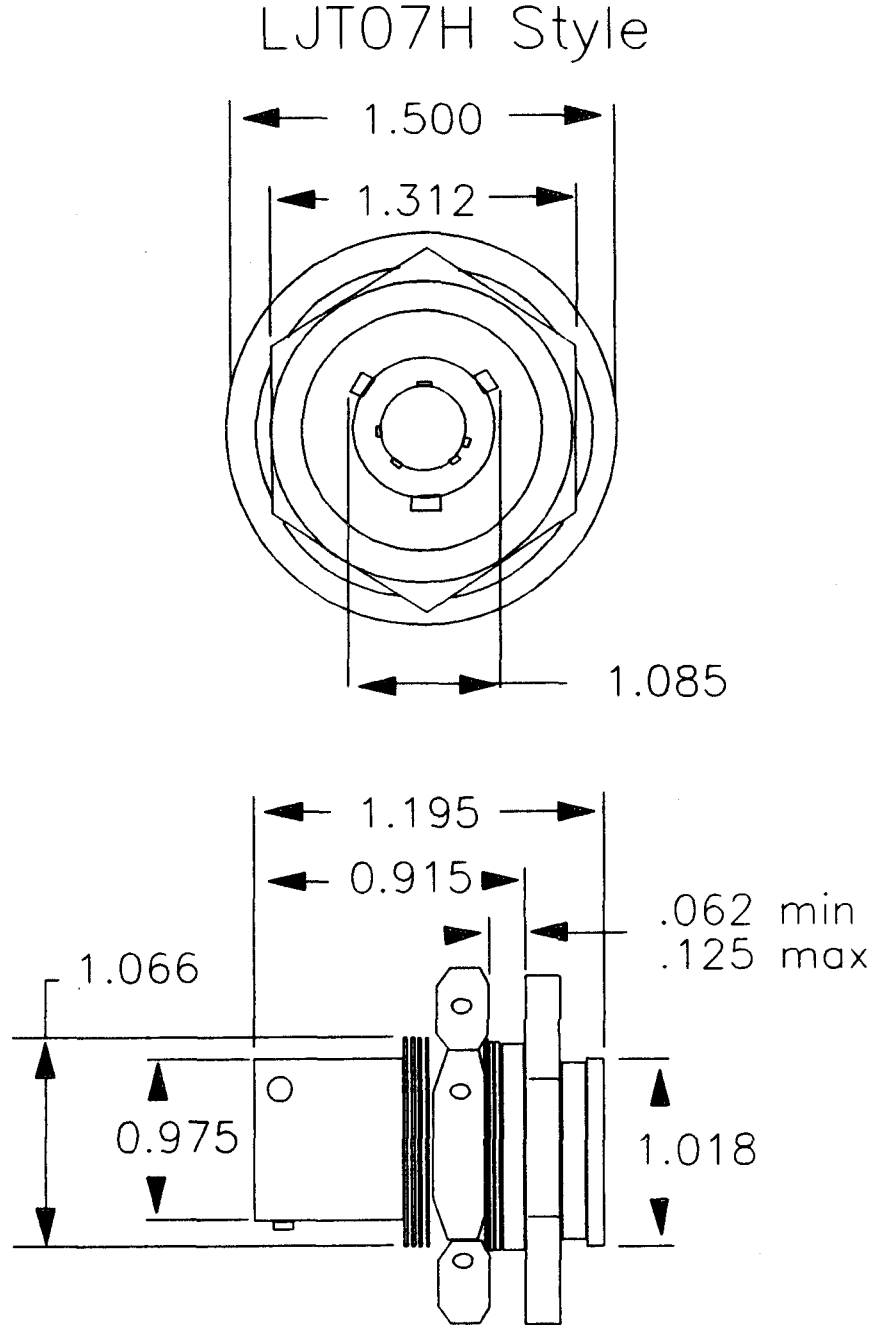


Figure 13 Connector with Dimensions

The dimensions of a connector are part of the connector selection database. The connectors in a given family-series combination have the same set of dimensions associated with them, because they have the same general shape. The individual dimensions are given cryptic names (a, b, c, ...) because it's difficult to create English descriptions.

The tasks being addressed here are (1) how can the drawings for these connectors be created, (2) how can they be displayed on a variety of terminals, and (3) how can the drawings be merged with their dimensions in the database. The rest of this document describes the methods researched for implementing these tasks, the approach selected, and the important details pertaining to the selected method.

10.1 Evaluations

Three techniques were evaluated for creating and displaying the connector drawings and dimensions. The first approach was to create and display the connector drawings using a graphics programming language. The second method was to create the drawings using an interactive graphics package and display the drawing and dimensions using Disspla or Telegraph. The last process evaluated was to create the drawings using an interactive graphics package and use the VMS operating system to display the drawing and dimensions. The pros and cons of each of these approaches follows. The last process was selected for implementation. The reasons for this choice and the important development details follow the evaluations.

10.1.1 Graphics Programming Language

The graphics programming language approach was to develop the code necessary to draw a connector. This code could be scaled, based on the dimensions from the database, to accurately represent each connector in the family-series combination. The dimensions from the database could be displayed where they belonged on the drawing. An example of what the connector might look like implemented by this method is shown in Figure 13.

About 103 family-series combinations are necessary to depict the 951 connectors initially input into the connector selection program. This implies that code would have to be created and maintained for 103 connectors, just to get started. The drawing code would have to be written using a graphics programming language. The standard appeared to be GKS at the time of this evaluation. GKS and PASCAL were available on 2551's VAXstation II/GPX, so development could be done there. However, for production the WhiteStar MicroVAX would have to purchase GKS and for maintenance an individual would need to be familiar with GKS and PASCAL.

The drawing number of the connector being reviewed could be passed from the Connector Selection Program to a VMS command procedure. This command procedure could query the database for the dimensions of the connector represented by the drawing number and place the information in a file. The drawing code could access this file to obtain the connectors dimensions.

The family and series of the connector being reviewed could be passed from the Connector Selection Program to the VMS command procedure to determine the drawing code to be executed for this particular drawing.

GKS is device-independent, so the connector drawings could be displayed on any graphics terminal. The terminal type could be determined and passed to the drawing code to produce the drawing for the engineers specific terminal type. This is extremely important at Sandia where there are many different types of graphics terminals. Those organizations without graphics terminals would not be able to view the connector drawings.

This approach provides well defined methods for performing the tasks required to create and display drawings and dimensions for the connector selection program. The drawings can be created using GKS and Pascal. The drawing dimensions can be passed to the drawing code via a file and the drawing can be displayed on virtually any terminal type.

10.1.2 Disspla

There are many interactive graphics packages on the market. The connector drawings could be created using one of these packages and then displayed by the connector selection program with the proper dimensions. To find out how these drawings could be displayed on a variety of terminals with dimensions from an external source, John Mareda (2644) was contacted.

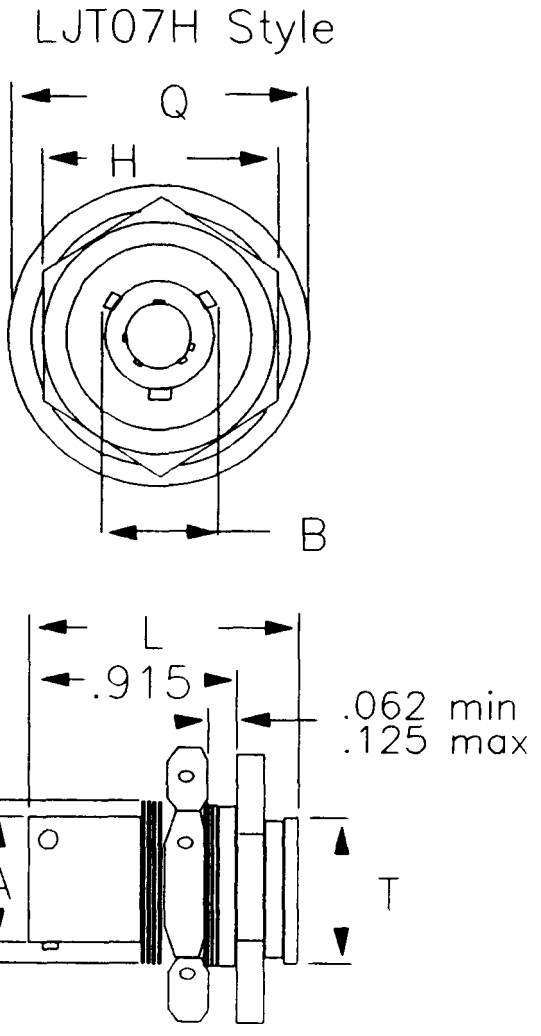
John said that the drawings could be created with SuperImage. SuperImage could output the drawing as a Computer Graphics iMage (CGM) file. The CGM file could then be displayed by Telegraph or Disspla on a variety of terminals.

SuperImage is a PC graphics package which is relatively inexpensive and very simple to learn and use. Telegraph and Disspla are graphics programming packages. Either package would cost approximately $2,500 for a licence on a MicroVAX.

The drawing number of the connector being reviewed could be passed from the Connector Selection Program to a VMS command procedure. This command procedure could query the database for the dimensions of the connector represented by the drawing number and place the information in a file. Disspla code could be written to access the dimensions in this file and display the information along with the drawing. The dimensions could be displayed as they are in Figure 14, or, with additional coding, as they are in Figure 13.

DIMEN	INCHES
A	.975
B	1.085
H	1.312
L	1.195
O	1.066
Q	1.500
T	1.018

7 RECORDS SELECTED.



----- PRTSCN TO PRINT -----
----- PRESS RETURN TO CONTINUE -----

Figure 14 Drawing & Dimensions

The family and series of the connector being reviewed could be passed from the Connector Selection Program to the VMS command procedure. This information could be used to determine the Disspla code to be executed for this particular connector.

Telegraph and Disspla are device-independent, so the connector drawings could be displayed on basically any graphics terminal. The terminal type could be determined and passed to Disspla to produce the drawing for the engineers specific terminal type.

This approach provides well-defined methods for performing the tasks required to create and display drawings and dimensions for the connector selection program. The drawings can be created using SuperImage. The drawing dimensions can be passed to Disspla or Telegraph code via a file, and the drawing can be displayed on virtually any terminal type.

10.1.3 Files

In order to supply drawings, approximately 103 connector drawings would have to be created. New drawings would have to be created in the future as new connectors were developed. This sounded like a draftsman's job, and there was no reason to duplicate the work. So, the idea of displaying the drawings created by drafting was investigated.

Greg Neugebauer (2854) was contacted initially to see if Anvil was capable of outputting a CGM file. Greg indicated that this could not be done currently, but they would be willing to make it possible in the future, if it was necessary. Anvil would currently output HP plot codes and Iges code. In the near future, they would have a method for creating HP Graphics Language (HPGL) code from the HP plot codes.

With this information, Telegraph and Disspla were evaluated to see if they would accept HP plot codes, Iges code, or HPGL. It turned out that Telegraph and Disspla would not accept any of these. Dal Jensen (2534) was consulted to see if any other software products would display HP plot codes, Iges code, or HPGL.

Dal pointed out that the Mass11 graphics processor and Baseview software were both capable of accepting HPGL code. Mass11 has utilities for outputting graphic files that can be displayed on a number of the major terminal types used at Sandia. Baseview can output graphic files that can be displayed on Regis, Sixel, and Tektronix terminals. Either package would cover most of the terminal types found at Sandia.

The files output by Mass11 or Baseview could be displayed on the proper device by simply typing the drawing file. The family and series of the connector being reviewed could be passed from the Connector Selection Program to the VMS command procedure. This information along with the terminal type could be used to determine the drawing file to be displayed for this particular drawing.

The drawing number of the connector being reviewed could be passed from the Connector Selection Program to a VMS command procedure. This command procedure could search the database for the dimensions of the

connector represented by the drawing number and place the information in a file. This file could then be displayed on top of the drawing or somehow be incorporated with the drawing file.

This approach meets the requirements for creating and displaying drawings and dimensions for the Connector Selection program. The drawings can be created by any software tool that will output HPGL code. The HPGL code can be translated by the Mass11 graphics processor or Baseview software to a file that can be displayed by Regis, Sixel, or Tektronix terminals. The drawing dimensions can be appended or incorporated with the converted file to display the drawing and dimensions together.

10.2 Decision

The graphics programming language approach was not implemented for several reasons. First and foremost, this process would duplicate the work being done by drafting. Forgetting the first reason, GKS would have to be purchased as well as a maintenance agreement. The creator and maintainer of the drawings would have know 2 programming languages, GKS and Pascal. Writing the code to create drawings is an abstract task, which implies it would take time to develop 103 initial drawings as well as additional drawings.

In summary, this method would duplicate efforts being done elsewhere, would cost money up front, would be a continuing cost, would require graphics and programming training and knowledge to create and maintain, and would take considerable time to develop and support.

The Disspla method was not implemented, either. This approach was not chosen because drafting would have to write code to translate HP plot codes to CGM code, Disspla or Telegraph would have to be purchased for the WhiteStar MicroVAX as well as a maintenance contract, and SuperImage was the only graphics software found that would output CGM code. The creator and maintainer of the drawings would have to know Disspla or Telegraph and SuperImage. Knowledge of SuperImage is necessary for creating the old connector drawings that drafting does not have available on Anvil.

In summary, this method would cost money up front, would be a continuing cost, would require Telegraph or Disspla and SuperImage training and knowledge to create and maintain, and would take time to develop the HP to CGM translation code.

The Files approach was implemented. This approach was selected because drawings can be created on a wide variety of software packages. (Anvil, SuperImage, Mass11 Draw, and Lotus Freelance are just a few examples.) No new software is necessary to develop or maintain this method. Drafting has Anvil and 2551 has SuperImage, Mass11 Draw, and Lotus Freelance for creating the drawings. The 891 VAX has the Mass11 graphics processor and Baseview software for translating the drawing from HPGL to files that can be displayed on most graphic terminal types. The command procedure for capturing the dimensions, determining the engineer's terminal type, and displaying the drawing are SQL and VMS

commands. Both VMS and SQL must be known to maintain the database itself.

In summary, this technique would not duplicate efforts being done by drafting, would not cost any money up front, would not be a continuing cost, and would not take considerable time to develop and support. This approach would require Baseview or Mass11 graphics processor training and knowledge to create and maintain. Drafting could be asked to create the drawings not available on Anvil, or they could be created using a PC interactive graphics package. (The latter would require some training.)

10.3 Implementation Details

To implement the files method a command procedure, Graphics.com, was written to accept a part number and a series. A copy of this command procedure can be found in Appendix J. The family name is not needed to determine the file to be displayed, since the series is a unique element. The part number is used to create a SQL batch file that queries the database for the parts dimensions and places them in a file.

The series is used as the drawing file name and the filename extension is based on the engineers terminal type which is determined by the "set term/inq" command. Thus, the series and the terminal type pick the file to be displayed.

To incorporate the dimensions with the Regis files, a Pascal program was written. This program adds dimensions from a file to the bottom of a specified Regis file with the proper quotations and escape sequences, creating a new file. The new file is displayed by the command procedure and then deleted.

A Tektronix manual or terminal was not available to determine how to add the dimensions to the Tektronix files. To avoid this problem, the Tektronix file is displayed and the dimensions are displayed on top. This is possible because the Tektronix does not scroll when it is in its graphics mode. To facilitate this process, the last point drawn on the drawing must be at the top.

The Connector Selection program drawing out block contains a trigger which passes the part number and the series to the graphics command procedure. Figure 15 summarizes the command procedures activities.

