NASA-CR-194241

p. 192

K-Base: A Hybrid Analogical/Semantic Modeler for Computer-Aided Design

FINAL REPORT
NAS 9-17808

GMS Technology
24 November, 1989

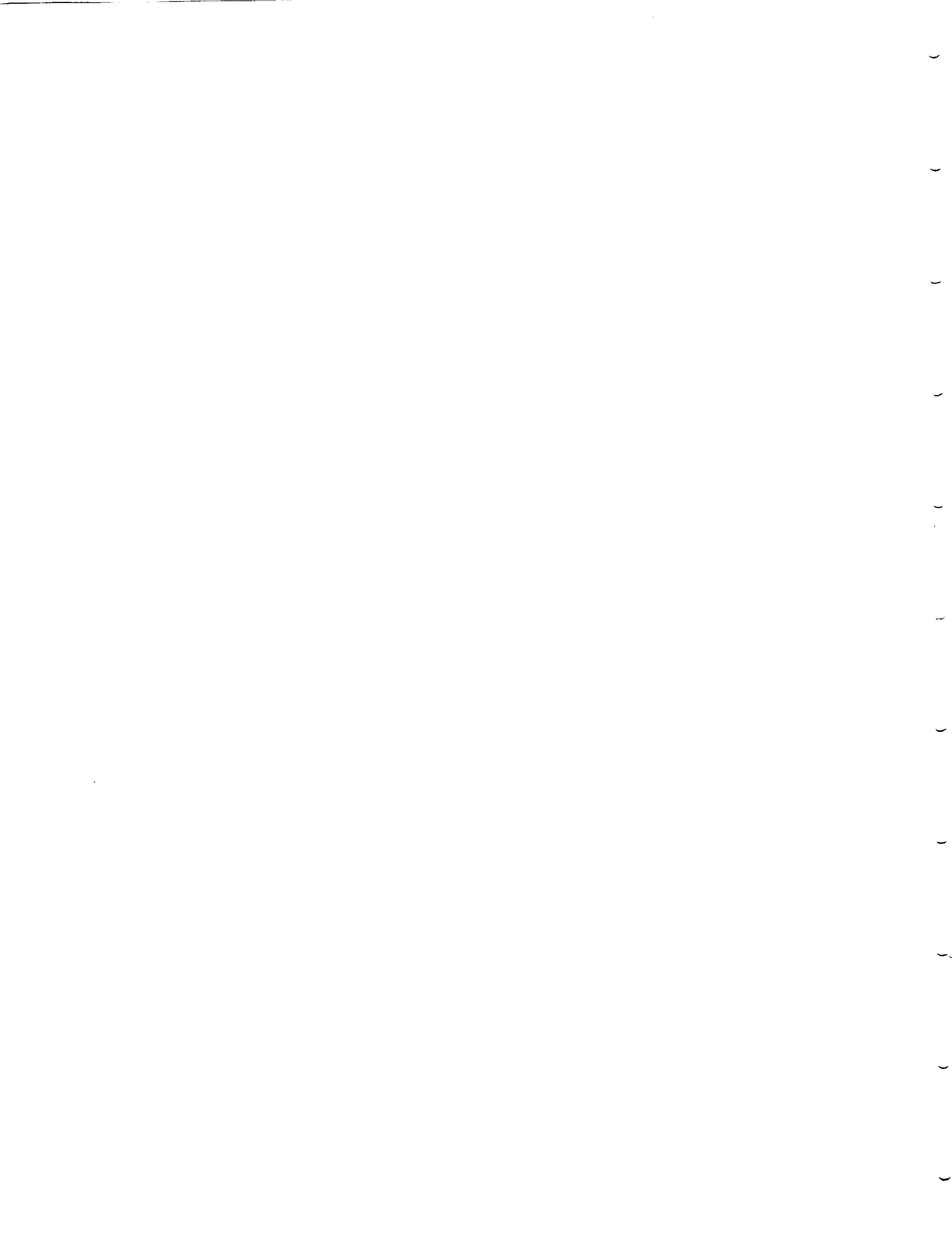
SBIR Phase II
SBIR-12-03-5297
Release date 11/24/91

(NASA-CR-194241) K-BASE: A HYBRID ANALOGICAL/SEMANTIC MODELER FOR COMPUTER-AIDED DESIGN Final Report (GMS Technology) 192 p

N94-70380

Unclas

Z9/61 0183159



K-Base: A Hybrid Analogical/Semantic Modeler for Computer-Aided Design

FINAL REPORT
NAS 9-17808

GMS Technology
24 November, 1989

Table of Contents

1. Project Summary 1-1
2. Background 2-1
    2.1 PLAID Environment 2-1
    2.2 DECnet/VAX Environment 2-2
3. Programming Paradigm Research 3-1
    3.1 Procedure-oriented Programming 3-1
    3.2 Object-oriented Programming 3-2
    3.3 Access-oriented Programming 3-4
    3.4 Rule-based Programming 3-4
    3.5 Multiple Paradigm Programming Systems 3-5
    3.6 Conclusions 3-6
4. Development Environment Search 4-1
    4.1 Criteria 4-1
    4.2 Product Evaluations 4-3
        4.2.1 C++ 4-3
        4.2.2 Common Lisp 4-4
        4.2.3 CLIPS - C Language Production System version 4.01 4-5
        4.2.4 DC-Representation Language (DC-RL) 4-8
        4.2.5 Flavors 4-18
        4.2.6 Knowledge Engineering Environment (KEE) 4-18
        4.2.7 LOOPS/XAIE 4-19
        4.2.8 New Flavors 4-20
        4.2.9 Nexpert Object 4-20
        4.2.10 PC-Scheme / Scoops 4-21
        4.2.11 Peabody 4-21
        4.2.12 SmallTalk-80 4-23
    4.3 Selected System Configuration 4-24
        4.3.1 MicroVAX II Boot Node 4-24
        4.3.2 VAXstation 2000 Workstations 4-24
        4.3.3 Local-Area VAXcluster (LAVC) 4-24
        4.3.4 Common LISP (Lucid) 4-25
        4.3.5 C Compiler 4-25
        4.3.6 FORTRAN Compiler 4-25
5. Work Performed 5-1
    5.1 K-Base Symbol Management System (KB/SMS) Specification 5-1
        5.1.1 Purpose of KB/SMS 5-1
        5.1.2 Approach 5-1
        5.1.3 Description of a Description File 5-2
        5.1.4 Data Entry 5-3
        5.1.5 Query Commands 5-3
        5.1.6 Report Generation 5-5
        5.1.7 Global Report Algorithm 5-6
        5.1.8 Example of a Global Search 5-8

PRECEDING PAGE BLANK NOT FILMED

5.1.9 Notes on Report Format: 5-11
5.2 Geometric Knowledge Enhancements 5-12
5.2.1 SITES 5-12
5.2.2 CONNECT command. 5-12
5.2.3 ATTACH / DETACH commands. 5-12
5.2.4 IRIS interface. 5-12
5.2.5 DESCRIPTION command. 5-12
5.2.6 DISPLAY file review. 5-13
5.2.7 RMS additions and improvements. 5-13
5.2.8 VIEW command improvements. 5-13
5.2.9 ROTATION of parts. 5-14
5.2.10 JOINT command. 5-14
5.2.11 SCALE command. 5-14
6. Conclusions 6-1
Appendix 1 -- Updated Multi-User Documentation A1-1
Appendix 2 -- Updated DMC Documentation A2-1
1. Updated DMC Routine Documentation. A2-1
2. Transformation Operations. A2-21
3. New and Updated User Commands for DMC. A2-25
Appendix 3 -- Rasterizer Software. A3-1
Appendix 4 -- DCRL Browser A4-1
Appendix 5 -- Scoops Evaluation. A5-1
Appendix 6: Multi-User Files Modified for KB/SMS A6-1
Bibliography B-1

1. Project Summary

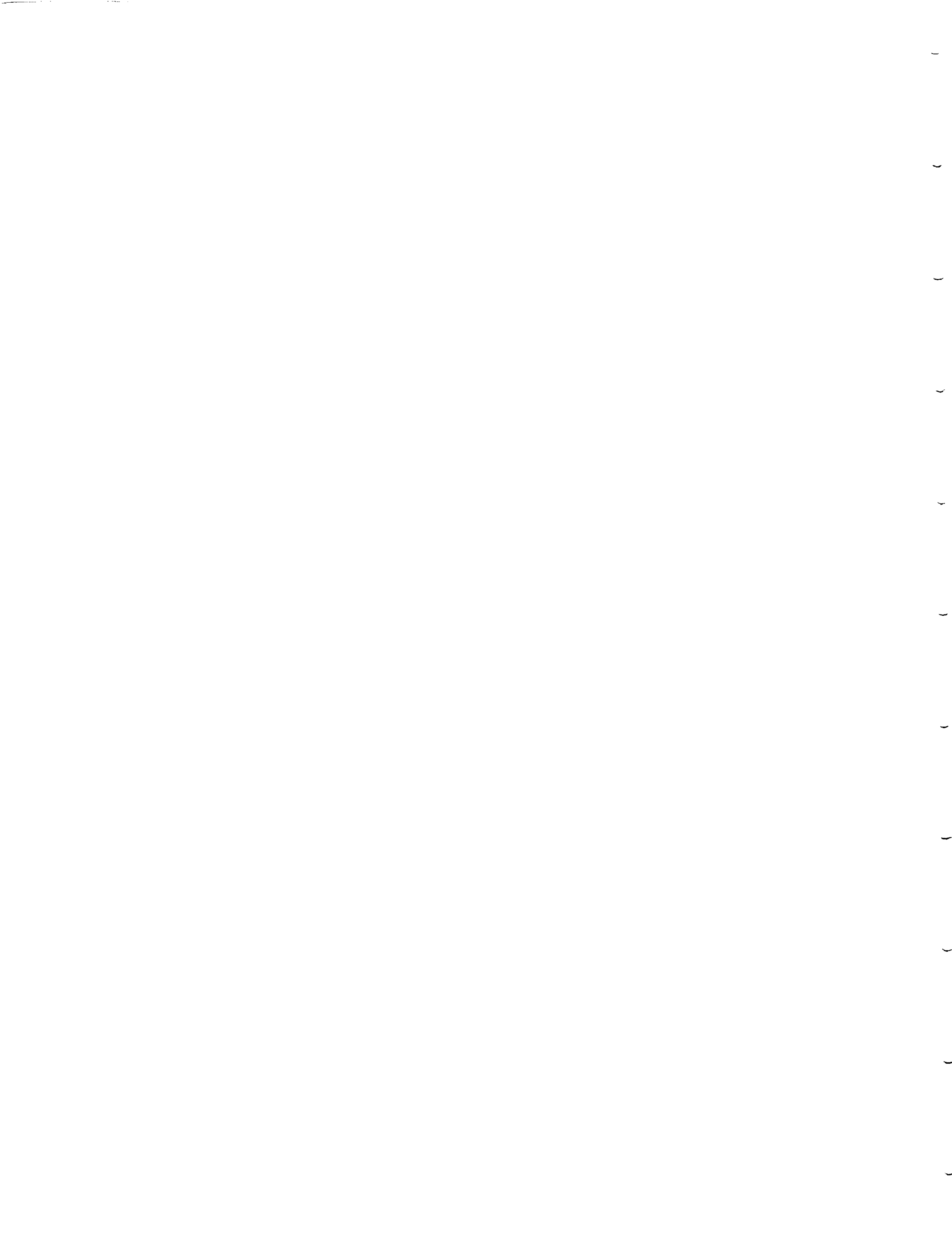
The K-Base project was an investigation of methods for improving the analytical and descriptive capabilities of Computer Aided Design (CAD) systems. The approach involved merging knowledge representation concepts from contemporary Artificial Intelligence (AI) research with the geometric modeling capabilities of the PLAID computer-aided design (CAD) system to provide a system capable of representing deep functional knowledge of a system of objects as well as its geometric appearance. The set of enhancements generated by the K-Base effort have been applied to the PLAID system which is currently in use by the Manned Systems Division at JSC.

In support of these objectives, GMS Technology conducted surveys of new programming paradigms which offer improved productivity and knowledge-representation schemes currently in use in AI research. Evaluations of available knowledge representation systems were conducted. These evaluations included the DC-Representation Language (a KL-ONE derivative) and Peabody.

A system of hardware and software was selected for use in the development of the K-Base software system. The selected development system consists of a loosely-coupled network of Digital Equipment Corporation (DEC) VAXstation 2000's and a DEC MicroVAX II with C, FORTRAN, and CommonLisp compilers.

Enhancements were made to the PLAID system which substantially extend its ability to model objects. The improvements fall into two categories: the association of symbolic information with each PLAID part and the addition of new geometric constructs. The utility of the new symbolic information is demonstrated by the addition of Multi-User commands which allow the location of PLAID models based upon symbolic information queries.

The PLAID software system is currently available through Cosmic. We envision the K-Base additions to PLAID being made available through the Cosmic distribution channel.



2. Background

The PLAID system was written for NASA by the principals of GMS Technology beginning in 1976, and has been continually enhanced since that time. As is typical of conventional CAD systems, the current implementation of PLAID provides a powerful working environment for implementation of simulation and analysis tasks.

The PLAID system relies almost exclusively upon geometric representations to provide descriptions of modeled objects. This limited knowledge representation produces a system which is able to provide users with a pictorial basis for analysis, but is largely manual in operation and is unable to store or analyze non-geometric information. The system is therefore unable to aid the user in analysis of the advisability or possible consequences of his actions. For example, a traditional CAD system contains no information which can differentiate a water pipe from an electrical conduit, and will gladly allow the user to connect them together. The result provides no warning of the disastrous results which could result from performing such an action in the real world.

Traditional AI research has, on the other hand, emphasized semantic/symbolic information to the almost complete exclusion of geometric information. Any geometric information provided to the user has been two-dimensional and intended for illustrative purposes only. No attempt has been made to include analyses of the geometry for purposes such as collision detection during object motion, inference of mass or moment of inertia from the volume of an irregularly shaped object, *etc.* AI systems must therefore have such information manually calculated and input by the user before it can be utilized by the reasoning process.

2.1 PLAID Environment

2.1.1. History.

PLAID is the name of a graphics analysis system used at the NASA JSC Manned Systems Division. Development of the system began in 1977 at the University of Texas at San Antonio as a NASA funded project. The original system was developed on a Data General Eclipse mini-computer using Tektronix display devices for graphic input and output. In 1978, the software was ported to a SEL mini-computer for on site utilization at NASA JSC. In 1980, the system was moved to a VAX 11/780.

2.1.2. Applications.

Since the time of its initial installation, PLAID has been used for many graphics analysis applications, such as vision analysis, reach analysis, and clearance analysis. In the case of the Challenger accident, PLAID images were used for image processing enhancements to identify fuel leaks. These images were also included in the Rogers Commission report.

2.1.3. Graphic Modules.

Currently, the PLAID system resides mainly on a VAX 11/785 running under VMS. It consists of three main modules. The BUILD module is used to construct primitive objects which define the geometry of a part. The DMC module (previously the COG module) is used to construct high level assemblies and sub-assemblies, called COG files, using the primitive objects constructed by BUILD. The COG files describe the transformations used to properly place primitives and other COG files (sub-assemblies) into an assembly. The DISPLAY module uses completed assemblies output by DMC. These completed assemblies are stored in TARGET files and used for generating perspective views with hidden lines removed.

2.1.4. File Management Module.

In addition to these three functional modules, there is file management system underlying each, which is used to partition the PLAID database into projects and user work areas. The system, known as MULTIUSER, permits the definition of new projects and user work areas with varying degrees of visibility for each user. However, due to the complex nature of the possible organization schemes and the high degree of integration of the other three modules, MULTIUSER must depend on specific features of the VMS operating system which is the primary environment for PLAID.

2.2 DECnet/VAX Environment

The PLAID system exists within a multi-processor VAXcluster computer system which allows numerous PLAID users to share a large database of PLAID models.

3. Programming Paradigm Research

GMS Technology conducted a survey of state-of-the art programming paradigms. The following sections summarize what was learned in that survey.

3.1 Procedure-oriented Programming

3.1.1 Description

Procedure-oriented programming is the classical paradigm of computer programming using one of the conventional programming languages (*e.g.* assembly language, FORTRAN, C). Procedure-oriented programming is characterized by its focus upon the sequential, algorithmic component rather than the data-structure component of the program.

There are many procedure-oriented languages from which to choose. The following discussion attempts to characterize the attributes of the mainstream languages as a group.

3.1.2 Strengths

Procedure-oriented programming is well understood and widely used. All of the major programming languages are implementations of the procedure-oriented paradigm. The huge majority of production quality software in use today was written in one of the following languages:

- Ada
- C
- COBOL
- BASIC
- FORTRAN
- Lisp
- Modula-2
- Pascal
- PL/I

Procedure-oriented programming systems are typically very efficient at execution time. Efficiency results from the fact that it is a key design goal of almost all procedure-oriented languages. The notable exceptions to this rule are the Lisp and BASIC languages.

3.1.3 Limitations

The price paid for execution-time efficiency is execution-time inflexibility which manifests itself in the areas of both data-structures and procedures.

In the data-structure area, procedure-oriented languages require that all data-types be known at compile-time. A new data-type can

be introduced only by editing the program source code, re-compiling it, and linking it.

Procedure-oriented languages lack the inheritance of attributes found in object-oriented languages. The programmer must implement a new data-type and all of the operations on that data-type from scratch. The advent of user-defined types which allows the programmer to define new types in terms of existing types helps alleviate this weakness but is far from being as powerful a construct as inheritance. (See Section 3.2)

Most procedure-oriented languages lack polymorphism. The designers of the Ada language did address this issue with the concept of generic procedures and operator over-loading. Generic procedures allow the programmer to write a single procedure which can be called with several different data-types. Even so, the type of a given object must be known at compile-time which limits the run-time flexibility of the program.

3.2 Object-oriented Programming

3.2.1 Description

The object-oriented programming paradigm focuses upon data rather than procedures. This approach is a complete inversion of the algorithmic focus of procedure-oriented languages.

Object-oriented programming systems have the following attributes [Pascoe 1986]:

- information hiding
- data abstraction
- dynamic binding (polymorphism)
- inheritance

Object-oriented languages bundle both data-structures and procedures into modules called *objects*. The internal structure of one type of object is completely hidden from objects of another type. The only access provided to an object is determined by the set of *messages* which the object will accept. The message specification is part of the object definition. This structure provides for both information hiding and data abstraction.

A new object type may be defined as a refinement or specialization of an existing object type. This mechanism, called *inheritance* lessens the programming burden of creating new type because the programmer need only modify the data-structures and procedures of the new (child) object which differ from those of the existing (parent) object.

Inheritance is also the basis of *polymorphism* in object-oriented systems. Because of restrictions placed upon the way in which descendants may differ from their ancestor objects, the programmer

may develop generic procedures which can operate on any object type in any given sub-tree of the object type hierarchy.

3.2.2 Strengths

The strengths of object-oriented programming languages are simply the attributes discussed above. The touted implication of these features is that programming projects may be completed much more quickly and that the resulting program will be more modular and maintainable.

Certain kinds of programming problems seem to fit very naturally into the object-oriented paradigm. Two of these are the implementation of windowed, graphical user interfaces and distributed applications.

Windowed interfaces may be structured such that each graphical object that appears on the display is represented internally by an object. The process of manipulating the graphical object is accomplished by sending the internal object a message specifying what it should do. For example, the user may wish to select an icon, then move it to another location on the screen. This is accomplished by the window manager sending messages to the object which say "select yourself", then "move yourself".

Another major advantage of object-oriented systems is that most of them provide for incremental modification. When a change is to be made to a component of the software system (*i.e.* an object), only the modified component must be compiled and there is no link step at all.

3.2.3 Weaknesses

The dynamic nature of object-oriented systems results in programs which are slower than comparable procedure-oriented programs on comparable processors. The primary reason for this is the overhead of message dispatching. Since there is no link step to resolve the absolute addresses of procedures. Their locations must be computed each time a message is received by an object.

Object-oriented programming languages are difficult or impossible to interface to existing programs written in procedural languages. The only effective solution to this problem is to completely rewrite the program in the context of the object-oriented environment. For the purposes of this project, it is simply not feasible to rewrite PLAID.

The inheritance of attributes through the class hierarchy creates the problem of extreme dependency between objects in an ancestor/descendant relationship. The problem arises when significant changes must be made to the ancestor. The programmer must be aware of the implications of the modification for every descendant of the modified object definition. This obstacle may be

overcome by creating a new object hierarchy, but can be a very cumbersome and laborious task.

Programmers who are skilled in the procedure-oriented paradigm often find the shift to the object-oriented paradigm very difficult. This problem has several roots. First, the structure of an object-oriented program is dramatically different from a procedure-oriented one. The idea of telling a data item to do something to itself seems very strange.

3.3 Access-oriented Programming

3.3.1 Description

The access-oriented programming paradigm is actually an adjunct to procedure-oriented programming. The key difference between an access-oriented language and its procedural host language is that data items may be made to execute procedures when they are accessed. For example, a procedure may be attached to a variable in such a way that the procedure is invoked each time the value of the variable is modified. In general, an access-oriented declaration consists of

- a variable definition
- an access procedure
- an access mode

3.3.2 Strengths

Access-oriented programming provides a mechanism for assuring that certain procedures are executed whenever designated data-items are accessed. This facility can be very useful in instances such as placing probes on data-items so that a display can be automatically updated when the item is modified.

3.4 Rule-based Programming

3.4.1 Description

The rule-based programming paradigm codifies expert knowledge into a set of if-then structures called *rules*. Each rule consists of the left-side or antecedent and a right-side or consequent. The left side is a boolean expression which evaluates to either TRUE or FALSE. If the left-side evaluates to TRUE, then the right-side is said to be triggered.

The inference engine of the rule-based system determines which rules are currently triggered and uses a rule selection strategy to determine which rule or rules to execute or *fire*. When a rule fires, the inference engine performs the actions specified by the right-side of the rule.

Rule-based systems are said to embody a shallow knowledge-representation. The rule-based system does not know anything about the domain to which it is applied than is built into the rules. If the rules do not cover a given situation, the inference engine must simply halt. There is no deep knowledge about the application domain from which to make inferences or to extend the rule set to include the exception.

3.4.2 Strengths

The rule-based paradigm offers a relatively simple method for codifying expert knowledge into an automated, computer-based solution.

3.4.3 Weaknesses

The rule-based programming paradigm is a radical departure from procedure-oriented programming. In a rule-based programming system, there are no procedures and no flow of control as in the other paradigms. While this structure lends itself to the construction of expert-systems for problem diagnosis, it is totally inappropriate for many problems which are commonly solved with procedure-oriented systems.

3.5 Multiple Paradigm Programming Systems

3.5.1 Description

Multiple paradigm programming systems incorporate two or more programming paradigms into a single environment. This is a pragmatic way of getting the best of several worlds. The user of such a system may utilize the capabilities of several paradigms in the solution of one problem, applying the best paradigm for approaching each problem fragment.

Many multi-paradigm programming systems currently exist. These range from procedure-oriented programming languages with object-oriented extensions (*e.g.* Common Lisp with Flavors, Common Loops, C++, Turbo Pascal 5.5, *etc.*) to complete programming environments which employ all of the paradigms within a graphical user interface environment (*e.g.* the Knowledge Engineering Environment from IntelliCorp) [Hailpern 1986] [Stefik 1986].

3.5.2 Strengths

The principal strength of multi-paradigm systems is that the most appropriate paradigm may be brought to bear on any given aspect of a problem.

The procedural language with extensions approach has the strength that it provides a growth path for systems written in the host language. These systems are almost always strict super-sets of the

host languages. As a result, programs written in the procedure-oriented subset may be easily ported to the multi-paradigm extension.

3.5.3 Weaknesses

Most multi-paradigm systems are a conglomeration of a procedure-oriented language with object-oriented and rule-based constructs tacked on. These systems are often syntactically obscure and confusing. Each such system represents a pragmatic solution to adding capabilities to the programming environment without throwing away a huge body of existing software.

3.6 Conclusions

There is no best programming paradigm. Each paradigm matches well with some types of problems and poorly with others.

Multi-paradigm programming systems are the wave of the future, but they have not yet arrived. The comprehensive multi-paradigm systems are either closed or based upon the Lisp language. Both solutions require that programs which are written in languages such as FORTRAN be scrapped or rewritten. For better or worse, there is no object-oriented (heaven help us) FORTRAN.

4. Development Environment Search

The implementation of K-Base would be facilitated by a development environment which supports several key features. The ideal development environment would provide all of the programming paradigms listed above, DECnet/VAX compatibility, and an integrated, windowed user-interface.

GMS Technology reviewed numerous products in search of an ideal programming environment. The criteria used to judge the best environment are enumerated in the following sections.

The development of an analogical/semantic modeler for CAD imposes certain unique requirements on the host software development environment. Simultaneous requirements exist for intensive numerical computation, data storage and retrieval, extensive graphics capabilities, transparent networking and non-numerical (or symbolic) processing. Also, it is most desirable that the selected development environment be portable across hardware platforms in order to take advantage of future advances in computer hardware. In addition, cost of acquisition was taken into account to insure compliance with budget limitations. These requirements are not adequately addressed by any single software development environment.

GMS Technology has evaluated several popular software development environments and where necessary their required hardware platforms for prospective use in development of K-Base. The systems reviewed include two basic types: software systems which can reside in multiple hardware environments, and integrated hardware/software environments.

4.1 Criteria

4.1.1 Multiple-paradigm Programming Environment

The availability of multiple programming paradigms would provide the greatest capability and flexibility in the development system. The appropriate programming paradigm can significantly reduce the effort required to produce specific features in the target system.

4.1.2 Strong Link to DECnet/VAX

A strong link to DECnet/VAX is required to integrate the developed software system into the existing PLAID environment. This linkage will allow the K-Base system to take advantage of the very substantial hardware and software resources which already exist in the Manned Systems Division computer system.

The DECnet/VAX linkage must include the following features:

4.1.2.1 File Transfer Capability

The development environment must provide for transferring (copying) files into and out of the VAX/VMS system environment. This feature will allow files containing both source code (programs) and data to be transported between the two environments.

4.1.2.2 Remote File Open

The development environment must provide the ability to directly open files across the network so that large PLAID database files may be accessed without duplicating their contents. This is relevant to reducing hardware resources in the K-Base development environment and maintaining the integrity of the PLAID/K-Base databases.

4.1.2.3 Inter-process Communication

The development environment must provide the ability of processes in the K-Base system to communicate directly with processes in the PLAID environment via DECnet task-to-task communication facilities. This is crucial to providing a seamless integration between new and existing software systems.

4.1.3 Inter-language Call Facility

The development environment must provide the capability to call procedures written in other languages (e.g. FORTRAN and C) as well as the capability to be called by procedures written in other languages. This feature is vital to the K-Base effort because of the large quantity of software available for processing the geometric data that exists in the PLAID environment. Thus, PLAID modules must be capable of accessing software developed under K-Base and K-Base modules must be capable of accessing PLAID modules.

4.1.4 Graphical User Interface (GUI)

The graphical user interface of the development environment must provide a uniform presentation of the software development tools and the software developed under the environment. The goal is to secure a SmallTalk-80 style of user interface environment which provides for the following features:

- Multiple Windows
- Bit-mapped Graphics
- Menu System
- Mouse Locator Device
- Applications Interface Library

4.2 Product Evaluations

The following sections summarize the information gathered during the product survey conducted by GMS Technology.

4.2.1 C++

C++ is an object-oriented extension to the ANSI Standard C programming language.

C++ (pronounced "see plus plus") is a portable, hybrid, object-oriented/procedure-oriented programming tool. The C++ language is intended to add object-oriented features to the existing C programming language as a strict superset. The goal is to make it possible to take advantage of object-oriented programming techniques while maintaining all the proven assets of the C language, including the large body of software written in C.

Several important compromises had to be made to achieve the stated intent of the language. These include the static definition of object types (classes), obfuscation of the message-passing paradigm, and the weaknesses in automatic storage management.

The benefits gained from the compromises are good run-time efficiency, portability, and upward compatibility with existing C programs. While these are major achievements, the fact that the PLAID software system is written in FORTRAN rather than C sharply reduces the benefits that might be gained by using C++ as the basis for new development.

Implementations of C++ are available for the VAX/VMS environment which make all of the features of the VMS environment available from C++. Programs written in C++ may make use of all of the DECnet/VAX network facilities by simply calling the appropriate VMS system services.

The definition of a standard set of classes and methods which would serve as an application programmer interface (API) to the VAXstation windowing system, or DECwindows, or X-Windows would have given C++ a distinct advantage in the realm of user-interface programming. Unfortunately, C++ has no standard support for any graphical user interface. The application programmer will therefore be forced to invent and implement the classes and methods required to utilize one of the most complex parts of the user-interface in addition to solving his/her central programming task.

Another major problem with C++ is that the definition of the language is not yet either complete or stable. Each implementor of C++ interprets the syntax and semantics of the language slightly differently. The result is that C++ code written under one implementation will not port to another implementation without significant change. Such compatibility problems are typical of

emerging technologies. C++ is not yet a mature software development tool.

4.2.2 Common Lisp

Common LISP was evaluated for use as a platform for K-Base in view of its wide acceptance by the artificial intelligence community, and because it is a portable, platform independent language which conforms to a vendor-independent standard.

The Common LISP environment chosen for serious evaluation was Lucid LISP, from Lucid, Inc. Lucid LISP was chosen because its environment is relatively consistent across multiple hardware/operating system platforms, it includes an object-oriented programming environment (Flavors) and includes a fairly comprehensive set of tools for dealing with bit mapped graphical displays, mice, pull-down menu systems, *etc.*

In the VAX/VMS environment, Lucid LISP operates either in a graphic workstation (VAXStation) environment or on character based displays. The VMS implementation also includes the ability to call routines written in conventional languages such as C or FORTRAN from the LISP environment, as well as the ability to call LISP routines from other languages.

Common LISP has a number of characteristics which recommend it for the K-Base project. LISP is a highly dynamic environment, in which variable bindings are determined at execution time. A single variable may, during a single execution session, contain a floating point number, a character string, a binary tree, an array, or any other data type. Memory management is completely automatic and is transparent to the programmer.

LISP code may be generated at execution time, and an interpreter is available so that the generated code may be executed immediately without any intervening compilation or linking steps. This level of interactivity allows facile implementation of such advanced features as frame-based systems with active slots, daemons which are activated automatically upon occurrence of certain conditions, *etc.*

A number of test programs were written in Lucid LISP to evaluate performance and interactivity. Several issues surfaced during this evaluation which reduced the programming team's perception of the utility of the Lucid LISP environment.

It was hoped that subsystems of K-Base could be implemented in LISP and subsequently embedded in a more comprehensive environment. The inter-language linkage, however, proved to be unwieldy. Due to the dynamics of the LISP environment, routines written in LISP require that linkage to other languages be performed at run time, thus burdening the user with the performance penalty of a linkage editing step during program execution. Worse yet, the inter-language linkage must be initiated in the LISP run-time environment. This complication precluded

convenient installation of a LISP subsystem into an extant environment. In addition, parameter passage between the different languages is both cumbersome and limited in functionality.

In addition to the linkage editing penalties, other performance problems surfaced. The Lucid LISP garbage collection system uses a half-space stop-and-copy model. At any given time, half of the allocated memory space is in use, while the other half is in reserve. When free space becomes exhausted, program execution is suspended and the garbage collection system proceeds to traverse the association lists and copy any used storage from the active half-space to the reserve half-space, discarding in the process any storage which is unreferenced. This copy operation is performed in its entirety before program execution can resume, often stopping operation for several minutes. The garbage collection process is extremely compute-intensive and will affect other users in a multi-user environment.

Compared to the FORTRAN environment to which the PLAID community has become accustomed, the numeric performance of Common LISP is rather poor. Migration of substantial portions of the PLAID functionality would, therefore, result in a perceived reduction in performance.

4.2.3 CLIPS - C Language Production System version 4.01

The Artificial Intelligence Section (AIS) at NASA/JSC has developed a rule-based expert system tool called CLIPS. This tool is a computer language designed for creating expert system applications. CLIPS was examined as a tool because of its portability and ease of integration into external systems. However, it was not clear whether its methodology could be used in a CAD knowledge base.

4.2.3.1 Portability

CLIPS is written in C using standard C library functions for its low level operations. This approach makes it very portability. CLIPS has been used on several platforms ranging from personal microcomputer's to large multiuser systems such as a VAX. It has been run under various operating systems such MSDOS, UNIX and VMS.

4.2.3.2 Ease of integration

The ease of integration is found in the fact that CLIPS can be both extended and embedded. The authors of CLIPS state that this was a primary reason for developing CLIPS. CLIPS can access and be accessed by other modules written in C, FORTRAN or ADA. (However, mixing languages requires advanced programming skills because of the differences in implementation of some data structures.)

4.2.3.2.1 Stand alone system

CLIPS can be run as a stand alone system (not embedded) where it can execute programs written in CLIPS. In this mode, CLIPS appears as a command line interpreter of CLIPS language programs.

4.2.3.2.2 User defined routines

CLIPS can be extended by linking user defined external functions with it. A routine named 'usrfuncs' is used to contain all references to user defined routines and is updated whenever a new routine is to become visible to CLIPS.

4.2.3.2.3 CLIPS as a library

CLIPS can be embedded in another program by linking its modules as a library to the calling program. This is the most useful feature when integrating with another large system where the impact of integration must be taken into account.

4.2.3.3 Applicability to a CAD knowledge base

It is not clear how a rule-base language system such as CLIPS can be used in the context of a CAD knowledge base. The following sections discuss issue.

4.2.3.3.1 CAD Knowledge Base

CAD knowledge bases tend to be object oriented because of the central occupation with objects. Object definitions and object relationships form the knowledge base. Users build and manipulate objects in a more or less step by step process. Parallel operations may be possible when dealing with more than one object, but because of the strong association of user interaction with objects and object interaction with objects, a cause and effect scenario usually develops.

4.2.3.3.2 CLIPS Knowledge Base

CLIPS represents knowledge as a set of rules (its knowledge base). The rules are matched to a list of facts and actions are taken as defined by the rule (inference engine). Its operations are performed in parallel in contrast to the more standard procedural (sequential) operations usual found in CAD systems.

4.2.3.3.3 User considerations

CLIPS defines a programming language and a user must learn to use the language to define the knowledge base and the facts which can be acted on. A CAD system will usually have in place some user interface (user friendly or not) which must be learned in order to use the system. Integration of another language or user interface such as CLIPS may introduce a considerable learning curve problem to the user.

4.2.3.3.4 An example

Consider, for example, the RMS (Remote Manipulator System) simulation problem. Currently, the RMS is modeled as a set of connected part nodes in the form of a tree. Each part down the length of the RMS is joined to the next part in a chain of parent-child relationships.

The joint angles computed by the RMS inverse kinematic routines are applied to each of the RMS parts which defines a joint. In order to properly update the transformations of each part defining the joints, the names of the parts must be known to the routine applying these joint angles. Currently a list in program memory of the parts must be defined (there are default names) by the user prior to a simulation run. If there is a change in the names, the user must update this list.

What the implication of this mode of operation is that for each RMS chain (there may be multiple configurations), the list must be made consistent by the user. Command files can be used to ease this task, but it would be helpful if the list could be part of the root node for each RMS chain of parts.

For example, a node may contain CLIPS language which can define the list of parts, the joint limits and the actions to be taken when a limit is reached (the RMS inverse kinematic routine deals with this, but in a special-purpose manner). Then the user can execute the node containing the CLIPS instructions with a set of joint angles as parameters with the rules of joint angle application embodied as part the data used to define an RMS model (a command file embedded in the data defining an RMS).

The problem with the use of CLIPS in this example, is the formulation of the CLIPS language by the user and applicability of the CLIPS paradigm to the RMS problem. No clear solution (if any) is evident. It might be easier to embed the current user interface command language in the data and use the currently language processor to execute them.

4.2.3.3.5 Future considerations

A solution to the user interface problem may be the integration of the CLIPS language with an existing user interface. The existing interface could generate CLIPS statements in response to appropriate user commands. However, this a non-trivial solution and would require some analysis.

4.2.4 DC-Representation Language (DC-RL)

DC-RL (Dave Cebula Representation Language) is a frame based knowledge representation language. It was developed at the University of Pennsylvania for research applications related to check-list processing. DC-RL is written in LISP and at the time its review it was operational for a VAX/VMS operating system environment.

4.2.4.1 Brief Description of DC-RL.

DC-RL is a derivation of the KL-ONE framed based system [Brachman 1985]. It is intended to provide a representation which can permit inferential operations, a structured semantic network and an external database referencing mechanism.

Concepts and roles are the basic elements of DC-RL. These elements are called objects. A concept object is a collection, class or thing. A role object is an the attribute associated with a concept. Concepts and roles are typed in order to extend the meaning a given concept or role relative to other concepts or roles in a given network or context. Typing is the defining of an object's access, semantic and definedness properties together a value representing the range of a concept's children or of a role's value.

For a concept, the access property defines storage and retrieval. The semantic property determines whether the concept is an instance, a class or a collection. The definedness property determines the level of completeness of values associated with a concept.

For a role, the access property determines how the role is to be used. The semantic property determines whether a role is an instance or a type of role. The definedness property determines the relevance of a role to the meaning or definition of the concept owning the role.

In addition to typing, DC-RL has a multiple inheritance paradigm for both concepts and roles. Concepts and roles can have any number of parents and inherit properties from these parents. However, roles are slightly different from concepts in that they are owned by a concept. This permits roles to have a dual form of inheritance. They may inherit via the owning

concept's family tree or they may inherit from their own family tree.

DC-RL also allows use of external database facilities to virtualize its universe network. It permits the imbedding of external interface functions to handle data conversion. These interfaces may be written in other languages (restricted to the linkage facilities of the version of LISP used and the operating system environment) and may access some commonly used data base management system.

4.2.4.2 Evaluation Procedure.

Inheritance is an important issue with a knowledge base for CAD because of the concern for objects and families of objects. It appeared that the inheritance features of DC-RL were the most interesting and useful; therefore the evaluation of DC-RL dealt with an example using basic multiple inheritance. For simplicity, only concepts were used in the example model. No role objects were needed as the basic multiple inheritance scheme works the same with both types of DC-RL objects.

4.2.4.3 Evaluation Results.

The multiple inheritance representation of DC-RL was satisfactory. The declarations of the parent-child relationships were straightforward although the syntax was not as clear. The multiple inheritance representation scheme was very powerful and easily handled the problem. However, The fact that the platform of DC-RL was LISP was a problem. As a consequence, the DC-RL declaration and access procedures can not be imbedded in other non-LISP environments.

4.2.4.4 DC-RL Browser - An Evaluation

See Appendix A-4.

4.2.4.5 Outline of sample data for use with DC-RL.

As a test for DC-RL, an outline describing some of the basic shuttle components and systems was written. An outline form is basically hierarchic and the components and systems can span multiple sections.

For example, components such as seats appear in the sub-sections, aft crew station, forward crew station and airlock. If viewing the shuttle from the context of seating arrangements, the emphasis of the outline is misplaced in that it is organized as physical sections of the shuttle.

Another example, is the reaction control system (RCS) which appears in the nose section and the tail section. The fact that the nose section and the tail section both contain RCS elements

is not obvious unless one scans the entire list of sections and sub-sections.

DCRL was used as way to organize the information in the outline such that a more flexible representation might provide more knowledge about component relationships.

4.2.4.6 Shuttle OV-103 Discovery

The shuttle OV-103 is organized spatially as the forward section, the payload bay section, the wing section and the tail section. These sections are then further divided until basic components are listed.

4.2.4.6.1 Forward Section

The forward section of the shuttle is that region forward of the payload section. The forward section is where the crew will spend most of their time.

4.2.4.6.1.1 Upper Deck

4.2.4.6.1.1.1 Aft Crew Station

Overhead viewports
Remote-Manipulator Translation Hand Controller
Remote-Manipulator Rotational Hand Controller
Orbitor Rotational Hand Controller
Payload Control Panel
Mission Specialist Seat
Payload Specialist Seat
Interdeck Access

4.2.4.6.1.1.2 Forward Crew Station

Mission Commander's Seat
Pilot's Seat
Flight Computer and Navigation Console
Navigation Unit

4.2.4.6.1.2 Lower Deck

Galley Space

4.2.4.6.1.2.1 Airlock

Interdeck access
Telescoping Escape Pole (new)
Extra Payload Specialists' seats (2)
Waste Management
Stowage Lockers
Avionics/Electronics Bay

4.2.4.6.1.3 Nose Section

4.2.4.6.1.3.1 Reaction Control System (RCS)

RCS Forward Thrusters
RCS Oxidizer Tank
RCS Helium Tank
RCS Hydrazine Fuel Tank
Phased-array Radar
Nosewheel Landing Gear (improved)

4.2.4.6.2 Payload Bay Section

The payload bay section is that region between the forward section and the tail section of the shuttle. It is used for storing the shuttle payloads, particularly deployable payloads. It is sometimes visited by the crew for EVA tasks.

4.2.4.6.2.1 Payload Bay Doors (2)

Radiators (4)

4.2.4.6.2.2 Remote Manipulator Arm

Elbow Video Camera (Videocam)
Extravehicular-activity Handhold
Getaway Special Canister
Aluminium Sheathing (Payload Bay lining)
Supports i.e. for Tracking and Data Relay Satellite (TDRS)

4.2.4.6.2.3 Below Payload Bay

Ventilator Liquid-Oxygen Tank
Fuel Cell Liquid-Hydrogen/Liquid-Oxygen Tanks

4.2.4.6.3 Wing Section

Main Landing Gear
Reinforced Carbon-Carbon Leading Edge
Elevon (Aluminum Honeycomb Structure)

4.2.4.6.4 Tail Section

The tail section contains the bulk of the shuttle's propulsion systems. There is no crew space in the tail section.

4.2.4.6.4.1 Space Shuttle Main Engines (3)

High-pressure Fuel Turbopump (improved)
Liquid-Hydrogen Supply Manifold
Liquid-Oxygen Supply Manifold

4.2.4.6.4.2 Auxiliary Power Hydrazine/Oxidizer Tanks

Fuel Cell

4.2.4.6.4.3 Reaction Control System (RCS)

RCS Oxidizer Tank
RCS Hydrazine Fuel Tank
RCS Aft Thrusters
RCS Helium Tanks (2)

4.2.4.6.4.4 Orbital Maneuvering System (OMS)

OMS Hydrazine Fuel Tank
OMS Oxidizer Tank
OMS Helium Tank
OMS Thruster

4.2.4.6.4.5 Rudder (Aluminum Honeycomb Structure)

Rudder/Speed Brake Power Unit
Rudder/Speed Brake
Rudder/Speed Brake Hydraulics

4.2.4.7 Tracking and Data Relay Satellite

C-Band Commercial Antenna
4.9 Meter K/S-Band Antenna (2)
2.0 Meter K-Band Ground-Link Antenna
Stowed Solar Array
Inertial Upper Stage

4.2.4.8 A sample of DCRL knowledge representation.

The following DCRL code was used to represent the information contained in the outline in section 3 above. The terms *universe* and *tout* are introduced from DCRL to provide a context for the shuttle-ov-103. The implication is that the shuttle-ov-103 lives in a universe which is a collection of universes found in tout (the top of universe tree).

The code given here is basically declarations of concepts (topics, data, categories, etc) and their relationships with other concepts. The example does not illustrate all of the capability of DCRL, but it does show those features of interest for a CAD knowledge base.

4.2.4.8.1 DCRL declarations.

```
;;; setting up a universe for shuttle-ov-103 to live in

{ concept universe
        is a collection of concept
        from tout }

{ concept shuttle-ov-103
        is a collection of concept
        from universe }

;;; now, set up a concept (category)
;;; called people-seats within shuttle-ov-103

{ concept people-seats
        is a collection of concept
        from shuttle-ov-103 }

;;; set up other categories within shuttle-ov-103

{ concept propulsion-system
        is a collection of concept
        from shuttle-ov-103 }

{ concept guidance-system
        is a collection of concept
        from shuttle-ov-103 }

{ concept fuel-system
        is a collection of concept
        from shuttle-ov-103 }

{ concept forward-section
        is a collection of concept
        from shuttle-ov-103 }

{ concept payload-section
        is a collection of concept
        from shuttle-ov-103 }

{ concept wing-section
        is a collection of concept
        from shuttle-ov-103 }

{ concept tail-section
        is a collection of concept
        from shuttle-ov-103 }

{ concept rms-system
        is a collection of concept
        from shuttle-ov-103 }
```

```
{ concept hand-controllers
        is a collection of concept
        from shuttle-ov-103 }

;;; set up categories within previously defined
;;; categories, etc

{ concept upper-deck
        is a collection of concept
        from forward-section }

{ concept lower-deck
        is a collection of concept
        from forward-section }

{ concept nose-section
        is a collection of concept
        from forward-section }

{ concept main-engines
        is a collection of concept
        from
           (tail-section
           propulsion-system) }

;;; note, here the reaction-control-system is
;;; defined such that is within several different
;;; categories at once

{ concept reaction-control-system
        is a collection of concept
        from
           (node-section
           tail-section
           propulsion-system) }

{ concept rudder
        is a collection of concept
        from
           (tail-section
           guidance-system) }

{ concept rcs-oxidizer-tank
        is a collection of concept
        from
           (reaction-control-system
           fuel-system) }

{ concept rcs-hydrazine-tank
        is a collection of concept
        from
           (reaction-control-system
           fuel-system) }
```

```
{ concept rcs-helium-tank
        is a collection of concept
        from
           (reaction-control-system
           fuel-system) }

{ concept rcs-aft-thrusters
        is a collection of concept
        from reaction-control-system }

{ concept oms-hydrazine-tank
        is a collection of concept
        from
           (orbital-maneuvering-system
           fuel-system) }

{ concept oms-oxidizer-tank
        is a collection of concept
        from
           (orbital-maneuvering-system
           fuel-system) }

{ concept oms-helium-tank
        is a collection of concept
        from
           (orbital-maneuvering-system
           fuel-system) }

{ concept oms-thruster
        is a collection of concept
        from orbital-maneuvering-system }

{ concept forward-crew-station
        is a collection of concept
        from upper-deck }

{ concept aft-crew-station
        is a collection of concept
        from upper-deck }

{ concept navigation-unit
        is a collection of concept
        from
           (forward-crew-station
           guidance-system) }

{ concept phased-array-radar
        is a collection of concept
        from
           (nose-section
           guidance-system) }
```

```
{ concept extra-mission-spec-seat
        is a collection of concept
        from
           (lower-deck
           people-seats) }

{ concept mission-spec-seat
        is a collection of concept
        from
           (aft-crew-station
           people-seats) }

{ concept payload-spec-seat
        is a collection of concept
        from
           (aft-crew-station
           people-seats) }

{ concept command-seat
        is a collection of concept
        from
           (forward-crew-station
           people-seats) }

{ concept pilot-seat
        is a collection of concept
        from
           (forward-crew-station
           people-seats) }

{ concept main-landing-gear
        is a collection of concept
        from wing-section }

{ concept nosewheel-landing-gear
        is a collection of concept
        from nose-section }

{ concept fuel-turbopump
        is a collection of concept
        from main-engines }

{ concept liquid-hydrogen-supply-manifold
        is a collection of concept
        from main-engines }

{ concept liquid-oxygen-supply-manifold
        is a collection of concept
        from main-engines }
```

```
{ concept rms-translation-hand-controller
        is a collection of concept
        from
            (aft-crew-station
            rms-system hand-controllers) }

{ concept rms-rotational-hand-controller
        is a collection of concept
        from
            (aft-crew-station
            rms-system hand-controllers) }

{ concept orbitor-rotational-hand-controller
        is a collection of concept
        from
            (aft-crew-station
            hand-controllers) }

{ concept rms-arm
        is a collection of concept
        from
            (payload-section
            rms-system) }

{ concept payload-bay-doors
        is a collection of concept
        from payload-section }

{ concept waste-management
        is a collection of concept
        from lower-deck }

{ concept tdrs
        is a collection of concept
        from universe }

{ concept antennas
        is a collection of concept
        from universe }

{ concept c-band-commercial-antenna
        is a collection of concept
        from
            (tdrs
            antennas) }

{ concept ks-band-antenna
        is a collection of concept
        from
            (tdrs
            antennas) }
```

```
{ concept K-band-ground-link-antenna
        is a collection of concept
        from
           (tdrs
           antennas) }

{ concept stowed-solar-array
        is a collection of concept
        from tdrs }

{ concept inertial-upper-stage
        is a collection of concept
        from tdrs }
```

These declarations form basic knowledge for use by other features on DCRL which are mainly access routines. However, the declarations provide a basis for analysis of knowledge representation schemes.

4.2.5 Flavors

Flavors is an object-oriented programming extension to the Common LISP language. As such, Flavors shares all the strengths and weaknesses of its parent environment. Programming in Flavors, as in Common LISP, allows the programmer a great deal of flexibility due to the late binding of variables. Flavors adds to this flexibility the object-oriented concepts of data abstraction, data encapsulation, multiple inheritance and procedure encapsulation.

The Flavors system offers a very dynamic system of class definition and object instancing. Class definitions may be altered and new objects instanced at execution time.

Difficulties with the Flavors system are a superset of those encountered with Common LISP. Although class definitions may be changed at execution time, instances of that class do not reflect the changes made to the parent class. In order to implement changes in the instances, they must be destroyed after their data has been copied to a new instance of the parent class.

At the time of the review, there was no standard for the Flavors system such as that which exists for Common LISP. Programs implemented with Flavors are, therefore, not completely portable. A standard definition for the language is said to be forthcoming, however, which would allow portable systems to be created.

4.2.6 Knowledge Engineering Environment (KEE)

The Knowledge Engineering Environment (KEE) system produced by IntelliCorp is a Lisp-based multiple-paradigm software development system. KEE incorporates the object-oriented, rule-based, access-oriented, and frame-based programming paradigms

into a software development environment which sports a sophisticated graphical user-interface.

KEE has been ported to all the major computer architectures which support Common Lisp including Sun workstations, VAXstations, and Symbolics 3600's.

The fact that KEE is Lisp-based makes the system susceptible to all of the problems associated with Lisp. Foremost among these problems are run-time inefficiency and the dreaded garbage-collection cycle.

Software written under the KEE system cannot be embedded within other applications. KEE provides its own required execution environment which is constructed atop the Lisp run-time environment. This architecture makes KEE difficult to interface with existing software systems such as PLAID.

4.2.7 LOOPS/XAIE

XAIE is an acronym for the Xerox Artificial Intelligence Environment. This programming environment, which includes the LOOPS object-oriented programming system, was evaluated on the only hardware platform upon which it is available: the Xerox 1186 AI Workstation. The LOOPS environment is similar to the Flavors environment described above; similar strengths and weaknesses apply.

The XAIE development environment has a number of attractive characteristics for utilization in the K-Base project. XAIE has been in development and use at Xerox Palo Alto Research Center for a number of years and is a very mature, well-developed programming environment. The system features excellent on-line help facilities, and a facility called DWIM, an acronym for Do What I Mean, which attempts to analyze typographical and syntactical errors entered by the user and suggest what the correct entry might have been. The system contains a toolbox of "gadgets and gauges", graphical input and output valuators which allow the user to easily view and manipulate program variables.

XAIE/LOOPS is Common LISP based, and shares the strengths and weaknesses of the Common LISP environment as described above. No languages other than Common LISP/Common LOOPS were available on the system.

Two major faults made XAIE unusable as a K-Base development tool: platform dependency and network communications deficiencies. The Xerox 1186 workstation upon which XAIE depends is both low in performance and limited in expansion capabilities. The system as evaluated could be expanded to a maximum configuration of 4Mb of RAM and 80 Mb of disk; both constraints are extremely restrictive in a LISP environment. In addition, network communications capabilities with VMS/DECNET were limited in scope and seemed to be poorly implemented. The

only networking facility available for communication with VMS was file transfer. As an example of implementation difficulties, networking documentation stated that communications with DECNET could be achieved only if the VMS system was running Version 3 DECNET software; at that time, the current version of VMS was Version 4.6, with Version 5 already announced.

4.2.8 New Flavors

New Flavors is the implementation of the Flavors system which is available on the Symbolics line of AI workstation products. New Flavors is an enhanced implementation of the Flavors system described above, and shares the strengths and weaknesses of that system.

The performance penalties associated with LISP/Flavors on the Symbolics hardware are not as great as on general purpose computer systems, as the Symbolics hardware is optimized for Common LISP and implements a concept known as "ephemeral garbage collection", which reduces (but does not eliminate) the need for the stop and copy garbage collection process as described above. In addition, Genera, the Symbolics operating and development environment, is both mature and rich in functionality. Excellent on-line help facilities are available.

Inter-operability with the VAX VMS environment, however is less than optimal. Network operations are limited to file transfers.

4.2.9 Nexpert Object

Nexpert Object is a multiple-paradigm, object-oriented programming environment available from Neuron Data, Inc. Nexpert shares a number of features with more expensive AI development environments, such as KEE, and features such advanced features as multiple inheritance of object classes, a rule-based reasoning system and an object database complete with an external representation. The system is available for multiple hardware platforms, including VAX/VMS workstations.

Nexpert initially appeared to be a promising platform for K-Base development. A demonstration copy was procured and evaluated on the IBM AT system.

Nexpert has a number of excellent features. The system includes an excellent user interface, with multiple windows and tools such as a hierarchy browser. Functionality of Nexpert is good, with excellent implementation of the object-oriented programming paradigm. Nexpert can also function as a knowledge base server for external applications written in high-level languages such as FORTRAN and C.

Upon careful evaluation, however, Nexpert was found to have a single flaw which precluded its use as a platform for K-Base.

Database access from foreign applications is read-only; only Nexpert applications may modify the knowledge base. Since the ability to modify the knowledge base from external applications (such as PLAID) was deemed to be critical to the success of K-Base, the principal advantage of Nexpert Object was nullified and no further evaluation was undertaken.

4.2.10 PC-Scheme / Scoops

PC-Scheme/SCOOPS is a Texas Instruments, Inc. implementation of the LISP-like programming language Scheme, which was introduced in 1975 by Gerald J. Sussman and Guy L. Steele which has been extended by the addition of SCOOPS, a LOOPS-like object-oriented programming system.. Scheme was the first dialect of LISP to fully support static scoping, first-class procedures and continuations, and was a precursor to the development of the Common LISP language. Scheme is relatively small, and derives most of its power from a small set of concepts. Its size permits it to be utilized effectively on a microcomputer system with limited memory.

PC-Scheme was purchased and evaluated on the IBM AT to assess the feasibility of employing the object-oriented programming paradigm for the K-Base project. Appendix 5 documents a familiar application implemented in Scheme/SCOOPS using (or perhaps abusing) an object-oriented approach.

Due to its dependence on the IBM PC hardware platform, PC-Scheme was not considered as a candidate for use in implementation of K-Base, but rather as a teaching and evaluation tool for use by the research team. It served well in this role, and is to be recommended for similar exercises in the future.

4.2.11 Peabody

Peabody is representation paradigm developed by University of Pennsylvania. Peabody is designed to model objects with constraints. It attempts to define a joint and constraint network representing segmented objects which can then be processed by a graph spanning algorithm to satisfy joint and constraint definitions for various scenarios involving object manipulation.

4.2.11.1 Environment

The Peabody environment is a collection of figures, segments, joints and constraints. It defines a graph in which the nodes are segments and the edges between the nodes are constraints and joints. Segments are defined as primitive objects with an associated geometric definition. Joints are defined as tightly bound connections between nodes. Their connection definitions are rigid. Constraints are defined as loosely bound connections between nodes. Their connection definitions are non-rigid whenever loops within the graph are to be resolved. Figures are

defined as sub-graphs of the environment graph and are connected to the environment only by constraints .

Connection definitions are formulated with the use of sites. Sites are defined as locations and attitudes relative to the local coordinate system of a segment. A joint or constraint definition is the binding of two sites on two different segments. Sites, connections and segments are all referenced by unique names given by the user to each of them.

Peabody is written in the C language and its user level syntax for defining its representations is C like in form. The Peabody representation (segments, connections and sites) is in the form of a text file which is used like a script by the graph resolving algorithm to build a spanning tree. This tree permits the articulation of the objects (segments) within the environment defined by the representation.

4.2.11.2 Problems with Peabody

With regard to the K-Base paradigm, the Peabody representation, in its form at the time of review, was problematic.

For example, it could not represent a hierarchy of objects as groups defined as a subassembly. It is true that a figure defined as a sub-graph could be viewed as a subassembly, however, sub-graphs could not be nested in other sub-graphs. In other words, the Peabody environment permitted only one level of assembly. In a complex environment, it is necessary to have many levels of assembly or sub-graphs. On the abstract level, single level hierarchies do not permit the richness of meaning derived from the inheritance and class definitions supplied by multiple level hierarchies. It was concluded that the Peabody paradigm is an excellent one, but it just did not go far enough with its representation.

In addition to the multiple level hierarchy problem, the use of text file formats as scripts was also a problem. The scripts are not conducive to a high degree of user interaction. If a change occurs in a definition of a component of the graph, the complete script must edited and then resubmitted to the graph spanning algorithm. This reduces user interaction greatly when the graph becomes large. The script itself, is not the problem; it is the question of how it is to be edited by the user in a highly interactive session which must be addressed.

4.2.11.3 Peabody Contributions

In its role as a prototyping tool for K-Base, the PLAID system incorporated several of the features found in the Peabody system, because of the similarity in the target problem addressed

by both Peabody and PLAID: the assembly and articulation of objects.

To add more knowledge to the current object representation in PLAID, the Peabody features of named site definitions and assignable joint attributes were incorporated into the multiple level hierarchy currently used by PLAID.

In the PLAID system, a site is a named location and attitude defined relative to the coordinate system local to the level of assembly at which the site is placed. In other words, the site is defined relative to all the other objects at the same level. The site is an object which is part of the collection of objects at a given level of assembly.

Joint attributes can be given to a site, defining its degrees of freedom and any corresponding limits. However, these sites with joint attributes are not quite like the constraint definitions found in Peabody. For example, explicit loops or cycles in the tree graph defined by PLAID are not possible; so the role of constraints as breaking points for such loops or cycles is not required. Any potential loops or cycles, such as a hand grabbing an object off the floor where the hand and the object are in the same tree graph, are resolved dynamically at the moment the object is attached to the hand. However, the explicit representation of cyclical relationships possible with Peabody does permit a more dynamic redefinition of the root of a given spanning tree graph.

4.2.11.4 Conclusion

The Peabody representation, in its form at the time of review, could not be used directly by the K-Base paradigm. However, many of its features were useful in exploring the areas of object representation and object articulation. Some of the features were incorporated into the PLAID prototyping tool and made available to its users.

4.2.12 SmallTalk-80

SmallTalk, the prototypical object-oriented software development environment, was developed at Xerox Palo Alto Research Center beginning in the 1970's. The visionaries who designed SmallTalk invented many of the key features of the user-friendly graphical user interfaces which now appear in numerous commercial products including the Apple Macintosh and the Microsoft Windows system. The SmallTalk vision, however, extends far beyond the user interface.

SmallTalk is a complete, self-contained software development system which is composed of a set of interlocking components. The key components of the system are an object-oriented programming language, a standard set of object classes defined in the language,

and the virtual SmallTalk machine. The predefined classes make it possible for a programmer to perform complex tasks in only a few lines of code written in the SmallTalk language.

The problem with the SmallTalk system is that it *is* an integrated programming environment. No provision is made for incorporating programs written in non-SmallTalk languages into the environment short of completely rewriting them.

4.3 Selected System Configuration

The system assembled as a vehicle for K-Base development consists of a closely-coupled network of four Digital Equipment Corporation 32-bit VAX microcomputers with a comprehensive set of software tools running under the VMS operating system. This system was chosen to assure complete compatibility with the target operating environment at JSC, and to provide an operating environment for PLAID, the primary client application for K-Base.

The system hardware consists of a MicroVAX II serving as a boot node and file server for a Local Area VAXcluster (LAVC) which includes three low-cost VAXStation 2000 workstations.

System software includes the proprietary VMS operating system, LAVC software, DECnet software, a DEC FORTRAN language compiler, a DEC C language compiler, and a Lucid LISP system from Lucid, Inc.

4.3.1 MicroVAX II Boot Node

The MicroVAX II boot node was provided by NASA as Government Furnished Equipment (GFE). The system as provided contained 9 Mb of RAM, three 71 Mb disk drives, eight serial ports, and an Ethernet interface. GMS added one 159 Mb disk drive and a DESTA thin-wire Ethernet adapter to complete the configuration.

4.3.2 VAXstation 2000 Workstations

The MicroVAX II system served three identical VAXstation 2000 workstations; one for each member of the research team. Each VAXstation included a thin-wire Ethernet interface, a bit-mapped graphic display, a mouse and workstation software as standard equipment. In addition to the standard equipment, each VAXstation was configured with 16Mb of RAM memory from Clearpoint, Inc. and a 71 Mb local disk drive to be used primarily for paging and swapping storage.

4.3.3 Local-Area VAXcluster (LAVC)

The LAVC software ties the four systems together via Ethernet as a VAXcluster, and provides most of the advantages of a VAXcluster environment without requiring expensive hardware interconnects between systems. VAXcluster advantages include simplified system management, elimination of redundant data storage for system

software, transparent access to disk drives which physically reside on remote machines and a close approximation to the target VAXcluster environment at JSC.

4.3.4 Common LISP (Lucid)

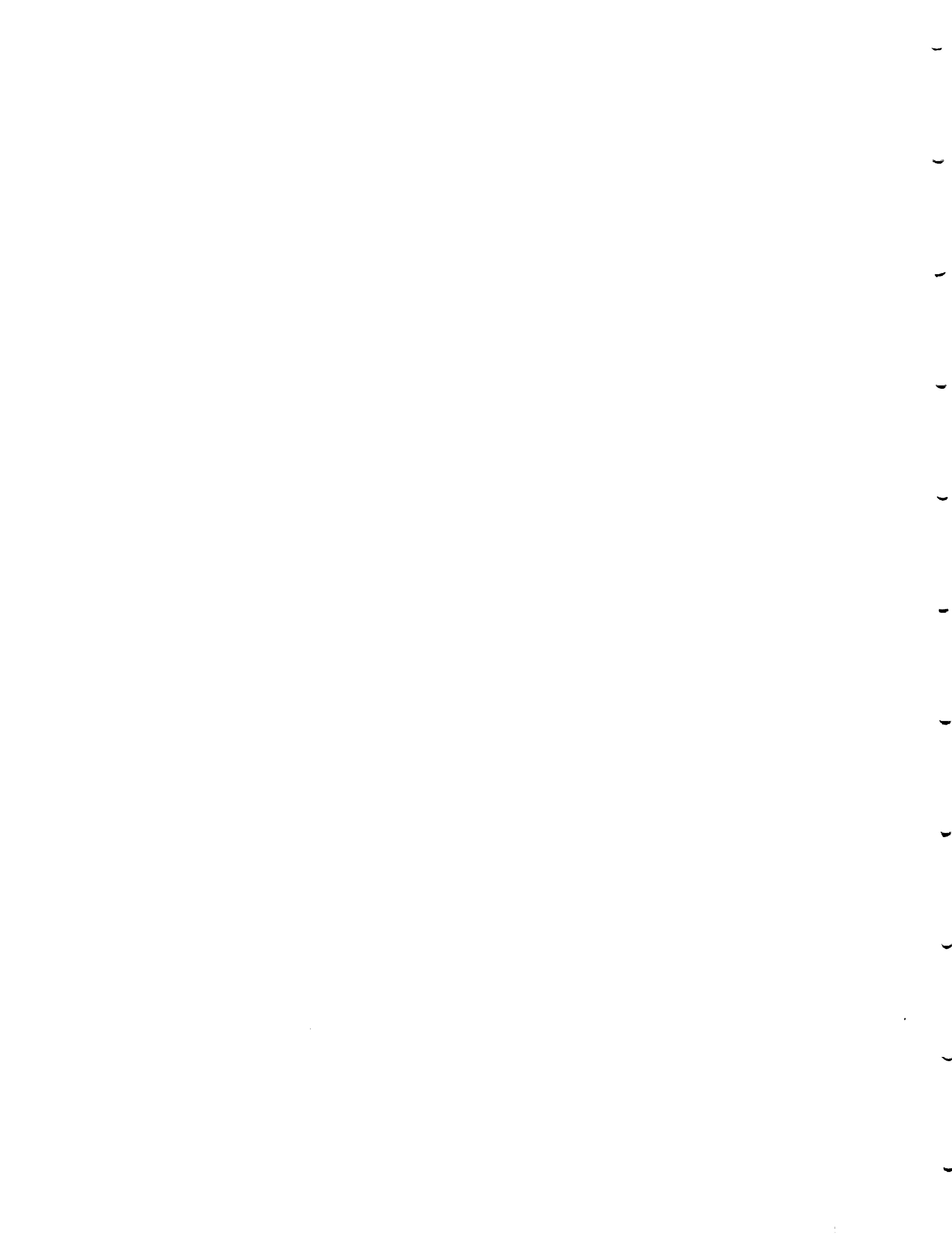
The LISP system chosen is Lucid LISP, from Lucid, Inc. The Lucid system was chosen over VAX LISP due to its inclusion of an implementation of the Flavors object-oriented programming system and its superior interface to the graphical user interface (GUI) of the VAXstations.