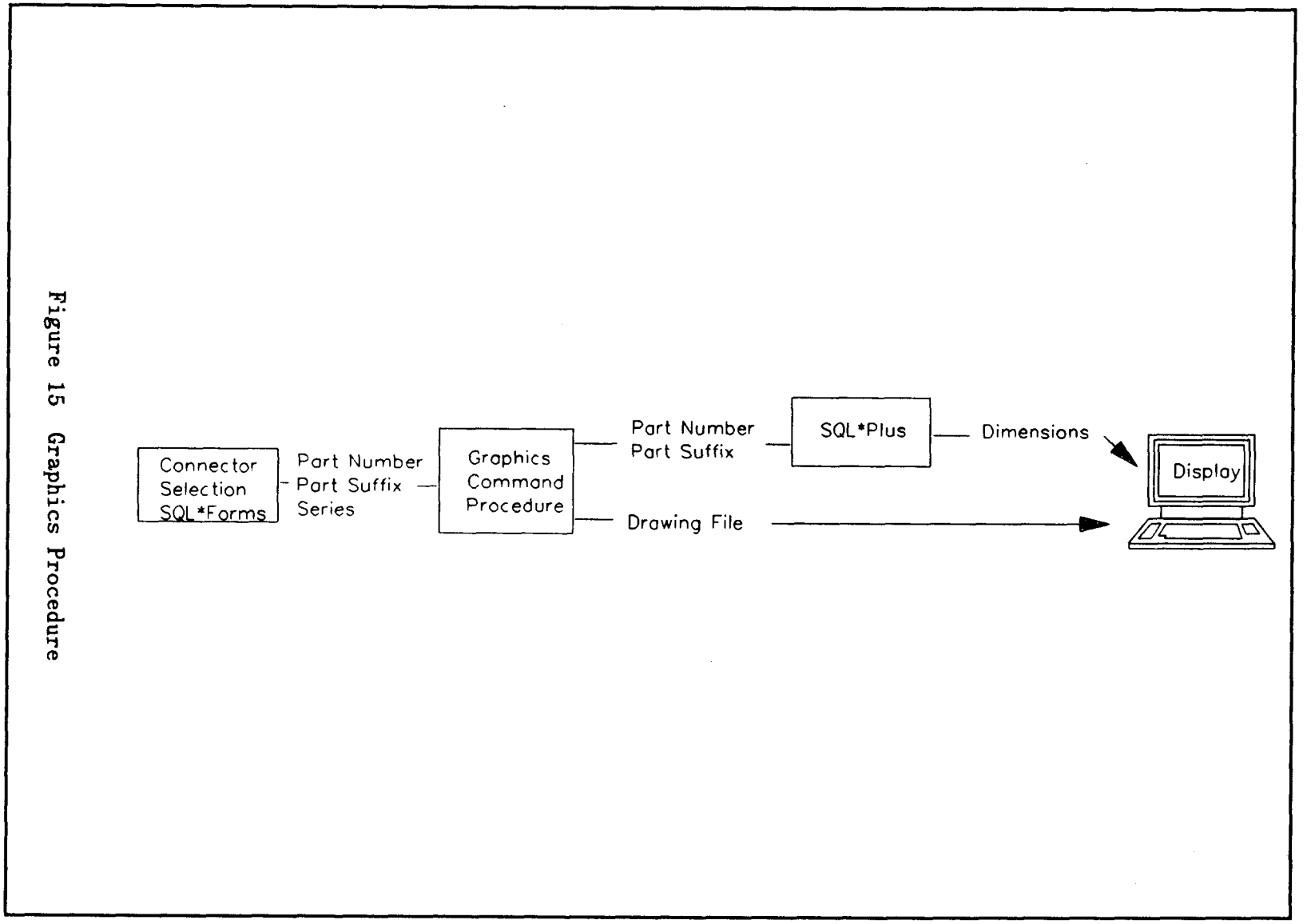


Figure 15 Graphics Procedure

The procedures followed to create the initial drawings are summarized here. (For a more detailed description, see the procedures document in this section.) A drawing is created using Freelance. A vertical view is used so that the drawing will fit on the screen with the dimensions displayed alongside. The output is to a file for an HP plotter and copied to the 891 VAX via an image copy. The file is input into Baseview. Once in Baseview, the file is rotated 90 degrees, shifted to the right as far as possible (the dimensions are displayed on the left), and a dot is placed on the drawing in the top center. (The dot is necessary to position the cursor for displaying the dimensions with the Tektronix files.) A Regis and a Tektronix output file are created.

The Regis file is modified to eliminate the command that clears the screen and returns to the text mode. The Regis file must have the commands to position the cursor and to display text added. The Tektronix file is altered to eliminate the command that appears to switch the Tektronix out of the graphics mode.

The two files are copied to the WhiteStar MicroVAX to the connector selection program subdirectory Graphics. Files received from Anvil would follow the same basic procedures. However, the typical Anvil drawing would have to be modified to discard the details and incorporate the dimension labels.

11. Report Facility

The Connector Selection program contains a vast amount of information about connectors. However, the only method for accessing this information is through the computer. There are times when this is not convenient, so a report facility was developed for the Connector Selection program. The report facility provides a hardcopy report of the information provided by the Connector Selection program for a given connector. The report command procedure and an example of a report generated by the report facility can be found in Appendix K. This section describes how the report facility was implemented.

Through a menu of options, the Connector Selection program asks the engineers what information they want to see about a selected connector. A new option was added, a "Display All the Information for Local Printing" for the report facility.

When this option is selected, the form trigger, PRINT_FILE, is executed. This trigger creates a command string that calls the command procedure, REPORT.COM, and passes it the connector part number and series.

The command procedures actions are illustrated by Figure 16 and the actual command procedure can be found in Appendix K. The command procedure creates a table, PART, and inserts the part number. The SQL*Report generation software, RPT, is invoked with the connector report program, REPORT.DAT.

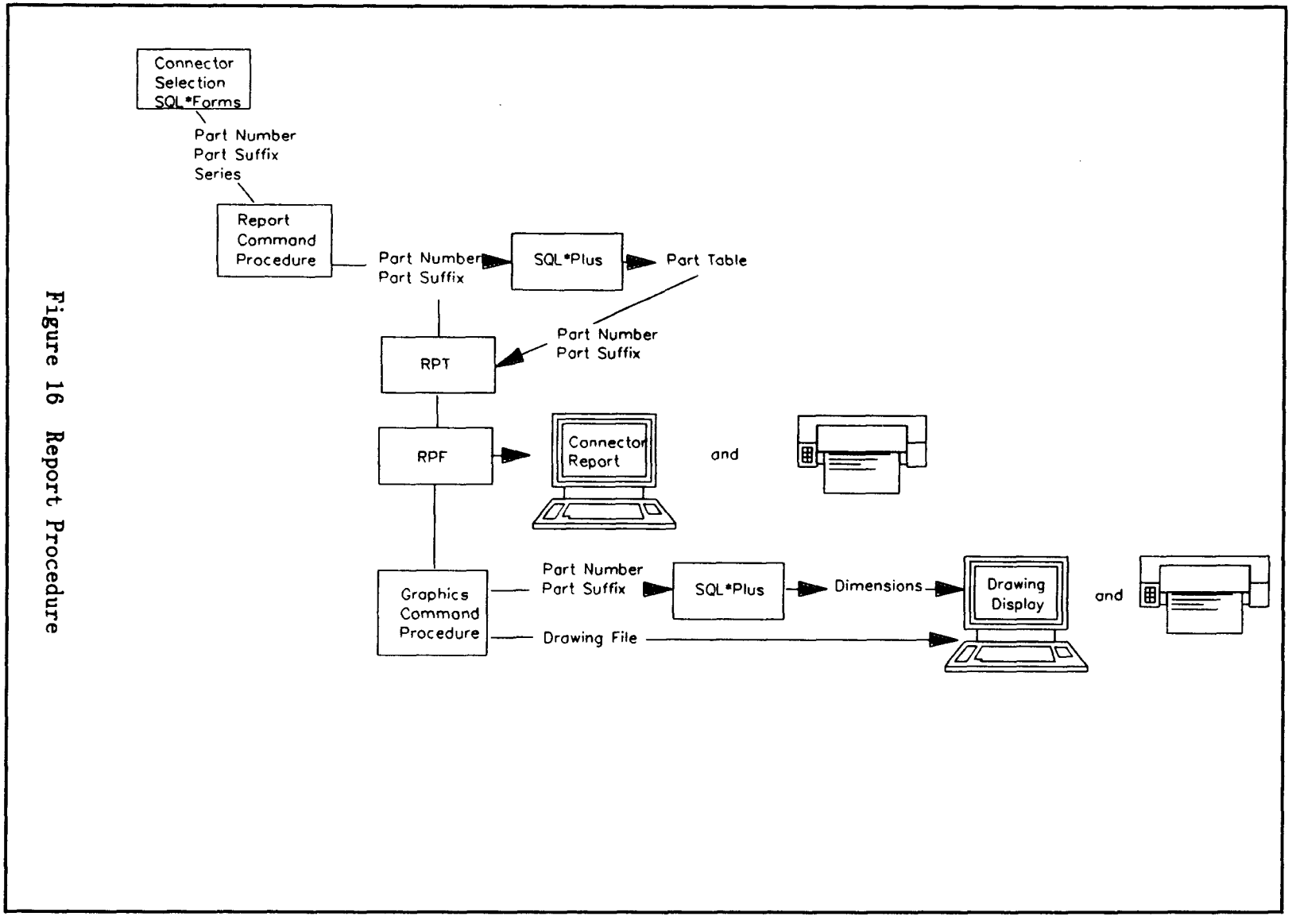


Figure 16 Report Procedure

The connector report program is a combination of SQL*Report RPT and RPF code. The RPT code accesses information in the database, and the RPF code formats the data for output. The RPT software is run to gather the information about the connector described by the part number in the table part and create a temporary file. The SQL*Report formatting software, RPF, is then executed with the temporary file to format and output a report to an output file. Together the two create a report containing data. The information is arranged in the same sequence it would be seen if an engineer looked at the output screens in order, except for the drawing which is displayed last.

The command procedure explains to an engineer how to get a printer ready to capture information displayed on the screen. When the engineer is ready, the report is displayed on the screen and printed on the engineer's printer. When the report is finished, the engineer is told how to tell the printer to stop capturing information displayed on the screen. The graphics command procedure is then called to display the connector drawing and dimensions. These can be captured on the engineer's printer, if it is a graphics printer. Control is then returned to the Connector Selection program.

12. Summary

The connector selection program was designed and developed to help engineers at Sandia select connectors that quickly and easily meet their requirements. The project development began with this basic concept in mind. Research found what information engineers commonly knew when they were looking for a connector and what additional information they needed to know to choose a connector. A conceptual model was created with the information gathered, reviewed by past and present engineers in the Interconnections Division to make sure that the information was represented correctly, then was transformed into ORACLE database record structures and indexes.

To input the connector data into the ORACLE database record structures two input screens were created. Connector input screens were created for the connector data, and accessory input screens were created for the connector accessory data. Connector data was gathered on hardcopies of the connector screens and input from these and the MC/SA Special Design Connector Catalog. Input/Change Forms have been created for new connector data and changes to the old connector data.

An extensive user interface was created for the engineers to access the information contained in the ORACLE database. The user interface is divided into three sets of screens: search, output, and accessory. These screens allow an engineer to input either a connector part number, SA number, or requirement that the connector must meet. The program will then list the connectors which match the part number, SA number, or requirements specified. The engineer may select any connector in that list and view additional information about the connector. Additional information includes general connector information, contact information, environment (PS) information, material information, a list of mating connectors, a drawing of the connector with dimensions, and accessories which fit the connector.

Several auxiliary programs are needed for the user interface screens to function. There is a Help command procedure and there are Help files that provide the engineer with detailed help. A graphics command procedure and drawings files allow the engineer to see a picture of the connector with its dimensions. And finally, a report command procedure and SQL*REPORT code create a report that can be printed on the engineer's local printer.

Together the database, the input screens, the output screens, the help facility, the graphics facility, and the report facility make up the connector selection program.

Appendix A: Data Survey Form

date: October 9, 1987

to: distribution

from: N. E. Sevier, 2544
E. D. Machin, 2544

subject: Connector Selection Data

Division 2544 is developing a connector selection database management system. This system will select connectors and mating connectors from a database of WR qualified connectors. Before we begin the design of this database we need some input from you, the ENGINEERS! If you have done some connector selection on your own, we would like to know what information you used to make your selection either for an original connector or for a mating connector. We would also like to know what information you want to know about a connector which has been selected.

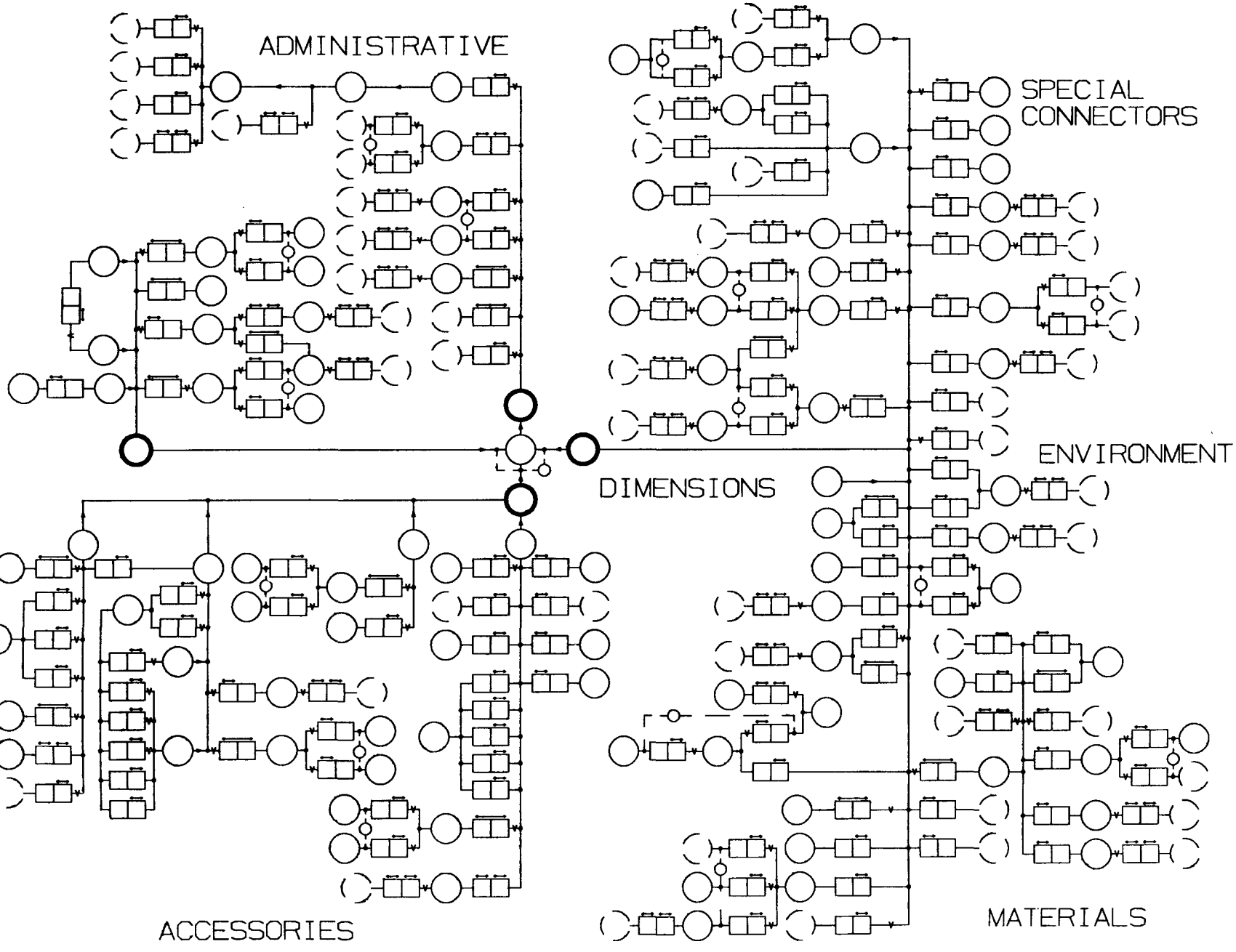
The attached sheets list some facts about connectors which can be found in the MC/SA Special Design Connector Catalog (orange catalog), AY drawings, or product specifications. Please check off those items which you have used or would like to use to select a connector or a mating connector. There is also a column where you can tell us what you would like to see output once a connector has been selected. Remember YOU are the people who will be using this program, so please take the time to fill this out carefully. If you have any questions, please feel free to call Nicole Sevier at 6-2993.

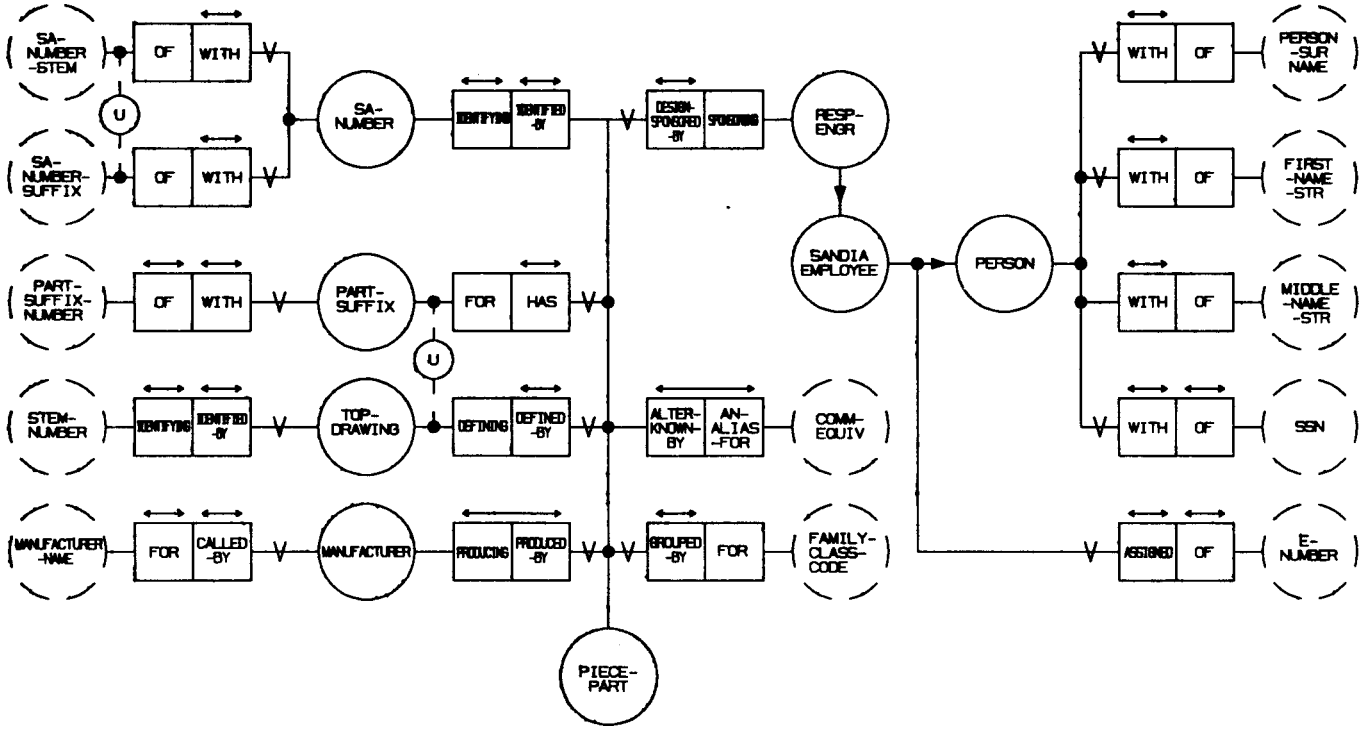
Information	Example	Mating Connector Selection	New Connector Selection	Output
General Administrative Information				
Family	LJT			
Drawing Number	358317			
Nomenclature	SA1438-12			
Manufacturer	BIC			
Manufacturer Part Code	358317			
Part Cost (BIC)				
Stock Quantity (BIC)				
Qualified Organization	2154			
Responsible Engineer	Work, R.			
Recommended Use Code	QER			
Status	Obsolete			
Use List				
Commercial Equivalent				
Electrical/Mechanical Information				
Connector Type	Receptacle			
Hermetic/Nonhermetic	Nonhermetic			
Mounting Type	Panel			
Materials for				
Shell (fused vitreous material)				
Insert (epoxy mold w/ glass fibers)				
Contacts (nickel w/ gold plating)				
Shell Size	15			
Insert Arrangement	15-15			
Insert Type/Pattern	Standard			
Type of contacts	Socket			