4.3.5 C Compiler

The VAX C language compiler was chosen as a K-Base development tool because of its compatibility with the UNIX and ULTRIX C languages and its efficient code generation.

4.3.6 FORTRAN Compiler

The VAX FORTRAN language compiler was chosen as a K-Base development tool because of its efficient code generation and its compatibility with existing PLAID software.



5. Work Performed

The implementation of K-Base may be divided into two broad functional areas; non-geometric (symbolic) knowledge enhancements and geometric knowledge enhancements. This section of the report is divided along those lines. We first consider the symbolic knowledge enhancements.

5.1 K-Base Symbol Management System (KB/SMS) Specification

5.1.1 Purpose of KB/SMS

The K-Base Symbol Management System (KB/SMS) is designed to extend the modeling capabilities of the PLAID system beyond the realm of purely geometric modeling while retaining the existing Multi-User contextual referencing mechanism. This goal requires that KB/SMS be a tightly-coupled component of the Multi-User PLAID system, utilizing and extending existing PLAID part access mechanisms.

KB/SMS also embodies features which aid in the maintenance of the large database of PLAID component files, target files, and display files by providing facilities for searching and reporting on parts based upon attributes of the parts. This facility will aid the PLAID user community in tracing the historical genesis and evolution of each part.

5.1.2 Approach

KB/SMS overcomes the symbolic information shortage by associating relevant textual information with each PLAID part file. The additional information for a given part is stored in a description file associated with the part file. Each description file contains a basic set of attribute fields which are common to all description files as well as optional fields which may be arbitrarily added by PLAID users.

The description files are managed by the Multi-User Manager process in conjunction with the various PLAID modules. The Manager process determines where the description files are stored in the file system. The PLAID modules are responsible for generating and maintaining these files. Please note that this distinction (Manager process vs. PLAID module) is invisible to the PLAID user.

New commands added the PLAID Multi-User Interface program provide access to a set of new search and report generation functions. The new functions allow parts to be searched for on the basis of information stored in the description files as well as on the basis of partname specification. Searches which combine partname searches with attribute-value searches provide a very powerful tool for part-file management.

5.1.3 Description of a Description File

A description file consists of a set of attribute name/value pairs. The attribute name is a character string which serves the same function as a field name in a database. The value part of each pair is a free-form block of text which is available for editing with a standard text editor.

Unlike a traditional database, KB/SMS allows any character string to be used as an attribute name and an arbitrarily long string of text to be used as its value. This has the benefit that the "database" is not restricted to a fixed set of fields; attributes specific to a particular item can be added at will. The negative side of this feature is that any given description file may contain errors in name spelling or total omission of data which would cause the query system to generate incomplete or incorrect reports.

We propose a basic skeleton for the description files which constitutes a required set of name/value pairs. The symbol management system requires that these names be defined in each description file and provides facilities for automatically maintaining them. Along with the required set of names, the user may define additional names as needed.

5.1.3.1 Description File Skeleton

```
::PARTNAME:: (name of the associated PLAID file)
::DOMAIN::(the name of the domain in which the file  resides)
::REAL_NAME::(the title of the object, like "Tracking and Data Relay
Satellite (TDRS)")
::DESCRIPTION::(a textual description of the object)
::CREATOR_NAME:: (the name of the person who created this  object).
::CREATION_DATE:: (the date on which this object was created)
::CREATION_FUR:: (the FUR designation under which the object was
created)
MOD_LIST: (a sequence of time/date stamped events which have
caused changes to the file)
DATE: (date the change was made)
NAME: (name of person making the change)
REASON: (reason for the change)
FUR: (for the change order)
ENDMODS:

(USER-DEFINED SYMBOLS GO HERE...)
```

5.1.4 Data Entry

A description file is created each time a new PLAID part file is created and updated each time a PLAID part file is modified. KB/SMS automatically maintains the validity of the symbols which are members of the skeleton symbol set.

5.1.4.1 Data Entry at File Creation Time

The symbol management system generates a new description file which contains a description file skeleton whenever a new PLAID part file is created. The newly created description file contains the fields shown in the Skeleton of a Description File. Information that is known by the software will be filled-in automatically (e.g. PARTNAME, DOMAIN, CREATOR_NAME, CREATION_DATE). The user is subsequently given an opportunity to edit the description file in order to input information into the remaining fields of the skeleton, or to add new fields to the description.

5.1.4.2 Data Entry at File Modification Time

The symbol management system updates each description file whenever the associated PLAID part file is modified. The default change to the description file is a new entry in the MODLIST section which indicates the account-name of the user making the modification along with the date of modification.

5.1.4.3 Data Entry at Description Editing Time

The symbol management system allows the user to edit all or part of a description file using a conventional text editor. The user must be aware that editing the description file can cause erroneous or incomplete query reports due to erroneous editing with the text editor.

5.1.5 Query Commands

Query commands provide facilities for generating reports based upon information in the description files. Two fundamental reporting modes are supported; global reporting and contextual reporting. Global queries traverse down the hierarchy of projects (domains), collecting information on each occurrence which meets the search criteria. Conversely, contextual queries traverse up the hierarchy of projects (domains).

5.1.5.1 Contextual Queries

Contextual queries are searches that are performed in the context of a specified project. These searches begin in a specified project domain and search up the context, locating all occurrences of the specified part(s). The search order for

contextual queries is the same as the search order of the Multi-User "RESO" (Resolve Partname) command.

5.1.5.1.1 Find All Occurrences of <part-expr>

This command searches the current context for all occurrences of parts which match the given <part-expr> and displays the filenames that it finds. The distinctive feature of this command is that it displays partname overloading so that if one file hides another in the context, both filenames are displayed with an annotation that describes which file hides which.

The syntax of this command is

FIND <part-expr>

where <part-expr> is any valid VMS filename expression which may include wild-cards.

5.1.5.1.2 Find All Occurrences of <part-expr> with <expr-list>

This command will perform the same file search as the previous command, then search the description files for name/value pairs which satisfy the <expression-list>. For example, to locate all primitives and assemblies with DESCRIPTION containing the string "space-station" one could enter the command:

FIND *.PRI,*.COG DESCRIPTION="*space-station*"

Note the asterisks in the description search string. An asterisk in any search string means that any string may occur in place of the asterisk. In general, the search strings may include any Unix-style regular expression.

5.1.5.2 Global Queries

Global queries are searches of an entire sub-tree of the project hierarchy. These queries begin at a specified root of the project tree and search downward through all project sub-trees. The order in which projects are searched is the same as the order of projects listed by the Multi-User "LCSF" (List Context Structure) command. Global queries will show how partnames are overloaded in each context that is scanned.

5.1.5.2.1 Find <part-expr>

This command searches downward in a project hierarchy to show all occurrences of a given set of files specified by <part-expr> as well as any file overloading. For example,

this command might be used to locate all primitive files which begin with "XP" as follows:

FIND/GLOBAL XP*.PRI

5.1.5.2.2 Find All Occurrences of <part-expr> with <expr-list>

This command performs the same hierarchy search as above, then selects specified files from those located based upon <expr-list>. For example, the command might be used to locate all primitive files with names beginning with "XP" which have PROJECT=TDRS with a command of the form:

FIND/GLOBAL XP*.PRI PROJECT=TDRS

5.1.6 Report Generation

The generation of reports is controlled by a single command which specifies what information will be gathered and where it will be stored. The syntax of the report specification command is be:

REPORT <name-list> <destination>

where

<name-list> is a list of the attribute names from which informations is to be collected, and

<destination> is the name of the file in which the report is to be placed. If no destination is specified, the report will be displayed on the terminal.

Once the report format and destination are specified, the required information is collected using the query commands described above.

5.1.6.1 Example of a Report Generation Session

Suppose that one wished to generate a report which listed all PLAID part files which were associated with flight STS-29. The following Multi-User dialog could be used to collect this information (assuming that "STS-29" is somewhere in the description files):

```
SETP root-project-name
REPORT PARTNAME STS-29.RPT
FIND/GLOBAL *.PRI,*.COG ANY="STS-29"
REPORT CLOSED
$PRINT STS-29.RPT
```

5.1.7 Global Report Algorithm

This section provides the outline of the algorithms used by the KB/SMS software to produce a global search report. The algorithms and data-structures are described in pidgin Pascal.

1. Traverse the Context Structure in pre-order fashion, scanning each domain for the specified files, and building an indexed file with records like this:

```
Record Part_Entry is
    name:       char*20, // Filename.
    seq_no:     char*8,  // Sequence number.
    domain:     char*63, // Where file was found.
    t_level:    integer, // Context tree level.
    o_level:    integer  // Occurrence level.
End_Record.
```

where `name` concatenated with `seq_no` is the primary key, `t_level` is the depth of domain_name in the traversal of the context, and `o_level` is the occurrence level which is initialized to zero.

2. After all domains have been scanned and the indexed file built, determine the occurrence level of each file as follows:

```
Procedure Set Occurrence Level:

Part_List: File of Part_Entry.
Parent, Child: Part_Entry.
    read-first-record from Part_List into: Parent.
    do:
        read-next-seqential from Part_List into: Child.
        while (Child.name == Parent.name),
        do:
        if child.t_level <= Parent.t_level
        then
            pop (Parent).                  /* pops Parent.name==NULL if
            stack is empty! */
        elseif Parent.domain is on the path of Child.domain,
        then
            Child.o_level := Parent.o_level + 1.
            rewrite Child record.
            push (Parent).
            Parent := Child.
            read next seq. record into Child.
        else
            pop (Parent).
        endif.
    enddo
    clear the stack.
    Parent := Child.
until (EOF encountered on the indexed file).
End Procedure.
```

3. Each file name now has an occurrence level associated with it. We can use the occurrence level numbers to correctly show the nested overloading of the names as follows:

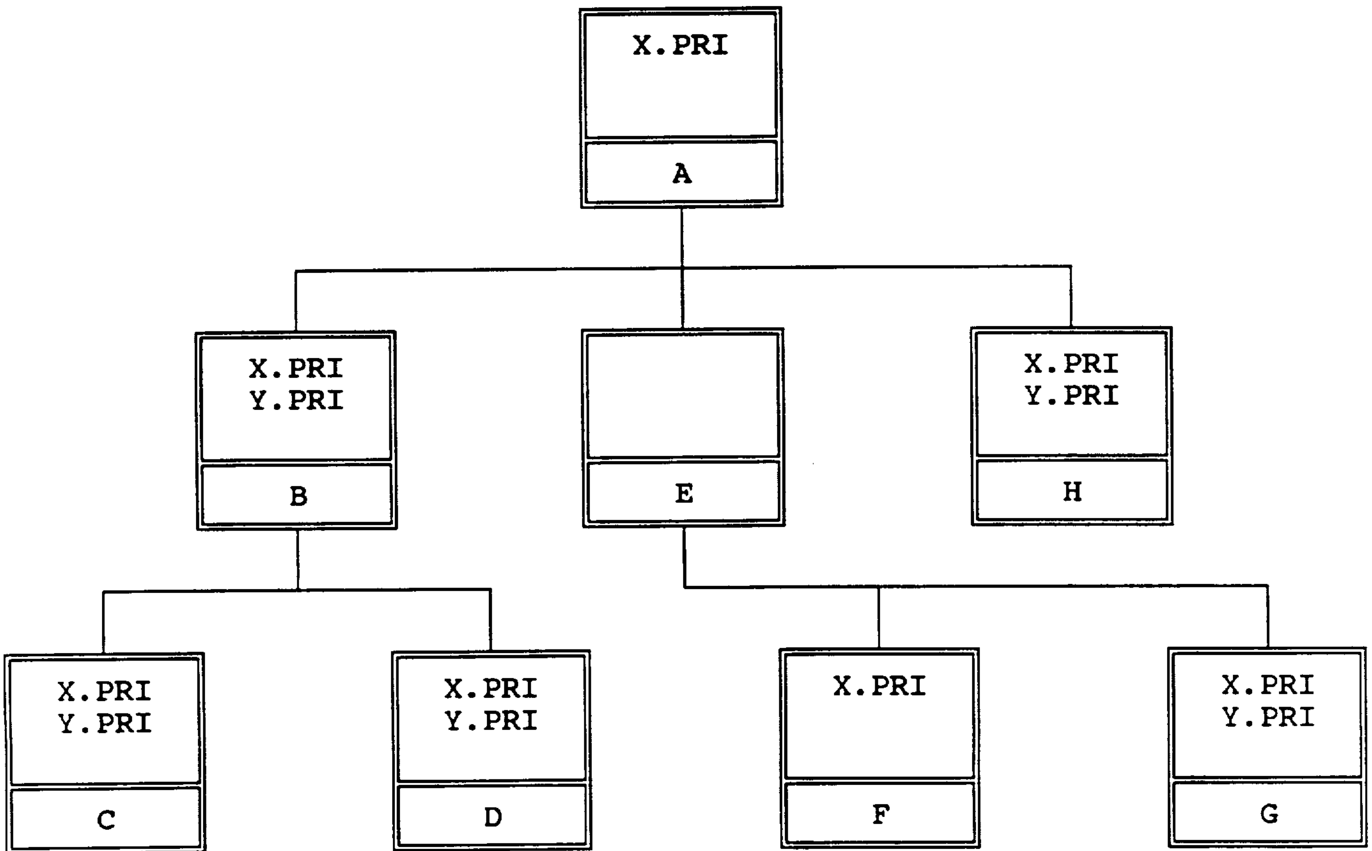
Procedure Display Overloading:

```
    Part_List: File of Part_Entry.
    reset the indexed file to the beginning.
    do:
        read the next sequential record into cur_*.
        tab the output as a function of cur_o_level.
        write the filename & domain name to output.
    until EOF on the indexed file.
End Procedure.
```

5.1.8 Example of a Global Search

This section elucidates the algorithms with which KB/SMS traverses a global context and produces a report. This information is meant for programmers who might be tasked with modifying the procedures. This process is, except for the output report, completely hidden from the PLAID user.

5.1.8.1 GIVEN: The following domain hierarchy and parts:



5.1.8.2 PRODUCE:

A report that shows the overloading of each of the parts in the global context of the A domain as follows:

```
Part         Domain       Level
----         ------       -----
X.pri          A            0
  X.pri        B            1
    X.pri      C            2
    X.pri      D            2
  X.pri        F            1
  X.pri        G            1
  X.pri        H            1

Y.pri          B            0
  Y.pri        C            1
  Y.pri        D            1

Y.pri          G            0

Y.pri          H            0
```

Step 1:

KB/SMS produces an indexed file which contains the serialized part names in the global scope of domain A. The sequence numbers are the result of a running count of part files located during a pre-order traversal of the tree of domains with root A:

Name	Domain	Seq#	T_Level	O_Level
X.pri	A	1	0	0
X.pri	B	2	1	0
X.pri	C	4	2	0
X.pri	D	6	2	0
X.pri	F	8	2	0
X.pri	G	9	2	0
X.pri	H	11	1	0
Y.pri	B	3	1	0
Y.pri	C	5	2	0
Y.pri	D	7	2	0
Y.pri	G	10	2	0
Y.pri	H	12	1	0

Step 2:

Run procedure `Set_Occurrence_Level` on the above file to determine the parent/child relationship which constitutes overloading to produce:

Name	Domain	Seq#	T_Level	O_Level
X.pri	A	1	0	0
X.pri	B	2	1	1
X.pri	C	4	2	2
X.pri	D	6	2	2
X.pri	F	8	2	1
X.pri	G	9	2	1
X.pri	H	11	1	1
Y.pri	B	3	1	0
Y.pri	C	5	2	1
Y.pri	D	7	2	1
Y.pri	G	10	2	0
Y.pri	H	12	1	0

Step 3:

Run procedure `Display_Overloading` to produce the desired report:

```
Part          Domain     Level

X.pri           A          0
  X.pri         B          1
    X.pri       C          2
    X.pri       D          2
  X.pri         F          1
  X.pri         G          1
  X.pri         H          1

Y.pri           B          0
  Y.pri         C          1
  Y.pri         D          1
Y.pri           G          0

Y.pri           H          0
```

5.1.9 Notes on Report Format:

1. Indentation of one partname under another means "is overloaded by". In the above example, X.pri in domain A is overloaded by X.pri in domain B, is overloaded by X.pri in Domain C. ("Overloaded by" is an obscure way of saying "is hidden by")

2. Two occurrences of a given partname which are at the same indentation level are not visible to one-another because the domains which contain them are "cousins" in the hierarchy.

3. A blank line between two occurrences of the same partname is emphasis that the corresponding parts do not share a common ancestor even though they have the same name. Look at the occurrences of Y.pri in the above report. The Y.pri's in domains C and D share Y.pri in domain B as a common ancestor, but the occurrences of Y.pri in domains G and H do not share ancestry with any of the first three occurrences and thus are likely to contain totally unrelated components.

5.2 Geometric Knowledge Enhancements

Geometric knowledge has been added to the object assembly process (DMC - Dynamic Motion COG). This additional knowledge is in the form of new information to be associated with each part, such as joint information; and new classes of objects such as sites. This additional knowledge permits a higher, more symbolic level of interaction between the model and the user.

5.2.1 SITES

A new COG record type called a SITE has been added. It is the same as normal COG record but has a reference name 'SITE'.

5.2.1.1 Use of sites.

These sites can be used as both objects and targets of DMC commands (see CONNECT, ATTACH, DETACH, RMS, VIEW). For example, a site can be used as an eye or camera location and viewpoint. A site can also be used as a location for grasping.

5.2.1.2 Viewing sites.

Sites are displayed as individual coordinate systems (three axes). The display of sites can be selectively enabled or disabled.

5.2.2 CONNECT command.

Used with sites to join two parts, this command permits the symbolic assembling of parts. For example, the user can define a connection point on a part (using sites) and later connect that part to another part using only the part names and their corresponding connection names.

5.2.3 ATTACH / DETACH commands.

The ATTACH and DETACH commands are now installed to permit the grasping of objects by the RMS as well as by the man-model. Using site definitions to specify grasping points, the user can attach a hand to an object.

5.2.4 IRIS interface.

Many of the additions to DMC mentioned here have been interfaced to the IRIS via the VAX ethernet.

5.2.5 DESCRIPTION command.

The user can add one line descriptions to COG records. This helps to better identify a part. More extensive assembly and part descriptions are forthcoming.

5.2.6 DISPLAY file review.

The user can display previously generated display files (created by the DISPLAY module of PLAID) for reviewing purposes.

5.2.7 RMS additions and improvements.

5.2.7.1 RMS joint name list.

A user defined joint list is a parameter to RMS. The user does not need to use a command file to update RMS joints. However, more work in this area is needed.

5.2.7.2 Symbolic articulation of RMS.

RMS can be directed to go to a given site name. The user does not need to enter coordinates and attitude information explicitly, as the site is used to encapsulate this information as a name.

5.2.8 VIEW command improvements.

5.2.8.1 VIEW from a site.

A viewpoint can be specified symbolically using a site which defines an eye (or camera) location and direction of view. This eye or camera site definition is being extended to include a 'camera or eye' field of view (angle of acceptance) which will be associated with the particular site.

5.2.8.2 VIEW attachment.

The current view point can now be attached to a particular site (defined as a camera or an eye) such that whenever the site is moved, the view will automatically change to reflect the new location of the eye or camera.

5.2.8.3 VIEW tracking.

View point tracking is being examined as a way to permit the automatic tracking of an object by the currently attached view point. Whenever the object being looked at moves, the viewpoint will automatically be updated so that it will follow the object. For example, the man-model could visually track an object until the eye joint limits are reached or until the head joint limits are reached.

5.2.9 ROTATION of parts.

5.2.9.1 Rotation of parts about an arbitrary axis.

An arbitrary axis of rotation can now be defined as well as the normal X, Y or Z axes. This permits limit checking for rotations.

5.2.9.2 Limit checking of rotations.

Rotational limits can now be specified (see JOINT command) and are checked when the limit checking is enabled.

5.2.10 JOINT command.

5.2.10.1 JOINT definitions for parts.

A part can now be defined as a joint with user defined axes of rotation, orders of rotation and rotational limits. The JOINT definition can be disabled or enabled.

5.2.10.2 Extensions for the JOINT command.

This command is being extended to permit the selective enabling or disabling (by axis) of limit checking. Translational limit checking is being examined as a new extension to the joint definitions.

5.2.11 SCALE command.

Differential scaling has been implemented which permits the parameterized sizing of normalized components. For example, the normalized man-model components (body segments) can be parameterized for various statistically determined body sizes.

6. Conclusions

The development of an analogical/semantic modeler for CAD imposes certain unique requirements on the host software development environment. Simultaneous requirements exist for intensive numerical computation, data storage and retrieval, extensive graphics capabilities, transparent networking and non-numerical (symbolic) processing. These requirements are not adequately addressed by any single software development environment.

A survey of state-of-the-art programming paradigms was conducted which examined the following paradigms:

- The Procedure-Oriented Programming Paradigm
- The Object-oriented Programming Paradigm
- The Access-oriented Programming Paradigm
- The Access-oriented Programming Paradigm
- Multi-Paradigm Programming Environments

There is no best programming paradigm. Each paradigm matches well with some problem domains and poorly with others. Multi-paradigm programming systems overcome this problem by incorporating several paradigms into a single programming environment, but they have not yet reached maturity. The comprehensive multi-paradigm systems are either closed or based upon the Lisp language. Both approaches require that the large body of programs which are written in languages such as FORTRAN be scrapped or rewritten.

GMS Technology reviewed numerous products in search of an ideal programming environment. The system assembled as a vehicle for K-Base development consists of a closely-coupled network of four Digital Equipment Corporation 32-bit VAX microcomputers with a comprehensive set of software tools running under the VMS operating system. This system was chosen to assure complete compatibility with the target operating environment at NASA JSC, and to provide an operating environment for PLAID, the primary client application for K-Base.

The implementation of K-Base may be divided into two broad functional areas; non-geometric (symbolic) knowledge enhancements and geometric knowledge enhancements.

6.1 Non-geometric Knowledge.

The K-Base Symbol Management System (KB/SMS) is designed to extend the modeling capabilities of the PLAID system beyond the realm of purely geometric modeling while retaining the existing Multi-User PLAID contextual referencing mechanism. KB/SMS embodies features which aid in the maintenance of the large database of PLAID component files, target files, and display files by providing facilities for searching and reporting on parts based upon attributes of the parts.

KB/SMS provides facilities for assigning arbitrarily many attribute name/value pairs to PLAID parts. Queries of the PLAID database may

then be performed based upon the values of specified attributes. Two Multi-User PLAID search algorithms were implemented; contextual and global searches. These facilities provide the PLAID user with a powerful tool for tracking the genesis and evolution of parts stored in the database.

6.2 Geometric Knowledge.

Geometric knowledge has been added to the object assembly process (DMC - Dynamic Motion COG) running on a VAX under VMS operating system. This module was chosen because it is currently used in production work and provides a good vehicle for moving concept to application. The added geometric knowledge permits the designation of new classes of objects and permits a more symbolic utilization of those objects.

It became clear during the course of our research that added geometric information should take the form of information that would better define the role of an object. Besides the normal definition of an object as a part within an assembly, new classes of objects were added. Joint information was added to some objects to define how the object could be articulated, objects were designated as sites which are named reference points on objects, and camera information was be embedded in sites to further designate its function or role.

The classifications provide role information to better define the relationship of objects to the world. When an object is designated by name in an operation, its role information qualifies its use. If an object designated as a joint is be moved as a joint, its joint limit information controls its behavior. If a site is to be used as an eye point its camera information can control the view port. Thus the user is using the embedded information of an object by only using its name in the correct context. This additional knowledge permits a higher, more symbolic level of interaction between objects and users and a richer level of knowledge representation for more advanced reasoning tasks. The user's specifications for object manipulation are easier to describe and understand when previously defined geometric information is accessible by name.

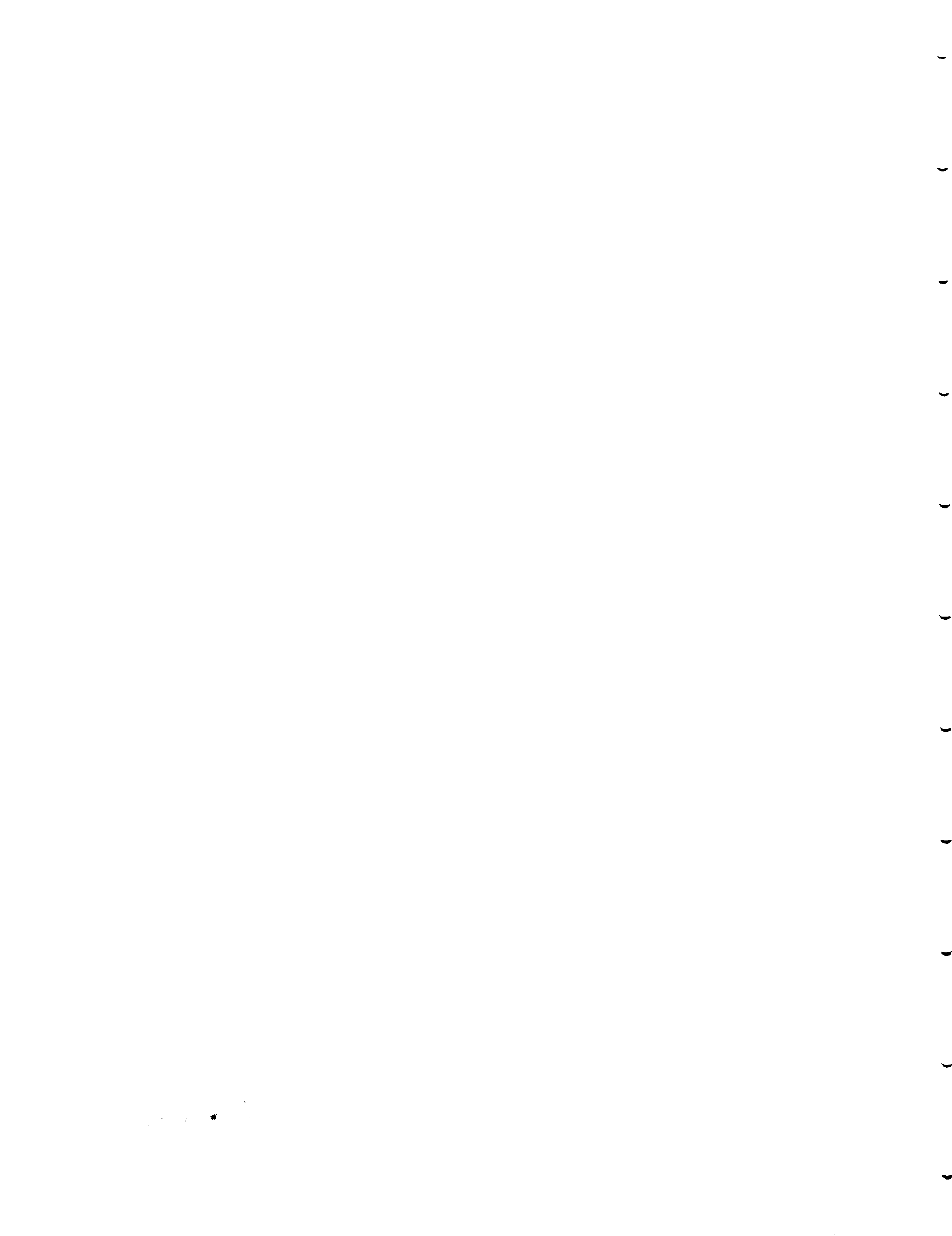
For example, the operation of reaching for an object is simplified whenever the user predefines a named site and then uses that site by name as a goal. Thus, whenever the goal is moved, the specification for reaching it is unchanged because the encapsulated information as to its location is automatically supplied whenever its name is used.

Viewing specifications are similarly simplified when a named site, which encapsulates the location, direction and camera specifications, is given the role as an eye point or camera. The eye point may move or its camera definition may change, but the user's viewing specifications will remain the same.

It is apparent that the encapsulation of geometric information as symbols improves the ease of use and the level of understanding for the user. In addition, it provides a link from the complex, less user-friendly representation of geometry to the symbolic, more user-friendly,

level. In addition, this symbolic level is more conducive to both artificial and human reasoning processes. In the complex world of CAD and CAE, the reduction of complexity means less human [illegible] areas better handled by computer and more utilization of a user's higher level reasoning.

ORIGINAL PAGE IS
OF POOR QUALITY



Appendix 1 -- Updated Multi-User Documentation

The following pages should be added to the Multi-User PLAID User's Guide at the end of Section 2.

ORIGINAL PAGE IS
OF POOR QUALITY

2.3. REPORT GENERATION COMMANDS

The report generation commands provide information on PLAID part files in entire sets of Multi-User domains. These commands provide a mechanism for listing files and retrieving information from the associated description files.

The general outline for generating a report is as follows:

1. Use the FORM/OPEN command to open the file which is to contain the report and optionally specify the description file fields which are to be reported upon. NOTE: if you wish for the output to be presented on your terminal, use a command like

FORM/OPEN TT: list-of-fields.

2. Use one or more FIND and/or DESCRIBE commands to cause output to be generated into the report file.

3. Use the FORM/CLOSE command to cause the report file to be released.

4. Use the DCL TYPE or PRINT commands to view the content of the report.

2.3.1. FORMAT_REPORT Command

FORM/OPEN filename [field_name_list]
FORM/CLOSE
FORM/SHOW

The Format Report command (FORM) causes a report destination file to be created and also allows the specification of the content of the report. The report filename is specified using the usual VMS file specifier (e.g. MYSTUFF.RPT).

The content of the report is specified as a list of description file field names.

NOTE: if a report generation command is issued when there is no report file open, the report will be directed to the terminal. Use the FORM/SHOW command to determine the status of the report file.

2.3.1.1. FORM/OPEN filename [field_name_list]

The FORM/OPEN command is specified BEFORE the report generation commands (i.e. FIND, DESCRIBE) are issued. The filename parameter specifies the name of the file into which the report will be stored. The optional field_name_list specifies which fields from the description files selected by the report generation commands will be copied into the report.

The field_name_list is an optional parameter which may be "ALL" (without the quotes), a single description file field name (without the enclosing double colons), or a list of field names with commas separating them from one another.

Parameters:

filename

the VMS filename into which the report will be stored

field_name_list (optional)

"ALL" or a comma-delimited list of field names which will copied to the output report file. "ALL" specifies that all fields of the description file will be included in the output report.