Information	Example	Mating Connector Selection	New Connector Selection	Output
Total Num. of Contacts	15			
Contact Resistance				
Contact Size	20 16			
Current Carrying Capacity	7.5 20			
Maximum Voltage Drop	25 30			
Number of Contacts to carry maximum current	1			
Hole Layout Code	2.045C			
Mating Connectors	358031			
Spring Finger Material				
Spring Finger Resistan				
Accessories				
Backshell Number	830644-00			
Connector Cover	204612-00			
Pin Protector				
Cable Clamp	353564-00			
Potting Mold				
Adapter	877236-005			
Angle	90 degrees			
Number	877113-004			
EMR Hardware				
Angle	90 degrees			
Adapter Ring Number	298466-00			
Elbow Cover	877025-008			
Ferrule	877024-018			
Shell				

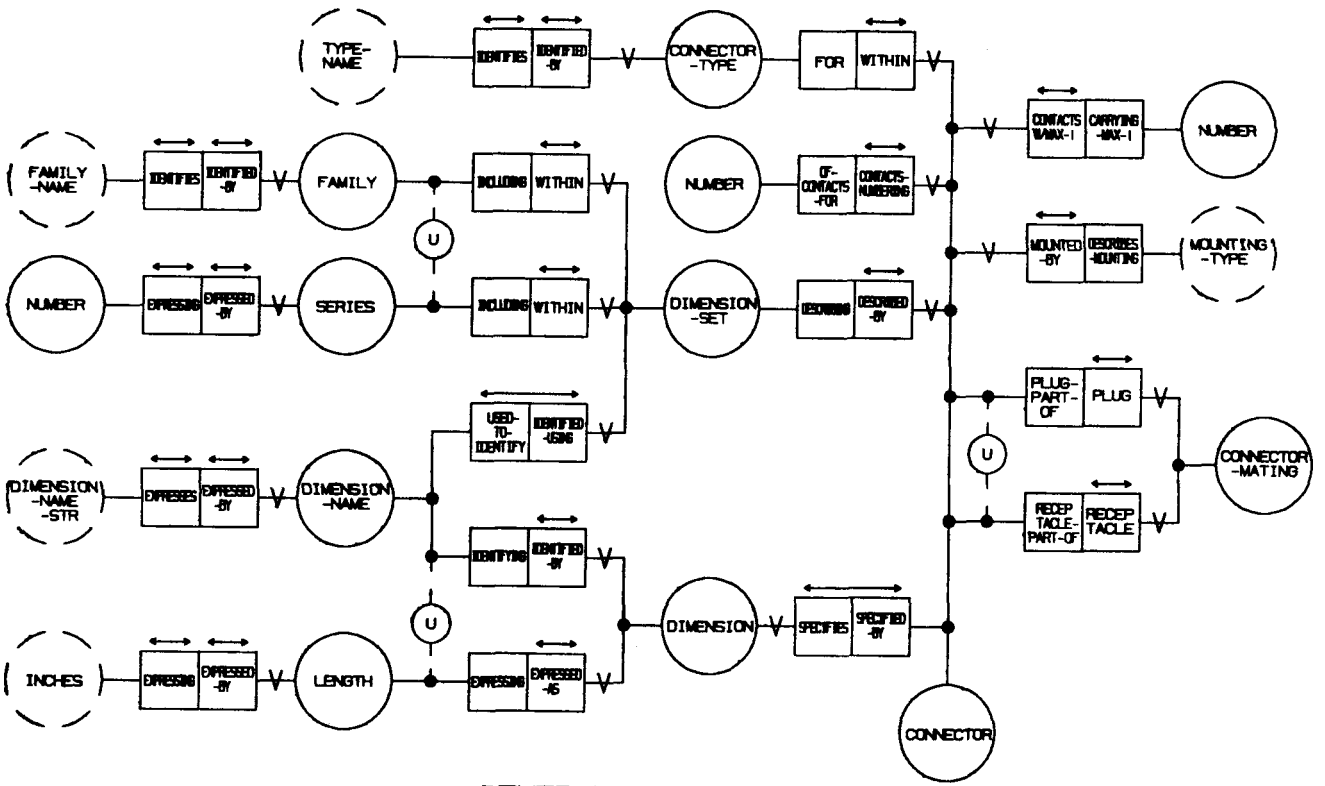
Information	Example	Mating Connector Selection	New Connector Selection	Output
Elbow Sleeve	877026-008			
Environmental Characteristics				
Dielectric Withstand Voltage	2120 Vdc Sea Level			
Maximum continuous operating temp.	350 f			
Air Leakage Rate	.001 cc/s			
Insulation Resistance	10000 Mohms			
Mechanical Shock	1000 g			
Vibration	Mil Std 202			
Durability	250 times			
Contact Retention	25 lbs			
Insert Retention	100 psig			
Moisture Resistance	Mil Std 202			
Coupling Pin Strength	50 lbs			
Operating Force				
Other (please specify)				

Appendix B: Conceptual Model

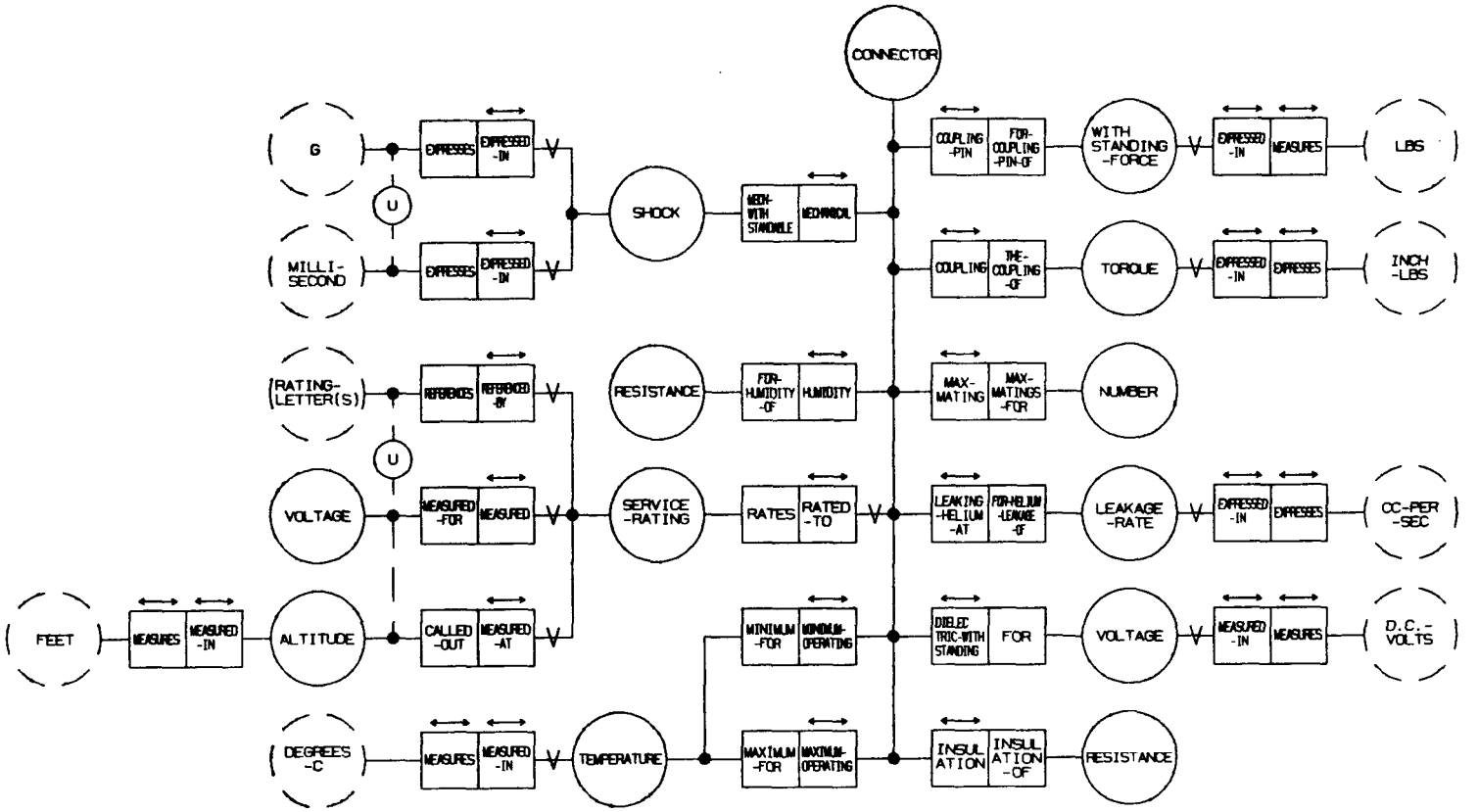

ADMINISTRATIVE
SPECIAL
CONNECTORS
ENVIRONMENT
DIMENSIONS
ACCESSORIES
MATERIALS


SA-
NUMBER
-STEM
OF
WITH
U
SA-
NUMBER-
SUFFIX
OF
WITH
SA-
NUMBER
IDENTIFYING
IDENTIFIED
-BY
DESIGN-
SPONSORED
-BY
SPONSORING
RESP-
ENGR
SANDIA
EMPLOYEE
PERSON
WITH
OF
PERSON
-SUR
NAME
WITH
OF
FIRST
-NAME
-STR
WITH
OF
MIDDLE
-NAME
-STR
WITH
OF
SSN
PART-
SUFFIX-
NUMBER
OF
WITH
PART-
SUFFIX
FOR
HAS
U
STEM-
NUMBER
IDENTIFYING
IDENTIFIED
-BY
TOP-
DRAWING
DEFINING
DEFINED
-BY
ALTER-
KNOWN-
BY
AN-
ALIAS
-FOR
COMM-
EQUIV
MANUFACTURER
-NAME
FOR
CALLED
-BY
MANUFACTURER
PRODUCING
PRODUCED
-BY
GROUPED
-BY
FOR
FAMILY-
CLASS-
CODE
ASSIGNED
OF
E-
NUMBER
PIECE-
PART

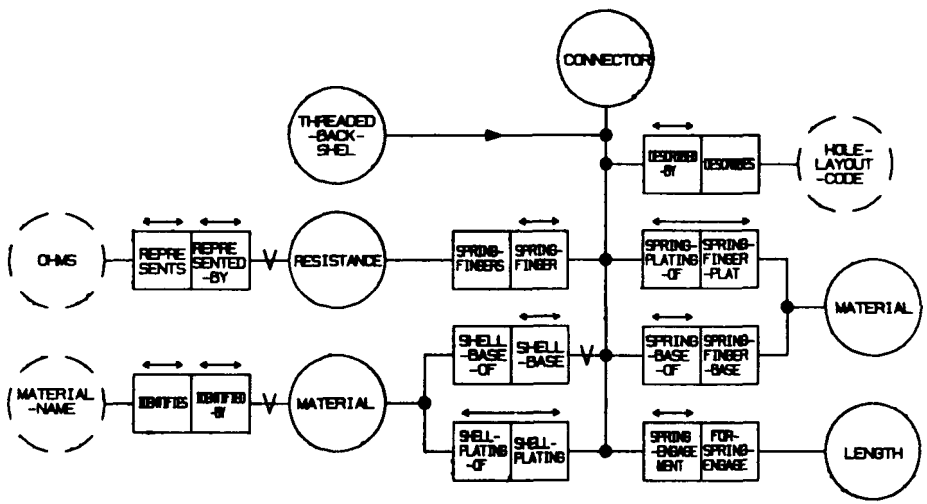
PIECE PARTS



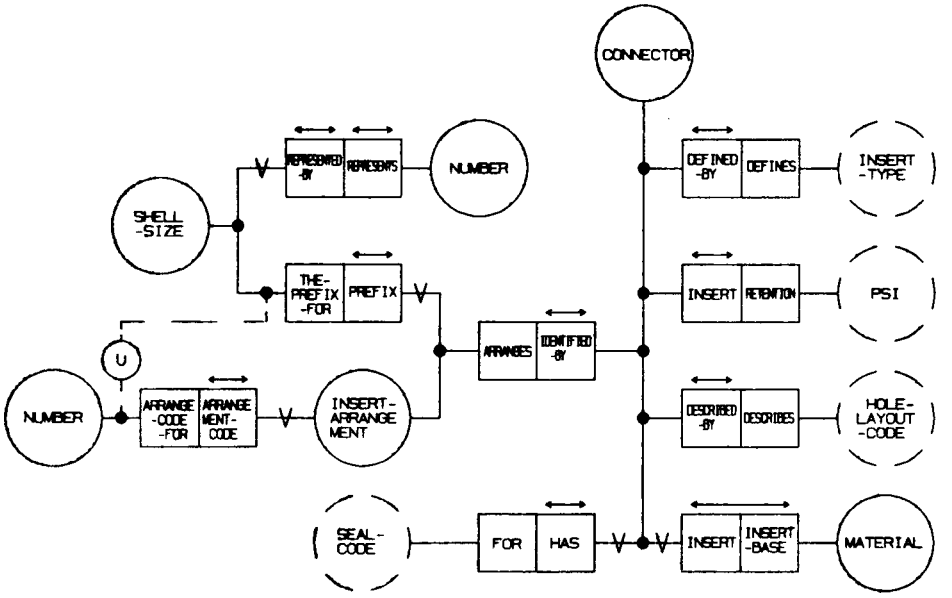
GENERAL INFORMATION



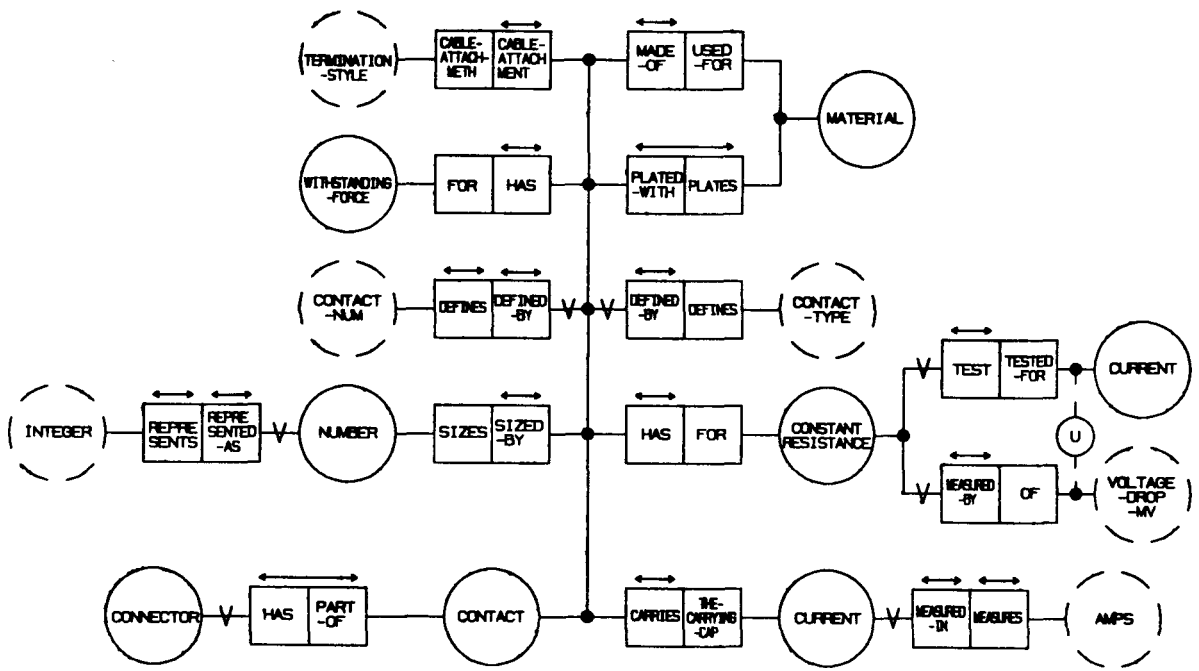
TEST REQUIREMENTS


CONNECTOR
THREADED-BACK-SHEL
HOLE-LAYOUT-CODE
OHMS
REPRESENTS
REPRESENTED-BY
RESISTANCE
SPRING-FINGERS
SPRING-FINGER
SPRING-PLATING-OF
SPRING-FINGER-PLAT
MATERIAL
SHELL-BASE-OF
SHELL-BASE
SPRING-BASE-OF
SPRING-FINGER-BASE
MATERIAL-NAME
MATERIAL
SHELL-PLATING-OF
SHELL-PLATING
SPRING-ENGAGEMENT
FOR-SPRING-ENGAGE
LENGTH

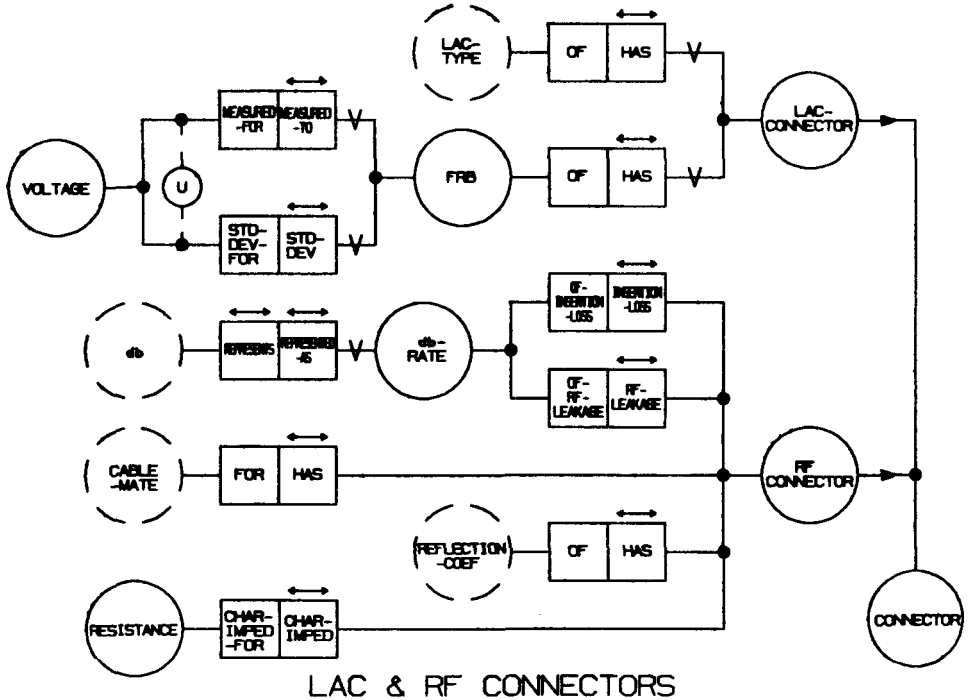
SHELLS


CONNECTOR
SHELL -SIZE
REPRESENTED -BY
REPRESENTS
NUMBER
THE-PREFIX -FOR
PREFIX
U
NUMBER
ARRANGE -CODE -FOR
ARRANGE MENT-CODE
INSERT-ARRANGE MENT
ARRANGES
IDENTIFIED -BY
DEFINED -BY
DEFINES
INSERT -TYPE
INSERT
RETENTION
PSI
DESCRIBED -BY
DESCRIBES
HOLE-LAYOUT -CODE
SEAL-CODE
FOR
HAS
INSERT
INSERT -BASE
MATERIAL

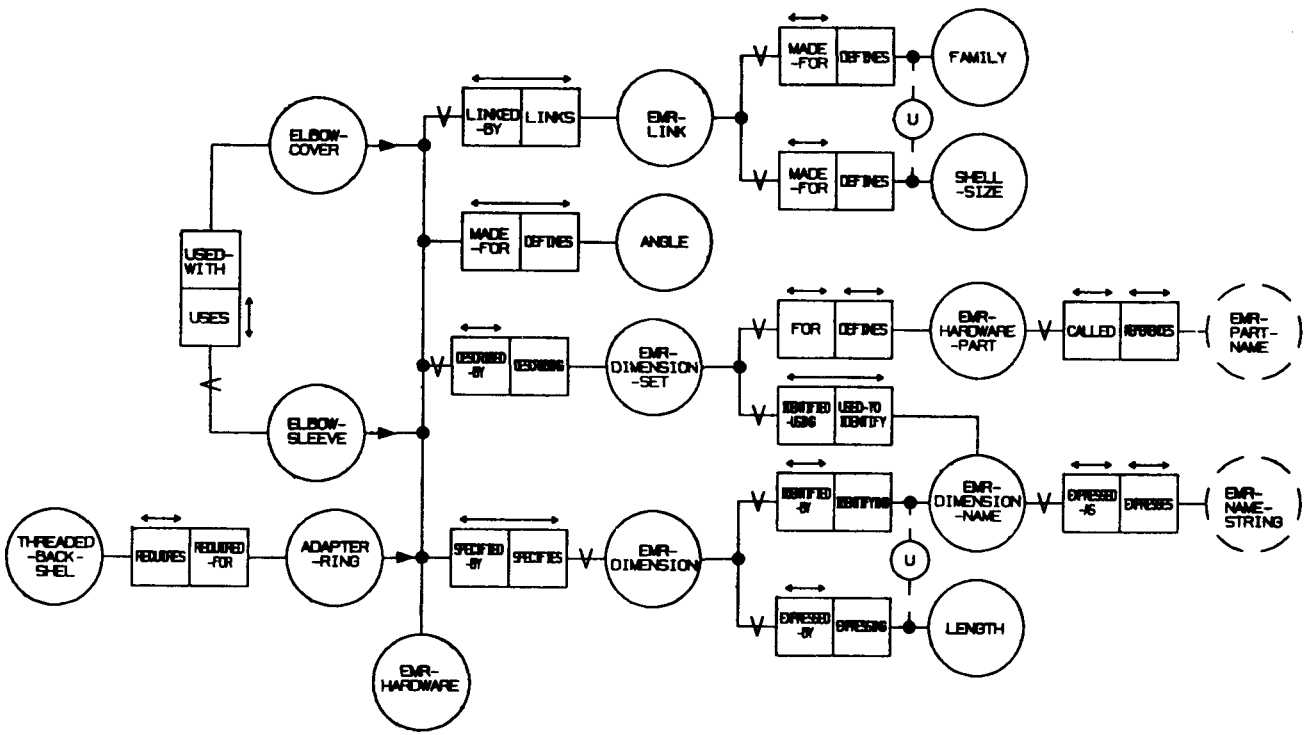
INSERTS


TERMINATION -STYLE
CABLE-ATTACH-METH
CABLE-ATTACH-MENT
MADE -OF
USED -FOR
MATERIAL
WITHSTANDING -FORCE
FOR
HAS
PLATED -WITH
PLATES
CONTACT -NUM
DEFINES
DEFINED -BY
DEFINED -BY
DEFINES
CONTACT -TYPE
INTEGER
REPRE SENTS
REPRE SENTED -AS
NUMBER
SIZES
SIZED -BY
HAS
FOR
CONSTANT RESISTANCE
TEST
TESTED -FOR
CURRENT
U
MEASURED -BY
OF
VOLTAGE -DROP -MV
CONNECTOR
HAS
PART -OF
CONTACT
CARRIES
CURRENT -CAP
CURRENT
MEASURED -IN
MEASURES
AMPS

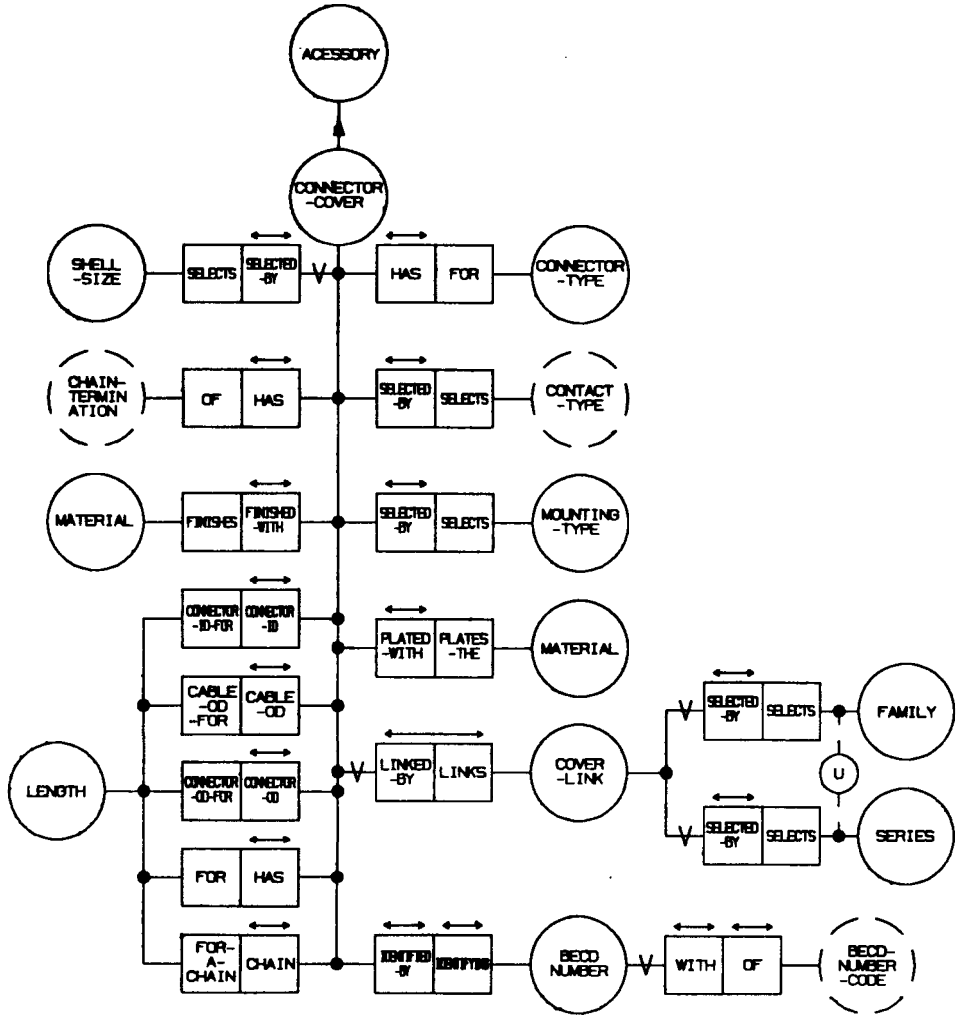
CONTACTS



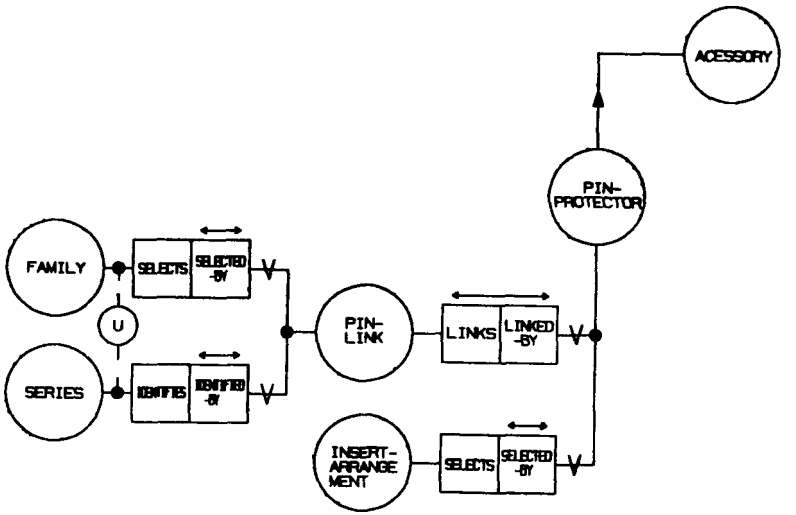
LAC & RF CONNECTORS



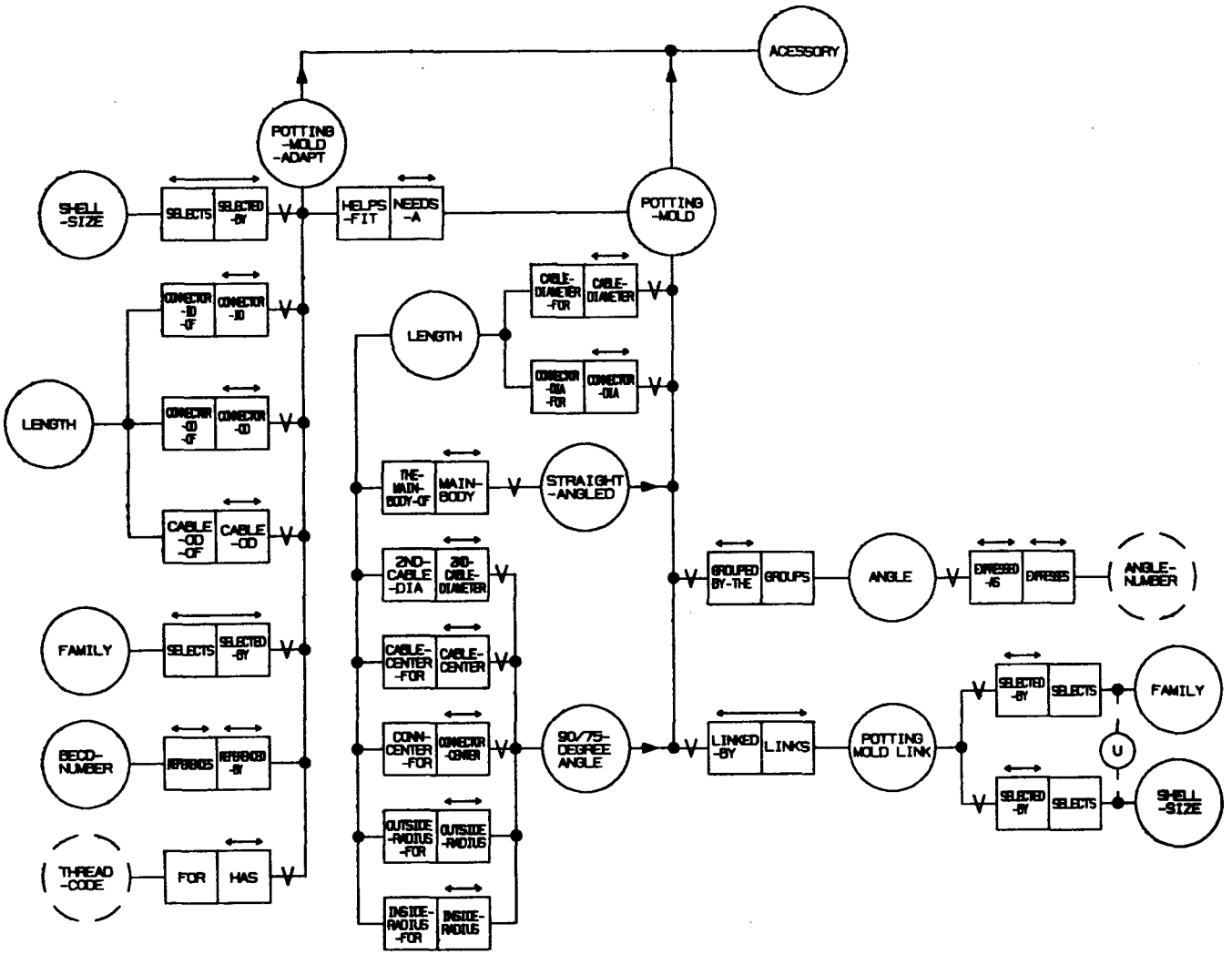
EMR HARDWARE



CONNECTOR COVERS



PIN PROTECTORS



POTTING MOLDS AND ADAPTER

Appendix C: Database Evaluation

The following characteristics were evaluated for each of the database management systems ORACLE, INGRES, BCSRim, RBase, IDMS/R, and Goldengate.

Costs - The cost is the published or list price of the database systems with the report generator, forms/applications generator, a link to an external language, and the graphics generator.

Version - The version number of the tool evaluated.

Design Compatibility - Compatibility with NIAM. The design methodology.

Name Lengths - The length of table names and the names of elements within the tables. The longer the names the more understandable they are and the easier they are to remember.

Data Types - What types are predefined for data? Those which are defined constrain and document the data.

Functions - The functions which assist in acquiring and manipulating data.

Constraints - The methods used to make sure the data in the database exists, is in the format required, and/or is within a specified range.

Security - The item which is checked to determine who can and can not access data.

Indexes - An index is a key for referencing a table.

Views - A view is a name for a defined query.

Compression - Does the database have an algorithm for keeping the storage space required as small as possible?

Query Language - The set of commands used to access data in a database.

Simplified Query Language - A subset of the query language which has been modified so it is easier for a casual user to understand and remember.

Example of Query Language - An example of the command required to find all the employees who make more than $40,000 per year. Employee is the table name. Ename is the name of the column in the table containing employees names. Esalary is the name of the column in the table containing employees salaries.

Menu Driven - Is the database driven mainly by menus, as opposed to the user having to type the commands out?

Report Generator - Is there a tool to assist in the arrangement and format of output reports?

Forms/Applications Generator - Is there a tool to assist in the development of forms or windows to acquire information from users, where the answer obtained from the user may lead to queries or they may lead to another form?

Interfaces to the Outside World - Precompilers for programming languages or links between programming languages and the database. These languages could be used to graphically display data from the database tool or to acquire information from the user.

PC Version - Is there a version of the database system which runs on a personal computer?

PC Compatibility - If a complete database application is created on the VAX version, will the same complete system execute on the PC version?

PC Runtime Version - Is there a version which will just execute completed applications on the PC, so the user need not purchase all the development tools?

Training - Is training included with the database?