Required Privileges:

ALL (all users)

Possible Errors and Warnings:

none

Example 1:

The following example opens a report file named MY_REPORT.RPT and selects ALL fields from the description files:

```
MONITOR> FORM/OPEN  MY_REPORT.RPT  ALL
Output_Filename: MY_REPORT.RPT
Field_Name_List: all
```

Example 2:

The following example opens a report file named GLOBAL.RPT and selects the creation date, project name, and modification list fields:

```
MONITOR> FORM/OPEN GLOBAL.RPT
                              NAME,PROJECT,MOD
Output_Filename: GLOBAL.RPT
Field_Name_List: NAME,PROJECT,MOD
MONITOR> DESC/GLOBAL J*.PRI ALL
...
MONITOR> FORM/CLOSE
K-Base Format Close
MONITOR> $TYPE GLOBAL.RPT
...
```

2.3.1.2. FORM/CLOSE

The FORM/CLOSE command closes the current report output file and releases it. NOTE: Nothing can be done with the report file at the VMS/DCL level until FORM/CLOSE has been issued. This is a result of the fact that Multi-User keeps the report file open until the FORM/CLOSE is issued.

Parameters:

none

Required Privileges:

ALL (all users)

See Also:

DESC(ribe) command
FIND command

Example:

```
MONITOR> FORM/CLOSE
Kbase Format Close
```

2.3.1.3. FORM/SHOW

The FORM/SHOW command displays the current state of the report file and associated list of field names to be included in the report. If no report file is currently open, a message to that effect is displayed.

Parameters:

none

Required Privileges:

ALL (all users)

See Also:

FORM/OPEN command
FORM/CLOSE command
DESC(ribe) command
FIND command

Example:

```
MONITOR> FORM/SHOW
Report in progress:
  Output filename: GLOBAL.RPT
  Field name list: NAME,PROJECT,MOD
```

2.3.2. FIND_FILES Commands

The FIND commands are designed to assist in locating files in the Multi-User environment by performing contextual and global searches of the context.

2.3.2.1. FIND/CONTEXT part_spec

The FIND/CONTEXT command is essentially the same as the Multi-User DIR command in that it searches up the context tree from the current default domain looking for files which match the given part_spec. The significant differences between FIND/CONTEXT and DIR are:

(1) FIND/CONTEXT locates every occurrence of each file matching the part_spec, tagging the hidden occurrences as "*hidden"

(2) FIND/CONTEXT will send its output to the currently open report file if one is currently open.

The product of the FIND/CONTEXT command is an alphabetical list of all filenames in the current context which match the given part_spec. If more than one occurrence of a given filename is found in the context, each occurrence is listed in order of occurrence in the context. Each occurrence after the first is tagged with "*hidden" to emphasize that it is obscured from view in the current context by another file of the same name.

2.3.2.2. FIND/GLOBAL part_spec

The FIND/GLOBAL command searches down the context hierarchy for all files matching part_spec. This search begins in the current default domain and proceeds to each tree of domains which is attached to the default domain. The order of domain searches is identical to the order of the list of domains provided by a command of the form "LCSF <current_domain>".

The product of this command is an alphabetical listing of occurrences of all files located which match the given part_spec. If more than one file is located with a given name, then an indentation scheme is used to show the logical dependencies between files with that name. There is a logical dependence between two occurrences of a filename if one file is a descendant of the other in the context structure tree. The idea here is that two occurrences of a given part name are really not related to one another if they do not share a common contextual ancestor.

Parameters:

part_spec

a VMS filename which may include any valid wildca rd characters (e.g. FIND *.PRI).

Privileges Required:

ALL (all users)

See Also:

FIND command
DESC(ribe) command
FORM command

Example: A Contextual FIND Command

```
MONITOR> SETP LEVEL1_2
MONITOR> FIND/CONTEXT *.PRI

------LIST OF PART NAMES WITH OVERLOADING----

<PART_NAME> <DOMAIN_NAME>

X.PRI     LEVEL1_2
   X.PRI      *hidden LEVEL0
       Y.PRI      LEVEL1_2

NOTE: @ REPRESENTS FOREIGN REFERENCE
```

Example: A Global FIND Command

```
MONITOR> SETP LEVEL0
MONITOR> FIND/GLOBAL *.PRI

   -----GLOBAL LIST OF PART NAMES WITH OVERLOADING----

        <PART_NAME>     <LVL>  <DOMAIN_NAME>

        X.PRI     0: LEVEL0
          X.PRI     1: LEVEL1_0
            X.PRI     2: LEVEL1_0_0
            X.PRI     2: LEVEL1_0_1
          X.PRI     1: LEVEL1_1_1
          X.PRI     1: LEVEL1_1_0
          X.PRI     1: LEVEL1_2
        Y.PRI     0: LEVEL1_0
          Y.PRI     1: LEVEL1_0_0
          Y.PRI     1: LEVEL1_0_1
        Y.PRI     0: LEVEL1_1_1
        Y.PRI     0: L1_1_0_0
        Y.PRI     0: LEVEL1_2

        NOTE: @ REPRESENTS FOREIGN REFERENCE
```

2.3.3. DESCRIBE_FILE

The DESC command allows the user to perform K-Base data retrievals from PLAID part description files. The user may qualify data retrieval by two regular expressions: one which matches field names, and one which matches the contents of a field whose name has previously been matched.

The listing may be performed either contextually or globally. The contextual search follows the normal part-name resolution strategy, searching from the current domain toward the root of the hierarchy; the global search proceeds from the current domain toward the leaves of the hierarchy tree.

Modifiers

/CONTEXT - Performs a contextual search. (Default)
/GLOBAL - Performs a global search.

Parameters

PART_SPEC

Specification(s) for files to be retrieved. May be either a single file specification or a comma-delimited list of file specifications. Each specification should be a standard VMS file specification; wild-cards are allowed.

FIELD_NAME_SPEC (optional)

A UNIX-style regular expression matching one or more description file field names.

CONTENT_SPEC (optional)

A UNIX-style regular expression matching the contents of any of the fields matched by FIELD_NAME_SPEC.

See Also

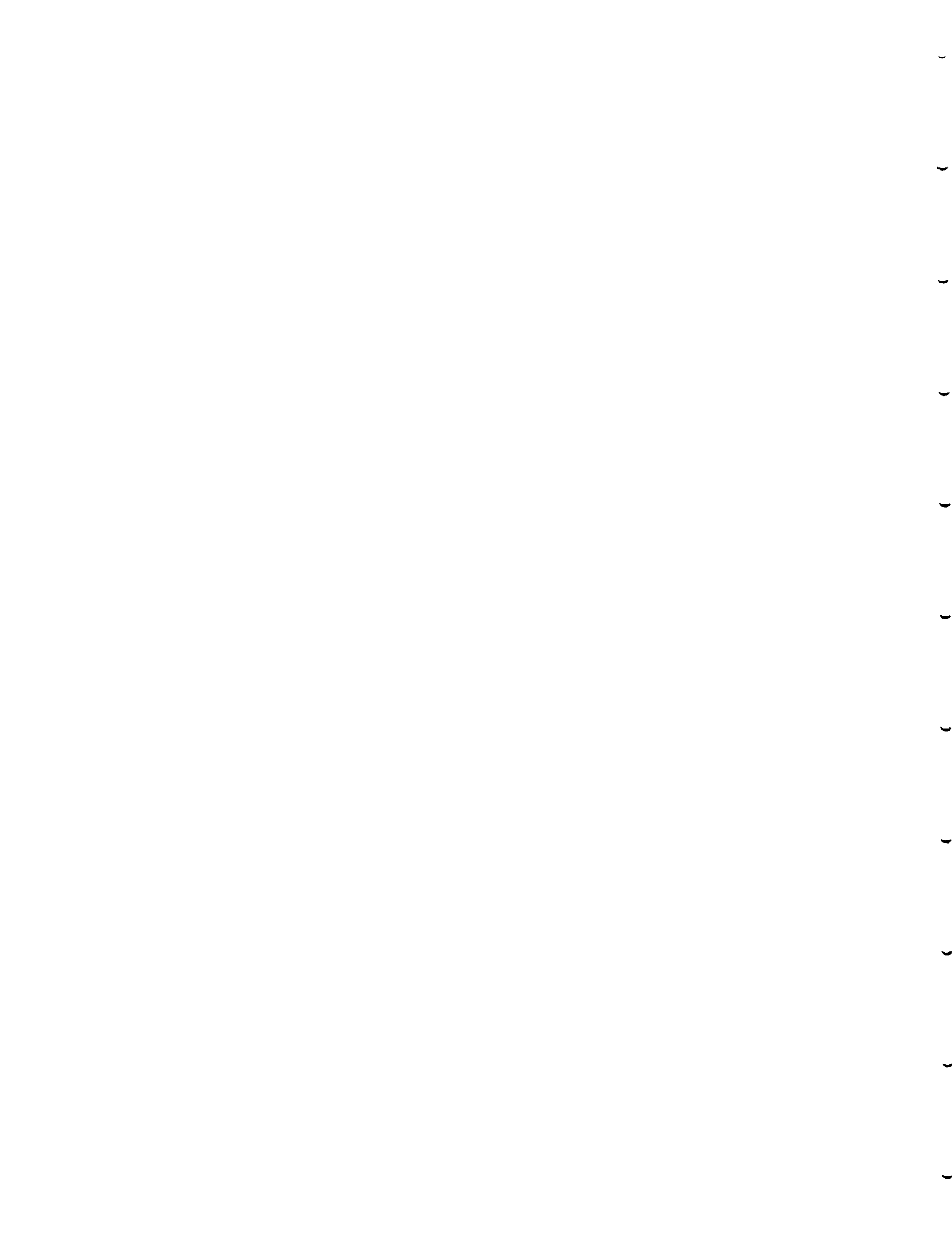
FORM command
FIND command

Examples: DESC/CONTEXT

This example generates a contextual report of all fields of all primitives named X.PRI.

```
MONITOR> FORM/OPEN  CONTEXT.RPT ALL
K-Base Format Open
MONITOR> DESC/CONTEXT X.PRI ALL
K-Base Describe Context
Part_Spec:    X.PRI
Field_Name_Spec: all
Content_Spec:
MONITOR> FORM/CLOSE
K-Base Format Close
MONITOR> $TYPE CONTEXT.RPT


   Domain: LEVEL1_0_0  Desc_name: X.PDF
  -----------------------------------------
   ::NAME:: X.PRI
   ::CREATED:: 17-APR-1989 18:15:37.78   [300,300]
   ::ACCOUNT::
   ::PROJECT:: DONALD$DUA1:[GALLAWAY.PLAID.LEVEL1_0_0]
   ::UIC:: [300,300]
   ::DESC::
   ::CONTENTS:: Rev:E  UOM: In.  Space used:   5.95%
     Object Count:   1
     Name code Description    Area Volume Ok?
     HBOX (52) Milled solid        26.742427.11373 YES
   ::RANGE::
     Cen.: 0.000000.000001.00000
     Min.:-1.00000         -1.000000.00000
     Max.: 1.000001.000002.00000
   ::VOLUME::7.11373 cu. In.
   ::AREA::         26.74242 sq. In.
   ::MOD::17-APR-1989 18:16:30.72   [300,300]
  -----------------------------------------
   Domain: LEVEL1_0  Desc_name: X.PDF
  -----------------------------------------
   ::NAME:: X.PRI
   ::CREATED:: 17-APR-1989 18:15:37.78   [300,300]
   ::ACCOUNT::
   ::PROJECT:: DONALD$DUA1:[GALLAWAY.PLAID.LEVEL1_0]
   ::UIC:: [300,300]
   ::DESC::
   ::CONTENTS:: Rev:E  UOM: In.  Space used:   5.95%
     Object Count:   1
     Name code Description    Area Volume Ok?
     HBOX (52) Milled solid        26.742427.11373 YES
   ::RANGE::
     Cen.: 0.000000.000001.00000
     Min.:-1.00000         -1.000000.00000
     Max.: 1.000001.000002.00000
   ::VOLUME::7.11373 cu. In.
   ::AREA::         26.74242 sq. In.
   ::MOD::17-APR-1989 18:16:05.07   [300,300]
  -----------------------------------------
   Domain: LEVEL0  Desc_name: X.PDF
  -----------------------------------------
   ::NAME:: X.PRI
   ::CREATED:: 17-APR-1989 18:15:37.78   [300,300]
   ::ACCOUNT::
   ::PROJECT:: DONALD$DUA1:[GALLAWAY.PLAID.LEVEL0]
   ::UIC:: [300,300]
   ::DESC::
   ::CONTENTS:: Rev:E  UOM: In.  Space used:   5.95%
     Object Count:   1
     Name code Description    Area Volume Ok?
     HBOX (52) Milled solid        26.742427.11373 YES
   ::RANGE::
     Cen.: 0.000000.000001.00000
     Min.:-1.00000         -1.000000.00000
     Max.: 1.000001.000002.00000
   ::VOLUME::7.11373 cu. In.
   ::AREA::         26.74242 sq. In.
   ::MOD::17-APR-1989 18:15:51.70   [300,300]
```



Appendix 2 -- Updated DMC Documentation

1. Updated DMC Routine Documentation.

Appendix two covers the areas of change for DMC. There are new and modified routines for accessing the workfile and part nodes. There are new and modified commands for the user.

1.1 DMC Workfile Access Routines.

The DMC program uses a workfile to store the assembly and subassembly files (PLAID files with extension 'COG') and to store the primitive object definition files (PLAID files with extension 'PRI') while a user is creating, editing or reviewing assemblies and subassemblies. These files are copied into the work file area when they are referenced by other files or when they are selected directly by the user.

At the program level, access to these work file components is done via a set of library routines (in DMC000.OLB). The following is a list of the routines and description of their function. The routines are written in Fortran so as to be compatiable with earlier PLAID modules, but they could be called from other languages via 'wrapper' routines.

1.1.1 c7init

Function:
Initialize work file access and buffer areas.

Parameters:
none

1.1.2 c7look(assy, part, ier)

Function:
Inquire in the work file for the existence of the given assembly or part. If the part name is blank then only the existence of the assembly is performed.

Parameters:

```
assy  = c*(*). Assembly name.(in)
part  = c*(*). Part name.(in)
ier   = i*2. Error code.(out)
             0 = ok
            >0 = not found
```

1.1.3 c7qpar(assy, part, ier)

Function:
Query the work file for the existence of the given part.

Parameters:
assy = c*(*). Assembly name.(in)
part = c*(*). Part name.(in)
ier = i*2. Error code.(out)
0 = Part found
18 = Part Not found

1.1.4 c7qass(assy, ier)

Function:
Query the work file for the existence of the given assembly.

Parameters:
assy = c*(*). Assembly name.(in)
ier = i*2. Error code.(out)
0 = ok
6 = Assembly not found

1.1.5 c7make(assy, part, ier)

Function:
Create an assembly or part. If the assembly does not exist then the assembly will be created before the part is created (if given).

Parameters:
assy = c*(*). Assembly name.(in)
part = c*(*). part name.(in)
ier = i*2. Error code.(out)
0 = ok
>0 = create error

1.1.6 c7read(assy, part, ier)

Function:
Read the given assembly or part from the work file into the current record buffer.

Parameters:
assy = c*(*). assembly name.(in)
part = c*(*). part name.(in)
ier = i*2. error code.(out)
0 = ok
>0 = read error

1.1.7 c7writ(assy, part, ier)

Function:
Write the current record buffer for the given assembly/part name to the work file.

Parameters:

```
assy  = c*(*). assembly name.(in)
part  = c*(*). part name.(in)
ier   = i*2. error code.(out)
          0 = ok
         >0 = write error
```

1.1.8 c7kill(assy, part, ier)

Function:

Delete the given assembly/part from the work file. If deleting an assembly, then all its parts will also be deleted.

Parameters:

```
assy  = c*(*). assembly name.(in)
part  = c*(*). part name.(in)
ier   = i*2. error code.(out)
          0 = ok
         >0 = delete error
```

1.1.9 c7maka(assy, ier)

Function:

Create assembly in the work file. The assembly will have no parts.

Parameters:

```
assy  = c*(*). Assembly name.(in)
ier   = i*2. Error code.(out)
          0 = ok
         11 = create error
```

1.1.10 c7puta(assy, ier)

Function:

Write the current record buffer to the named assembly in the work file.

Parameters:

```
assy  = c*(*). assembly name.(in)
ier   = i*2. error code.(out)
          0 = ok
         >0 = write error
```

1.1.11 c7geta(assy, ier)

Function:

Read the given assembly into the current record buffer.

Parameters:
    assy   = c*(*). assembly name.(in)
    ier   = i*2. error code.(out)
        0 = ok
        >0 = read error

1.1.12 c7dela(assy, ier)

Function:
Delete the named assembly from the work file. The assembly record is not removed from the file but is flagged as deleted. The parts records, however, are removed from the file.

Parameters:
    assy   = c*(*). assembly name.(in)
    ier   = i*2. error code.(out)
        0 = ok
        >0 = delete error

1.1.13 c7makp(assy, part, ier)

Function:
Create the named part in the work file. The record buffer for this part is assumed to be properly initialized.

Parameters:
    assy   = c*(*). assembly name.(in)
    part   = c*(*). part name.(in)
    ier   = i*2. error code.(out)
        0 = ok
        >0 = create error
            20 = part already exists

1.1.14 c7getp(assy, part, ier)

Function:
Read the given part record into the current record buffer.

Parameters:
    assy   = c*(*). assembly name.(in)
    part   = c*(*). part name.(in)
    ier   = i*2. error code.(out)
        0 = ok
        >0 = read error

1.1.15 c7putp(assy, part, ier)

Function:
Write the current record buffer into the name part record in the work file.

Parameters:
assy = c*(*). assembly name.(in)
part = c*(*). part name.(in)
ier = i*2. error code.(out)
0 = ok
>0 = write error

1.1.16 c7delp(assy, part, ier)

Function:
Delete the named part from the work file. The part record in the work file is removed from the file and not flagged as with the assembly records.

Parameters:
assy = c*(*). assembly name.(in)
part = c*(*). part name.(in)
ier = i*2. error code.(out)
0 = ok
>0 = delete error

1.1.17 c7nxta(assy, ier)

Function:
Read the next assembly in sequence from the work file.

Parameters:
assy = c*(*). assembly name.(in)
ier = i*2. error code.(out)
0 = ok
>0 = read error
-1 = end of file

1.1.18 c7nxtp(assy, part, ier)

Function:
Read the next part under the given assembly. Routine returns error conditions for end of assembly, as well as, end of file.

Parameters:
assy = c*(*). assembly name.(in)
part = c*(*). part name.(in)
ier = i*2. error code.(out)
0 = ok
>0 = read error
-1 = end of file
-2 = end of assembly

1.1.19 c7next(assy, part, ier)

Function:
Read the next assembly or part from the work file.

Parameters:
    assy = c*(*). assembly name.(in)
    part = c*(*). part name.(in)
    ier = i*2. error code.(out)
        0 = ok
        >0 = read error
        -1 = end of file
        -2 = end of assembly

1.1.20 c7pbuf(record, l)

Function:
Store the given record area into the current record buffer for the work file. If the current record buffer is larger than the specifed length the input record, the record buffer is zeroed filled. Current record buffer in 512 bytes (256 words).

Parameters:
    record = i*2. Record area to store.(in)
    l = i*2. Length of record area in words.(in)

1.1.21 c7gbuf(record, l)

Function:
Copy the contents of the current record bufferinto the given record area up to the given length. If the record buffer is smaller than the receiving buffer then the receiving buffer is zeroed filled.

Parameters:
    record(l) = i*2. Receiving record area.(out)
    l = i*2. length of record area in words.(in)

1.1.22 c7stat(istat)

Function:
Returns the current status field in the current record buffer. The status currently has meaning for assembly header records only.

Parameters:
    istat = i*2.Status.(out)
        0 = no change
        1 = modified
        2 = deleted

1.1.23 c7flag(assy, istat)

Function:
Update the current status of the given assembly.

Parameters:
assy = c*(*). Assembly name.(in)
istat = i*2. New status.(in)
0 = not modified
1 = assembly modified
2 = assembly deleted

1.1.24 c7clea

Function:
Delete all the current assemblies and parts in the work file. Only assembly and part records are effective. All the records are removed from the work file (not flagged).

Parameters:
none

1.1.25 c7file(name, ier)

Function:
Create an indexed file for storing processed primitives referenced by the current part structure tree.

Parameters:
name = c*(*). File name.(in)
ier = i*2. Error code.(out)
0 = ok
>0 = Fortran error code

1.1.26 c7clwf

Function:
Close the current work file.

Parameters:
none

1.1.27 c7keys(assy, part, key)

Function:
Given the assembly and part names, construct a valid record key for the work file. Note: extensions on the assembly name, such as '.COG' are not passed onto the key name.

Parameters:
assy = c*(*). assembly name.(in)
part = c*(*). part name.(in)
key = c*(*). record key.(out)

1.1.28 c7noex(assy, name, ext)

Function:
Strip the extension, if any, from the given assembly name and place results into a new new name.

Parameters:
assy = c*(*). assembly name.(in)
name = c*(*). New assembly name.(out)
ext = c*(*). Extension to strip.(in)

1.1.29 c7mkex(assy, name, ext)

Function:
Add the extension, if any, to the given assembly name and place results into a new new name.

Parameters:
assy = c*(*). assembly name.(in)
name = c*(*). New assembly name.(out)
ext = c*(*). Extension to add.(in)

1.1.30 c7gcog(name)

Function:
Get the cog file name or assembly name from current record buffer. The extension '.COG' is implied, so it must be appended before using as a file name.

Parameters:
name = c*(*). Assembly or cog file name.(out)

1.1.31 c7gpar(part)

Function:
Get the current part name from the current record buffer.

Parameters:
part = c*(*). Name of part.(out)

1.1.32 c7ppar(part)

Function:
Replace the current part name in the record buffer with the given part name.

Parameters:
part = c*(*). New part name.(in)

1.1.33 c7vref(ref, type, ier)

Function:
Verify the given reference name as a valid reference and return the type of reference if so.

Parameters:
ref = c(*). Reference name to check.(in)
type = i*2. type of reference, if valid.(out)
1 = primitive
2 = assembly
3 = site
ier = i*2. Error code.(out)
0 = ok
26 = invalid reference

1.1.34 c7gref(ref)

Function:
Get the reference name from the current record buffer. Note: the extension for the reference is preserved so as to permit type checking.

Parameters:
ref = c*(*). reference name.(out)

1.1.35 c7pref(ref)

Function:
Replace the current reference name in the record buffer with the new reference name.

Parameters:
ref = c*(*). New reference name.(in)

1.1.36 c7gmat(m)

Function:
Get the local transformation from the current record buffer. The matrix is of the form:

m(3,5)

where
m(1,1) thru m(3,3) is rotation matrix,
m(1,4) thru m(3,4) is scaling
m(1,5) thru m(3,5) is translation

Parameters:
m(3,5) = r*4. Transformation matrix.(out)

1.1.37 c7pmat(m)

Function:
Replace the local transformation matrix in the current record buffer with the given matrix. current record buffer. The matrix is of the form:

m(3,5)

where
m(1,1) thru m(3,3) is rotation matrix,
m(1,4) thru m(3,4) is scaling
m(1,5) thru m(3,5) is translation

Parameters:
m(3,5) = r*4. New transformation matrix.(in)

1.1.38 c7grot(r)

Function:
Get rotation matrix component of part transformation from the current record buffer.

Parameters:
r(3,3) = r*4. Rotation matrix.(out)

1.1.39 c7prot(r)

Function:
Put rotation matrix component of part transformation into the current record buffer.

Parameters:
r(3,3) = r*4. Rotation matrix.(in)

1.1.40 c7gtra(t)

Function:
Get translation component of transformation from current record buffer.

Parameters:
t(3) = r*4. Translation vector.(out)

1.1.41 c7ptra(t)

Function:
Put translation component of transformation into current record buffer.

Parameters:
t(3) = r*4. Translation vector.(in)

1.1.42 c7gsca(s)

Function:
Get scaling component of transformation from current record buffer.

Parameters:
s = r*4. scaling.(out)

1.1.43 c7psca(s)

Function:
Put scaling component of transformation into current record buffer.

Parameters:
s(3) = r*4. Scaling.(in)

1.1.44 c7gdsca(s)

Function:
Get differential scaling from current record buffer.

Parameters:
s = r*4. scaling.(out)

1.1.45 c7pdsca(s)

Function:
Put differential scaling into current record buffer.

Parameters:
s(3) = r*4. Scaling.(in)

1.1.46 c7gcol(color)

Function:
Get the color name from the current record buffer. The color name is assumed to be a valid name taken from the current color name file in the Plaid system.

Parameters:
color = c*(*). Color name.(out)

1.1.47 c7pcol(color)

Function:
Replace the color name in the current record buffer with the

given color name. The color name is assumed to have been validated.

Parameters:
 color = c*(*). New color name.(in)

1.1.48 c7pdesc(desc)

Function:
 Put part description in current record buffer.

Parameters:
 desc = c*(*). Description.(in)

1.1.49 c7gdesc(desc)

Function:
 Get the part description from the current part record buffer.

Parameters:
 desc = c*(*). Description.(out)

1.1.50 c7gtyp(type)

Function:
 Get the reference type of the reference name in the current record buffer. Currently there are valid types.

 0 = neither
 1 = primitive
 2 = assembly
 3 = site point
 4 = eye point
 5 = general reference point

Parameters:
 type = i*2. type code.(out)

1.1.51 c7newp(part)

Function:
 Initialize the current record buffer to be new part record. The record buffer is zeroed out, then updated with the given part name, a null reference (spaces), the identity transformation and the color 'white'.

Parameters:
 part = c(*). New part name.(in)

1.1.52 c7pad(s, l)

Function:
Replace all characters below 32 and above 127 ascii with spaces.

Parameters:
s(l) = c(*) name to pad.(in/out)
l = i*2. length of name.(in)

1.1.53 c7sbuf

Function:
Save the current record buffer into an alternate record buffer area.

Parameters:
none

1.1.54 c7rbuf

Function:
Replace the current record buffer with the record buffer previous saved into the alternate record buffer area. (see c7sbuf)

Parameters:
none

1.1.55 c7gclr(scolor)

Function:
Get the color name from the color stack. The color name is assumed to be a valid name taken from the current color name file in the Plaid system.

Parameters:
scolor= c*(*). Color name.(out)

1.1.56 c7pcam(fov, aspect)

Function:
Update the field of view and aspect ratio of the camera definition. The field of view of the camera definition is the angle in degrees between the vertical frame size and the focal point. The aspect ratio is (horizontal frame size) / (vertical frame size).

Parameters:
fov = r*4. Field of view in degrees.(in)
aspect = r*4. Aspect ratio.(in)
Horz / Vertical

1.1.57 c7gcam(fov, aspect)

Function:
Get the current camera definitions. The field of view of the camera definition is the angle in degrees between the vertical frame size and the focal point. The aspect ratio is (horizontal frame size) / (vertical frame size). If the camera definition was not user defined, the default values will be returned.

Default:
fov = 50.0 degrees
aspect = 1.0

Parameters:
fov = r*4. Field of view in degrees.(out)
aspect = r*4. Aspect ratio.(out)
Horz / Vertical

1.2 DMC Assmbly Node Access Routines.

Assemblies and subassemblies are a collection of nodes which describe objects and their relationships with other objects. At the program level, access to the components in a node is done via a set of library routines from DMC000.OLB. These routines are written in Fortran to be compatiable with older PLAID modules, but can be callable from other langauges via a set of 'wrapper routines'.

The node is stored as 512 byte (fixed length) record. These routines access this record via a record buffer which is assumed to contain the currently selected part (see DMC work file access routines). The routines do not perform any input or output to a file, they change only the record buffer. It was intended that the routines hide the details of the record buffer description.

1.2.1 j7init(type)

Function:
Initialize the joint definition in the part record buffer.

Parameters:
type = i*2. Sets the joint type field.(in)
0 = disables joint
1 = joint is enabled

1.2.2 j7iaor()

Function:
Initializes the axes of rotation to

#1 = <0,0,0> <1,0,0> x axis
#2 = <0,0,0> <0,1,0> y axis
#3 = <0,0,0> <0,0,1> z axis

in the current part record buffer.

Parameters:
none

1.2.3 j7pcode(code)

Function:
Put the given joint type code in current record buffer. The type code is used enable/disable joint information. Will be later used to provided a classification of joint information.

Parameters:
code = i*2. Type code.(in)
0 = joint information not initialized (disabled)
1 = joint is enabled and joint information is valid.

1.2.4 j7gcode(code)

Function:
Get the joint type code from current record buffer.

Parameters:
code = i*2. Joint type code.(out)
0 = joint info not initialized
1 = has joint information

1.2.5 j7prlc(ano, switch)

Function:
Turn rotation limit checking on or off.

Parameters:
ano = i*2. Axis id number.(in)
switch = i*2. Switch.(in)
0 = limit checking off
1 = limit checking on

1.2.6 j7grlc(ano, switch)

Function:
Get the rotation limit checking switch.

Parameters:
ano = i*2. Axis id number.(in)
switch = i*2. Switch.(out)
0 = limit checking off
1 = limit checking on

1.2.7 j7pooc(switch)

Function:
Turn order of rotation checking on or off.

Parameters:
switch = i*2. Switch.(in)
0 = order checking off
1 = order checking on

1.2.8 j7gooc(switch)

Function:
Get the order of rotation checking switch.

Parameters:
switch = i*2. Switch.(out)
0 = order checking off
1 = order checking on

1.2.9 j7ioor()

Function:
Initialize the order of rotation field in the current part buffer. The order is set to 1, 2 and 3 or x, y and z.

Parameters:
none

1.2.10 j7poor(order)

Function:
Put the order of rotation into the current part record buffer. For example, the order is specified in the array parameter as

order(1) = 3
order(2) = 1
order(3) = 2

which corresponds to the order of rotation Z, X and Y.

Parameters:
order(3) = i*2. Order array.(in)

1.2.11 j7goor(order)

Function:
Get the order of rotation from the current part record buffer.

Parameters:
order(3) = i*2. Order of rotation.(out)

1.2.12 j7paor(number, v1, v2)

Function:
Put the axis of rotation in the current part record buffer. The axis of rotation is identified by an axis id number. The maximum number of axes is 3.

Parameters:
number = i*2. Axis id number.(in)
1, 2 or 3
v1(3) = r*4. Origin of axis.(in)
v2(3) = r*4. End point of axis.(in)

1.2.13 j7gaor(number, v1, v2)

Function:
Get the specified axis of rotation from the current part record buffer.

Parameters:
number = i*2. Axis id number.(in)
1, 2 or 3
v1(3) = r*4. Origin of axis.(out)
v2(3) = r*4. End point of axis.(out)

1.2.14 j7purl(limit)

Function:
Put the upper rotation limits for x,y,z or the three axes of rotation into the current part record buffer. The limits are in degrees.

Parameters:
limit(3) = r*4. Upper rotation limits (deg).(in)

1.2.15 j7gurl(limit)

Function:
Get the upper rotation limits in degrees from current part record buffer.

Parameters:
limit(3) = r*4. Upper rotation limits (deg).(out)

1.2.16 j7plrl(limit)

Function:
Get the lower rotation limits in degrees from the current part record buffer.

Parameters:
limit(3) = r*4. Lower rotation limits (deg).(in)

1.2.17 j7glrl(limit)

Function:
Get the lower rotation limits in degrees from current part record buffer.

Parameters:
limit(3) = r*4. Lower rotation limits (deg).(out)

1.2.18 j7prot(angles)

Function:
Put the rotation angle accumulations in degrees into the current part record buffer.

Parameters:
angles(3) = r*4. Rotation angle accumulations.(in)

1.2.19 j7grot(angles)

Function:
Get the rotation angle accumulations in degrees from the current part record buffer.

Parameters:
angles(3) = r*4. Rotation angle accumulations.(out)

1.2.20 j7aorm(axis, angle, r)

Function:
Compute the rotation matrix using the given axis id number and the given angle of rotation.

Parameters:
axis = i*2. Axis id number.(in)
angle = r*4. Angle of rotation in degrees.(in)
r(3,5) = r*4. Rotation matrix.(out)

1.2.21 j7rlim(ano, angle, diff, ier)

Function:
Check rotation limits.

Parameters:
ano = i*2. Id number of axis.(in)
angle = r*4. Angle to check in degrees.(in)
diff = r*4. Angle change.(out)
Set to upper or lower limit
when angle exceeds them.
ier = i*2. Error code.(out)
0 = ok
69 = upper limit exceeded
70 = lower limit exceeded

1.2.22 n7pshow(code)

Function:
Switch the "show site" on/off. If switch is off site will not be displayed. The default is off.

Parameters:
switch = i*2. Switch.(in)
0 = do not display site.
1 = display site.

1.2.23 n7gshow(switch)

Function:
Get the "show site" switch.

Parameters:
switch = i*2. Switch.(out)
0 = do not display site
1 = display site

1.2.24 n7pdiff(code)

Function:
Switch the "show site" on/off. If switch is off site will not be displayed. The default is off.

Parameters:
switch = i*2. Switch.(in)
0 = disable differential scaling
1 = enable differential scaling

1.2.25 n7gdiff(switch)

Function:
Get differential scaling switch.

Parameters:
switch = i*2. Switch.(out)
0 = off
1 = on

1.2.26 n7pcam(code)

Function:
Flag node as having a camera or eye definition defined by the user.

Parameters:
switch = i*2. Switch.(in)
0 = no user defined camera definition
1 = user defined camera definition

1.2.27 n7gcam(switch)

Function:
Get camera definition flag. If the flag is set then camera definition is user defined.

Parameters:
switch = i*2. Switch.(out)
0 = no user defined definition
1 = user defined definition

1.2.28 n7pmodify(switch)

Function:
Put the value of the modified/created switch into part record buffer.

Parameters:
switch = i*2. Switch value.(in)
0 = not modified/created
1 = modified/created

1.2.29 n7gmodify(switch)

Function:
Get the value of the modified/created switch from part record buffer.

Parameters:
switch = i*2. Switch value.(out)
0 = not modified/created
1 = modified/created

2. Transformation Operations.

The connecting, attaching and detaching of objects within DMC requires the computation of a rigid body transformation for one or both the objects involved (operations are pairwise). Each part can be located anywhere in part structure tree and must have a common root.

Given these conditions, each part in the tree has three transformations associated with it; a local transformation (stored with the part), a global transformation, computed from the transformations of parts up the tree and a composite transformation, computed from the local and global transformations. The transformation operations of connecting, attaching and detaching use these three transformations.

2.1 Affine or rigid body transformations.

The components of an affine transformation system are

R = rotation matrix for attitude. (3x3)
t = translation vector for positioning. (1x3)
s = scaling for size. (scalar)

To transform a point

$$p = [x\ y\ z] \text{ to } p' = [x'\ y'\ z']$$

using an affine transformation

$$p' = p * R * s + t$$

To concatenate affine transformations, let **G** and **L** be affine transformations such that **L** is transformed by **G**

$$(R_L * s_L + t_L) * R_G * s_G + t_G$$

Removing parenthesis

$$R_L * R_G * s_L * s_G + t_L * R_G * s_G + t_G$$

then

$$R_C = R_L * R_G$$

$$t_C = t_L * R_G * s_G + t_G$$

$$s_C = s_L * s_G$$

form the components of a new composite affine transformation **C** which is the concatentation of **G** and **L**

2.2 Connection operation.

Let site **A** be defined by the affine transformation

$R_A * s_A + t_A$

and site **B** be defined by the affine transformation

$R_B * s_B + t_B$

To connect site **A** to site **B**, a new transformation must be computed which will transform site **A** to site **B** (a three point to three point transformation).

1. Undo the affine transformation currently defined by site **A** using its inverse

$R^{-1}_A = R_{TA}$ (transpose of RA)

$t_A \quad = -t_A$

$s^{-1}_A = 1 / s_A$

The inverse affine transformation of a point **p'** at site **A** is

$p \quad = (p' - t_A) * R^{-1}_A * s^{-1}_A$

2. Apply to site **A** the transformation currently defined by site **B**. The transformation of a point **p** at site **A** to site **B** is

$p' \quad = p * R_B * s_B + t_B$

3. The composite transformation for connecting site **A** to site **B** is derived from

$[(p - t_A) * R^{-1}_A * s^{-1}_A] * R_B * s_B + t_B$

$[p * R^{-1}_A * s^{-1}_A - t_A * R^{-1}_A * s^{-1}_A] * R_B * s_B + t_B$

$p * R^{-1}_A * s^{-1}_A * R_B * s_B - t_A * R^{-1}_A * s^{-1}_A * R_B * s_B + t_B$

where

$R \quad = R^{-1}_A * R_B$

$t = -t_A * R^{-1}_A * s^{-1}_A * R_B * s_B + t_B$

$s = s^{-1}_A * s_B$

are the elements of the transformation for site **A** such that site **A** is connected to site **B**.

2.3 Detach Operation.

The detach operation permits removal of part **A** from its current assembly sub-tree and attachment to the sub-tree of another assembly. The detach operation is functionally the same as the attach operation, but the objects of the operation are different. The part **A** is detached to an assembly **B** rather than to part within assembly **B**.

Given

$R_A * s_A + t_A$ = transformation of part **A**.

$R_B * s_B + t_B$ = transformation of assembly **B**.

then

$$R_C = R_A * R^{-1}_B * s_A * s^{-1}_B$$

$$t_C = (t_A - t_b) * R^{-1}_B$$

$$s_C = s_A * s^{-1}_B$$

are the components of the new affine transformation for part **A** when included as a part within the assembly **B**.

2.4 Attach Operation.

The attach operation permits the joining of part **A** at site **A** with part **B** at site **B**. The attach operation requires that the original attitude and position of part **A** must be preserved at the moment of attachment to part **B**; but the transformation of part **A** will, however, be defined relative to the global or assembly transformation for part **B**. This is in contrast to the connect operation which permits modification of the attitude and position to achieve connection of part **A** at site **A** to site **B**. The attach operation is the more general form of the detach operation.

Given

$R_A * s_A + t_A$ = global transformation for part **A**.

$R_{SA} * s_{SA} + t_{SA}$ = global transformation for site on part **A**.

$R_B * s_B + t_B$ = global transformation for part **B**.

$R_{LB} * s_{LB} + t_{LB}$ = local transformation for site on part **B**.

A global transformation is defined in the world coordinate system of the root assembly referencing both parts **A** and **B**. A local transformation is defined relative to the coordinate system of the assembly containing the part or site.

Then

$$R_C = R_A * R^{-1}_B$$

$$t_C = -t_{SA} * R_A * R^{-1}_B * s_{SA} * s^{-1}_B + t_{LB}$$

$$s_C = s_{SA} * s^{-1}_B$$

are the components of the new affine transformation for part **A** relative to the assembly for part **B**, such that site **A** is attached to site **B**.

3. New and Updated User Commands for DMC.

User commands for DMC have been updated and several new commands have been added to permit the utilization of new geometric and symbolic constructs. The user may find these commands via the online help facility of DMC (the HELP command) or by issuing the command followed by a '?'.

3.1 ATTACH

The ATTACH command causes the specified part to become a part in the assembly at the specified site. The command assumes that both part and site are part of some larger assembly selected with the OPEN command. After the command has successfully executed, the site name becomes the name of the attached part and the site reference is replaced with the reference of the attached part. The attached part's original position in the assembly tree becomes a site with the original part's name. Both part and site must already exist. (see DETACH command)

Form:

ATTACH part site1 site2

part The part to be attached.

site1 Attachment site on part to mate with site2

site2 The destination site to mate with site1

3.1.1 Example

Assume the following part tree PLBAY.COG

PLBAY.LOAD with reference LOAD.COG

PLBAY.RMS with reference RMS.COG

LOAD.S1 with reference SITE
(this part is a site definition)

LOAD.S2 with reference LOAD.PRI

within RMS.COG part tree is the part

J7.ENDEFF with reference SITE
(this part is a site definition)

To attach PLBAY.LOAD to the end effector site J7.ENDEFF on the RMS, enter the command

ATTACH PLBAY.LOAD LOAD.S1 J7.ENDEFF

If successful, the part J7.LOAD will be created using the part definition PLBAY.LOAD. The part PLBAY.LOAD within PLBAY will become a site definition (its reference will be changed from LOAD.COG to SITE). The part J7.ENDEFF will remain the same. Note, if the part J7.LOAD already exists, then it will deleted then recreated.

3.2 DETACH

The DETACH command permits the removal of a part from its current assembly and attached to another assembly. If the receiving assembly is not specified, it will be placed in currently opened assembly. The DETACH command is functional the same as the ATTACH command, except the objects of the operation are different (see ATTACH command).

Form:

DETACH part [assembly]

part The part to be detached.

assembly The assembly to be attached to.(optional)

3.2.1 Example

Usage:

DETACH J7.LOAD BASE

Will cause the part named J7.LOAD to be detached from the assembly J7 and become the part BASE.LOAD in the assembly BASE. J7.LOAD in the assembly J7 will be redefined as a site.

3.3 DISPLAY

The DISPLAY command permits the reviewing of display files created in the DISPLAY module. The command has a set of options to control aspects of the display. The valid options are ZOOM, STATUS, KEEP, DASH, NODRAW and CONFLICT. Up to four optionscan selected at the same time.

Form:

DISPLAY file [option1..4] [scale]

file The name of a display file.
The implied extension is '.DSP'.

option1..4 One to four of the display options.

scale Optional scale factor.

3.3.1 options

The options control display file presentation. Up to three options can be selected at the same time for a given command line.

ZOOM permits the definition of a zoom area prior to drawing a display file.

KEEP disables the clearing of the screen prior to drawing a display file.

STATUS permits the output of the status information about the display file.

DASH permits the display of hidden lines as dashed lines in hidden line display files.

CONFLICT enables the displaying of conflict points in hidden line display files.

NODRAW disables drawing of the display file and outputs status information only.

3.3.2 Example

Usage:

DISPLAY SWITCH STATUS ZOOM

The display file SWITCH.DSP will be drawn using a zoom area definition. The status information will also be output.

3.4 CLEAR

The CLEAR command is used to initialize all or portions of the transformation associated with the named part or with the part currently being edited. It can also be used to initialize the joint information for a part. In addition, the reference and color names associated with the part can also be cleared.

Form:

CLEAR [part] option

part The name of the part to be cleared.

option The keywords ROT, TRA, SCA, ALL, REF
or COL

3.4.1 ALL

The ALL keyword causes rotation angles to be set to zero, the translation values to be set to zero and the scale to be set to one.

3.4.2 ROTATE

The ROT keyword causes the rotation angles for the named part to be set to zero. All three angles for the three axes are cleared. The joint angle accumulators are also set to zero (see JOINT command).

3.4.3 TRANSLATE

The TRA keyword causes the translation values for the named part to be set to zero. All three translation values for the three axes are cleared.

3.4.4 SCALE

The SCA keyword causes the scale value for the named part to be set to one.

3.4.5 URL

The URL keyword causes the upper rotation limit values to be set to zero (see JOINT command).

3.4.6 LRL

The LRL keyword causes the lower rotation limit values to be set to zero (see JOINT command).

3.4.7 AXIS

The AXIS keyword causes the user defined axes of rotation to be set to their initialized state (see JOINT command).

Axis #1 endpoints are set to (0,0,0) and (1,0,0) corresponding to the X axis.

Axis #2 endpoints are set to (0,0,0) and (0,1,0) corresponding to the Y axis.

Axis #3 endpoints are set to (0,0,0) and (0,0,1) corresponding to the Z axis.

3.4.8 ORDER

The ORDER keyword causes the order of rotation to set to its initialized state of 1, 2 and 3 (see JOINT command).

3.4.9 REFERENCE

The REF keyword causes the reference name for part to set to spaces.

3.4.10 COLOR

The COL keyword causes the color name for the selected part to be set to spaces. A color name set to spaces causes the selected part to use any previously used color name for the part's default color. The global default color for all parts is WHITE.

3.5 CONNECT

The CONNECT command is used to mate the specified part and site to another part and site. The translation and rotation components to accomplish mating are computed using the site definitions (see SITE command). Note that only the transformation of the first named part is updated. The second part is assumed to be the fixed part.

Form:

CONNECT part-a site-a part-b site-b

part-a The name of the part to be attached.

site-a The name of the site on part-a.

part-b The name of the part to be attached to.

site-b The name of the site on part-b.

3.5.1 Example

The following sequence of commands are example session using the CONNECT command.

```
* define assembly a1 with two parts P1 and P2
DEFINE A1.P1 A2.COG
DEFINE A1.P2 A3.COG
* define assembly a2
DEFINE A2.P1 PART2.PRI
* define site on assembly a2
SITE A2.P2
* orient site
TRANSLATE A2.P2 -1 0 0
ROTATE A2.P2 Z 45
* define assembly a3
DEFINE A3.P1 PART3.PRI
* define site on assembly a3
SITE A3.P2
* orient site
TRANSLATE A3.P2 10 3 12
ROTATE A3.P2 Y -90
* mate part p2 to p1 in assembly a1
PARTA=A1.P2
SITEA=A3.P2
PARTB=A1.P1
SITEB=A2.P2
CONNECT PARTA SITEA PARTB SITEB
* parta now has a new transformation mating it
to partb
```

3.6 DESCRIPTION

The DESCRIPTION command has two forms. The first is used to add a description to the given part. The second form is used to add a description to the description file corresponding the given Plaid data file; or to edit all fields of the description file by spawning a VAX edit session if the description argument is not given.

Form 1:

DES [part_name] ["description"]

part_name The name of the part to be described.

description The new description in double quotes for the named part. Max 78 char.

Form 2:

DES option file ["description"]

option The keywords PRIMITIVE, COG, TARGET
or DISPLAY to indicate the Plaid data
file type.

file The name of a Plaid data file.

description Optional description of Plaid
data file in double quotes.
Max 78 char.

If not given a VAX edit session will be spawned to edit the description file.

3.6.1 Example

Form 1:

DES PANEL.SWITCH "Panel light control switch"

EDIT PANEL.SWITCH

DES "Panel light control switch"

Form 2:

DES TARGET XYZ "This is my target file"

This command will replace or add the given description to the description file XYZ.TDF which is associated with the target file XYZ.TAR.

DES TARGET XYZ

This command will cause a VAX edit session to be spawned which will allow editing of the description file XYZ.TDF which is associated with the target file XYZ.TAR.

3.7 JOINT

The JOINT command is used to set the joint information for the specified part. The joint information consists of upper and lower rotation limits, order of rotation specification, joint angles accumulators and user defined axes of rotation.

Form:

JOINT part option [arguments]

part Name of the part to receive joint
information.

option A keyword specifying which joint
information fields are to updated.

args Variable length list of values to use for
updating depending on the option.

3.7.1 INIT

The INIT keyword causes the named part's joint information to be initialized.

Example:

JOINT assy.part INIT

3.7.2 ENABLE

The ENABLE keyword causes joint rotation limit checking to be turned on. If no axis number is given then all axes will be enabled.

Example:

JOINT assy.part ENABLE axis_no

3.7.3 DISABLE

The DISABLE keyword causes joint rotation limit checking to be turned off. If no axis number is given then all axes will be disabled.

Example:

JOINT assy.part DISABLE axis_no

3.7.4 STATUS

The STATUS keyword causes the named part's joint information to be typed on the console.

3.7.5 URL

The keyword URL causes the upper rotation limits of the named part to be set to the given angles. The angles are assumed to be in degrees.

Example:

JOINT assy.part URL x_angle y_angle z_angle

3.7.6 LRL

The keyword LRL causes the lower rotation limits of the named part to be set to the given angles. The angles are assumed to be in degrees.

Example:

JOINT assy.part LRL x_angle y_angle z_angle

3.7.7 ORDER

The keyword ORDER is used to define the order in which rotations are to be applied. The default order is 123. This order information does not currently affect the rotations perform with the ROT command as these operations can be performed in any order. Defining an order of rotation causes the rotational order flag to be set (see DISABLE and ENABLE)

Example:

JOINT assy.part ORDER XYZ

or

JOINT assy.part ORDER 1 2 3

3.7.8 DISABLE

The DISABLE keyword causes the rotational order flag to be turned off (set to zero). The order of rotation is unchanged, but order of rotation is not imposed.

3.7.9 ENABLE

The ENABLE keyword causes the rotational order flag to be set (set to one). The order of rotation currently defined is in effect.

3.7.10 ROTATE

The ROTATE keyword is used to set the joint angle accumulators to the given set of angles. The angles are assumed to be in degrees. These accumulators are used for comparisons to the joint angle limits. These joint angle accumulators are updated whenever a rotation about a user defined axis occurs (see JOINT name AXIS command form).

Example:

JOINT assy.part ROT x_angle y_angle z_angle

3.7.11 AXIS

The AXIS keyword is used to define an axis of rotation (arbitrarily placed in space). There can be up to three axes defined. The angles applied to these axes are accumulated in the joint angle accumulators for joint limit comparison if enabled.

Example:

JOINT assy.part AXIS number x,y,z x,y,z

number The id number of the axis.

x,y,z The first endpoint or origin of axis.

x,y,z The second endpoint of axis.

3.8 RMS

The RMS command provides an interface to the RMS joint angle computation routines. If the user has components currently defined in the DMC work file which are also currently defined in the RMS joint definition list (see RMS DEFINE), those parts can be optionally updated with joint angles computed by RMS. (see RMS module documents for details on RMS and joint angles). The command hasseveral options and forms depending on the keyword used. The IRIS interface can take a slightly different form of the RMS command. (see IRIS-form)

3.8.1 IRIS-form

The IRIS interfaces with the RMS in a different manner to permit utilization of hardware features available on it.

Form:

RMS a1 a2 a3 a4 a5 a6 [step]

a1..a6 The joint angles in degrees.

step The number of steps for each joint angle to reach the given location and attitude.

3.8.1.1 Example

The IRIS form of the RMS command makes the following assumptions

o DMC is in IRIS direct mode.

o The parts of the arm and their respective axes are named

J1.R4 on y

J2.R6 on x

J3.R8 on x

J4.R10 on x

J5.R12 on y

J6.R14 on z

The command has optional step argument to provide movement in small increments.

To get a series of RMS actions, a command file of the form

RMS ITF x y z x y z

RMS SHY SHP ELP WRP WRY WRR 100

RMS ITF x y z x y z

RMS SHY SHP ELP WRP WRY WRR 100

...

can be run.

3.8.2 ATF

The ATF keyword executes the RMS auto-trajectory function. The auto-trajectory function generates a series of joint angles over a time period required to move the end effector of the RMS arm from the current location and attitude to a new location and attitude.

Form:

RMS ATF part steps [sub-steps][file]

part The name of the part to reach with RMS.

steps The number of calls to ATF to reach part.

sub-steps The number of steps to use for positioning between each call to ATF (optional, default is 1.0).

file The name of a command file ('.CMD') to receive output of ATF (optional, no default).

3.8.3 KDG

The KDG keyword is used to output a location and attitude for a given set of joint angles. The command is for information only and does not update any parts or disturb the current RMS end effector location and attitude or RMS joint angles.

Form:

RMS KDG angle1 ... angle6

angle1..angle6 The six joint angles in degrees of RMS arm.

3.8.4 INIT

The keyword INIT clears all accumulated RMS joint angle values and updates all parts specified in thejoint definition list.(see RMS DEFINE)

Form:

RMS INIT

3.8.5 DEFINE

The DEFINE keyword is used to review and/or modify the current joint definition list. A joint definition list is a list of the names and axes for the joints in RMS. This list is used to select and update parts if in the work area. If no arguments follow the keyword DEFINE then the current joint definitions are listed.

Form:

RMS DEFINE [DEFAULT]

DEFAULT The optional keyword causes the current joint definition list to be initialize with the default list.

or

RMS DEFINE joint_number joint_name joint_axis

joint_number The sequence number of the joint.

joint_name The assembly and part name of joint. Must be in the form assembly.part

joint_axisThe axis of rotation for joint.

3.8.5.1 Example

To change the third entry in the current joint definition list

RMS DEF 3 J3.R8 X

To review current list

RMS DEFINE

To set list to default

RMS DEFAULT

The default list is

RMS DEF 1 J1.R4 Y

RMS DEF 2 J2.R6 X

RMS DEF 3 J3.R8 X

RMS DEF 4 J4.R10 X

RMS DEF 5 J5.R12 Y

RMS DEF 6 J6.R14 Z

3.8.6 ITF

The keyword ITF invokes the RMS joint angle routine. The keyword is followed by the destination location and attitude. A part name may be used in place of numeric values; in which case, the location and the attitude of the part will be used. An optional site name defined within the given part's sub-assembly definition, can be used for determining the location and attitude information send to RMS. The optional keyword UPDATE causes immediate update of the parts specified in the joint definition list.(see RMS DEFINE) The optional steps size value is used to increment the the joint angles with positioning the end effector. This useful for smoothing the motion for animation.

3.8.6.1 Explicit_values

This form of the RMS ITF command is used to input explicit location and attitude information for positioning the end effector.

RMS ITF x y z x y z [UPDATE][steps]

x y z The location to be reached by
the RMS.

x y z The attitude, in degrees, of the
location.

UPDATE Optional keyword causing the
immediate update of the parts
specified in the joint
definition list.
(see RMS DEFINE)

steps Optional step size to use in moving
RMS end effector.

3.8.6.2 Symbolic_values

This form of the RMS ITF command permits the selection of location and attitude information to be done symbolically with the use of part names and site names.

RMS ITF part [site] [UPDATE][steps]

part The name of a part to reach with RMS.

site Optional name of a site within in the
given part's sub-assembly. The site
will used to determine position and
attitude of end effector of RMS.

UPDATE Optional keyword causing the
immediate update of the parts
specified in the joint definition
list.(see RMS DEFINE)

steps Optional step size to use in
moving RMS end effector.

3 Examples

The following are some examples of the use of the RMS ITF command. Note, that when the UP argument is specified it is assumed that the user has input an RMS assembly into the

work file and the part names of the arm components have been defined with the RMS DEFINE command.

1. Using explicit input

RMS ITF 9 628 -1090 0 0 0 up 10

This command will cause the RMS end effector to be positioned at x=9, y=628 and z=-1090 with attitude x=0, y=0 and z=0. The 'UP' argument will cause the RMS arm components defined with the RMS DEFINE command will be updated with the appropriate joint angle information. The step size 10 will used to update the joint angle information in 10 steps.

2. Using symbolic input

RMS group1 blk.p2 up 5

This command will cause the RMS end effector to be positioned at the location and with the attitude of the site BLK.P2 in the sub-assembly named GROUP1. The 'up' argument specifies that the RMS arm components are to be updated in 10 steps.

3.8.6.3 Command_files

The RMS module routine returns the results in variables defined in the DMC language processor. These variables contain the joint angle changes from the previous RMS ITF command. The variables are named

SHY=shoulder yaw

SHP=shoulder pitch

ELP=elbow pitch

WRP=wrist pitch

WRY=wrist yaw

WRR=wrist roll

A command file (call it MOVE.CMD) of the form

ROT J1.R4 Y SHY

ROT J2.R6 X SHP

ROT J3.R8 X ELP

ROT J4.R10 X WRP

ROT J5.R12 Y WRY

ROT J6.R14 Z WRR

can be executed to position the RMS to the new position (from its previous position). For a series of RMS actions, a command file can be run with the following contents

RMS ITF x y z x y z

RUN MOVE.CMD

RMS ITF x y z x y z

RUN MOVE.CMD

..

where MOVE.CMD is the command file described above.

3.8.7 PLAID

The keyword PLAID sets the RMS system to interpret input values and output values in the coordinate system used by Plaid.

Form:

RMS PLAID

3.8.8 ORBITOR

The keyword ORBITOR set the RMS system to interpret input values and output values in the coordinate system used for the orbitor.

Form:

RMS ORBITOR

3.8.9 STATUS

The keyword STATUS displays the current RMS joint angles.

Form:

RMS STATUS

3.8.10 UPDATE

The keyword UPDATE causes the update of the parts selected from the current joint definition list (see RMS DEFINE).

Form:

RMS UPDATE

3.9 SET

The SET command is used to change various control values for the DMC program. Currently some of SET options are applicable only to the IRIS or IMI.

3.9.1 PERSPECTIVE

The PERSPECTIVE keyword is used to set the viewing mode to perspective projection. View point definitions of eye point and view direction are used to control viewing. (See VIEW command)

Form:

SET PER switch

switchThe keywords ON or OFF.

3.9.2 SPEED

Form:

SET SPEED s1 s2 s3

s1 Rotation speed.(0 to 1)

s2 Translation speed.(0 to .01)

s3 Scale speed.(0 to .01)

3.10 SHOW

The SHOW command permits the display of information about features of DMC and Plaid depending on the options and arguments given to the command. The information is displayed on the command console.

3.10.1 DESCRIPTION

The DESCRIPTION option is used to display the contents of the description file corresponding to a given Plaid data file. The valid Plaid data files have the extensions '.PRI', '.COG', '.TAR' or '.DSP' , the default extension is '.COG'.

Form:

SHOW DESCRIPTION data_file

data_file The name of a Plaid data file.
The default extension is '.COG'.

3.10.2 PROJECT

The PROJECT option is used to display the current multiuser project name and account name.

Form:

SHOW PROJECT

3.10.3 USER

The USER option is used to display the current multiuser account information, such as priviledges, etc.

Form:

SHOW USER

3.11 SITE

The SITE command permits the definition of a part as a site. The command denotes the part as a site by setting the part reference to the name 'SITE'.

A site is a location in space with a given attitude. Sites can be translated and rotated just like a normal part. For example, sites are used by the CONNECT command to mate two parts. A site can be optionally visible (default is invisible).

Form:

SITE site [visibility]

site The name of the site. The site name
follows the same rules as a part name.

visibility The optional keywords ON or OFF to
control whether the site is shown.

3.11.1 Example

SITE site-x

The command creates a part defined as a site with name site-x. The reference name will be the name SITE.

3.12 STATUS

The STATUS command is used to display the current environment of the DMC program. It will show the name of the current default assembly and current default part (if any).

Form:

STATUS

3.13 TARGET

The TARGET command is used to traverse the named assembly and generate the named target file. The target file is used by the display processor in the PLAID system to generate hidden line and hidden surface views from various viewpoints. The target file contains all the transformed primitives referenced by the given root assembly.

Form:

TARGET assembly target

assembly The name of an existing assembly.

target The name of a target file to receive the output of the traversal process. If thefile already exists, the user will be asked to continue.

3.14 VERSION

Output to the console the current version number of the DMC program.

Form:

VERSION

3.15 VIEW

The VIEW command is used to modify the parameters for viewing an object and/or to actually execute the drawing process for an object. There are two ways of viewing an object, viewing with perspective projection and viewing with orthograhic projection. When viewing in perspective mode, the defining of an eye point and view direction are used to establish a view point. When viewing in orthographic mode, predefined (i.e. FRONT, RIGHT, etc.) and user defined (i.e. ROTATE, TRANSLATE, ZOOM, etc.) viewing specifications and directives are used to establish a view point. (Note, that the DRAW command is a subset of the VIEW command in that it is not used to define eye points and eye directions for perspective viewing, see DRAW).

3.15.1 Explicit-viewpoints.

The eye point location and the view direction can be explicitly entered as a single command. See also VIEW FROM, VIEW TO and VIEW HEAD.

Form:

```
VIEW from_x from_y from_z
       to_x to_y to_z
       [angle]
```

from_x,from_y,from_z The eye point location.

to_x, to_y, to_z The 'to' point for view direction.

angle The head roll angle in deg.

3.15.2 Predefined-views.

There are predefined views to provide for standard views.

Form:

VIEW view-name

view-name One of the eight predefined views.

FR (front)

RI (right)

LE (left)

TO (top)

BO (bottom)

RE (rear)

IS (isometric)

DI (dimetric)

3.15.3 FROM

The FROM keyword is used to select a site for determining eye point location, view direction and field of view. If the view point is currently attached (see VIEW ATTACH), then the view point will

become attached to the given site. An explicit x, y or z location may be given in place of a site.

Form:

VIEW FROM site

site The name of a site to view from.

or

VIEW FROM x y z

x,y,z The location of the eye point.

3.15.4 SITE

See VIEW FROM.

3.15.5 EYE

See VIEW FROM

3.15.6 TO

The TO keyword is used to select a site for determining the view direction for the view point computation. If the view point is currently tracked (see VIEW TRACK), then the view point will track the given site. An explicit x,y and z location may be given in place of the site name.

Form:

VIEW TO site

or

VIEW TO x y z

x,y,z The explicit location to view 'to'.

3.15.7 HEAD

The HEAD keyword is used update the head roll angle of the current viewpoint. However, this value will be overridden if the view point is attached or tracked.

Form:

VIEW HEAD angle

angle The head angle in degrees.

3.15.8 ROLL

See VIEW HEAD.

3.15.9 ATTACH

The ATTACH keyword is used to specify the part or site name to be attached to the view point calculation routine. If any part within the currently selected assembly is moved, the position and attitude of the current selected site to view from (see VIEW FROM) will be used to compute new view point information.

Form:

VIEW ATTACH [site]

site The optional name of a site to used as the location of the eye point, the direction of view and field of view.

3.15.9.1 Example

The following command will attach the view point to a given site.

VIEW ATTACH `PLBAY.EYE`

The site plbay.eye will used to compute the view point formation with the location and attitude of plbay.eye determining the location the eye point and the direction of view. The camera definition of plbay.eye will be used to determine field of view.

The following set of commands are equivalent to the above example.

VIEW FROM `PLBAY.EYE`

3.15.10 VIEW ATTACH

3.15.11 DETACH

The DETACH keyword is used to release the currently attached site. The view point information will not be automatically updated whenever a part is moved within the currently selected assembly. However, the DETACH command will not change the current state of the view point.

Form:

VIEW DETACH

3.15.12 TRACK

The TRACK keyword is used to attach a site to the view point calculation for determining the direction of the view point. This operation is analogous to the ATTACH keyword in that, whenever a part within the currently selected assembly is moved; the view point will automatically be recomputed using the currently attached and track sites. Selecting a site to be tracked will override the any previously determined direction of view.

Form:

VIEW TRACK [site]

site The optional name of a site to be used for computing the view point direction.

3.15.12.1 Example

The following command is used to track a site.

VIEW TRACK J7.EYE

The site `J7.EYE` will be tracked by recomputing the view point direction whenever a part is moved within the currently selected assembly

3.15.13 UNTRACK

The UNTRACK keyword is used reverse the effects of the VIEW TRACK command. The view point direction will be not be changed by this command, but it will not longer automatically computed.

Form:

VIEW UNTRACK

3.15.14 STATUS

The STATUS keyword will output the current values and state of the view point parameters.

Form:

VIEW STATUS

3.15.15 ROTATE

Rotation of the scene using an explicit argument or the joystick is accomplished with ROT argument.

form:

VIEW ROT [x y z] [step]

x, y, z Rotation angles in degrees.

step Number of steps to rotate.

3.15.16 TRANSLATE

The TRA keyword permits the scene to be moved in x and y and optionally in z. When in perspective mode translation in z has the effect of scaling the scene. When the amount is not given the joystick can be used to supply translation values.

form:

VIEW TRA [axis] [amount] [step]

axis Axis specification; X, Y or Z.

amount The amount of translation.

step Number of steps to translate.

3.15.17 SCALE

The SCA keyword permits the scene to be scaled relative to the view center. The scale value is absolute (not accumulative) when it is entered explicitly. When scaling is not explicitly entered then the joystick can be used to provide values for scaling.

Form:

VIEW SCA [scale]

scale A positive scale value.
(not accumulative)

3.15.18 RESET

The RESET keyword will cause the current viewing transformation to be initialized to a front view with scale of 1.0. The view center is not affected.