Data Dictionary - A data dictionary is a tool which keeps track of what is in the database, the table names and their structure, the data types, the constraints, etc...).

Tutorials - On a scale of excellent-good-fair-ok-poor, how well did the online or manual tutorials assist the developer in learning the database and its tools?

Appearance - An opinion of the overall appearance of the database to a user.

First Impression - An impression of the database and its tools after a quick review of the manuals.

Attributes	Oracle	Ingres
Version	1	1.4
PC Version	$ 1,300	$ 1,150
MicroVAX Version	$ 17,500	$ 34,000
VAX 8600 Version	$140,000	$136,000
Design Compatibility	Yes	Yes
Name Lengths	30 Char	12 Char
Data Types	Char, Integer, Float, Money, Date	Char, Integer, Float, Money, Date
Functions	Mathematical, Boolean, Conversion	Mathematical, Boolean, String, Conversion
Constraints	No null, Data types	Data types
Security	User ID	User ID
Indexes	Yes	Yes
Views	Yes	Yes
Compression	No	Yes
Query Language	SQL	Quel
Simplified Query Language	SQL + & Easy SQL	
Example of Query Language	SELECT ename,esalary FROM employee WHERE esalary > 40000	RANGE OF e IS employe RETRIEVE (e.ename, e.esalary) WHERE e.esalary > 40000
Menu Driven	Yes	Yes
Report Generator	Yes	Yes
Forms/Applications Generator	Yes	Yes (VAX/microVAX)
Interfaces to the Outside World	Pascal, C, PL1, Fortran, Cobol	Pascal, C, Fortran, Cobol, Basic
PC Version	Yes, AT extended mem	Yes
PC Compatibility	Yes	Yes

Attributes	Oracle	Ingres
PC Runtime Version	No	No
Training	Yes	Yes
Data Dictionary	Yes	No
Tutorials	Good	Good
Appearance	Very Nice	Good
First Impression	English like and straight forward	English like with a programming complexity

Attributes	BCSRim	RBase
Version	7.0	1.1
PC Version		$700 (Run Time $50)
MicroVAX Version	$ 12,000	
VAX 8600 Version	$ 25,000	
Design Compatibility	Yes	Yes
Name Lengths	6 Char	8 Char
Data Types	Char, Integer, Float, Money, Date, Time	Char, Integer Float, Money, Date, Time
Functions	Mathematical, Boolean, Conversion	Mathematical, Boolean, Conver, Financial
Constraints	Programmed rules	Programmed rules
Security	Passwords	Passwords
Indexes	Yes	Yes
Views	No	Yes (5 tables)
Compression	Yes	No
Query Language	BCSRim	RBase
Simplified Query Language		Clout
Example of Query Language	SELECT ename,esalary FROM employee WHERE esalary gt 40000	SELECT ename,esalary FROM employee WHERE esalary gt 40000
Menu Driven	Yes	Yes
Report Generator	Yes	Yes (PC only)
Forms/Applications Generator	Yes (VAX/microVAX)	Yes (PC only)
Interfaces to the Outside World	Fortran	Pascal, C, Fortran
PC Version	Yes (RBase)	Yes (Only)
PC Compatibility	For Data Only	Yes

Attributes	BCSRim	RBase
PC Runtime Version	Yes (RBase)	Yes
Training	No	No
Data Dictionary	No	Yes
Tutorials	OK	OK
Appearance	Good	Good
First Impression	English like, but appears to be complex	English like, but appears to be complex

Attributes	IDMS/R	Goldengate
Version	10.2	1.3
PC Version		$600
MicroVAX Version		
VAX 8600 Version	$265,000 (IBM 4381)	
Design Compatibility	Yes	No
Name Lengths	32 Char	255 Char
Data Types	Char, Integer, Float	Char, Integer, Float, Money, Date
Functions	Mathematical, Boolean, Conversion	Mathematical, Boolean, Financial
Constraints	Data definition	Data type
Security	User ID	User ID
Indexes	Yes	No
Views	Yes	No
Compression	Yes	No
Query Language	OLQ (SQL)	Goldengate
Simplified Query Language	OLQ novice	
Example of Query Language	SELECT ename,esalary FROM employee WHERE esalary > 40000	SELECT ename = * #and# esalary > 40000
Menu Driven	Yes	Yes
Report Generator	Yes (IBM only)	Yes (PC only)
Forms/Applications Generator	No	No
Interfaces to the Outside World	Cobol, PL1, Assembler Fortran, RPG II	
PC Version	Yes (Goldengate)	Yes (Only)
PC Compatibility	For Data Only	Yes

Attributes	IDMS/R	Goldengate
PC Runtime Version	No	No
Training	Yes	Yes
Data Dictionary	Yes	No
Tutorials		Good
Appearance		OK
First Impression	SQL with a Menu System	Like dBASE III. It is really a spread-sheet not a database.

Appendix D: Tables and Indexes

Tables

```
create table CONNECTOR (cc_per_sec_helium_leakage_of number,
                dc_volts_for number not null,
                degrees_c_maximum_for number not null,
                degrees_c_minimum_for number,
                family_name_describing char(15) not null,
                g_mech_withstandable number,
                hole_layout_code_describes char(7),
                inches_spring_engage_of number,
                inch_lbs_the_coupling_of number,
                insert_type_defines char(15) not null,
                integer_arranges_a number(3) not null,
                integer_arranges number(3) not null,
                integer_contacts_for number(3) not null,
                series_describing char(10) not null,
                integer_max_matings_for number(4),
                internal_service_rating_code char(6) not null,
                integer_max_i_contacts_for number(4),
                lbs_for_coupling_pin number,
                material_name_spring_finger_ba char(15),
                material_name_the_shell_base char(15),
                milli_second_mech_withstandabl number,
                mounting_type_describes_mounti char(25) not null,
                ohms_for_humidity_of number,
                ohms_for_insulation_of number not null,
                ohms_spring_fingers_for number,
                part_suffix_number_for number(3) not null,
                psi_retention_for number,
                seal_code_for char(11) not null,
                stem_number_defining number(6) not null,
                type_name_for char(25));

create table CONNECTOR_MATING (
                part_suffix_numb_a_plug_part_o number(3) not null,
                part_suffix_nu_a_receptacle_pa number(3) not null,
                stem_number_a_plug_part_of number(6) not null,
                stem_number_a_receptacle_part number(6) not null);

create table CONN_CONTACTS (contact_ref_part_of number(6) not null,
                part_suffix_number_has number(3) not null,
                stem_number_has number(6) not null);

create table CONN_DIMENSIONS
               (dimension_name_str_specifies char(5) not null,
                inches_specifies number not null,
                part_suffix_numbe_specified_by number(3) not null,
                stem_number_specified_by number(6) not null);

create table CONN_INSERT_MAT
               (material_name_the_insert_base char(15) not null,
                part_suffix_number_insert number(3) not null,
                stem_number_insert number(6) not null);
```

```
create table CONN_SHELL_PLATE (
                material_name_shell_plating_fo char(15) not null,
                part_suffix_numb_shell_plating number(3) not null,
                stem_number_shell_plating number(6) not null);

create table CONN_SPRING_PLATE (
                material_name_spring_finger_pl char(15) not null,
                part_suffix_nu_spring_plating number(3) not null,
                stem_number_spring_plating_of number(6) not null);

create table CONTACT (amps_carrying_cap number not null,
                amps_for number not null,
                contact_ref_defines number(6) not null,
                contact_type_defines char(10) not null,
                integer_sizes number(2) not null,
                lbs_for number not null,
                material_name_used_for char(15) not null,
                termination_st_cable_attach_me char(15) not null,
                voltage_drop_mv_for number not null);

create table CONTACT_PLATE_MAT
               (contact_ref_plated_with number(6) not null,
                material_name_plates char(15) not null);

create table DIM_SET_DIM_NAME
               (dimension_name_used_to_identi char(5) not null,
                family_name_identified_using char(15) not null,
                series_identified_using char(10) not null);

create table LAC_CONNECTOR (dc_volts_of_a number not null,
                dc_volts_of number not null,
                lac_type_of char(15) not null,
                part_suffix_number_for number(3) not null,
                stem_number_defining number(6) not null);

create table SERIES (series_represents char(10) not null);

create table SHELL_SIZE (integer_represents number not null);

create table CONN_PERSON (first_name_str_of char(25) not null,
                middle_name_str_of char(25),
                person_sur_name_of char(25) not null,
                ssn_of number(9) not null);

create table PIECE_PART_A (family_class_code_for char(25) not null,
                part_suffix_number_for number(3) not null,
                ssn_sponsoring number(11) not null,
                stem_number_defining number(6) not null);

create table PIECE_PART_B (part_suffix_number_for number(3) not null,
                sa_number_stem_of char(6) not null,
                sa_number_suffix_of number(3),
                stem_number_defining number(6) not null);
```

```
create table PP_COMM_EQUIV (comm_equiv_an_alias_for char(20) not null,
                part_suffix_numbe_alt_known_by number(3) not null,
                stem_number_alt_known_by number(6) not null);

create table PP_MAN (manufacturer_name_producing char(25) not null,
                part_suffix_number_produced_by number(3) not null,
                stem_number_produced_by number(6) not null);

create table RF_CONNECTOR (cable_mate_for char(20),
                db_insertion_loss_for number,
                db_rf_leakage_for number,
                ohms_char_imped_for number,
                part_suffix_number_for number(3) not null,
                reflection_coef_of number,
                stem_number_defining number(6) not null);

create table CONN_SANDIA_EMPLOYEE (e_number_of number(5) not null,
                ssn_of number(9) not null);

create table SERVICE_RATING (dc_volts_measured_for number not null,
                feet_called_out number not null,
                internal_service_rating_code char(6) not null,
                rating_letters_references char(3) not null);

create table SEQNOS(tablename char(30) not null,
                maxseqno number(6) not null);

create table DESIGN_ENGINEER (first_name_str_of char(25) not null,
                middle_name_str_of char(25),
                person_sur_name_of char(25) not null,
                division number(4) not null,
                stem_number_defining number(6) not null,
                part_suffix_number_for number(3) not null);

create table MOUNTS (mounting_types char(25) not null);

create table SEAL_CODES(codes char(11) not null);

create table INSERTS(insert_types char(15) not null);

create table BASE_MATERIALS(material_types char(15) not null);

create table PLATING_MATERIALS(plating_types char(15) not null);

create table SHAPE(name char(15) not null);

create table TYPE(name char(15) not null);

create table CONTACT_TYPE(name char(15) not null);

create table DEGREES(name char(1) not null);

create table OHMS(name char(15) not null);
```

```
create table EMR_HARDWARE_TYPES (emr_part_name_references char (20) not null);

create table EMR_HARDWARE_ANGLE (stem_number_made_for number (6) not null,
        part_suffix_number_made_for number (3) not null,
        angle_number_defines number not null);

create table EMR_HARDWARE (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        emr_part_name_describing char (20) not null);

create table EMR_HARD_DIM (stem_number_specified_by number (6) not null,
        part_suffix_numbe_specified_by number (3) not null,
        emr_name_string_specifies char (10) not null,
        inches_specifies number not null);

create table EMR_HARD_LINK (stem_number_made_for number (6) not null,
        part_suffix_number_made_for number (3) not null,
        shell_size_defines number (3) not null,
        family_defines char (15) not null);

create table EMR_SET_NAMES (emr_part_name_identified_using char (20) not null,
        emr_name_string_used_to_id char (10) not null);

create table PIN_PROTECTOR (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        shell_size_selects number (3) not null,
        dash_number_selects number (3) not null);

create table POTTING_MOLD_ADAPT (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        thread_code_for char (20) not null,
        cable_id number not null,
        connector_od number not null,
        cable_od number not null,
        becd_number char (15));

create table POTTING_MOLD_ADAPT_LINK (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        shell_size_selects number (3) not null,
        family_selects char (15) not null);

create table POTTING_MOLD (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        stem_number_helps_fit number (6),
        part_suffix_number_helps_fit number (3),
        angle_number_groups number not null,
        inches_cable_diameter_for number not null,
        inches_connector_dia_for number not null);

create table POTTING_MOLD_LINK (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        shell_size_selects number (3) not null,
        family_selects char (15) not null);
```

```
create table ANGLED_POTTING_MOLDS (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        second_cable_diamter number not null,
        cable_center number,
        conn_center number,
        outside_radius number,
        inside_radius number);

create table CONNECTOR_COVER (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        connector_type char (25),
        shell_size_selects number (3) not null,
        contact_type char (10),
        mounting_type char (25),
        chain_termination_of char (15),
        material_finishes char (15),
        material_plates char (15),
        connector_id number,
        connector_od number,
        cable_od number,
        length number,
        chain_length number,
        becd_number_identifying char (15));

create table ADAPTER_RINGS (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        stem_number_required_for number (6) not null,
        part_suffix_required_for number (3) not null);

create table COVER_LINK (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        family_selects char (15) not null,
        series_selects char (10) not null);

create table PIN_PROTECTOR_LINK (stem_number_defining number (6) not null,
        part_suffix_number_for number (3) not null,
        family_selects char (15) not null,
        series_selects char (10) not null);
```

Indexes

```
create unique index CONN_INDEX on CONNECTOR
(stem_number_defining, part_suffix_number_for) NOCOMPRESS;

create index CONN_SELECT on CONNECTOR
(stem_number_defining, part_suffix_number_for,
family_name_describing, series_describing, integer_contacts_for)
NOCOMPRESS;

create unique index PLUG_MATING_INDEX on CONNECTOR_MATING
(stem_number_a_plug_part_of, part_suffix_numb_a_plug_part_o,
stem_number_a_receptacle_part, part_suffix_nu_a_receptacle_pa)
NOCOMPRESS;
```

```
create index RECEPTACLE_MATING_INDEX on CONNECTOR_MATING
(stem_number_a_receptacle_part, part_suffix_nu_a_receptacle_pa,
stem_number_a_plug_part_of, part_suffix_numb_a_plug_part_o)
NOCOMPRESS;

create unique index CONTACT_REF_INDEX on
CONN_CONTACTS(contact_ref_part_of, stem_number_has,
part_suffix_number_has) NOCOMPRESS;

create index CONTACT_PART_NUMBERS_INDEX on CONN_CONTACTS
(stem_number_has, part_suffix_number_has, contact_ref_part_of)
NOCOMPRESS;

create unique index CONN_DIMENSION_INDEX on CONN_DIMENSIONS
(stem_number_specified_by, part_suffix_numbe_specified_by,
dimension_name_str_specifies, inches_specifies) NOCOMPRESS;

create unique index CONN_INSERT_INDEX on CONN_INSERT_MAT
(stem_number_insert, part_suffix_number_insert,
material_name_the_insert_base)  NOCOMPRESS;

create unique index CONN_SHELL_INDEX on CONN_SHELL_PLATE
(stem_number_shell_plating, part_suffix_numb_shell_plating,
material_name_shell_plating_fo) NOCOMPRESS;

create unique index CONN_SPRING_INDEX on CONN_SPRING_PLATE
(stem_number_spring_plating_of, part_suffix_nu_spring_plating,
material_name_spring_finger_pl) NOCOMPRESS;

create unique index CONTACT_INDEX on CONTACT
(contact_ref_defines) NOCOMPRESS;

create unique index CONTACT_PLATING_INDEX on CONTACT_PLATE_MAT
(contact_ref_plated_with, material_name_plates) NOCOMPRESS;

create unique index DIM_SET_INDEX on DIM_SET_DIM_NAME
(family_name_identified_using, series_identified_using,
dimension_name_used_to_identi)  NOCOMPRESS;

create index FAMILY_INDEX on DIM_SET_DIM_NAME
(family_name_identified_using) NOCOMPRESS;

create unique index LAC_INDEX on LAC_CONNECTOR
(stem_number_defining, part_suffix_number_for) NOCOMPRESS;

create unique index SERIES_INDEX on SERIES  (series_represents)
NOCOMPRESS;

create unique index SHELL_SIZE_INDEX on SHELL_SIZE
(integer_represents) NOCOMPRESS;

create unique index CONN_PERSON_INDEX on CONN_PERSON (ssn_of)
NOCOMPRESS;
```

```
create unique index PIECE_PART_A_INDEX on PIECE_PART_A
(stem_number_defining, part_suffix_number_for) NOCOMPRESS;

create unique index PIECE_PART_B_INDEX on PIECE_PART_B
(stem_number_defining, part_suffix_number_for) NOCOMPRESS;

create index SA_SELECT_INDEX on PIECE_PART_B
(sa_number_stem_of, sa_number_suffix_of) NOCOMPRESS;

create unique index COMM_EQUIV_INDEX on PP_COMM_EQUIV
(stem_number_alt_known_by, part_suffix_numbe_alt_known_by,
comm_equiv_an_alias_for) NOCOMPRESS;

create unique index MAN_INDEX on PP_MAN
(stem_number_produced_by, part_suffix_number_produced_by,
manufacturer_name_producing) NOCOMPRESS;

create unique index RF_INDEX on RF_CONNECTOR
(stem_number_defining, part_suffix_number_for) NOCOMPRESS;

create unique index CONN_SANDIA_SSN_INDEX on CONN_SANDIA_EMPLOYEE
(ssn_of) NOCOMPRESS;

create unique index SERVICE_RATING_INDEX on SERVICE_RATING
(internal_service_rating_code) NOCOMPRESS;

create unique index SHAPE_INDEX on SHAPE(name) NOCOMPRESS;

create unique index TYPE_INDEX on TYPE(name) NOCOMPRESS;

create unique index CONTACT_TYPE_INDEX on CONTACT_TYPE(name)
NOCOMPRESS;

create unique index DEGREES_INDEX on DEGREES(name) NOCOMPRESS;

create unique index OHMS_INDEX on OHMS(name) NOCOMPRESS;

create unique index MOUNT_INDEX on MOUNTS(mounting_types)
NOCOMPRESS;

create unique index SEAL_INDEX on SEAL_CODES(codes) NOCOMPRESS;

create unique index INSERT_INDEX on INSERTS(insert_types)
NOCOMPRESS;

create unique index BASE_MATERIAL_INDEX on
BASE_MATERIALS(material_types) NOCOMPRESS;

create unique index PLATING_MATERIAL_INDEX on
PLATING_MATERIALS(plating_types) NOCOMPRESS;

create unique index DESIGN_INDEX on
DESIGN_ENGINEER(stem_number_defining, part_suffix_number_for)
NOCOMPRESS;
```

```
create unique index SEQ_INDEX on SEQNOS (tablename, maxseqno)
NOCOMPRESS;

create unique index EMR_TYPE_INDEX on EMR_HARDWARE_TYPES
(emr_part_name_references);

create unique index EMR_ANGLE_INDEX on EMR_HARDWARE_ANGLE
(stem_number_made_for, part_suffix_number_made_for,
angle_number_defines);

create unique index EMR_HARDWARE_INDEX on EMR_HARDWARE
(stem_number_defining, part_suffix_number_for);

create unique index EMR_DIM_INDEX on EMR_HARD_DIM
(stem_number_specified_by, part_suffix_numbe_specified_by,
emr_name_string_specifies, inches_specifies);

create unique index EMR_LINK_INDEX on EMR_HARD_LINK
(stem_number_made_for, part_suffix_number_made_for,
shell_size_defines, family_defines);

create unique index EMR_SET_INDEX on EMR_SET_NAMES
(emr_part_name_identified_using, emr_name_string_used_to_id);

create unique index PIN_PROTECTOR_INDEX on PIN_PROTECTOR
(stem_number_defining, part_suffix_number_for,
shell_size_selects, dash_number_selects);

create unique index PIN_PROTECTOR_LINK_INDEX on
PIN_PROTECTOR_LINK (stem_number_defining, part_suffix_number_for,
family_selects, series_selects);

create unique index MOLD_ADAPT_INDEX on POTTING_MOLD_ADAPT
(stem_number_defining, part_suffix_number_for);

create unique index MOLD_ADAPT_LINK_INDEX on
POTTING_MOLD_ADAPT_LINK (stem_number_defining,
part_suffix_number_for, shell_size_selects, family_selects);

create unique index MOLD_INDEX on POTTING_MOLD
(stem_number_defining, part_suffix_number_for);

create unique index MOLD_LINK_INDEX on POTTING_MOLD_LINK
(stem_number_defining, part_suffix_number_for,
shell_size_selects, family_selects);

create unique index ANGLED_MOLD_INDEX on ANGLED_POTTING_MOLDS
(stem_number_defining, part_suffix_number_for);

create unique index COVER_INDEX on CONNECTOR_COVER
(stem_number_defining, part_suffix_number_for);
```

```
create unique index COVER_LINK_INDEX on COVER_LINK
(stem_number_defining, part_suffix_number_for, family_selects,
series_selects);
```