Form:

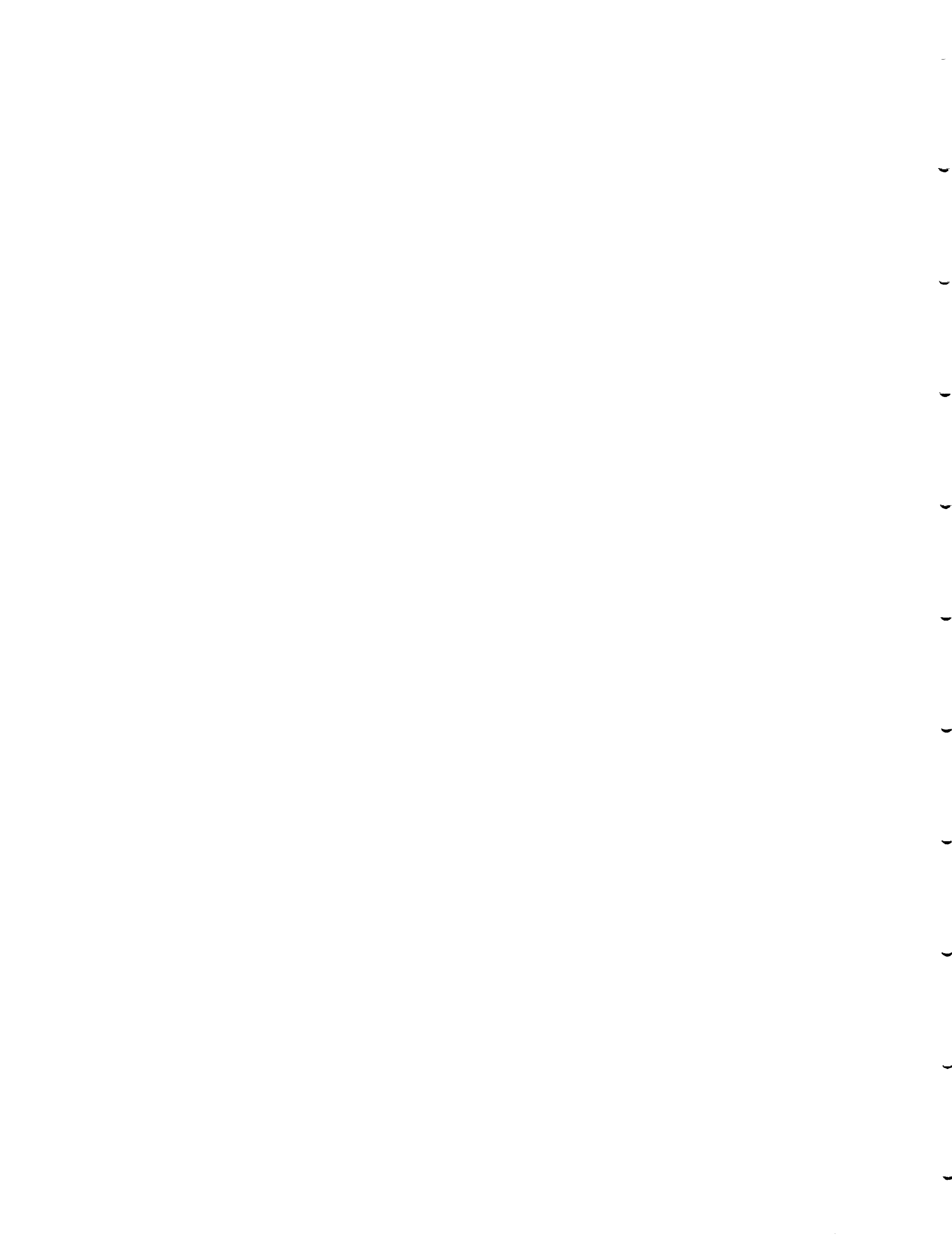
VIEW RESET

3.15.19 CENTER

Form:

VIEW CENTER [x y z]

x,y,z The new center of the view.



Appendix 3 -- Rasterizer Software.

During the course of research, experimentation and development, it became necessary to generate hardcopies of bitmapped and graphic images. The graphics display device was the VaxStation 2000 monitor and the available hardcopy device was an HP LaserJet II laser printer. To get hardcopy images from the VaxStation to the laser printer, a sixel formatted bitmap (the DEC standard bitmap format) had to be converted to the bitmap format of the HP LaserJet II. The following description is the system developed and used by GMS for this function.

In addition, software was developed to output PLAID DISPLAY files to the HP Laserjet II printer. This program is capable of plotting DISPLAY files in either portrait and landscape mode and at any level of resolution of which the printer is capable. The program prompts for user input, and should be fairly self-explanatory as its functionality is rather limited. Printer output is directed to the printer queue HP$LASERJET.

Functional Description.

The program "SIXELDUMP.EXE" is the utility which reads a file containing a sixel formatted bitmap, converts it to an HP LaserJet II bitmap and dumps it to the LaserJet II printer. The **"Sixel Dumping Monitor"** is the Vax command file named **"SIXELHP.COM"** which is submitted to the batch queue to execute this program whenever a file appears in the directory

"DISK$USER1:[SIXELDUMPS]".

This directory is named by the command file and may be changed. It is assumed that this file is a sixel formatted bitmap; unpredictable results may occur if the file is not. The **"Sixel Dumping Monitor"** executes repeatedly (in this implementation, it executes every ten seconds) looking for files to dump to the printer. It will terminate when it finds a file named **"STOP.NOW"**.

User Interface Description.

Workstation Setup.

The user of a VaxStation 2000 workstation can perform screen dumps to the HP LaserJet printer using the Vax Windowing System (VWS) screen dumping facility built into the workstation. However, in order to perform these screen dumps to the laser printer, the user must set up the workstation. The procedure is

1. Move the mouse cursor to a blank area of the screen (gray) and click the left mouse button. The menu titled **"Workstation Options"** will appear.

2. Move the mouse to the option labeled **"Set up the Workstation"** and click the left mouse button. The menu titled **"Workstation Setup"** will appear.

3. Move the mouse to the option labeled **"Printer Set Up"** and click the left mouse button. The menu title **"Printer Setup"** will appear.

4. Move the mouse to the option labeled "**Aspect Ratio**" and click the left mouse button. A menu displaying the aspect ratio options will appear. Select the aspect ratio 1 to 1 using the left mouse button.

5. Move the mouse to the option labeled "**Enter new printer destination**" on the "**Printer Setup**" menu and click the left mouse button. A window will appear showing the current name of the device to receive the sixel bitmaps. Enter the name

"DISK$USER1:[SIXELDUMPS]MYPIC.R75A1"

Entering the name will not overwrite the current name until the carriage return is pressed. Also, the file name can be permanently saved as part the general workstation setup using the "**Save current settings**" option on the "**Workstation Options**" menu.

Performing a Screen Dump.

To perform a screen dump, select the "**Print (portion of) screen**" label on the "**WorkStation Options**" menu. An arrowhead will appear (different from the normal arrow cursor of mouse). Move the arrowhead to the upper left corner of the portion of the screen to dump. Press and hold the left mouse button and move the arrowhead to the lower right corner of the portion of the screen to dump. Release the left mouse button and wait until the normal mouse cursor returns. After several minutes, the screen dump will be output to the laser printer (this may take some time depending on the size of screen area dumped and on the resolution selected).

Aspect Ratio and Resolution.

The user can select various combinations of aspect ratios and resolutions. This is done by using a naming convention for the file extenstion of the file named by the user to receive the sixel formatted bitmaps. For example, the extension "**R75A1**" will be dumped to laser printer at the resolution of 75 dots per inch with an aspect ratio of 1 to 1. The 1 to 1 aspect ratio option in the "**Printer Setup**" menu must match the file extension selction. The valid extentions are

R75A1 - 75 dpi with 1 to 1 aspect
R75A2 - 75 dpi with 2 to 1 aspect
R150A1 - 150 dpi with 1 to 1 aspect
R150A2 - 150 dpi with 2 to 1 aspect
R300A1 - 300 dpi with 1 to 1 aspect
R300A2 - 300 dpi with 2 to 1 aspect

For example, the file named "**MYFILE.R150A2**" will be dumped to the laser printer at a resolution of 150 dots per inch with an aspect ratio of 2 to 1.

Vax command file 'Sixel Dumping Monitor'.

This command file should be executed by the system startup file. It will place the 'Sixel Dumping Monitor' in the system batch queue and will monitor the directory DISK$USER1:[SIXELDUMPS]. It will execute every ten seconds, looking for files to dump to the laser printer.

SIXELHP.COM

```
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$ ! How to Submit Sixel Dumping Monitor
$ !
$ ! submit/nolog/noprint/notify -
$ !       sys$specific:[sysmgr]sixelhp.com
$ ! write sys$output "Sixel Dump Monitor (SDM) running"
$!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
$ Start:
$   set noon
$   define hp$laserjet sys$print
$   sixeldump :== $disk$user1:[plaid.exe]sixeldump.exe
$ Top:
$   wait 00:00:10
$ Middle:
$   name = f$search( "Disk$user1:[SixelDumps]*.*;*" )
$   if name .eqs. "" then goto top
$   if name .eqs. "DISK$USER1:[SIXELDUMPS]STOP.NOW;1" then goto Done
$   open/read/err=Middle ch &name
$   close ch
$   sixeldump 'name
$   delete &name/nolog
$   goto Middle
$ Done:
$   delete/nolog &name
$ exit
```

Sixel Dumping Program and Support Routines.

The following listings form the kernel of the dumping utility. They are compiled as follows:

```
$FORTRAN/CONT=99/I4
```

Listing:

SIXELDUMP.FOR

```
        program sixeldump
c Program intended to accept a DEC SIXEL graphics file as input
C and output to an HP LaserJet II.
c
c Sam Smith
c 3-Feb-88
c
c Bitmap Memory
c
        byte          bitmap(300,3000)
        integer*2     resolution,xdim,ydim
        common/BITMAP/bitmap,resolution,xdim,ydim
c
c I/O Channels
c
        integer*4     tinp
        integer*4     tout
        integer*4     finp
        integer*4     fout
        data tinp/1/tout/2/finp/3/fout/4/
c
c Misc.
c
        character*80 input_name(1)
        character*80 ext
        integer*2        line
c
c jcm05-feb-1988 changed dimension from 1500 to 3000
c to handle extra resolution
c
        byte             sixels(3000)
        integer*2        scount
        integer*2        i
        integer          n
        integer          ierr,eof
        integer          xunit
        integer          yunit
        integer          sixelmax
        integer          aspect
c
c Initialization
c
        data xdim/1200/ydim/1500/resolution/2/
c
c jcm05feb-1988 added these constants to deal with
c multi-resolution situation
c
        data xunit / 600 /
        data yunit / 750 /
        data sixelmax / 3000 /
c
c Begin
c
        open(unit=tinp,
     &      name='SYS$INPUT ',
     &      status='OLD',
     &      err=9000)
c
        open(unit=tout,
     &      name='SYS$OUTPUT ',
     &      status='OLD',
     &      err=9010)
c
        open(unit=fout,
     &      name='sys$scratch:$$laser_jet.dat ',
     &      status='NEW',
     &      form= 'FORMATTED',
     &      err=9020)
c
c Write banner
c
c       write(tout,10)
10      format(' * Sixeldump V1.0 *')
c
c Get file name and open input file
c
        call getarg(input_name,1,n)
```

```
c
c check filename extension for resolution specification
c
c default resolution is 150 dots per inch
c
        call fparse( input_name(1), ext )
        aspect = 2
        resolution = 2
        if( index( ext, 'R75'  ) .gt. 0 ) resolution = 1
        if( index( ext, 'R150' ) .gt. 0 ) resolution = 2
        if( index( ext, 'R300' ) .gt. 0 ) resolution = 4
        if( index( ext, 'A1' ) .gt. 0 ) aspect = 1
        if( index( ext, 'A2' ) .gt. 0 ) aspect = 2
c
c compute dimensions for bitmap based on
c file extension which is a request for
c laserjet resolution
c
        xdim = resolution * xunit
        ydim = resolution * yunit
c
c open sixel file
c
        call sixelopen(input_name,finp,ierr)
        if(ierr.ne.0)go to 9030
c
c Clear bitmap
c
        call clearmap
        line = 0
c
c Read data records & convert
c
30000   continue
        call sixelread( finp, sixels, sixelmax, scount, eof )
        if (eof.ne.0)go to 40000
c
        do 30099 i=0,scount-1
         call sxtohp( i, line, sixels(i+1), aspect )
30099   continue
        line = line + 1
c
c Loop through remaining vectors
c
        go to 30000
c
c Write bitmap to printer and stop
c
40000   continue
        close (unit=finp)
c       write(tout,170)
170     format('+Writing to output file...       ')
        call outmap(fout)
        close (unit=fout)
        call lib$spawn(
     & 'PRINT/PASS/QUE=HP$LASERJET/DEL SYS$SCRATCH:$$LASER_JET.DAT'
     &                )
c       write(tout,40020)
40020   format(' Processing complete.               ')
c
        call exit
c
c Error conditions
c
9000    write(*,9001)
9001    format(' ***** Error opening SYS$INPUT')
        stop
c
9010    write(*,9011)
9011    format(' ***** Error opening SYS$OUTPUT')
        stop
c
9020    write(tout,9021)
9021    format(' ***** Error opening $$LASER_JET.DAT')
        stop
c
9030    write(tout,9031)input_name
```

```
9031    format(' ***** Error opening ',a64)
        stop
c
9040    write(tout,9041)
9041    format(' ***** Error writing $$LASER_JET.DAT')
        stop
c
9050    write(tout,9051)
9051    format(' ***** Error opening $$LASER_JET.COM')
        stop
c
9060    write(tout,9061)
9061    format(' ***** Error writing $$LASER_JET.COM')
        stop
c
        end
```

```
      subroutine sxtohp( sixx, sixy, sixdat, aspect )
c
c Parameters
c
      integer*2  sixx,sixy
      byte       sixdat
    integer*2  aspect
c
c Locals
c
      integer*2  hpx,hpy
      integer*2  i
      integer*2  mask
      integer*2  element
      byte       onebyte(2)
      equivalence (element,onebyte(1))
c
c Begin
c
      onebyte(1) = sixdat-63
      if(onebyte(1) .eq. 0)return
      hpx = sixx
      mask = 1
      do 99 i = 0, 5
        if( iand(element,ishft(mask,i)) .ne. 0 )then
            if( aspect .eq. 1 ) then
                hpy = sixy * 6 + i
                call setbit( hpx, hpy )
            endif
            if( aspect .eq. 2 ) then
                hpy = sixy * 12 + i * 2
                call setbit( hpx, hpy )
                hpy = hpy + 1
                call setbit( hpx, hpy )
            endif
        endif
99    continue
      return
      end
```

```
c
      subroutine fparse( name, ext )
c
      character*(*) name
      character*(*) ext
c
      n = index( name, ':' )
      if( n .eq.0 ) n = len(name)
      do 10 i = n, 1, -1
         if( name(i:i) .eq. '.' ) then
              ext = name(i+1:n-1)
              return
         endif
10    continue
      return
      end
```

SIXELIB.FOR

```
c*********************************************************
c sixelread( c1, out, last, count, eof )
c Author: James C. Maida
c Date:  3-FEB-1988 12:39:16.90
c Function:
c   Read a line of data from sixel file and
c   return a byte array of the sixel data.
c   Line from sixel file is processed to remove
c   compression and escape characters.
c
c   Sixel Line Format:
c      esc P 1 q .... sixel data .... esc \
c      ---------                      -----
c      start                          end of data
c                            (decimal 144 can be alternate
c                                     end of data)
c
c   Compression Format:
c      !nnnnC
c   where
c      '!' indicates start of duplicate character
c          count
c      nnnn is the count (1 to 5) in ascii numbers
c           (one byte per number)
c      C is the character to duplicate.
c
c   Raster format:
c      All sixel data has ascii byte value of 63 to 127.
c      To map to pixels, etc.
c      1. subtract 63 from sixel byte
c      2. each sixel byte represents a column of six
c         scan lines.(low order bit is 1st scan line of the six)
c
c Parameters:
c   c1         = i*4. Input channel for sixel file.(in)
c   out(count) = b*1. Byte array.(out)
c   count      = i*4. Number of bytes in array.(out)
c   eof        = i*4. End file indicator.(out)
c                     0 = ok
c                     1 = end of file
c*********************************************************
      subroutine sixelread( c1, out, last, count, eof )
      implicit integer (a-z)
c parameters
      integer c1
      byte out(1)
      integer last
      integer count
      integer eof
c common
c functions
c locals
      character*2048 line
      character*1 chr
      integer eol
c begin
      count = 0
      eof = 0
c
c read a line from sixel file
c
      read( c1, 2, end = 99 ) line
2     format( a )
c
c process line for starting escape sequences
c and duplicate character counts
c
      now = 1
      do 5 i = 1, len(line)
c
c get characters
c
         call getbyte( line, now, next, len(line), chr, nc, eol )
c
c end of line or end of sixel data ?
c
```

```
          if( eol .ne. 0 ) then
              if( eol .eq. 2 ) eof = 1
              return
          endif
c
c character count and character are processed into
c byte array.
c
          if( nc .gt. 0 ) then
              do 10 j = 1, nc
                 count = count + 1
c
c exceeded length of byte array ?
c
                 if( count .gt. last ) return
                 out(count) = ichar(chr)
10            continue
          endif
          now = next
5       continue
        return
c
c hard end of file
c
99      continue
        eof = 2
        return
        end
c*****************************************************
c sixelopen( filename, c1, ier ) - open sixel file
c Author: James C. Maida
c Date:  3-FEB-1988 12:39:16.90
c Function:
c   Open sixel file in read-only mode.
c   File is assumed sequential, with variable
c   length records. File is opened formatted.
c Parameters:
c   filename = c(*). File name of sixel file.(in)
c   c1       = i*4. Channel to open file on.(in)
c   ier      = i*4. Open error code.(out)
c                    0 = ok
c                   >0 = Fortran open error code.
c*****************************************************
      subroutine sixelopen( filename, c1, ier )
c parameters
      character*(*) filename
      integer c1
      integer ier
c common
c functions
c locals
      character*80 fname
c begin
      fname = filename
      open( unit = c1,
     1       file = fname,
     2       access = 'sequential',
     3       status = 'old',
     4       readonly,
     5       form = 'formatted',
     6       iostat = ier )
      return
      end
c*****************************************************
c getbyte( line, now, next, last, char, count, eol )
c Author: James C. Maida
c Date:  3-FEB-1988 12:39:16.90
c Function:
c   Process sixel line for escape sequences and
c   duplicate characters.
c Parameters:
c   line = c(*). Sixel input line.(in)
c   now  = i*4. Current character pointer.(in)
c   next = i*4. Pointer to next character.(out)
c   last = i*4. Length of sixel line.(in)
c   char = c*1. Character from line.(out)
c   count = i*4. Character count for character returned.(out)
```

```
c   eol   = i*4. End of line or end of sixel data.(out)
c                0 = ok
c                1 = end of line
c                2 = end of sixel data
c*****************************************************
      subroutine getbyte( line, now, next, last, char, count, eol )
c parameters
      character*(*) line
      integer now, next, last
      character*1 char
      integer count, eol
c common
c functions
c locals
      integer c
      character*10 num
c begin
      eol = 0
      char = ' '
      count = 0
      next = now + 1
c
c end of sixel line ?
c
      if( next .gt. last ) then
          eol = 1
          return
      endif
      c = ichar(line(now:now))
c
c end of sixel line ?
c
      if( c .eq. 144 ) then
          eol = 2
          return
      endif
c
c escape sequences to be removed ?
c
      if( c .eq. 27 ) then
c
c end of sixel data
c
          if( line(next:next) .eq. '\' ) then
              eol = 2
              return
          endif
c
c skip to start of sixel data
c
          do 7 i = next, last
             if( line(i:i) .eq. 'q' ) then
                 next = i + 1
                 return
             endif
7         continue
c
c no sixel data ?
c
          eol = 2
          return
      endif
c
c end of line
c
      if( c .eq. 45 ) then
          eol = 1
      endif
c
c valid sixel data ?
c
      if( c .ge. 63 ) then
          char = line(now:now)
          count = 1
          return
      endif
c
```

```
c duplicate character count flag ?
c
      if( line(now:now) .eq. '!' ) then
c
c extract count as character string
c
            do 10 i = next, last
               if( line(i:i) .lt. '0' .or. line(i:i) .gt. '9' ) then
                  num = line(next:i-1)
                  k = i - next
c
c convert to binary number
c
                  read(num,11) count
11                format(i<k>)
                  char = line(i:i)
                  next = i + 1
                  return
               endif
10          continue
      endif
      return
      end
c*****************************************************
c getarg( arg, n ) - get argument from command line
c Author: James C. Maida
c Date:  3-FEB-1988 12:39:16.90
c Function:
c   Get arguments from the command line.
c   The main routine should be executed at
c   DCL level as follows:
c   $ progname :== $ disk$user:[user]progname.exe
c   then enter
c   $ progname arg1 arg 2 ...
c Parameters:
c   arg(maxarg) = c*(*) Argument list.(out)
c   maxarg      = i*4. Length of argument list.(in)
c   n           = i*4. Number of arguments returned.(out)
c*****************************************************
      subroutine getarg( arg, maxarg, n )
c parameters
      character*(*) arg(1)
      integer       maxarg
      integer       n
c locals
      character*132 a
      integer       l
      integer status
      integer argcount
      integer lib$get_foreign
c begin
      call lib$get_foreign( a, 'Input file : ', argcount )
      l = argcount
c
c no arguments ?
c
      if( l .lt. 1 ) then
         n = 0
         arg(1) = ' '
         return
      endif
      kk = 0
      kkk = 1
      do 10 i = 1, l
        if( kkk .eq. 1 ) then
           kkk = 0
           k = i
        endif
        if( a(i:i) .eq.' ' .or. i .eq. l ) then
           kk = kk + 1
           if( kk .gt. maxarg ) then
              n = kk - 1
              return
           endif
           if( i .lt. l ) arg(kk) = a(k:i-1)
           if( i .eq. l ) arg(kk) = a(k:i)
           kkk = 1
```

```
         endif
10    continue
      n = kk
      return
      end
```

```
HPLIB.FOR
c Subroutines to manipulate bitmaps destined for the
c LaserJet II printer
c
c Sam Smith
c 29-Jan-87
c
       subroutine setbit(x,y)
c
c Parameters
c
       integer*2   x,y
c
c Bitmap Memory
c
       byte        bitmap(300,3000)
       integer*2   resolution,xmax,ymax
       common/BITMAP/bitmap,resolution,xmax,ymax
c
c Locals
c
       integer*2   byte_num,bit_num
       integer*2   element,mask
       byte        onebyte(2)
       equivalence (onebyte(1),element)
c
c Begin
c
       if((x .lt. 0)    .or.
     &    (x .ge. xmax) .or.
     &    (y .lt. 0)    .or.
     &    (y .ge. ymax)) return
       byte_num = x / 8 + 1
       bit_num  = mod(x,8)
       onebyte(1) = bitmap(byte_num,y+1)
       mask = 128
       mask = ishft(mask,-bit_num)
       element= ior(element,mask)
       bitmap(byte_num,y+1) = onebyte(1)
       return
       end
```

```
      subroutine clearmap
c
c Subroutine clears bitmap to zero (all white)
c
c Sam Smith   29-Jan-88
c
c Bitmap Memory
c
      byte        bitmap(300,3000)
      integer*2   resolution,xmax,ymax
      common/BITMAP/bitmap,resolution,xmax,ymax
c
      do 99 i = 1,300
        do 88 j = 1,3000
          bitmap(i,j) = 0
88      continue
99    continue
      return
      end
```

```
      subroutine outmap(ich)
c
c Subroutine writes bitmap to HP LaserJet II printer
c
c Sam Smith   29-Jan-88
c
c Parameters
      integer*4   ich
c
c Bitmap Memory
c
      byte        bitmap(300,3000)
      integer*2   resolution,xmax,ymax
      common/BITMAP/bitmap,resolution,xmax,ymax
c
c Locals
c
      byte        esc,ff
      integer*2   ymult
      integer*2   i,j,k
      logical     blank
      data esc/27/ff/12/
c
c Begin
c
c
c Calculate y-address multiplier
c
      ymult = 4/resolution
c Reset printer
c
      write(ich,10)esc,esc,esc,esc,esc
10    format('+',a1,'E',a1,'&l66P',a1,'&l0o2E',
     &         a1,'&l1H',a1,'&l1X',a1,$)
c
c Init to proper graphic resolution
c
      if(resolution .eq. 1)write(ich,11)esc
11    format('+',a1,'*t75R',$)
      if(resolution .eq. 2)write(ich,12)esc
12    format('+',a1,'*t150R',$)
      if(resolution .eq. 4)write(ich,13)esc
13    format('+',a1,'*t300R',$)
c
c Write data to printer
c
      do 99 i=1,ymax
c
c Check for blank line
c
        blank = .true.
c
        do 77 j=1,(resolution)*75
          if (bitmap(j,i) .ne. 0) blank = .false.
77      continue
c
        if (.not.blank)then
c
c Set cursor address & enter graphic mode
c
          write(ich,20)esc,esc,(i-1)*ymult,esc,esc,resolution*75
20        format('+',a1,'*p0X',a1,'*p',i4,'Y',
     &           a1,'*r1A',a1,'*b',i3,'W',$)
c
c Send one line to printer
c
          do 88 j=0,((resolution-1)*75),75
            write(ich,30)(bitmap(j+k,i),k=1,75)
30          format('+',75a1,$)
88        continue
c
c Leave graphics mode
c
          write(ich,40)esc
40        format('+',a1,'*rB',$)
c
```

```
           endif
c
99      continue
        return
        end
```

PLAID DISPLAY File Rasterizer for HP Laserjet II.

The following listings form the kernel of the rasterizer utility. They are compiled as follows:

```
$FORTRAN/CONT=99/I4
```

Listing: RASTERIZE.FOR

```
      program rasterize
c Program intended to accept PLAID Display file as
c input and output to an HP LaserJet II.
c
c Sam Smith
c 27-Jan-87
c
c Input buffers for header/data records
c
      integer*4   drec(8)
      integer*2   dhwrec(16)
      equivalence (drec(1),dhwrec(1))
c
c Declarations for header record 1
c
      integer*2   dtype
      integer*2   dview
      equivalence (dhwrec(1),dtype)
      equivalence (dhwrec(2),dview)
c
c Declarations for header record 2
c     real*4      dscale
      real*4      droll
      real*4      dvpt(6)
      equivalence (drec(1),dscale)
      equivalence (drec(2),droll)
      equivalence (drec(3),dvpt(1))
c
c Declarations for data records
c
      integer*2    dwin
      integer*2    dkind
      integer*2    dpoly
      integer*2    dedge
      real*4       dv1(3)
      real*4       dv2(3)
      equivalence (dhwrec(1), dwin)
      equivalence (dhwrec(2), dkind)
      equivalence (dhwrec(3), dpoly)
      equivalence (dhwrec(4), dedge)
      equivalence (dhwrec(5), dv1(1))
      equivalence (dhwrec(11),dv2(1))
c
c Setup information for record unblocking routine
c
      integer*4 icb(4)
      integer*4 ibuff(192)
      common/CB001/ibuff
c
c Bitmap Memory
c
      byte        bitmap(300,3000)
      integer*2   resolution,xdim,ydim
      common/BITMAP/bitmap,resolution,xdim,ydim
c
c I/O Channels
c
      integer*4   tinp
      integer*4   tout
      integer*4   finp
      integer*4   fout
      data tinp/1/tout/2/finp/3/fout/4/
c
c Misc.
c
      character*80 input_name
      character    yesno
      real*4       xmin,ymin
      real*4       xmax,ymax
      real*4       xsize,ysize
      real*4       x_doffset,y_doffset
      integer*2    x_poffset,y_poffset
      integer*2    xrange,yrange
      real*4       scale_factor
      logical      portrait
      integer*2    ix1,iy1
      integer*2    ix2,iy2
```

```
        integer*4    vectors
        integer      i
c
c Initialization
c
        data xrange/600/yrange/750/
c
c Begin
c
        open(unit=tinp,
     &      name='SYS$INPUT ',
     &      status='OLD',
     &      err=9000)
c
        open(unit=tout,
     &      name='SYS$OUTPUT ',
     &      status='OLD',
     &      err=9010)
c
        open(unit=fout,
     &      name='sys$scratch:$$laser_jet.dat ',
     &      status='NEW',
     &      form= 'FORMATTED',
     &      err=9020)
c
c Write banner
c
        write(tout,10)
10      format(' * Rasterizer V1.1 *')
c
c Query for resolution
c
1       write(tout,11)
11      format(/' 1 -->  75 dpi'/
     &          ' 2 --> 150 dpi'/
     &          ' 3 --> 300 dpi'/
     &          ' Select resolution: ',$)
        read(tinp,12)resolution
12      format(i5)
        if((resolution .lt. 1) .or. (resolution .gt. 3)) then
          write(tout,13)
13        format(' ***** Invalid selection *****')
          go to 1
        end if
        resolution = 2**(resolution-1)
        xdim = resolution * xrange
        ydim = resolution * yrange
c
c Calculate X & Y offsets to center image
c
        x_poffset = xdim/2
        y_poffset = ydim/2
c
c Get file name and open input file
c
        write(tout,20)
20      format(/' Enter input file name: ',$)
        read(tinp,30)input_name
30      format(a80)
c
        open(unit=  finp,
     &      name=  input_name,
     &      access='SEQUENTIAL',
     &      form=  'UNFORMATTED',
     &      status='OLD',
     &      readonly,
     &      err=   9030)
c
c Initialize blocking variables
c
        icb(1) = 184
        icb(2) = 8
        icb(3) = 0
        icb(4) = 0
c
c Read header record 1
c
```

```
        call pl8cb(finp,'R',drec,icb,ibuff,ierr)
        if (ierr .ne. 0) then
          write(tout,35)input_name
35        format(' ***** Error reading ',a64)
          stop
        end if
        write(tout,40)dtype,dview
40      format(/' Type: ',a2,', Perspective: ',i1)
c
c Read header record 2
c
        call pl8cb(finp,'R',drec,icb,ibuff,ierr)
        if (ierr .ne. 0) then
          write(tout,35)input_name
          stop
        end if
        write(tout,50)dscale,droll
50      format(' Scale: ',f10.4,', Head Roll: ',f10.4)
        write(tout,60)dvpt(1),dvpt(2),dvpt(3)
60      format(' View from: ',f10.4,', ',f10.4,', ',f10.4)
        write(tout,70)dvpt(4),dvpt(5),dvpt(6)
70      format(' View to:   ',f10.4,', ',f10.4,', ',f10.4)
c
c Initialize min/max values
c
        xmin = 1e30
        xmax = -1e30
        ymin = 1e30
        ymax = -1e30
        vectors = 0
        write(tout,80)
80      format(/' Calculating min/max...')
c
c Read data records & check for error/end
c
10000   call pl8cb(finp,'R',drec,icb,ibuff,ierr)
        if (ierr .eq. 1) go to 20000
        if (ierr .eq. 2) then
          write(tout,35)input_name
          stop
        end if
c
c Check for invisible edges & penetration points
c
          if(dkind.ne.0)go to 10000
          if(dedge.eq.1)go to 10000
c
c Update min/max values & vector count
c
        if(dv1(1) .lt. xmin)xmin = dv1(1)
        if(dv2(1) .lt. xmin)xmin = dv2(1)
        if(dv1(1) .gt. xmax)xmax = dv1(1)
        if(dv2(1) .gt. xmax)xmax = dv2(1)
        if(dv1(2) .lt. ymin)ymin = dv1(2)
        if(dv2(2) .lt. ymin)ymin = dv2(2)
        if(dv1(2) .gt. ymax)ymax = dv1(2)
        if(dv2(2) .gt. ymax)ymax = dv2(2)
        vectors = vectors + 1
c
        go to 10000
c
c Got min/max, make portrait/landscape decision and calculate scale factor
c
20000   write(tout,90)vectors,xmin,xmax,ymin,ymax
90      format('+Vectors: ',i12/
     &         ' X-min: ',f10.4,', X-max: ',f10.4/
     &         ' Y-min: ',f10.4,', Y-max: ',f10.4)
        xsize = xmax - xmin
        ysize = ymax - ymin
c
        x_doffset = (xmax + xmin) / 2
        y_doffset = (ymax + ymin) / 2
c
        if (ysize .gt. xsize) then
          write(tout,100)
100       format(' Portrait mode suggested... ')
          portrait = .true.
```

```
        else
          write(tout,110)
110       format(' Landscape mode suggested... ')
          portrait = .false.
        end if
c
c See if user wants to override portrait/landscape decision
c
        write(tout,120)
120     format(' Override? (Y or N): ',$)
        read(tinp,130)yesno
130     format(a)
        if((yesno.eq.'Y').or.(yesno.eq.'y'))portrait=.not.portrait
c
c Inform user of orientation and calculate scale factor
c
        if(portrait)then
          write(tout,140)
140       format(' Portrait mode selected.')
          scale_factor=min(((1/xsize)*xdim),
     &                     ((1/ysize)*ydim))
        else
          write(tout,150)
150       format(' Landscape mode selected.')
          scale_factor=min(((1/xsize)*ydim),
     &                     ((1/ysize)*xdim))
        endif
c
c Re-initialize record blocking info
c
        rewind (unit=finp)
        icb(1) = 184
        icb(2) = 8
        icb(3) = 0
        icb(4) = 0
c
        write(tout,160)
160     format(/' Processing vectors...')
c
c Skip over header records
c
        call pl8cb(finp,'R',drec,icb,ibuff,ierr)
        if (ierr .ne. 0) then
          write(tout,35)input_name
          stop
        end if
        call pl8cb(finp,'R',drec,icb,ibuff,ierr)
        if (ierr .ne. 0) then
          write(tout,35)input_name
          stop
        end if
c
c Clear bitmap
c
        call clearmap
c
c Read data records, scale & rasterize
c
30000   continue
          call pl8cb(finp,'R',drec,icb,ibuff,ierr)
          if (ierr .eq. 1) go to 40000
          if (ierr .eq. 2) then
            write(tout,35)input_name
            stop
          end if
c
c Check for invisible edges & penetration points
c
          if(dkind.ne.0)go to 30000
          if(dedge.eq.1)go to 30000
c
c Center & scale
c
          ix1 = (dv1(1) - x_doffset) * scale_factor
          iy1 = (dv1(2) - y_doffset) * scale_factor
          ix2 = (dv2(1) - x_doffset) * scale_factor
          iy2 = (dv2(2) - y_doffset) * scale_factor
```

```
c
c Rasterize
c
          if(portrait)then
            ix1 = ix1 + x_poffset
            iy1 = ydim - (iy1 + y_poffset)
            ix2 = ix2 + x_poffset
            iy2 = ydim - (iy2 + y_poffset)
            call bresen(ix1,iy1,ix2,iy2)
          else
            ix1 = ydim - (ix1 + y_poffset)
            iy1 = iy1 + x_poffset
            ix2 = ydim - (ix2 + y_poffset)
            iy2 = iy2 + x_poffset
            call bresen(iy1,ix1,iy2,ix2)
          endif
c
c Loop through remaining vectors
c
        go to 30000
c
c Write bitmap to printer and stop
c
40000   continue
        close (unit=finp)
        write(tout,170)
170     format('+Writing to output file...       ')
        call outmap(fout)
        close (unit=fout)
        call lib$spawn(
     & 'PRINT/PASS/QUE=HP$LASERJET/DEL SYS$SCRATCH:$$LASER_JET.DAT'
     &              )
        write(tout,40020)
40020   format(' Processing complete.              ')
c
        call exit
c
c Error conditions
c
9000    write(*,9001)
9001    format(' ***** Error opening SYS$INPUT')
        stop
c
9010    write(*,9011)
9011    format(' ***** Error opening SYS$OUTPUT')
        stop
c
9020    write(tout,9021)
9021    format(' ***** Error opening $$LASER_JET.DAT')
        stop
c
9030    write(tout,9031)input_name
9031    format(' ***** Error opening ',a64)
        stop
c
9040    write(tout,9041)
9041    format(' ***** Error writing $$LASER_JET.DAT')
        stop
c
9050    write(tout,9051)
9051    format(' ***** Error opening $$LASER_JET.COM')
        stop
c
9060    write(tout,9061)
9061    format(' ***** Error writing $$LASER_JET.COM')
        stop
c
        end
```

```
      subroutine bresen(x1,y1,x2,y2)
c
c Performs a simple, floating point based substitute for
c Bresenham's algorithm to rasterize input vectors into a bitmap.
c Assumes vectors have been scaled and offset properly.
c
c Sam Smith  29-Jan-88
c
c
c Parameters
c
      integer*2   x1,y1,x2,y2
c
c Locals
c
      integer*2   xsize,ysize
      integer*2   xpos,ypos,numdot
      real*4      xincr,yincr
      real*4      xaddr,yaddr
      integer*2   i
c
c Begin
c
      xsize = iabs(x2 - x1)
      ysize = iabs(y2 - y1)
c
c Handle case of zero-length vector
c
      if((xsize .eq. 0) .and. (ysize .eq. 0))then
        call setbit(x1,y1)
        return
      else
c
c Increment by one on x, a fraction on y
c
        if(xsize .gt. ysize)then
          numdot = xsize
          if(x2 .ge. x1) then
            xincr = 1
          else
            xincr = -1
          end if
          if(y2 .ge. y1) then
            yincr = float(ysize) / float(xsize)
          else
            yincr = float(-ysize) / float(xsize)
          endif
c
        else
c
c Increment by one on y, a fraction on x
c
          numdot = ysize
          if(y2 .ge. y1) then
            yincr = 1
          else
            yincr = -1
          end if
          if(x2 .ge. x1) then
            xincr = float(xsize) / float(ysize)
          else
            xincr = float(-xsize) / float(ysize)
          endif
        endif
c
c Increments calculated, set bits as appropriate
c
        xaddr = x1
        yaddr = y1
c
        do 99 i = 1,numdot
          xpos = xaddr
          ypos = yaddr
          call setbit(xpos,ypos)
          xaddr = xaddr + xincr
          yaddr = yaddr + yincr
```

```
99        continue
c
      endif
      return
      end
```

Appendix 4 -- DCRL Browser

Example: The STS-Orbitor Logical Hierarchy

GMS Technology evaluated the DCRL knowledge representation language for use in the K-Base project. The evaluation was was conducted in two parts. The first part of the evaluation consisted of the construction of a knowledge-base of the major components of the space shuttle. We first present the logical shuttle hierarchy, then the DCRL code which represents the hierarchy.

The second part of the evaluation consisted of writing a program which would make use of the representation constructed in part 1. For this exercise we chose to write a graphical browser which provides the user an interactive interface to the knowledge representation.

The DCRL Browser allows the user to interactively peruse the knowledge network using the mouse to designate objects of interest. The entire network may be viewed or just a small portion may be selected to simplify the display. The Browser code runs on the VAXstation 2000 in the Lucid Common Lisp environment.

- Shuttle OV-103 Discovery
    - Forward Section
        - Upper Deck
            - Aft Crew Station
                - Overhead viewports
                - Remote-Manipulator Translation Hand Controller
                - Remote-Manipulator Rotational Hand Controller
                - Orbitor Rotational Hand Controller
                - Payload Control Panel
                - Mission Specialist Seat
                - Payload Specialist Seat
                - Interdeck Access
            - Forward Crew Station
                - Mission Commander's Seat
                - Pilot's Seat
                - Flight Computer and Navigation Console
                - Navigation Unit
        - Lower Deck
            - Galley Space
            - Airlock
                - Interdeck access
                - Telescoping Escape Pole (new)
            - Extra Payload Specialists' seats (2)
            - Waste Management
            - Stowage Lockers
            - Avionics/Electronics Bay
        - Nose Section
            - Reaction Control System (RCS)
                - RCS Forward Thrusters
                - RCS Oxidizer Tank
                - RCS Helium Tank
                - RCS Hydrazine Fuel Tank
            - Phased-array Radar
            - Nosewheel Landing Gear (improved)
    - Payload Bay Section
        - Payload Bay Doors (2)
            - Radiators (2-4?)
        - Remote Manipulator Arm
            - Elbow Video Camera (Videocam)
        - Extravehicular-activity Handhold
        - Getaway Special Canister
        - Aluminium Sheathing (Payload Bay lining)
        - Supports i.e. for Tracking and Data Relay Satellite (TDRS)
        - Below Payload Bay
            - Ventilator Liquid-Oxygen Tank
            - Fuel Cell Liquid-Hydrogen/Liquid-Oxygen Tanks
    - Wing Section
        - Main Landing Gear
        - Reinforced Carbon-Carbon Leading Edge
        - Elevon (Aluminum Honeycomb Structure)
    - Tail Section
        - Space Shuttle Main Engines (3)
            - High-pressure Fuel Turbopump (improved)
            - Liquid-Hydrogen Supply Manifold
            - Liquid-Oxygen Supply Manifold
        - Auxiliary Power Hydrazine/Oxidizer Tanks
        - Fuel Cell
        - Reaction Control System (RCS)
            - RCS Oxidizer Tank
            - RCS Hydrazine Fuel Tank
            - RCS Aft Thrusters
            - RCS Helium Tanks (2)
        - Orbital Maneuvering System (OMS)
            - OMS Hydrazine Fuel Tank
            - OMS Oxidizer Tank
            - OMS Helium Tank
            - OMS Thruster
        - Rudder (Aluminum Honeycomb Structure)
            - Rudder/Speed Brake Power Unit
            - Rudder/Speed Brake
            - Rudder/Speed Brake Hydraulics
- Tracking and Data Relay Satellite
    - C-Band Commercial Antenna
    - 4.9 Meter K/S-Band Antenna (2)
    - 2.0 Meter K-Band Ground-Link Antenna
    - Stowed Solar Array
    - Inertial Upper Stage

DCRL Representation of the Shuttle Hierarchy

```
{concept universe
         is a collection of concept
         from tout
}

{concept shuttle-ov-103
         is a collection of concept
         from universe
}
{concept people-seats
         is a collection of concept
         from universe
}
{concept propulsion-system
         is a collection of concept
         from shuttle-ov-103
}
{concept guidance-system
         is a collection of concept
         from shuttle-ov-103
}

{concept fuel-system
         is a collection of concept
         from shuttle-ov-103
}

{concept forward-section
         is a collection of concept
         from shuttle-ov-103
}

{concept payload-section
         is a collection of concept
         from shuttle-ov-103
}

{concept wing-section
         is a collection of concept
         from shuttle-ov-103
}

{concept tail-section
         is a collection of concept
         from shuttle-ov-103
}
{concept rms-system
         is a collection of concept
         from shuttle-ov-103
}

{concept hand-controllers
         is a collection of concept
         from shuttle-ov-103
}
{concept upper-deck
         is a collection of concept
         from forward-section
}

{concept lower-deck
         is a collection of concept
         from forward-section
}

{concept nose-section
         is a collection of concept
         from forward-section
}

{concept main-engines
         is a collection of concept
         from (tail-section propulsion-system)
}
```

```
{concept reaction-control-system
        is a collection of concept
        from (nose-section tail-section propulsion-system)
}

{concept orbital-maneuvering-system
        is a collection of concept
        from (tail-section propulsion-system)
}

{concept rudder
        is a collection of concept
        from (tail-section guidance-system)
}

{concept rcs-oxidizer-tank
        is a collection of concept
        from (reaction-control-system fuel-system)
}

{concept rcs-hydrazine-tank
        is a collection of concept
        from (reaction-control-system fuel-system)
}

{concept rcs-helium-tank
        is a collection of concept
        from (reaction-control-system fuel-system)
}

{concept rcs-aft-thrusters
        is a collection of concept
        from reaction-control-system
}

{concept oms-hydrazine-tank
        is a collection of concept
        from (orbital-maneuvering-system fuel-system)
}

{concept oms-oxidizer-tank
        is a collection of concept
        from (orbital-maneuvering-system fuel-system)
}

{concept oms-helium-tank
        is a collection of concept
        from (orbital-maneuvering-system fuel-system)
}

{concept oms-thruster
        is a collection of concept
        from orbital-maneuvering-system
}

{concept forward-crew-station
        is a collection of concept
        from upper-deck
}

{concept aft-crew-station
        is a collection of concept
        from upper-deck
}

{concept navigation-unit
        is a collection of concept
        from (forward-crew-station guidance-system)
}

{concept phased-array-radar
        is a collection of concept
        from (nose-section guidance-system)
}

{concept extra-mission-spec-seat
```

```
        is a collection of concept
        from
        (lower-deck people-seats)
}

{concept mission-spec-seat
        is a collection of concept
        from
        (aft-crew-station people-seats)
}

{concept payload-spec-seat
        is a collection of concept
        from (aft-crew-station people-seats)
}

{concept command-seat
        is a collection of concept
        from (forward-crew-station people-seats)
}

{concept pilot-seat
        is a collection of concept
        from (forward-crew-station people-seats)
}

{concept main-landing-gear
        is a collection of concept
        from wing-section
}

{concept nosewheel-landing-gear
        is a collection of concept
        from nose-section
}

{concept fuel-turbopump
        is a collection of concept
        from main-engines
}

{concept liquid-hydrogen-supply-manifold
        is a collection of concept
        from main-engines
}

{concept liquid-oxygen-supply-manifold
        is a collection of concept
        from main-engines
}

{concept rms-translation-hand-controller
        is a collection of concept
        from (aft-crew-station rms-system hand-controllers)
}

{concept rms-rotational-hand-controller
        is a collection of concept
        from (aft-crew-station rms-system hand-controllers)
}

{concept orbitor-rotational-hand-controller
        is a collection of concept
        from (aft-crew-station hand-controllers)
}

{concept rms-arm
        is a collection of concept
        from (payload-section rms-system)
}

{concept payload-bay-doors
        is a collection of concept
        from payload-section
}

{concept waste-management
```

```
        is a collection of concept
        from lower-deck
}

{concept tdrs
        is a collection of concept
        from universe
}

{concept antennas
        is a collection of concept
        from universe
}

{concept c-band-commericial-antenna
        is a collection of concept
        from (tdrs antennas)
}

{concept ks-band-antenna
        is a collection of concept
        from (tdrs antennas)
}

{concept K-band-ground-link-antenna
        is a collection of concept
        from (tdrs antennas)
}

{concept stowed-solar-array
        is a collection of concept
        from tdrs
}

{concept inertial-upper-stage
        is a collection of concept
        from tdrs
}
```

DC-RL Browser Display of Shuttle Example

Browsing Tree: TOUT Date: 10/10/88 Time: 11:40:39 Sorted: NIL

Figure A4.1 Shuttle Hierarchy without Boxes

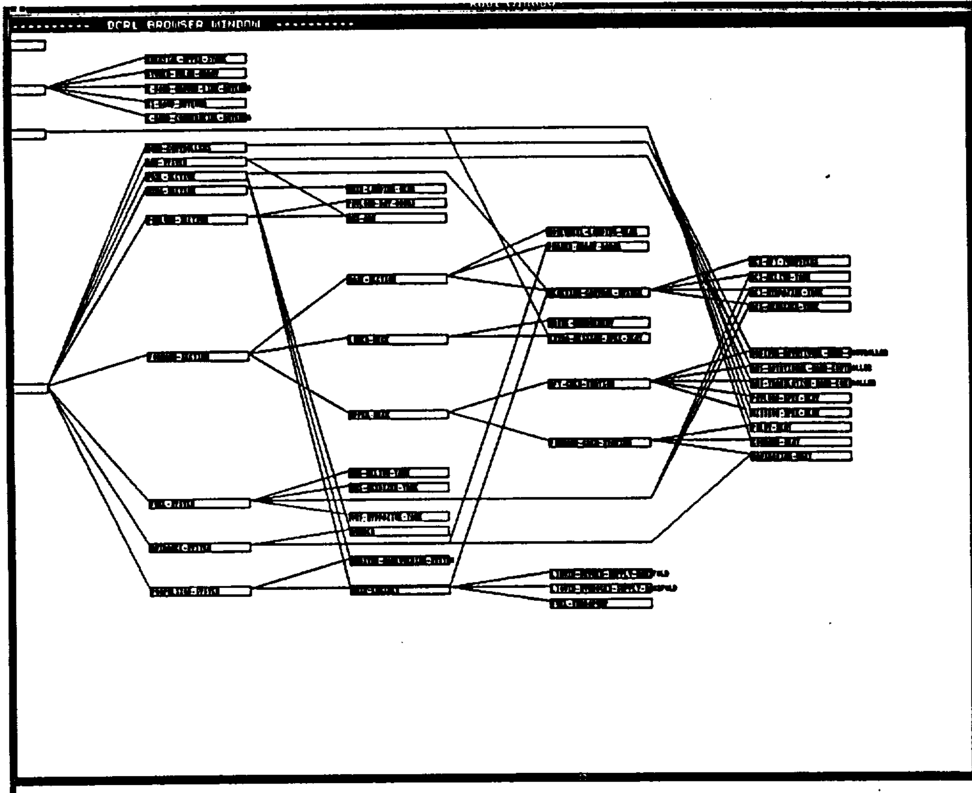


Figure A4.2 Shuttle Hierarchy with Boxes

The Code of the DCRL Browser

```
;;;---------------------------------------------------------------
;;; DCRL-Browse.Lisp                                          RAG/10-Oct-88
;;;
;;; This is a prototype of the DC-RL Tree/Net Browser for
;;; K-Base.
;;;
;;;
;;;
;;;---------------------------------------------------------------
(defvar *tree-root-node* 'tout)                          ; the root of the DCRL environment
(defvar *Tree-Display-Root* 'tout)      ; the root of the current tree
(defvar *tree-max-levels* 100)
(defvar *tree-structure* nil)
(defvar *tree-node-set* *concepts*)
(defvar *tree-max-level* 0)
(defvar *tree-level-lists*)
(setf *tree-level-lists* (make-array *tree-max-levels*))
(defvar *tree-level-counts*)
(setf *tree-level-counts* (make-array *tree-max-levels*))
(defvar *tree-traversal-queue* (list 'universe))
(defvar *tree-process-queue* '(universe))
(defvar *tree-current-breadth* 0)
(defvar *tree-current-depth* 0)
(defvar *tree-cell-width* 200)          ; the width of a cell of a diagram
(defvar *tree-cell-height* 8)           ; cell height of a diagram
(defvar *tree-window-handle* nil)        ; lisp handle for the diagram
                                         ; window.
(defvar *Tree-Sort-Order* nil)
                                                   ; Deter. the sort-order of sub-
trees
(defvar *Tree-Display-Font*
        (find-font 'small-roman))
(defvar *Tree-Diagram-Size* 'medium)
(defvar *Active-Display-Cells* nil)

(defvar *Cell-MB-Menu* nil)             ; The pop-up menu to display
                                         ; middle mouse button is depressed.
(defvar *Cell-RB-Menu* nil)             ; The pop-up menu to display
                                         ; right mouse button pressed.
```

```
;;;-------------------------------------------------------------
;;; DCRL-Browse  --- Browse a DCRL class hierarchy given the root
;;;                  of the hierarchy to be browsed (?).
;;;
;;;-------------------------------------------------------------

(defun DCRL-Browse (root)

  (setf *tree-display-root* root)
  (setf *tree-process-queue* nil)
  (setf *tree-current-depth* 0)
  (setf *tree-current-breadth* 0)
  (setf *active-display-cells* nil)

  ;;; Add the top of the universe to the process queue to begin
  ;;; the breadth-first traversal.

  (queue-add-list (list root))
  (tree-clear-levels *concepts*)
  (tree-process-node root (get root 'concept-parent) 0 0)
  (tree-traverse-by-breadth)
  (tree-sort-children root)
  (tree-clear-level-lists)
  (tree-build-level-lists)
  (setf *tree-current-breadth* (tree-calculate-width root))
  (setf *tree-current-breadth* (Tree-Layout-Diagram root 1))
  (delete-diagram-window)
  (make-diagram-window)
  (make-diagram-menus)
  (tree-draw-diagram root)
  (tree-draw-links root)
)

(defun delete-viewport-from-tree (vp tree)
  (cond ((null tree) nil)
        ((equal vp (car tree))(cdr tree))
        ((listp (car tree)) (cons (delete-viewport-from-tree vp (car tree))
                                  (delete-viewport-from-tree vp (cdr tree))
                            )
        )
        (T (cons (car tree) (delete-viewport-from-tree vp (cdr tree))))
  )
)
```

```
;;; ---------------------------------------------------------------
;;; Tree-Traverse-By-Breadth -- Recursive traverser
;;; ---------------------------------------------------------------
;;;
(defun Tree-Traverse-By-Breadth ()
  (let ((child-list nil)
            (symbol-level nil)
            (current-node nil)
            )

    ;;; Get the next node from the front of the queue and process it.

    (setf current-node (queue-get-item))

;;; (princ (format nil "Breadth-Trav: processing node -- ~a ~%"
;;;                        (symbol-name current-node))
;;;            )

    (cond ((null current-node))
              (t (setf child-list (tree-get-children current-node))
                 (setf symbol-level (get current-node 'tree-level))
                 (setf *tree-current-depth* (max *tree-current-depth* symbol-level))
                 (dolist (child child-list)
                   (Tree-Process-Node child current-node (+ symbol-level 1) 0)
                   )
                 (queue-add-list child-list)
                 (Tree-Traverse-By-Breadth)
                 )

              )
    )
  )
```

```
;;; Tree-Clear-Levels -- set the TREE-LEVEL property to NIL
;;;                      and the TREE-SERIAL property to NIL for all
;;;                      concepts in the knowledge-base.

(defun Tree-Clear-Levels (concept-list)
  (let ((concept-list concept-list))

;;; (princ (format nil
;;;                          "Initializing Concept: ~A ~%" (symbol-name (car concept-
list))))

    (cond ((null concept-list) "Finished")
          (T (setf (get (car concept-list) 'tree-level) 0)
             (setf (get (car concept-list) 'tree-serial) 0)
             (setf (get (car concept-list) 'region) nil)
             (setf (get (car concept-list) 'tree-order-value) 0)
             (setf (get (car concept-list) 'tree-diagram-parent) nil)
             (setf (get (car concept-list) 'tree-nd-level-span) 0)
             (Tree-Clear-Levels (cdr concept-list))
             )
          )
    )
  )
```

```
;;; Get-Children -- given a DCRL node symbol, return the list of its children.
;;;

(defun get-children (node)
  (get node 'concept-child)
  )



(defun tree-test ()
  (setf *tree-structure* '(universe))
  (setf (car *tree-structure*) (cons (car *tree-structure*)
                                                          (list (get (car *tree-
structure*)
                                                                      'concept-child)
                                                                )
                                                          )
            )
  )


;;; Tree-Get-Children -- Given the name of a concept in the DCRL concept
;;;                       network, Return the list of that concept's
;;;                       children.

(defun Tree-Get-Children (node)
  (get node 'Concept-Child)
  )

;;; Tree-Get-Parents --  Given the name of a concept in the DCRL concept
;;;                       network, return the list of that concept's
;;;                       parent concepts.

(defun Tree-Get-Parents (node)
  (get node 'Concept-Parent)
  )

;;; Tree-Process-Node -- Process a node during the traversal by setting
;;;                       "tree-level" and "tree-serial" properties.

(defun Tree-Process-Node (concept parent level serial)
  (setf (get concept 'tree-diagram-parent) parent)
  (setf (get concept 'tree-level) level)
  (setf (get concept 'tree-serial) serial)
  )
```

```
;;;-----------------------------------------------------------
;;; Tree-Calculate-Width
;;;                  Calculate the width of each sub-tree by
;;;                  performing a depth-first traversal of the DCRL
;;;                  class hierarchy tree. This program recursively
;;;                  defines the width of a tree as the sum of the widths of
;;;                  its sub-trees. The width of a tree with no
;;;                  children (a leaf) is defined as 1.
;;;
;;;                  Each node has an associated property,
;;;                  'tree-breadth, which contains this width value.
;;;-----------------------------------------------------------

(defun Tree-Calculate-Width (root)
  (let ((child-sum 0)
         (child-list nil)
         (diagram-parent nil)
         (non-diagram-children nil)

         )

    (cond ((null root) 0)
              (T (setf child-list (get-children root))

                 ;;; calculate the width of each sub-tree of root

                 (dolist (child child-list)
                   (setf diagram-parent (get child 'tree-diagram-parent))
                   (cond ((equal root diagram-parent)
                               (setf child-sum (+ child-sum
                                                                     (tree-calculate-
width child))
                                                      )
                               )
                              ;;; There is a non-diagram child of this node
                              ;;; so, make note of the fact. Also, calculate the
                              ;;; maximum distance between this root and its deepest
                              ;;; non-diagram child and store this value in the
                              ;;; property 'tree-nd-level-count.

                              (T (setf non-diagram-children T)
                                   (setf (get root 'tree-nd-level-span)
                                         (max (get root 'tree-nd-level-span)
                                                   (- (get child 'tree-level)
                                                      (get root 'tree-level)
                                                      )
                                                   )
                                         )
                                   )
                              )
                   )


                 ;;; If root had one or more non-diagram children that are more
                 ;;; than one level beneath it, add
                 ;;; one to its width to leave room for the link to children
                 ;;; that are more than one level below root.

                 (cond ((and non-diagram-children
                                    (> (get root 'tree-nd-level-span) 1)
                                    )
                            (setf child-sum (+ child-sum 1))
                            (setf (get root 'non-diagram-children) t)
                            )
                           (T (setf (get root 'non-diagram-children) nil))
                           )

                 ;;; Store the breadth of this tree in the 'tree-breadth property
                 ;;; of ROOT.

                 (setf (get root 'tree-breadth)
                          (max 1 child-sum)
                          )
                 )
              )
    )
```

```
;;; ------------------------------------------------------------
;;; Tree-Sort-Children
;;;                  Sorts the children of root by sub-tree size
;;;                  ('by-size), or by the number of non-diagram
;;;                  children ('by-nd-size) or does not sort them
;;;                  (nil).
;;; ------------------------------------------------------------

(defun Tree-Sort-Children (root)

  (cond ((equal *tree-sort-order* 'by-size)
           (tree-sort-children-by-size root)
           )
          ((equal *tree-sort-order* 'by-nd-size)
           (tree-sort-children-by-nd-size root)
           )
          ((equal *tree-sort-order* 'by-alpha)
           (tree-sort-children-by-alpha root)
           )
          (T nil)
        )

)
```

```
;;;------------------------------------------------------------
;;; Tree-Sort-Children-by-Size
;;;                 Sort the children of each node by the number
;;;                 of descendants of that node. This
;;;                 operation puts the nodes with the largest
;;;                 number of descendants at the top of diagram.
;;;
;;;                 Each node has an associated property,
;;;                 'tree-order-value, which contains this count
;;;                 value.
;;;------------------------------------------------------------

(defun Tree-Sort-Children-by-Size (root)
  (let ((child-sum 0)
          (child-list nil)
          )

    (cond ((null root) 0)
              (T (setf child-list (get-children root))

                 ;;; recursively calculate the number of children
                 ;;; of each sub-tree of root.

                 (dolist (child child-list)

                   (setf child-sum (+ (+ child-sum
                                                         (tree-sort-children-by-size
child))
                                                      1)
                          )
                   )

                 )
              )

    ;;; Store the number of descendants for the tree in the
    ;;; 'tree-order-value property of the root.

;;;    (princ (format nil
;;;                        "Child-Sort -- Node: ~a has ~a children~%"
;;;                        (symbol-name root)
;;;                        child-sum)
;;;           )

    (setf (get root 'tree-order-value) child-sum)
    (order-children-by-size root)
    (eval child-sum)

    )
  )
```

```
;;; ------------------------------------------------------------------
;;; Tree-Order-Subtrees-by-Size
;;; ------------------------------------------------------------------

(defun Tree-Order-Subtrees-by-Size (root)
  (Order-Children-by-size root)
  (dolist (child (get-children root))
    (Tree-Order-Subtrees-by-Size child)
    )
  )
```

```lisp
;;; ------------------------------------------------------------------
;;; Order-Children-by-Size
;;;      Sort a given list into descending order on the value of a given property.
;;;      The current implementation of this routine uses the Lisp SORT function.
;;; ------------------------------------------------------------------

(defun Order-Children-by-Size (node)
  (let ((temp-list nil)
            (order-list nil)
            )

    ;;; Get the list of children of the given node and sort it in
    ;;; descending order into a temporary list.

    (setf temp-list (get node 'concept-child))
    (setf temp-list
              (stable-sort temp-list
                              #'>
                              :key #'(lambda (x)
                                                    (get x 'tree-order-value))
                              )
              )

    (setf order-list temp-list)

;;;    The following code places the largest sub-trees in the middle
;;;    of the list...

;;;    (dotimes (nth-item (list-length temp-list))
;;;      (cond ( (evenp nth-item)
;;;               (setf order-list (append order-list
;;;                                                   (list (nth nth-item temp-
list)
;;;                                                              )
;;;                                                   )
;;;                          )
;;;               )
;;;             ( (oddp nth-item)
;;;               (setf order-list (cons (nth nth-item temp-list)
;;;                                                order-list)
;;;                          )
;;;               )
;;;             )
;;;      )

    (setf (get node 'concept-child) order-list)

;;;    (princ (format nil
;;;                         "--- Size-Sort -- Child Order for Node: ~a ----~%"
;;;                         (symbol-name node)
;;;                         )
;;;           )
;;;    (dolist (child (get node 'concept-child))
;;;
;;;      (princ (format nil
;;;                           "Child: ~a  Sub-tree Size: ~a~%"
;;;                           (symbol-name child)
;;;                           (get child 'tree-order-value)
;;;                           )
;;;             )
;;;      )

    )
  )
```

```
;;;-----------------------------------------------------------------
;;; Tree-Sort-Children-by-ND-Size
;;;                  Sort the children of each node by the number
;;;                  of non-diagram (ND) children of that node. This
;;;                  operation puts the nodes with the largest
;;;                  number of ND links in the center of diagram so
;;;                  that they will hopefully look better!
;;;
;;;                  Calculate the number of non-diagram children
;;;                          of each sub-tree by
;;;                  performing a depth-first traversal of the DCRL
;;;                  class hierarchy tree.
;;;
;;;                  Each node has an associated property,
;;;                  'tree-order-value, which contains this count
;;;                  value.
;;;-----------------------------------------------------------------

(defun Tree-Sort-Children-by-ND-Size (root)
  (let ((child-sum 0)
           (child-list nil)
           (diagram-parent nil)
           )

    (cond ((null root) 0)
              (T (setf child-list (get-children root))

                 ;;; calculate the number of non-dia references
                 ;;; of each sub-tree of root. The number of ND
                 ;;; children of a node is the sum of the ND
                 ;;; children of the root plus the number of
                 ;;; ND children of its sub-trees.

                 (dolist (child child-list)
                   (setf diagram-parent (get child 'tree-diagram-parent))
                   ;;; If this child is a ND child, then increment the
                   ;;; ND child count (but do NOT traverse that sub-tree).
                   ;;; Otherwise, traverse the sub-tree looking for more
                   ;;; ND links.
                   (cond ((not (equal diagram-parent root))
                              (setf child-sum (+ child-sum 1))
                              )
                             (T
                              (setf child-sum (+ child-sum
                                                                 (tree-sort-children-
by-nd-size child))
                                                  )
                              )
                             )
                   )
                 )
              )
    ;;; Store the number of ND children for the tree in the
    ;;; 'tree-order-value property of the root.

    ;;;(princ (format nil
    ;;;                     "ND-Kids -- Node: ~a has ~a ND-Kids~%"
    ;;;                    (symbol-name root)
    ;;;                    child-sum)
    ;;;          )
    (order-children-by-nd-number root)
    (setf (get root 'tree-order-value) child-sum)

    )
  )
```

```lisp
;;; -------------------------------------------------------------------------
;;; Tree-Order-Subtrees-by-ND-Children
;;; -------------------------------------------------------------------------

(defun Tree-Order-Subtrees-by-ND-Children (root)
  (Order-Children-by-ND-Number root)
  (dolist (child (get-children root))
    (Tree-Order-Subtrees-by-ND-Children child)
    )
  )

;;; -------------------------------------------------------------------------
;;; Order-Children-by-ND-Number
;;;      Sort a given list into descending order on the value of a given property.
;;;      The current implementation of this routine uses a
;;;      bubble-sort algorithm.
;;; -------------------------------------------------------------------------

(defun Order-Children-by-ND-Number (node)
  (let ((temp-list nil)
           (order-list nil)
           )

    ;;; Get the list of children of the given node and sort it in
    ;;; descending order into a temporary list.

    (setf temp-list (get node 'concept-child))
    (setf temp-list
            (stable-sort temp-list
                         #'>
                         :key #'(lambda (x) (get x 'tree-order-value))
                         )
            )

    (dotimes (nth-item (list-length temp-list))
      (cond ( (evenp nth-item)
                (setf order-list (append order-list
list)                                               (list (nth nth-item temp-
                                                              )
                                                    )
                      )
                )
            ( (oddp nth-item)
                (setf order-list (cons (nth nth-item temp-list)
                                                  order-list)
                      )
                )
            )
      )

    (setf (get node 'concept-child) order-list)

;;;    (princ (format nil
;;;                        "--- Child Order for Node: ~a ----~%"
;;;                        (symbol-name node)
;;;                        )
;;;           )
;;;    (dolist (child (get node 'concept-child))
;;;
;;;      (princ (format nil
;;;                          "Child: ~a  ND-Value: ~a~%"
;;;                          (symbol-name child)
;;;                          (get child 'tree-order-value)
;;;                          )
;;;             )
;;;      )

  )
 )
```

```
;;;-----------------------------------------------------------
;;; Tree-Sort-Children-by-Alpha
;;;                     Sort the children of each node by the string
;;;                     value of the node's symbolic name.
;;;
;;;                     Each node has an associated property,
;;;                     'tree-order-value, which contains this count
;;;                     value.
;;;-----------------------------------------------------------

(defun Tree-Sort-Children-by-Alpha (root)
  (let ((child-list nil)
        )

    (cond ((null root) 0)
          (T (order-children-by-alpha root)
             (setf child-list (get-children root))

             ;;; recursively calculate the number of children
             ;;; of each sub-tree of root.

             (dolist (child child-list)
                  (tree-sort-children-by-alpha child)
               )
              )

            )

    ;;; Debug I/O

;;;    (princ (format nil
;;;                        "Alpha Child-Sort -- Node: ~a has ~a children~%"
;;;                        (symbol-name root)
;;;                        child-sum)
;;;            )

    )
  )

;;; ---------------------------------------------------------------------
;;; Order-Children-by-Alpha
;;;      Sort a given list into descending order on the value of a given property.
;;;      The current implementation of this routine uses the Lisp SORT function.
;;; ---------------------------------------------------------------------

(defun Order-Children-by-Alpha (node)
  (let ((temp-list nil)
        )

    ;;; Get the list of children of the given node and sort it in
    ;;; descending order into a temporary list.

    (setf temp-list (get node 'concept-child))
    (setf temp-list
          (stable-sort temp-list
                       #'string<
                       :key #'(lambda (x)
                                       (symbol-name x))
                       )
          )

    (setf (get node 'concept-child) temp-list)

;;;    (princ (format nil
;;;                        "--- Child Order for Node: ~a ----%"
;;;                        (symbol-name node)
;;;                        )
;;;             )
;;;    (dolist (child (get node 'concept-child))
;;;
;;;      (princ (format nil
;;;                          "Child: ~a~%"
;;;                          (symbol-name child)
;;;                          )
;;;              )
;;;      )
```

```
;;; ----------------------------------------------------------------
;;; Tree-Clear-Level-Lists -- Clear the lists of the nodes at each level
;;;                           of the concept network.
;;;
;;; ----------------------------------------------------------------

(defun Tree-Clear-Level-Lists ()
  (let ((max-levels (length *tree-level-lists*))
        )

    ;;; Initialize the lists for each level to nil

    (dotimes (tree-level max-levels)
      (setf (aref *tree-level-lists* tree-level) nil)
      )

    (dotimes (tree-level max-levels)
    (setf (aref *tree-level-counts* tree-level) 0)
      )
    )
  )

;;; Tree-Build-Level-Lists -- construct a list of the nodes at each level
;;;                           of the concept network. This is accomplished
;;;                           by accessing each of the concepts listed in
;;;                           the list *concepts*.
;;;

(defun Tree-Build-Level-Lists ()
  (let ((node-level 0)
        (level-count 0)
        )

    ;;; Build a list of nodes at each level by looking at the "tree-level"
    ;;; property of each node listed in *concepts*.

    (dolist (current-node *concepts*)
      (setf node-level (get current-node 'tree-level))
      (setf (aref *tree-level-lists* node-level)
            (cons current-node (aref *tree-level-lists* node-level))
            )

      (setf level-count (aref *tree-level-counts* node-level))
      (setf level-count (+ level-count 1))
      (setf (aref *tree-level-counts* node-level) level-count)
      (setf (get current-node 'tree-serial) level-count)
      )
    )
  )
```