Appendix E: Input/Update Form

Connector Selection Input or Change Form

General Information

* Type: _____ (plug/receptacle)
* Family: _____ (LJT, JT, PT, R&P,...)
* Series: _____ (00,02,06,07,07H,...)
* Mounting Type: _____ (see list below)
* Seal Code: _____ (hermetic/nonhermetic)
* Insert Arrangement: ____ - ____ (shell size - no. contacts/dash no.)
* Insert Type: _____ (subminiature, miniature, standard, double density)
* Number of Contacts: _____
* Number of Contacts carrying max current: _____
* Service Rating: _____ (I, II, M, ...)
* Sandia Number: _____-_____

  Hole Layout Code: _____

  SA/MC Number:_____-_____

Material Information

* Shell Base Material: _____ (see list below)

  Shell Platings: _____ (see list below)

  _____

  _____

* Insert Materials: _____

  _____

  _____

  Spring/Grounding Finger Base Material: _____

  Spring/Grounding Finger Plating Materials: _____

  _____

  _____

Environment Information

* Dielectric Withstanding Voltage: _____ V DC
* Insulation Resistance: _____ Megohms
  Spring Engagement: _____ Inches
  Spring Finger Resistance: _____ Ohms
  Air/Helium Leakage: _____ cc/sec
  Insert Retention: _____ psig
  Durability: _____ times coupled and uncoupled
  Physical/Mechanical Shock: _____ g
  _____ milliseconds
  Coupling Pin Strength: _____ inch
  _____ lbs
  Moisture/Humidity Resistance: _____ Megohms
* Maximum Operating Temperature: _____ Degrees C/F
  Minimum Operating Temperature: _____ Degrees C/F

Contact Information (per contact size)

* Type: _____ (pin/socket)
* Size: _____
* Current Rating: _____ amps
* Termination Style: _____ (solder cups, weldable,...)
* Base Material: _____
* Test Current: _____ amps
* Test Voltage: _____ mV
* Retention: _____ lbs
  Plating Materials: _____
  _____
  _____

Responsible Engineer

Name: _____

Division: _____

Misc.

LAC Type: _____

LAC FRB: _____

RF Insertion Loss: _____ db

RF Leakage: _____ db

RF Reflection Coefficient: _____

RF Characteristic Impedence: _____ ohms

RF Compatible Cable: _____

Manufacturer: _____

Commercial Equivalent: _____

Mating Connectors: _____(Drawing Numbers)

_____

_____

_____

* Dimensions: Letter Inches(see note below)

Families

Dwarf	LV
JT	HV
LJT	RF
R&P	PC Board
PT	Micro Miniature

Mounting Types

Jam Nut	Square Flange 2 Hole
Square Flange 4 Hole	8 Hole Flange
4 Hole R&P	2 Hole R&P
Threaded Panel	Weld Flange
Encapsulation	Solder
Cable Mount	Bulkhead
PC Board	Right Angled PC Board

Base Materials

Stainless Steel	Steel
Aluminum	Aluminum Alloy
Alloy 52	Beryllium Copper
Copper Alloy	Plastic
Diallyl Phthalate	Phenolic
Epoxy	Glass Filled Epoxy
Glass	Brass
Ceramic	High Impact Epoxy
PTFE	PCTFE
Teflon	Bronze
Nylon Molding	Molded Plastic

Plating Materials

Gold	Silver

Nickel	Cadmium
Tin	Copper
Aluminite	

Dimensions

To specify dimensions, a letter and a value are needed. The letter indicates what the dimension is for. The value is the length, diameter, or radius given to your connector for the specified dimension. The letters used for a given connector can currently be found in the SA/MC Special Design Connector Catalog or the Connector Selection Graphics notebook. If no letters are given for the connector you are entering, then call Nicole Sevier at 6-2993. You should know the values.

Appendix F: PC to VAX Transfer Description

To transfer Oracle applications on the PC to the MicroVAX it is necessary to transfer the following:

1. The application tables and rows of data.
2. The application views and indexes.
3. The application user interface or SQL*FORMS file.

To acquire the application tables and rows use the Oracle export procedures. Export only the tables defined by your application. Do not export system tables.

The file created by the export procedure is a binary file. To transfer a binary file to the MicroVAX, the 891 Building VAX (sav49) and polygon poly-COM/240 terminal emulation and file transfer software were used. The 891 Building VAX was used because the MicroVAX did not have any file transfer software available. The procedures followed to make this transfer are described in detail below.

1. Use polygons TRM routine to emulate a terminal. This is done by typing:

   ```
   trm
   ```

   and setting up all the screens so the two machines are talking the same language. Once this is done, press F3 to start the emulating.
2. Log in to the 891 Building VAX.
3. Proceed to the directory where you want the files transferred to.
4. Invoke the 891 Building VAX file transfer software, HST. This is done by typing:

   ```
   hst
   ```

5. Press Alt-F10 X to exit the polygon TRM routine.
6. Transfer the binary file, BINARY.DMP, from the PC to the 891 Building VAX file, VAX.DMP by typing:

   ```
   xfr BINARY.DMP/i to VAX.DMP/im
   ```

7. Invoke the polygon TRM software by typing:

   ```
   trm
   f3
   ```

8. To exit the 891 Building VAX file transfer software, type Ctrl-Z.

To get this binary file, VAX.DMP from the 891 Building VAX to the MicroVAX you need use only the copy command. This can be done by typing:

```
copy VAX.DMP sav248::*
```

To transfer the views and the indexes it is necessary to have the ASCII files which were used to create them originally. These can be transferred to the MicroVAX using most terminal emulation software; such as, polygon, vterm, kermit, etc...

The polygon software and the 891 Building VAX file transfer software were used to transfer these ASCII files. The only difference between a binary file transfer and an ASCII file transfer are the options used (/i and /im). For example, to transfer the ASCII file, PC_INDEXES.SQL to VAX_INDEXES.SQL the xfr command would be as follows:

```
xfr PC_INDEXES.SQL to VAX_INDEXES.SQL
```

The same transfer command can be used for the views, too. The copy command, as shown above, will get these files from the 891 Building VAX to the MicroVAX.

To transfer the application user interface you must first generate an inp file on the PC using SQL*FORMS development software. This file can be transferred as an ASCII file to the MicroVAX as described above.

Once the inp file is on the MicroVAX you may load it using the SQL*FORMS development software or generate an executable forms file. The first choice will allow you to manipulate the SQL*FORMS application on the MicroVAX. The latter only allows you to run the SQL*FORMS application on the MicroVAX.

To create an executable forms file from the transferred inp file, VAX.inp, type:

```
iag VAX -to
```

To run this file use runforms, as was used with the PC Oracle software.

Appendix G: CRT Files & Key Definition Description

Creating New CRT Files

This document describes the method used to create a new CRT file. A new CRT file was created to map the keys used on a VT100 to the following SQL*FORMS operator functions:

Action	Key
Next Field	Return
Previous Field	Left Arrow or Numeric 4
Next Record	Down Arrow or Numeric 2
Previous Record	Up Arrow or Numeric 8
Next Screen	PgDn or Numeric 3
Previous Screen	PgUp or Numeric 9
Previous Menu	Home or Numeric 7
Clear Field	Numeric 5
Help	PF1
Invoke Search	PF2
List of Values	PF3
Exit	PF4

The simplest way to access the ORACLE tables associated with CRT definitions was to use the CRT forms provided with ORACLE. The CRT.inp file was found in the sys$oracle directory. An executable version of this form was created by typing:

```
iag crt -to
```

Once this file was created, the CRT form was created by typing:

```
runform crt system/password
```

The user must have system privileges to access the ORACLE tables associated with CRT definitions.

After the CRT form was invoked, the execute query key was pressed to have all the CRT files loaded. This list was scanned through using the Next Record key to see all the CRT files that were available. Page 239 of the ORACLE Database Administrators Guide has a list of all the CRT files defined by ORACLE.

The CRT forms contains two screens in sequence. The first defines how the screen will appear to the engineer on a given terminal device. The screen defines how a box will be displayed on a given terminal device and maps the keys on the keyboard to ORACLE software functions.

```
        Crt Name:                              Type: CHAR

  Columns: 80    Lines: 24    Mode Line: 24    Message Line: 23    Base: 1

   Terminal Setup:                    Term Reset:
      Graphics On:                   Graphics Off:
       Protect On:                    Protect Off:
           2X Top:                      2X Bottom:

    Clearscreen:            Clear Line:              Clear to EOL:
  Create Window:             Scroll Up:              Scroll Down:
     C-L Offset:              Goto C-L:                  C-L Map:

  Cursor Left:          Right:              Up:             Down:

  Video  Bold:          Flash:            Revrs:           Under:
 Attribs  Off:              1:               2:              3:
        1+2+3:            1+2:             2+3:            1+3:


        Char Mode: Replace  Page 1                    Count: *0
```

CRT Screen: Page 1

```
                  Graphics (Y/N):                          Screen    Printer
                                              Vertical line:
                                            Horizontal line:
Crt Name                                  Upper left corner:
                                         Upper right corner:
                                          Lower left corner:
                                         Lower right corner:
                                                Left-side T:
                                               Right-side T:
                                                  Upright T:
                                                 Inverted T:
                                                  Cross (+):


  Prod  Fn
  Code  Cd  Function Description       Escape Sequence  Comments


        Char Mode: Replace  Page 2                    Count: *0
```

CRT Screen: Page 2

For a formal definition of these screens and the ORACLE CRT system, see pages 71-88 of the ORACLE Database Administrator's Guide.

The method used for creating the new CRT file was to copy the existing VT100 CRT file, except for the key definitions for the SQL*FORMS operator.

To do this, the VT100 CRT information was found on the first screen. A new record was created for the new CRT file, by pressing the create record key. The duplicate record key was pressed, to duplicate the information in the previous record (VT100). The next step was to change the name of the CRT being defined to the new CRT name, TVT100 (for test VT100).

The box drawing information was input for the new CRT. To copy the VT100 box drawing information access to the field which states what CRT the box drawing information is for is needed (so it could be changed from VT100 to TVT100). ORACLE did not create the form with "Modify Access" to this field. The CRT form should be modified to allow this, but to get around it the same values that were given for the VT100 CRT were input for the TVT100 CRT.

The same problem existed for the key definitions. The forms did not allow access to the field that defines what CRT file the keys being displayed correspond to. Thus, new keys had to be input; they could not be copied and then altered. The next few paragraphs describe each of the fields used to map a key to an ORACLE software function.

Look at the lower section of the second CRT screen. The first field is the product code. The product code is the code for the ORACLE tool that the key definitions are being made for: in this case, the SQL*FORMS operator. The product code for this is IAP.

The next field is the function code. The function codes available for the SQL*FORMS operator or IAP can be found on page 236 of the ORACLE Database Administrator's Guide.

The function description field is filled in automatically, based on what is in the function code field.

The Escape Sequence tells ORACLE what key this function will be associated with. The escape sequence associated with a given key depends on what terminal emulator or device is being used. To find out what the escape sequences were for a VT100, he August 4, 1987 VTERM 4010 manual was consulted. This information can be found in Chapter 12: The Keyboard, Mouse, and Terminal Screen, tables 12-2 (page 135) and 12-3 (page 137).

The comments field normally explains what keyboard key the function has been associated with, because the escape sequence is relative to the device.

The following table illustrates the keys as they were defined for the new CRT file TVT100:

Function Code	Description	Escape Sequence	Comments
CA	Clear Form/Rollback	\e0w	Numeric 7
CM	Insert/Replace	\e0n	. or Del
DB	Delete Backward		Delete
DE	Display Error	\e01	,
DK	Show Function Keys	*\ek	Escape k
EQ	Enter Query	\e0P	PF1
LV	List Field Values	\e0R	PF3
ML	Left	\e0D	<-
MR	Right	\e0C	->
NB	Next Block	\e0s	Numeric 3
NF	Next Field	\015	Return
NF	Next Field	\011	Tab
NR	Next Record	\e0B	Down Arrow
NR	Next Record	\e0r	Numeric 2
P	Print	*\ep	Escape p
PB	Previous Block	\0y	Numeric 9
PF	Previous Field	\010	Back Space
PF	Previous Field	\e0t	Numeric 4
PR	Previous Record	\e0A	Up Arrow
PR	Previous Record	\e0x	Numeric 8
Q	Execute Query	\e0Q	PF2
R	Redisplay Page	*\er	Escape r
X	Exit/Cancel	\e0S	PF4
X	Exit/Cancel	\032	Ctrl Z
CF	Clear Field	\e0u	Numeric 5

A copy of the keyboard layout as it appears using TVT100 is shown in the VT100 map on the next page.

All the key definitions shown above were input into the key definition section (on the lower section of the second CRT screen), the records were committed by pressing the commit key, and the CRT forms were exit using the exit key.

VT100 Keyboard Layout

F1 Help	F2 Invoke Search
F3 List of Values	F4 Exit
F5	F6
F7 Previous Record	F8 Next Record
F9	F10

Next Field = Return

Previous Field = Shift Tab

7 Previous Menu	8 Previous Record	9 Previous Screen	*
4 Previous Field	5 Clear Field	6	–
1	2 Next Record	3 Next Screen	+
0		.	

with NUM

To actually create a TVT100 CRT file that could be used by engineers, the new TVT100 CRT must be run through the CRT program. This was done by typing the following:

```
CRT TVT100 system/password
```

The CRT file could now be used when running a SQL*FORMS application. It could not be used with any of the other ORACLE tools. Key definitions were only defined for SQL*FORMS applications. Additional key definitions could be included for other ORACLE tools.

Implementing Variable Key Definitions

To implement different key definitions for the connector selection program the following procedures were followed.

To start off, the trigger functions that were used by the connector selection program were reviewed and the following list was created:

Function	Trigger Name
Next Field	Key-Nxtfld
Previous Field	Key-Prvfld
Next Record	Key-Nxtrec
Previous Record	Key-Prvrec
Next Block	Key-Nxtblk
Previous Block	Key-Prvblk
List of Values	Key-Listval
Help	Key-Entqry
Exit	Key-Exit
Previous Menu	Key-Clrfrm
Execute Query	Key-Exeqry

The functions help, list of values, and exit were defined as form triggers since the same action is performed regardless of the current block or field. The remaining functions were defined as block triggers or field triggers, depending on where they occurred and the actions they were to perform.

A table named Keys was defined as follows:

```
Create table KEYS (TERMINAL_TYPE char(10) not null,
                   NXTFLD char(5),
                   PRVFLD char(5),
                   NXTREC char(5),
                   PRVREC char(5),
                   NXTBLK char(5),
                   PRVBLK char(5),
                   LISTVAL char(5),
                   HELP char(5),
                   EXIT char(5),
                   MENU char(5),
                   EXEQRY char(5));
```

The following information was placed in this table:

Terminal	Nxtfld	Prvfld	Nxtrec	Prvrec	Nxtblk	Prvblk	Listv	Help	Exit	Menu	Exeqr
PC	Enter	^-Tab	v	^	PgDn	PgUp	F3	F1	F4	Home	F2
VT100	Enter	^-Tab	v	^	PgDn3	PgUp9	F3	F1	F4	Home7	F2
VT220	Enter	F12	v	^	Next	Prev	F13	Selct	PF4	F17	Find

To implement this idea, two blocks were created. One block, KEYS, was created to perform the query to the KEYS table. This query looks in the engineer's table TERMINAL_TYPE (field TERMINAL) to find out what terminal type the engineer has. Based on this information, the block searches the table KEYS for the corresponding key definitions.

The second block, OUTPUT_KEYS, contains the fields on all the screens where the key definitions are to be displayed. The names of these fields were created by concatenating the key name with the fields page number. For example, nxtblk_10 displays the nxtblk key (PgDn, PgDn3, or Next) on page 10 of the application. The values to be placed in these fields are "copied from" the corresponding field in the KEYS block.

Thus, to get the keys displayed initially the following actions were performed:

```
Goblk keys;              -- Go to the block KEYS
Exeqry;                  -- Find the appropriate key
                            definitions
Goblk output_keys;       -- Copy the key definitions to all
                            the screens.
```

Note that whenever a clear form function is performed, the above actions must be invoked, again.

The type of CRT file used must match the type of keys displayed by the keys blocks. This was done in the following manner:

1. Determine the engineer's terminal type via set term/inq (or by asking the engineer if set term/inq doesn't work);
2. Have the engineer create the table TERMINAL_TYPE with the field TERMINAL;
3. Insert the engineer's terminal type into the table TERMINAL_TYPE;
4. Invoke the application with the appropriate CRT file;
5. And when the engineer is finished with the application, drop the table TERMINAL_TYPE

See the User interface Command Procedure for more information.

Appendix H: User Interface Command Procedure

RunConsel

```
$set noon
$set nover
$! ---------------------- Run Consel. com ----------------------------------
$!
$! Created:
$!    Date:  November 7, 1988
$!    Author:  N. E. Sevier
$! Modified:
$!    Date:  April 14, 1989
$!    Author:  N. E. Sevier
$!    Changes:  Set logicals to reference the disk sd: since users are on usr:
$!              Need a logical name for the first form called, consel1g
$!
$! This routine will make sure that the user's terminal type is set
$! based on the terminal s/he is currently using.
$! Once the terminal type is known, the forms application must be told
$! what type of terminal the user has.  The only medium which both SQL*FORMS
$! and VMS have in common is the database.  Thus, we will use SQL*Plus to
$! create a table, terminal_type, in Oracle.  This table will contain the users
$! terminal type.  Currently, only VT100 and VT220 are supported by this
$! command file.
$! The forms application will be invoked with the appropriate crt file.
$! When the application is complete, SQL*Plus will be used to delete the table
$! terminal_type.
$!
$! A logical name must be created so that oracle can find the second part of
$! the connector selection form, consel2v.  Consel2v is in the subdirectory
$! consel and the only way to tell oracle this is to create a logical name
$! so that consel2v = [conuser]consel2v.  Thus, a process logical name
$! is created by this command procedure.
$!
$! A logical name must also be created so that oracle can find the third part
$! of the connector selection form, acc_output.  Acc_output is in the sub
$! directory consel and the only way to tell oracle this is to create a logical
$! name so that acc_output = [conuser]acc_output.  Thus, a process
$! logical name is created by this command procedure.
$!-------------------------------------------------------------------------------
$!
$! Create logical name so consel2v and acc_output can be found.
$!
$assign/nolog/table=lnm$process_table sd:[conuser]consel1g consel1g
$assign/nolog/table=lnm$process_table sd:[conuser]consel2g consel2g
$assign/nolog/table=lnm$process_table sd:[conuser]acc_output acc_output
$!
$! Find out the user's terminal type
$!
$set term/inq
$if .not. $status then goto askuser
$pid = ""
$loop:
```

```
$   on error then goto error_message
$   term = f$getjpi(f$pid(pid),"TERMINAL")
$   if term .eqs. "" then goto loop
$end_loop:
$num = f$getdvi(term,"DEVTYPE")
$!
$! Branch so that the SQL*FORMS can be called with the proper crt file
$!
$if num .eqs. 96 then goto vt100
$if num .eqs. 98 then goto vt100
$if num .eqs. 110 then goto vt200
$!
$! The system cannot determine what type of terminal the engineer has, so this
$! procedure must ask the user.
$!
$askuser:
$type sys$input

                    Terminal Types Available

          VT100  -  Includes all VT100 Series Terminals or Emulators
          VT200  -  Includes all VT200 and VT300 Series Terminals or Emulators

$inquire choice "Enter your terminal type (VT100 or VT200)"
$choice = f$edit(choice,"UPCASE")
$!
$! Branch to so that SQL*FORMS can be invoked with the proper crt file.
$!
$on warning then goto error_message
$goto 'choice'
$!
$! Invoke SQL*FORMS with the Test VT100 crt file (defines keys commonly
$! found on a PC keyboard)
$!
$vt100:
$write sys$output "You are running as a VT100"
$sqlplus -silent ""/ @sd:[conuser]create_vt100
$define/user sys$input sys$command
$runform consel1g -qc TVT100 ""/
$sqlplus -silent ""/ @sd:[conuser]drop_terminal
$@sd:[conuser]log
$exit
$!
$! Invoke SQL*FORMS with the VT220 crt file.
$!
$vt200:
$write sys$output "You are running as a VT220"
$sqlplus -silent ""/ @sd:[conuser]create_vt220
$define/user sys$input sys$command
$runform consel1g -qc VT220 ""/
$sqlplus -silent ""/ @sd:[conuser]drop_terminal
$@sd:[conuser]log
$exit
$!
```

```
$! Prints out an error message if the user enters an invalid terminal type.
$!
$error_message:
$on warning then goto error_message
$write sys$output " "
$write sys$output "Unknown Terminal Type"
$write sys$output " "
$goto askuser
$exit
```

Create VT100

```
create table terminal_type (terminal char (5) not null);
insert into terminal_type values ('VT100');
exit;
```

Create VT220

```
create table terminal_type (terminal char(5) not null);
insert into terminal_type values ('VT220');
exit;
```

Drop Terminal

```
drop table terminal_type;
exit;
```

Appendix I: Help Command Procedure

Help.Com

```
$!*********************  Display Help . com  ************************
$!
$!  Created:
$!    Author:  Nicole E. Sevier, 2551
$!    Date:  April 13, 1989
$!
$!  Revised:
$!    Author:
$!    Date:
$!    Notes:
$!
$!  This procedure is called by SQL*FORMS when a user presses the help button.
$!  The help trigger must determine from the table connector_help what
$!  file to display based on the current form, block, and field.  The table
$!  connector_help has the following fields: help_file, form, block, and field.
$!  The current form, block, and field are stored in the system variables
$!  system.current_form, system.current_block, and system.current_field.
$!
$!  This command procedure simply displays the help_file for the form, block,
$!  and field the user is located on.
$!
$!*****************************************************************************
$type/page sd:[conuser.help]'P1'
$exit
```

Sample Help Text

This is the search selection menu. This is where you select the type of search you would like to perform. There are five search methods provided for you. Each of these is described below:

Fields:

Search Number - To select one of these search numbers, you must type in the number to the left of the description that best describes the search you want to perform. Once you have typed in the number, the Invoke Action key described in the key definition area should be pressed. This tells the system that you have made a selection. After you have pressed the Invoke Action key, a screen will appear where you may enter your search conditions. The types of searches available are:

1. Search by SA number - If you know all or part of the Sandia Apparatus (SA) number or the MC number for a connector that you want information about, then select this option. The SA number is a number that is described as: 'SA' followed by 4 digits followed by a dash, '-', and a dash number. For example, SA1457-5 is an SA number for a connector. The stem, SA1457, usually references a group of connectors that have the same general characteristics, whereas the dash number identifies the

individual connectors in the group. The same is true for Major Component (MC) numbers, except the first two characters are "MC," instead of "SA."

2. Search by Drawing number - If you know all or part of a Sandia drawing number for a connector that you want information about, then select this option. The Sandia drawing number is a 6 digit number that references the top drawing (usually the Automated Materials List or AML) for a specific connector. The top drawing will make reference to or lead you to all other drawings associated with that connector.

3. Search by General Information - If you know any or all of the following information then you want to select this option.

   a. connector family (Dwarf, LJT, JT, HV, RF, ...)
   b. connector type (plug, receptacle, or adapter)
   c. whether the connector needs to be hermetic or nonhermetic
   d. the mounting type for the connector
   e. the shell size for the connector
   f. the contact style (pins or sockets)
   g. the number of contacts
   h. the current rating for the described contacts

   If you know more about the connector than what is listed here, you may want to search by specific information.

4. Search by Specific Information - If you know any or all of the general information and any of the following, then you want to search for connectors using this option.

   a. maximum operating temperature
   b. insulation resistance
   c. dielectric withstanding voltage (in V DC)
   d. insert arrangement dash number
   e. shell base and plating materials
   f. insert material
   g. spring finger base and plating materials

Key Definitions:

Invoke Action - will tell the program that you have made a selection and that you are ready to continue.

Help - will display help text.

Exit - will terminate the program.

Appendix J: Graphics Command Procedure

Graphics.Com

```
$!---------------------- Graphics . Com -------------------------------
$!
$! Creation:
$!   Author:  N. E. Sevier
$!   Date:  February 24, 1989
$!
$! Modified:
$!   Author:  N. E. Sevier
$!   Date:  April 13, 1989
$!   Notes:  To take advantage of a graphics subdirectory.
$!
$! This command file will accept information from SQL*FORMS and utilize that
$! information to make a database query using SQL*PLUS.  The output from the
$! query to SQL*PLUS goes to a file, sqlfile.  The drawing file, drawing_file,
$! is selected.  The sqlfile and the drawing file are displayed according to
$! the terminal type the user has.
$!
$! SQL*FORMS calls this command procedure passing it the following parameters:
$!      P1 = Part Stem Number
$!      P2 = Part Suffix Number
$!      P3 = Connector Series
$!
$! General Flow:
$!
$! An SQL command file, getdim.sql, is created to query the database for
$! the connector dimensions.  Which connector is identified by p1 & p2.
$! Connector dimensions are stored as a set of dimension names and values.
$!
$! Once the SQL command file has been created and closed, SQL*Plus is
$! invoked with this file as input.  When SQL*Plus is finished, the SQL
$! command file is deleted.
$!
$! The terminal type the user has is determined and verified by the user.
$!
$! The drawing file to be used is defined by the connector series.  This
$! has been passed to this command procedure by SQL*FORMS.  The type of
$! drawing file is determined by the terminal type.
$!
$! If the user is using VTERM, then the tek4014 type drawing file will be
$! displayed.  The drawing files will tell VTERM to switch to emulating a
$! tektronix terminal and display the drawing.  The drawings have been
$! positioned so that the sqlfile can be displayed on the same screen to
$! the right of the connector drawing.  When the user is finished looking
$! at the drawing, s/he is asked to acknowledge this by pressing return.
$! The terminal is then reset to the VT100 mode.  The sqlfile is deleted
$! and the command procedure will exit.
$!
$! If the user has a VT Regis terminal type, then the Regis-type drawing files
$! will be used.  A program will be executed to append the dimensions in the
$! sqlfile to the drawing file.  This revised drawing file will be displayed on
```

```
$! the screen.  The revised drawing file and the sqlfile will be deleted and
$! the command procedure will exit.
$!
$! SQL*FORMS provides the facility that states "Press a function key to ..."
$! return control to SQL*FORMS.
$!
$!-------------------------------------------------------------------------
$!     Create the SQL command file
$!
$open/write sql_file getdim.sql;
$!
$!     Set up Column Headings
$!
$ write sql_file "column dimension_name_str_specifies heading 'Dimension'"
$ write sql_file "column inches_specifies format 999.999 heading 'Inches'"
$!
$!     Turn off output to the terminal
$!
$ write sql_file "set termout off;"
$!
$!     Send output to the file oracle_file.dat
$!
$ write sql_file "spool oracle_file.dat;"
$!
$!     Create the query using the part number parameters
$!
$ write sql_file "select dimension_name_str_specifies, inches_specifies"
$ write sql_file "from ops$nesevie.conn_dimensions"
$ p1 = "where stem_number_specified_by = " + p1
$ p2 = "and part_suffix_numbe_specified_by = " + p2 + ";"
$ write sql_file p1
$ write sql_file p2
$!
$!     Close the output file
$!
$ write sql_file "spool out;"
$!
$!     Exit SQL*Plus
$!
$ write sql_file "exit;"
$!
$!     Close the SQL command file
$!
$close sql_file
$!
$!     Invoke SQL*Plus with the command file
$!
$sqlplus -silent ""/ @getdim.sql;
$!
$!     Delete the SQL command file
$!
$delete getdim.sql;
$!
$!     Draw the appropriate connector drawing & display the dimensions.
```

```
$!     ------------------------------------------------------------
$!
$!     This section will make sure that the user's terminal type is set
$!     based on the terminal s/he is currently using.
$!     Once the terminal type is known, the corresponding drawing type will be
$!     created and displayed, if possible.
$!
$set term/inq
$set term/form/nowrap
$pid = ""
$loop:
$   on error then goto error_message
$   term = f$getjpi(f$pid(pid),"TERMINAL")
$   if term .eqs. "" then goto loop
$end_loop:
$num = f$getdvi(term,"DEVTYPE")
$if num .eqs. 96 then goto vt100
$if num .eqs. 98 then goto vt100
$if num .eqs. 110 then goto vt200
$askuser:
$type sys$input

                    Terminal Types Available

          VT100  -  Includes all VT100 Series Terminals or Emulators
          VT200  -  Includes all VT200 and VT300 Series Terminals or Emulators

$inquire choice "Enter your terminal type (VT100 or VT200)"
$choice = f$edit(choice,"UPCASE")
$on warning then goto error_message
$goto 'choice'
$!
$!     If the terminal type is a PC running VTERM, then print the TEK4014
$!     version of the connector drawing and the dimensions.
$!
$vt100:
$!
$!     Ask if the user is using VTERM in VT/TEK mode.  If so, then graphics
$!     are possible.  Otherwise, they are not.
$!
$on warning then goto vt100
$inquire/nopunctuation vttek -
 "Are you running VTERM in VT/TEK mode? (Enter 'Y' or 'N'): "
$vttek = f$extract(0,1,f$edit(vttek,"UPCASE"))
$if vttek .nes. "Y" then goto end
$!
$p3 = "sd:[conuser.graphics]" + p3 + ".tek4014;"
$on warning then goto nofileexists
$test = f$file_attributes(p3,"fid")
$!
$!     Display the drawing
$!
$type 'p3'
$!
```

```
$!     Display the dimensions
$!
$type oracle_file.dat
$!
$!     Skip 5 lines
$!
$write sys$output " "
$write sys$output " "
$write sys$output " "
$write sys$output " "
$write sys$output " "
$!
$!     Pause until user is finished looking at the drawing.
$!
$write sys$output "-----     PrtScn to Print      -----"
$inquire/nopunctuation next "----- Press Return to Continue -----"
$!
$!     Reset VTERM to VT100 mode.
$!
$write sys$output "
$!
$!     Clean up and exit.
$!
$goto cleanup
$!
$!     If the terminal type is a VT240 or better, then append the dimensions
$!     to the REGIS-version of the drawing and display it.  Be sure to delete
$!     the latest version.
$!
$vt200:
$!
$!     Ask the user if they are using a VT240 series terminal or a Regis type
$!     terminal.  If so, graphics can be displayed.  Otherwise, it is not
$!     possible.
$!
$on warning then goto vt200
$inquire/nopunctuation vttype -
    "Do you have a VT REGIS graphics terminal (240, 241, 330, 340)? (Y/N) :"
$vttype = f$extract(0,1,f$edit(vttype,"UPCASE"))
$if vttype .nes. "Y" then goto end
$!
$!     Create the drawing file name
$!
$p3 = "sd:[conuser.graphics]" + p3 + ".regis"
$!
$on warning then goto nofileexists
$test = f$file_attributes(p3,"fid")
$copy 'p3' drawing_file.dat;
$!
$!     Append the dimensions to the drawing file
$!
$run sd:[conuser.graphics]add_dimensions
$!
$!     Display the drawing and dimensions
```

```
$!
$type drawing_file.dat
$!
$!     Wait until the user is finished viewing the drawing before continuing.
$!     The first P is not displayed.  It positions the text at the left margin.
$!
$!inquire/nopunctuation next "P----- Press Return to Continue  -----"
$!
$!     Delete the appended drawing file
$!
$delete drawing_file.dat;
$goto cleanup
$!
$error_message:
$! Prints error message if user enters an invalid terminal type
$write sys$output " "
$write sys$output "Unknown Terminal Type"
$write sys$output " "
$goto askuser
$!
$nofileexists:
$write sys$output " "
$write sys$output " "
$write sys$output "  Sorry, no drawing currently exists for this connector."
$write sys$output "  A drawing file is being created and will be available"
$write sys$output "  in the future."
$write sys$output " "
$write sys$output " "
$!
$cleanup:
$!
$!     Delete the SQL output file
$!
$delete oracle_file.dat;*
$!
$!     Return control back to SQL*FORMS.
$!
$end:
$exit
```

Add Dimensions

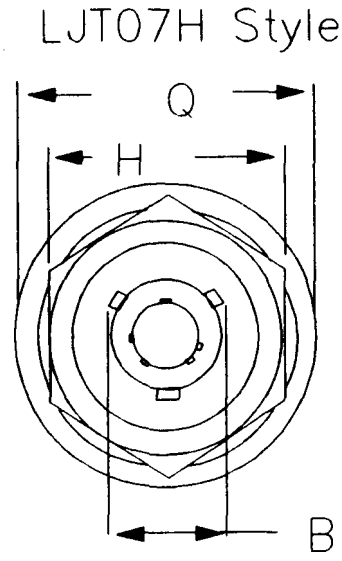
```
program convert_and_append(drawing_file, oracle_file);
(*************************************************************************)
(*                                                                       *)
(*  Original:                                                            *)
(*    Date Written:  February 15, 1989                                   *)
(*    Author:  N. E. Sevier, 2551                                        *)
(*                                                                       *)
(*  Modified:                                                            *)
(*    Date Modified:                                                     *)
(*    Author:                                                            *)
(*                                                                       *)
(*  This program will read the oracle file and append its contents to the  *)
```

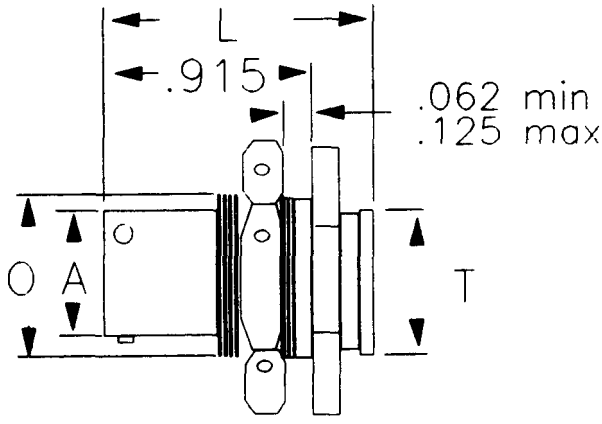
```
(*  drawing file in the following form.                                    *)
(*                                                                         *)
(*          'oracle line <CR><LF>                                          *)
(*           oracle line <CR><LF>                                          *)
(*                 ...                                                     *)
(*           oracle line <CR><LF>                                          *)
(*           '                                                             *)
(*************************************************************************)

var
  data: varying[50] of char;
  drawing_file: text;
  oracle_file: text;

begin
  open(oracle_file,history:=readonly);
  reset(oracle_file);
  extend(drawing_file);
  write(drawing_file,'''');
  while not eof(oracle_file)
    do begin
         readln(oracle_file,data);
         writeln(drawing_file,data,''(10),''(13))
       end;
  writeln(drawing_file,'''');
  writeln(drawing_file,''(27),'/');
  close(drawing_file);
  close(oracle_file)
end.
```

Sample Graphics Output

DIMEN	INCHES
A	.975
B	1.085
H	1.312
L	1.195
O	1.066
Q	1.500
T	1.018

7 RECORDS SELECTED.





----- PRTSCN TO PRINT -----
----- PRESS RETURN TO CONTINUE -----

Appendix K: Report Command Procedure

Report.Com

```
$!---------------------- Report . Com -------------------------------
$!
$! Creation:
$!   Author:  N. E. Sevier
$!   Date:  April 27, 1989
$!
$! Modified:
$!   Author:
$!   Date:
$!   Notes:
$!
$! This command file will accept a part number from SQL*FORMS and use this
$! information to create a report containing all the information known about
$! the connector with that part number.  This information will be displayed
$! on the screen for the user to capture with his/her printer or PC, since no
$! printer is available on the WhiteStar MicroVAX.  The connector drawing is
$! then found and displayed on the user terminal for printing via print screen.
$!
$! Input:
$!
$! SQL*FORMS calls this command procedure passing it the following parameters:
$!      P1 = Part Stem Number
$!      P2 = Part Suffix Number
$!      P3 = Series
$!
$! General Flow:
$!
$! An SQL command file, create_part.sql, is written to create a table
$! containing the part number.  The part number inserted into the table
$! is identified by p1 & p2.
$!
$! Once the SQL command file has been created and closed.  SQL*Plus is
$! invoked with this file as input.
$!
$! The ORACLE report generator, RPT, is then called to find all the information
$! about this part using the file report.dat.  RPT generates an output file,
$! report.tmp.  This file is then formatted by the ORACLE report text formatter,
$! RPF.  RPF creates the formatted output report in the file report.out.
$!
$! The user is notified to turn on his/her printer and the file report.out is
$! displayed on the user's terminal.  The user is notified to turn his/her
$! printer off.  Unfortunately, this notification will become part of the
$! user's report, but until WhiteStar attaches a printer to their machine this
$! will have to do.
$!
$! To clean up, the following files are deleted: create_part.sql, report.tmp,
$! and report.out.  The table part number table is deleted, using drop_part.sql.
$!
$! If the user is told that the drawing is going to be fetched and to print it,
$! s/he should use PrtScn.  The command procedure graphics.com finds
```

```
$! and displays the drawing.
$!
$! The user is returned to SQL*FORMS when this command procedure
$! completes
$!
$!-------------------------------------------------------------------------
$!     Create the SQL command file
$!
$ write sys$output ""
$ write sys$output "** Placing the Part Number in a mail box. **"
$ write sys$output ""
$!
$ open/write sql_file create_part.sql;
$!
$!     Create Part Table
$!
$ write sql_file "create table part (stem number(6) not null,"
$ write sql_file "suffix number(3) not null);"
$!
$!     Insert Part Number into the Part Table
$!
$ write sql_file "insert into part values ("
$ temp = p1 + ", " + p2 + ");"
$ write sql_file temp
$!
$!     Exit SQL*Plus
$!
$ write sql_file "exit;"
$!
$!     Close the SQL command file
$!
$ close sql_file
$!
$!     Invoke SQL*Plus with the command file
$!
$ sqlplus -silent ""/ @create_part.sql;
$!
$!     Run RPT with sd:[conuser.report]report.dat
$!
$ write sys$output ""
$ write sys$output "** Creating the Report **"
$ write sys$output ""
$!
$ define/user sys$input sys$command
$ rpt sd:[conuser.report]report.dat report.tmp ""/
$!
$!     Run RPF with report.tmp to create report.out
$!
$ write sys$output ""
$ write sys$output "** Formatting the Report **"
$ write sys$output ""
$!
$ rpf report.tmp report.out
$!
```

```
$!     Notify user to turn on his/her printer or method for capturing
$!     the output.
$!
$ type sys$input

 ** The report has now been created.                          **
 **    -  Turn ON your Printer!                               **
 **    -  Tell your printer to print whatever appears on the  **
 **       screen from now on.  Usually, Ctrl-PrtScn           **

 **            Press return when you are ready.              **
$ inquire/nopunctuation next ""
$!
$!     Type the file
$!
$ type report.out
$!
$!     Notify user to turn off his/her printer or method for capturing
$!     the output.
$!
$ write sys$output ""
$ inquire/nopunctuation next -
"** Tell the printer to stop printing (usually Ctrl-PrtScn) THEN press return"
$!
$!     Clean House
$!
$ write sys$output ""
$ write sys$output "** Cleaning House **"
$ write sys$output ""
$!
$ delete create_part.sql;*
$ delete report.tmp;*
$ delete report.out;*
$ sqlplus -silent ""/ @sd:[conuser.report]drop_part.sql
$!
$!      Get the drawing for this connector.
$!
$ type sys$input

**  The drawing and dimensions for this connector will be displayed.     **
**  If you have a graphics printer then you may send this to your        **
**  printer by pressing PrtScn after the connector is drawn on your      **
**  screen.                                                              **

**  The EMR Hardware and Potting Mold drawings will not be displayed.    **
**  To obtain a hardcopy of these, go to these screens and press PrtScn  **


** Getting the Drawing **

$!
$@sd:[conuser]graphics.com 'p1' 'p2' 'p3'
$!
$! Return to SQL*FORMS
```

```
$!
$exit
```

Report Code

The report code can be found on the WhiteStar MicroVAX in the connector user report directory, or contact Nicole E. Sevier, 2551, for a copy.

Sample Report

Connector Selection Program Output

For: 358215 -

General Information
- SA Number: SA1457 - 5
- Family: LJT
- Series: LJT07H
- Type: RECEPTACLE
- Insert Arrangement: 15 - 18
- Number of Contacts: 18
- Number of Contacts carrying maximum current: 18
- Mounting Type: JAM NUT
- Hole Layout Code: 12.016C
- Responsible Engineer: JOHNNY R. BACA
  Division: 2551
- Manufacturers: BCO
- Service Rating: JTI

Letter	Altitude	V dc
I	0	2120.00

Contact Information
- Size: 20
- Type: SOCKET
- Termination: SOLDER CUP
- Retention: lbs
- Current Rating: 5.00 amps
- Contact Resistance:
  - Test Current: 5.00 amps
  - Test Voltage: 45.00 mV drop
- Base Material: STEEL
- Plating Materials: GOLD
  NICKEL

Product Specification Information
- Dielectric Withstanding Voltage (DWV): 2,120 V dc
- Insulation Resistance (IR): 1000 M ohms min
- Temperature Range: to degrees c
- Spring Engagement: .11600000 inch max
- Air/Helium Leakage: 1.00E-08 cc/sec
- Insert Retention: psi g

Durability: 250 times coupled and uncoupled
Physical/Mechanical Shock: 2400 g min
.800 ms max
Coupling Pin Strength: lbs min
inch dia
Moisture Resistance: 100 M ohms min after testing

Materials
Shell Base: STAINLESS STEEL
Spring Finger Base:

Mating Connectors

Drawing Number	SA Number
358032-	SA1445-13

Accessories

EMR Hardware:

Part Type: ADAPTER RING
Part Number: 298466 -
Angle: 90
Dimensions:

Letter	Inches
A	.8640
B	.8840
C	.9140
D	.9890

Part Type: SHELL
Part Number: 877023 - 8
Angle: 0
Dimensions:

Letter	Inches
A	.5080
B	.8540
C	.9380

Part Type: FERRULE
Part Number: 877024 - 18
Angle: 0
Dimensions:

Letter	Inches
A	.5060
B	.2170

Part Type: FERRULE
Part Number: 877024 - 18
Angle: 90
Dimensions:

Letter	Inches
A	.5060
B	.2170

Part Type: FERRULE
Part Number: 877024 - 22
Angle: 0
Dimensions:

Letter	Inches
A	.5060
B	.3120

Part Type: FERRULE
Part Number: 877024 - 22
Angle: 90
Dimensions:

Letter	Inches
A	.5060
B	.3120

Part Type: FERRULE
Part Number: 877024 - 28
Angle: 0
Dimensions:

Letter	Inches
A	.5060
B	.3870

Part Type: FERRULE
Part Number: 877024 - 28
Angle: 90
Dimensions:

Letter	Inches
A	.5060
B	.3870

Part Type: ELBOW COVER
Part Number: 877025 - 8
Angle: 90
Dimensions:

Letter	Inches
A	.2540
B	.8750
C	.2640
D	.9580

Part Type: ELBOW SLEEVE
Part Number: 877026 - 8
Angle: 90
Dimensions:

Letter	Inches
A	.2540
C	.8540
D	.4370
E	.8750
F	.5290
G	.0940
H	.4990

Potting Mold Adapters:
Part Number: 877236 - 5
BECD Number: 10-150912-14
Thread Code: 13/16-20 UNEF-2B
Cable End Inside Diameter: .8520
Cable End Outside Diameter: .8920
Connector End Outside Diameter: .9540

Potting Molds:

Part Number: 877113 - 4
Angle: 90 degrees
Cable End Diameter: .3030 inches
Connector End Diameter: .8920 inches
Second Cable End Diameter (possible ellipse): .8500 inches
Length from cable end to the center of the bend: .6620 inches
Length from connector end to the center of the bend: .7810 inches
Outside Radius: .1510 degrees
Inside Radius: degrees

Part Number: 877237 - 5
Angle: 0 degrees
Cable End Diameter: .6830 inches
Connector End Diameter: .8920 inches

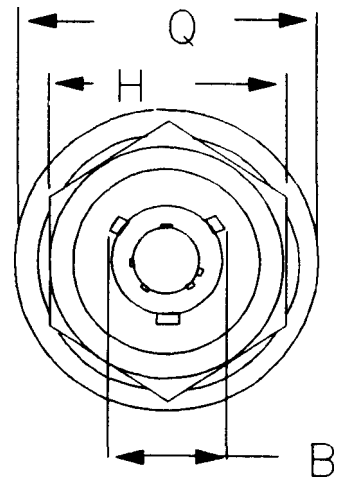
Part Number: 877238 - 4
Angle: 0 degrees
Cable End Diameter: .6500 inches
Connector End Diameter: .8920 inches

Part Number: 877239 - 5
Angle: 75 degrees
Cable End Diameter: .6250 inches
Connector End Diameter: .8920 inches
Second Cable End Diameter (possible ellipse): .7810 inches
Length from cable end to the center of the bend: .6560 inches
Length from connector end to the center of the bend: .7340 inches
Outside Radius: .4310 degrees
Inside Radius: .2500 degrees

Part Number: 877240 - 6
Angle: 90 degrees
Cable End Diameter: .8920 inches
Connector End Diameter: .8920 inches
Second Cable End Diameter (possible ellipse): .8920 inches
Length from cable end to the center of the bend: inches
Length from connector end to the center of the bend: inches
Outside Radius: degrees
Inside Radius: .0150 degrees

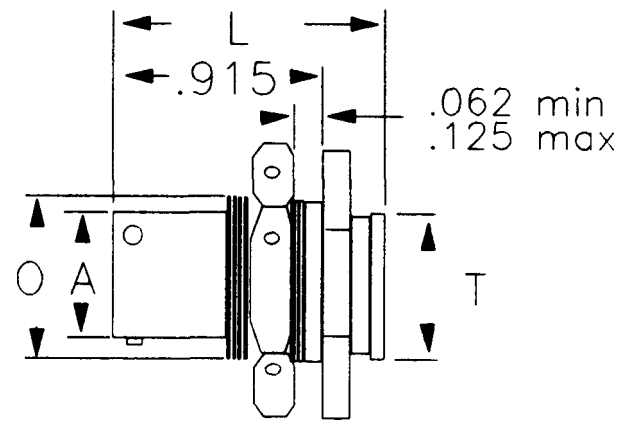
Part Number: 877240 - 7
Angle: 90 degrees
Cable End Diameter: .4580 inches
Connector End Diameter: .8920 inches
Second Cable End Diameter (possible ellipse): .9660 inches
Length from cable end to the center of the bend: inches
Length from connector end to the center of the bend: inches
Outside Radius: degrees
Inside Radius: .0150 degrees

LJT07H Style



DIMEN	INCHES
A	.975
B	1.085
H	1.312
L	1.195
O	1.066
Q	1.500
T	1.018

7 RECORDS SELECTED.



----- PRTSCN TO PRINT -----
----- PRESS RETURN TO CONTINUE -----

References

Models: Connector Selection, Drawing S92977, Issue B.
Information Structure Diagram, Fastener Selection Requirements, Drawing FC-R20586, Issue A.
PREMIS, Drawing R12818, Issue B.

Sevier, Nicole E. Developing a Connector Selection DBMS Using NIAM. SAND88-0272, presented at the ASME Computers & Engineering Conference, 7/88.

Sevier, Nicole E. Connector Selection Program User's Guide, Version 1.0. SAND89-1286.

Sevier, Nicole E. and Drozdick, William. Fastener Selection Requirements. Drawing R20586, 10/89.

Sharp, John K., Orman, John L. and Stevens, Norman H. Information Engineering: How Sandia is Developing the CIM Database (the NIAM Approach). SAND89-0532C, presented at the Fifth Annual CAD/CIM Database Conference, 3/89.

Stevens, Norman H. "NIAM in Relational Modeling." Database Programming and Design. June, 1989. pp. 11-15.

DISTRIBUTION:

Allied Signal Aerospace Corp. (3)
Kansas City Division
Attn: R. M. Schaefer
M. L. Smith
D. L. Hobbs
P.O. Box 1159
Kansas City, MO 64141

2534 D. H. Jensen
2534 S. Carroll
2500 R. L. Schwoebel
2550 C. F. Gibbon
2551 D. E. Carnicom
2551 N. E. Sevier (5)
2810 D. W. Doak
2812 A. C. Bernadino
2812 T. F. Ezell
2812 S. L. K. Rountree
2820 G. Carli
2825 J. K. Sharp
2825 J. L. Orman
2825 N. H. Stevens
2826 A. J. Ahr
2850 D. L. McCoy
2854 R. E. Thompson
2854 G. L. Neugebauer
2858 W. Drozdick
3141-1 C. L Ward (8)
For DOE/OSTI
3141 S. A. Landenberger (5)
3151 W. I. Klein (3)
8524 J. A. Wackerly