PRECEDING PAGE BLANK NOT FILMED

```
;;; Tree-List-Levels -- List all nodes by level number. This routine uses the
;;;                       level lists generated by Build-Level-Lists and
;;;                       stored in *tree-level-lists*.

(defun Tree-List-Levels ()
  (let (
          (node-serial 0)
          (node-list nil)
          (node-parent-list nil)
          (node-child-list nil)
          )
    (dotimes (level (length *tree-level-lists*))
      (cond ((null (aref *tree-level-lists* level)) "finished")
            ((not (null (aref *tree-level-lists* level)))
               (terpri)
               (princ (format nil "Level ~a with nodes:~%" level))
               (setf node-list (aref *tree-level-lists* level))
               (dolist (node-symbol node-list)
                 (setf node-serial (get node-symbol 'tree-serial))
                 (princ (format nil
                                         "Node: ~a  Level: ~a   Serial# ~a  Pos ~a
ND-Kids: ~a~%"
                                         (symbol-name node-symbol)
                                         level
                       node-serial
                                         (get node-symbol 'tree-diagram-column)
                                         (get node-symbol 'tree-order-value)
                                         )
                       )
                 (setf node-parent-list (tree-get-parents node-symbol))
                 (setf node-child-list (tree-get-children node-symbol))
                 (princ (format nil "     with parents: ~%"))
                 (dolist (next-link node-parent-list)
                      (princ (format nil "          ~a ~%" next-link))
                      )
                 (princ (format nil "     and children: ~%"))
                 (dolist (next-link node-child-list)
                      (princ (format nil "          ~a ~%" next-link))
                      )
                 )
               )
             )
      )
    )
  )
```

```
;;; -----------------------------------------------------------------
;;; Tree-Layout-Diagram -- Determine the positions of tree nodes within
;;;                        each level of the tree so that the tree can
;;;                        be mapped to the display.
;;;
;;; Return-Value:          This function returns the incremented starting
;;;                        column of the next node of the diagram.
;;; -----------------------------------------------------------------

(defun Tree-Layout-Diagram (node starting-column)
  (let ( (width 0)
           (node-column 0)
           (non-diagram-children nil)
           (nd-level-span 0)
           (current-start-column starting-column)
           (diagram-parent nil)
           (child-width 1)
           )

    ;;; Position node in the center of an area large enough to contain this
    ;;; node's entire tree.
    ;;; Store the node's column position in the property 'tree-diagram-column.

    (setf width (get node 'tree-breadth))
    (setf node-column
            (floor
             (+ starting-column
                (/ width 2))
             )
            )
    (setf (get node 'tree-diagram-column) node-column)

    (setf non-diagram-children (get node 'non-diagram-children))
    (setf nd-level-span (get node 'tree-nd-level-span))

    ;;; Determine the positions of node's sub-trees.

    (dolist (current-child (get-children node))
      (setf diagram-parent (get current-child 'tree-diagram-parent))
      (setf child-width (get current-child 'tree-breadth))

      ;;; Only position diagram-children of this node. Leave non-diagram
      ;;; children where they are.

      (cond ((equal node diagram-parent)

                  ;;; If this tree has non-diagram children, then leave the
                  ;;; space directly under the root open so that the link to
                  ;;; the non-diagram children will not cross any of the direct
                  ;;; (diagram) children.

                  (if (and non-diagram-children
                                 (> nd-level-span 1)
                                 (and (>= node-column current-start-column)
                                       (<= node-column (+ current-start-column
                                                              (- child-width
1)
                                                              )
                                           )
                                       )
                                 )
                         (setf current-start-column
                               (+ node-column 1)
                               )
                         )
                  (setf (get current-child 'tree-start-column) current-start-column)
                  (setf current-start-column
                         (Tree-Layout-Diagram current-child current-start-column)
                         )
                  )
                 ;;; Otherwise, this is a non-diagram child so set its width to one
column.
                 (T (setf child-width 1))
                 )
     )
;;; The return value of this invocation is the next available "column" in the tree
diagram.
```

```
          (+ current-start-column 1)
          )
        )
```

```
(defun queue-add-list (lst)
  (setf *tree-process-queue* (append *tree-process-queue* lst)))


(defun queue-get-item ()
  (let ((return-item nil))
    (setf return-item (car *tree-process-queue*))
    (setf *tree-process-queue* (cdr *tree-process-queue*))
    return-item))


;;; Make-Diagram-Window -- Creates a Lucid Lisp window to accomodate a
;;;                                DCRL Tree which is "breadth" cells wide and
;;;                        "depth" cells (levels) deep.

(DEFUN Make-Diagram-Window ()
  (let ( (diagram-width 0)
            (diagram-height 0)
            )
    (setf diagram-width  (* *Tree-Cell-Width*  (+ *tree-current-depth* 2)))
    (setf diagram-height (* *tree-cell-height* (+ *tree-current-breadth* 10)))

    (INITIALIZE-WINDOWS :HEIGHT
                                     800
                                     :WIDTH
                                     1010
                                     :LABEL
                                     "* DCRL Browser Window *")

    "Make a window in root window"

    (SETF *tree-window-handle*
            (let ((time-list (multiple-value-list (get-decoded-time))))
              (MAKE-WINDOW :X
                               0
                               :Y
                               0
                               :VIEWPORT-WIDTH
                               (min 950 diagram-width)
                               :VIEWPORT-HEIGHT
                               (min 750 diagram-height)
                               :WIDTH
                               diagram-width
                               :HEIGHT
                               diagram-height
                               :SCROLL
                               T
                               :TITLE
                               (format nil
                                       "  Browsing Tree: ~a  Date: ~a/~a/~a  Time:
~a:~a:~a Sorted: ~a"
                                              (symbol-name *tree-display-root*)
                                              (nth 4 time-list)
                                              (nth 3 time-list)
                                              (mod (nth 5 time-list) 100)
                                              (nth 2 time-list)
                                              (nth 1 time-list)
                                              (nth 0 time-list)
                                              (symbol-name *tree-sort-order*)
                                              )
                               )
              )
      )
    )
  )
```

```
;;; ---------------------------------------------------------------
;;; Delete-Diagram-Window
;;; ---------------------------------------------------------------

(defun Delete-Diagram-Window ()
  (when (viewportp *tree-window-handle*)
    (deactivate-viewport *tree-window-handle*)
    (clear-bitmap-active-regions (viewport-bitmap *tree-window-handle*))
    (delete-viewport *tree-window-handle*)

;;;     TROUBLESOME STUFF???
    (setf (viewport-children (root-viewport))
              (delete-viewport-from-tree *tree-window-handle*
                                                        (viewport-children (root-
viewport))
                                                        )
              )

    (setf *tree-window-handle* nil)
    )
  )

;;; ---------------------------------------------------------------
;;; Make-Diagram-Menus
;;;          Generate the pop-up menus that are needed to manipulate the
;;;          tree diagram.
;;; ---------------------------------------------------------------

(defun Make-Diagram-Menus ()

  ;;; Define the Middle-Button (-MB-) menu

  (if (null *cell-mb-menu*)
      (setf *cell-mb-menu*
              (make-pop-up-menu '(("Draw Small Display"    small    )
                                            ("Draw Medium Display"   medium  )
                                            ("Draw Large Display"    large   )
                                            ("Sort by Size"      by-size )
                                            ("Sort by ND-Child" by-nd-size)
                                            ("Sort by Alpha"     by-alpha)
                                            ("No Sort"           nil-sort)
                                            ("QUIT"              quit    )
                                            )
                                  )
              )
      )
  ;;; Define the Right-Button (-RB-) menu

  (if (null *cell-rb-menu*)
      (setf *cell-rb-menu*
              (make-pop-up-menu '(("Draw Sub-Tree"     sub-tree)
                                            ("Draw from Root"   root     )
                                            ("Show Concept"     show     )
                                            )
                                  )
              )
      )
)
```

```
;;; ---------------------------------------------------------------------
;;; Tree-Draw-Node -- Draw a node of the tree in the tree diagram window
;;; ---------------------------------------------------------------------

(defun Tree-Draw-Node (node)
  (let ((node-level (get node 'tree-level))
            (node-column (get node 'tree-diagram-column))
            (node-x 0)
            (node-y 0)
        (cell-origin nil)
        (cell-corner nil)
            (cell-width (truncate (/ *tree-cell-width* 2)))
            (cell-height (truncate (* *tree-cell-height* 0.666)))
            (boole-op boole-1)
            )

    (setf node-x (* node-level  *tree-cell-width*))
    (setf node-y (* node-column *tree-cell-height*))

    (stringblt *tree-window-handle*
                  (make-position (+ node-x 1)
                                             (- node-y 2))
                  *Tree-Display-Font*
                  (symbol-name node)
                  :operation boole-op
                  )

    (cond ( (not (equal *Tree-Diagram-Size* 'small))
                (draw-line *tree-window-handle*
                            (make-position node-x
                                                  node-y)
                            (make-position node-x
                                                  (- node-y cell-height))
                            :operation boole-op
                            )
                (draw-line *tree-window-handle*
                            (setf cell-origin (make-position node-x
                                                                 (- node-y cell-
height)))
                            (make-position (+ node-x cell-width)
                                                  (- node-y cell-height))
                            :operation boole-op
                            )
                (draw-line *tree-window-handle*
                            (make-position (+ node-x cell-width)
                                                  (- node-y cell-height))
                            (make-position (+ node-x cell-width)
                                                  node-y)
                            :operation boole-op
                            )
                (draw-line *tree-window-handle*
                            (setf cell-corner (make-position (+ node-x cell-width)
                                                                 node-y))
                            (make-position node-x
                                                  node-y)
                            :operation boole-op
                            )
                )

              (T
                (setf cell-origin (make-position node-x
                                                                 (- node-y cell-
height)))
                (setf cell-corner (make-position (+ node-x cell-width)
                                                                 node-y))
                )

      )
     (Make-Node-Mousey node cell-origin cell-corner)
   )
 )
```

```
;;;
;;; Make-Node-Mousey -- makes the diagram box an active region
;;; SLS - 3-Oct-88

(defun Make-Node-Mousey (node cell-origin cell-corner)
  (setf (get node 'region) (make-the-region cell-origin cell-corner))
  (setf *active-display-cells* (cons node *active-display-cells*))
  )

;;; Actually make the region here
;;;

(defun make-the-region (cell-origin cell-corner)
  (make-active-region

   (make-region :origin cell-origin
                :corner cell-corner
                )

   :bitmap (viewport-bitmap *tree-window-handle*)

   ;; Invert region on entry
   ;;
   :mouse-enter-region
   #'(lambda (viewport active-region mouse-event x y)
       (declare (ignore mouse-event x y))
       (bitblt-region (viewport-bitmap viewport) active-region
                      (viewport-bitmap viewport) active-region
                      boole-c1
                      )
       )

   ;; Invert region back to normal on exit
   ;;
   :mouse-exit-region
   #'(lambda (viewport active-region mouse-event x y)
       (declare (ignore mouse-event x y))
       (bitblt-region (viewport-bitmap viewport) active-region
                      (viewport-bitmap viewport) active-region
                      boole-c1
                      )
       )

   :mouse-middle-down
   #'(lambda (viewport active-region mouse-event x y)
       (declare (ignore viewport mouse-event x y))
       (let ((choice nil)
             )
            (with-asynchronous-method-invocation-allowed
             (setf choice (car (pop-up-menu-choose *cell-mb-menu*)))
             (cond ((equal choice 'small)
                    (tree-set-display-size 'small)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'medium)
                    (tree-set-display-size 'medium)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'large)
                    (tree-set-display-size 'large)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'by-size)
                    (setf *tree-sort-order* 'by-size)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'by-nd-size)
                    (setf *tree-sort-order* 'by-nd-size)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'by-alpha)
                    (setf *tree-sort-order* 'by-alpha)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'nil-sort)
                    (setf *tree-sort-order* 'nil-sort)
                    (dcrl-browse *tree-display-root*))
                   ((equal choice 'quit)
                    (delete-diagram-window)
                    )
                   (T nil)
                   )
             )
```

```
          )
     )
 :mouse-right-down
 #'(lambda (viewport active-region mouse-event x y)
     (declare (ignore viewport mouse-event x y))
     (let ((choice nil)
                (node   nil)
                )
            (setf node (get-region-name active-region *active-display-cells*))
            (with-asynchronous-method-invocation-allowed
             (setf choice (car (pop-up-menu-choose *cell-rb-menu*)))
             (cond ((equal choice 'sub-tree)(dcrl-browse node))
                        ((equal choice 'root)  (dcrl-browse 'tout))
                        ((equal choice 'show)  (show-concept node))
                        (T nil)
                        )
             )
            )
     )
 )
)
```

```
;;; Get-Region-Name -- Returns the name of an active region
;;;

(defun get-region-name (r l)
  (cond ((null l) nil)
           ((equal (get (car l) 'region) r) (car l))
           (T (get-region-name r (cdr l)))
           )
  )


;;; Tree-Leaf-p -- Returns T if the given node is a leaf of the
;;;                DCRL tree and nil otherwise.

(defun Tree-Leaf-p (node)
  (let ((child-list (get-children node))
           )
    (null child-list)
    )
  )

;;;-----------------------------------------------------------------
;;; Tree-Draw-Diagram
;;;-----------------------------------------------------------------

(defun Tree-Draw-Diagram (root)
  (let ((child-list nil)
           )

    (tree-draw-node root)
    (setf child-list (get-children root))

    ;;; calculate the width of each sub-tree of root

    (dolist (child child-list)
             (if (equal root (get child 'tree-diagram-parent))
                 (tree-draw-diagram child)
                 )
             )

    )
  )
```

```
;;;-----------------------------------------------------------
;;; Tree-Draw-Links
;;;-----------------------------------------------------------

(defun Tree-Draw-Links (root)
  (let ((parent-level 0)
           (parent-column 0)
           (child-level 0)
           (child-column 0)
           (pseudo-level 0)
           (pseudo-column 0)
           (boole-op boole-1)
           )

    (cond ((null root) nil)
             (T (setf parent-level (get root 'tree-level))
                (setf parent-column (get root 'tree-diagram-column))
                (dolist (child (get-children root))
                  (setf child-level (get child 'tree-level))
                  (setf child-column (get child 'tree-diagram-column))

                  (cond (
                    ;;; If the parent node is more than one level above the child
node,
                            ;;; then draw a line down to where a parent tail would be
if the
                            ;;; parent were at child-level + 1 and disperse the
connections to
                            ;;; the children from that point.

                            (> (- child-level parent-level) 1)
                            ;;; Draw a line from the tail of the parent to the tail
of
                            ;;; a non-existant (pseudo-) node at one level above the
child.

                            (setf pseudo-level (- child-level 1))
                            (setf pseudo-column parent-column)
                            (draw-line *tree-window-handle*
                                                  (tree-calc-node-tail parent-level
parent-column)
                                                  (tree-calc-node-tail pseudo-level
pseudo-column)
                                                  :operation boole-op
                                                  )
                            (draw-line *tree-window-handle*
                                                  (tree-calc-node-tail pseudo-level
pseudo-column)
                                                  (tree-calc-node-head child-level
child-column)
                                                  :operation boole-op
                                                  )
                            )
                           ;;; Otherwise, draw the connecting links directly from the
                           ;;; tail of the parent to the head of the child.

                           (T (draw-line *tree-window-handle*
                                                    (tree-calc-node-tail parent-
level parent-column)
                                                    (tree-calc-node-head child-level
child-column)
                                                    :operation boole-op
                                                    )
                                     )
                           )
                  (tree-draw-links child)
                  )
                )
             )
    )
  )
```

```
;;; Tree-Calc-Node-Tail -- Compute the position of the tail of a node and
;;;                        return it as a Lucid Lisp Position structure
;;;                        suitable for the drawing routines.

(defun Tree-Calc-Node-Tail (node-level node-column)
  (let ((node-tail-x 0)
          (node-tail-y 0)
          )

    (setf node-tail-x (- (* (+ node-level 1)
                                      *tree-cell-width*)
                                   (truncate (/ *tree-cell-width* 2))
                                   )
             )
    (setf node-tail-y (- (* node-column
                                      *tree-cell-height*)
                                   (truncate (/ *tree-cell-height* 2))
                                   )
             )
    (make-position node-tail-x node-tail-y)
    )
  )

(defun Tree-Calc-Node-Head (node-level node-column)
  (let ((node-head-x 0)
          (node-head-y 0)
          )
    (setf node-head-x (* node-level
                                   *tree-cell-width*)
             )
    (setf node-head-y (- (* node-column
                                      *tree-cell-height*)
                                   (truncate (/ *tree-cell-height*
                                                      2))
                                   )
             )
    (make-position node-head-x node-head-y)
    )
  )

;;; Draw-Part -- Draws the nodes and the links for any sub-tree of the network

(DEFUN DRAW-PART
       (NODE)
       (CLEAR-BITMAP *TREE-WINDOW-HANDLE*)
       (TREE-DRAW-DIAGRAM NODE)
       (TREE-DRAW-LINKS NODE))
```

```
;;; -----------------------------------------------------------------------
;;; DCRL-Load
;;;           Loads a DCRL description file after clearing the contents of the
;;;           currently-defined universe.
;;;
;;; NOTE: THIS FUNCITON DELETES THE ENTIRE CONTENT OF THE EXISTING DCRL UNIVERSE!
;;;-----------------------------------------------------------------------

(defun DCRL-Load (filename)
  (DCRL-Clear-Universe)
  (load filename)
  (DCRL-Browse *tree-root-node*)
)

;;; -----------------------------------------------------------------------
;;; DCRL-Clear-Universe
;;;             Remove all concepts from the current DCRL Universe
;;; -----------------------------------------------------------------------

(defun DCRL-Clear-Universe ()
  (let ((node nil))
    (dolist (node *concepts*)
      (destroy-concept node)
      )
    )
  )

;;; -----------------------------------------------------------------------
;;; CLS
;;;           Clear the Diagram window and *tree-window-handle*
;;; -----------------------------------------------------------------------

(defun CLS ()
  (leave-window-system)
  (setf *tree-window-handle* nil)
  )

;;; -----------------------------------------------------------------------
;;; Tree-Set-Display-Size
;;; -----------------------------------------------------------------------

(defun Tree-Set-Display-Size (size)
  (cond ( (equal size 'small)
          (setf *Tree-Diagram-Size* 'small)
          (setf *Tree-Cell-Height* 8)
          (setf *Tree-Cell-Width* 100)
          (setf *Tree-Display-Font* (find-font 'small-roman))
          )

        ( (equal size 'medium)
          (setf *Tree-Diagram-Size* 'medium)
          (setf *Tree-Cell-Height* 15)
          (setf *Tree-Cell-Width* 200)
          (setf *Tree-Display-Font* (find-font 'small-roman))
          )

        ( (equal size 'large)
          (setf *Tree-Diagram-Size* 'large)
          (setf *Tree-Cell-Height* 22)
          (setf *Tree-Cell-Width* 300)
          (setf *Tree-Display-Font* (find-font 'bold-roman))
          )
        )
  )

(Tree-Set-Display-Size 'small) ;;; Initialize the display to small.
```

Appendix 5 -- Scoops Evaluation

A brief evaluation of SCOOPS, the PC-SCHEME OOPS package was conducted on personal computers before the VAXStation system was in place in order to evaluate the feasibility of an object-oriented approach to programming for K-Base. Implementation of a problem familiar to computer science students, the "Towers of Hanoi" problem, was undertaken in order to provide a familiar basis for review of the programming platform. Readers not familiar with this programming example may refer to almost any computer science textbook which introduces the concept of recursion.

The familiar "Towers of Hanoi" problem can be solved using an Object-Oriented paradigm in the following fashion:

The problem can be decomposed into a series of interactions between objects representing Disks, Pegs, and the Game. Multiple instances of Disks and Pegs are required; multiple instances of the game may optionally co-exist (in different screen windows, for example).

Game play operates as follows: A game is instantiated and a PLAY-GAME message (for a specified number of disks) is sent to it requesting it to begin play. For purposes of discussion, this instance of Game will be called myGame.

MyGame will then create instances of the three required pegs and as many disks as required. (The environment contains default names for the first 12 disks and the Source, Intermediate, and Destination pegs.) INITIALIZE messages are sent to the three pegs requesting that they initialize and draw themselves. RESERVATION messages are then sent to the source peg requesting that it send an INVITATION to each disk from the largest to the smallest in succession in order to initialize object data structures and the screen display.

Implementation of the requisite game moves is as follows: When myGame wishes to cause a move of a disk to a particular peg, a MOVE message is sent to that disk with the name of the destination peg. The disk, knowing its current peg, undraws itself and sends a GOODBYE message to that peg, causing the peg to adjust its data structure and redraw its vacated portion. The disk then sends an RESERVATION message to the target peg, informing it of the disk's pending arrival. The peg then computes the destination address of the top of its pile and sends an INVITATION message containing those coordinates to the disk. Upon receipt of the INVITATION, the disk adjusts its data structure and draws itself at the proper location. This process continues until myGame completes.

Implementation

Implementation of this approach requires the following class definitions and methods:

Class GAME:

Instances:	myGame
Instance Variables:	Num-Disks
Methods:	Play-Game(Num-Disks)

Class DISK:

Instances:	Disk1..Disk12
Class Variables:	Height
Instance Variables:	Width X-Pos Y-Pos Color On-Peg
Methods:	Move(peg-name) Invitation(X-Pos,Y-Pos)

Class PEG:

Instances:	Source, Intermediate, Destination
Class Variables:	Height Width
Instance Vairables:	Base-X Base-Y Color Top-Level
Methods:	Initialize Reservation(Disk-ID) Goodbye

```
;;; Load the SCOOPS environment

(fast-load "scoops.fsl")

;;; Define names for objects to be used

(define peg1 '())
(define peg2 '())
(define peg3 '())

(define disk1 '())
(define disk2 '())
(define disk3 '())
(define disk4 '())
(define disk5 '())
(define disk6 '())
(define disk7 '())
(define disk8 '())
(define disk9 '())
(define disk10 '())
(define disk11 '())
(define disk12 '())

;;; Define the GAME class

(define-class Game
                 (instvars (Num-Disks 0)
                 )
)

;;; Methods for class Game

;;; Method Game Play-Game is used to start play of a game.
;;; It instantiates three pegs and n disks. The pegs are
;;; initialized, and the disks are moved to peg 1. Function
;;; PLAY-HANOI is then called to implement the game logic
;;; and send appropriate messages to the disks.

(define-method (game play-game) (n)
    (eval(list 'set! (make-peg-name 1) (make-instance peg)
    (eval(list 'set! (make-peg-name 2) (make-instance peg)
    (eval(list 'set! (make-peg-name 3) (make-instance peg)
    (set! Num-Disks n)
    (make-disks n)
    (set-video-mode! 4)
    (clear-graphics)
    (set-palette! 1 1)
    (draw-box -159 -89 159 -81 3)
    (send (eval (make-peg-name 1)) initialize 1)
    (send (eval (make-peg-name 2)) initialize 2)
    (send (eval (make-peg-name 3)) initialize 3)
    (init-disks n)
    (gc T)
    (play-hanoi n 1 2 3)
    (gc T)
```

```
    (set-video-mode! 3)
)

;;; Utility function MAKE-DISKS is called to instantiate n
;;; disks.

(define make-disks
    (lambda (n)
        (cond
            ((zero? n) '())
            (else (eval (list 'set!
                                                                    (make-
                    disk-name n)
                                                                    (make-
                    instance disk 'width n

                                            'color (1+ (remainder n
                    3)))
                                                        )
                                )
                                (make-disks (-1+ n))
            )
        )
    )
)

;;; Utility function INIT-DISKS is called to move disks to
;;; peg one.

(define init-disks
    (lambda (n)
        (cond
            ((zero? n) '())
            (else
                (send (eval (make-peg-name 1)) reservation n)
                (init-disks (-1+ n))
            )
        )
    )
)

;;; Utility function PLAY-HANOI implements the game logic.

(define play-hanoi
    (lambda (n s i d)
        (cond
            ((zero? n) '())
            (else
                (play-hanoi (-1+ n) s d i)
                (send (eval (make-disk-name n)) move d)
                (play-hanoi (-1+ n) i s d)
            )
        )
    )
)
```

```
(compile-class Game)

;;; Define class Disk

(define-class Disk
    (classvars          (height 10))
    (instvars           width
                           X-pos
                           Y-pos
                           (color 'white)
                           (On-Peg 1))
    (options
        (inittable-variables width color)
    )
)

;;; Methods for class Disk

;;; Method Disk Move is invoked by the Game when it is
;;; desired to move a disk to a new location.  The disk
;;; erases itself from its current location, says goodbye to
;;; its current peg and makes a reservation on the target
;;; peg.

(define-method (Disk Move) (peg-num)
    (draw-box      (- X-Pos (* Width 3) 5) Y-Pos
                         (+ X-Pos (* Width 3) 5) (+ Y-Pos 9) 0)
    (send (eval (make-peg-name On-Peg)) goodbye)
    (send (eval (make-peg-name peg-num)) reservation width)
    (set! On-Peg peg-num)
)

;;; Method Disk Invitation is invoked by a peg when the disk
;;; is invited to move itself to that peg. The disk sets its
;;; X,Y position according to the invitation and draws
;;; itself on the target peg.

(define-method (Disk Invitation) (x y)
    (set! X-pos x)
    (set! Y-pos y)
    (draw-box      (- X-Pos (* Width 3) 5) Y-Pos
                         (+ X-Pos (* Width 3) 5) (+ Y-Pos 9) color)
)

;;; Utility function MAKE-DISK-NAME constructs a disk name
;;; from its identifying number.

(define make-disk-name
    (lambda (n)
        (string->symbol
            (string-append "DISK" (integer->string n 10))
        )
    )
)
```

```
(compile-class Disk)

;;; Define class Peg

(define-class Peg
    (classvars      (height 130)
                          (width 10)
    )
    (instvars       Base-X
                          (Base-Y -80)
                          (color 3)
                          (Top-Level 0)
    )
)

;;;
;;; Define methods for class Peg
;;;

;;; Method Peg Initialize draws the peg and initializes its
;;; Top-Level

(define-method (Peg Initialize) (n)
    (set! Top-Level 0)
    (set! Base-X (- (* (- n 1) 100) 100))
    (draw-box       (- Base-X (/ width 2)) Base-Y
                          (+ Base-X (/ width 2)) (+ Base-Y height)
                          color
    )
)

;;; Method Peg Goodbye is invoked by the disk when leaving
;;; the peg. The peg adjusts its Top-Level and re-draws its
;;; vacated portion.

(define-method (Peg Goodbye) ()
    (set! Top-Level (-1+ Top-Level))
    (draw-box       (- Base-X (/ width 2)) (+ Base-Y (* Top-
    Level           10))
                          (+ Base-X (/ width 2)) (+ Base-Y (* Top-
    Level           10) 9) color)
)

;;; Method Peg Reservation is invoked by a disk arriving at
;;; the peg.  The peg adjusts its top level and sends the
;;; disk the location of the top of its pile.

(define-method (Peg Reservation) (d)
    (send
        (eval (make-disk-name d))
        Invitation Base-X (+ Base-Y (* Top-Level 10))
    )
    (set! Top-Level (1+ Top-Level))
)
```

```
;;; Utility function MAKE-PEG-NAME creates a peg name given
;;; its number.

(define make-peg-name
    (lambda (n)
        (string->symbol
            (string-append "PEG" (integer->string n 10)))
    )
)

(compile-class Peg)

;;; Utility function DRAW-BOX draws a box filled with COLOR
;;; from (X1,Y1) to (X2,Y2)

(define draw-box
    (lambda (x1 y1 x2 y2 color)
        (position-pen x1 y1)
        (set-pen-color! color)
        (draw-filled-box-to x2 y2)
    )
)

(define myGame (make-instance game))
```

Appendix 6: Multi-User Files Modified for KB/FMS

This section lists and briefly describes changes to components of the Multi-User PLAID software system. The filenames listed are in the directory

Directory DONALD$DUA1:[PLAID.CMSMULTI]

ACCESS_FILE_IN_DOMAIN.FOR

This file contains a routine which allows a PLAID module to open a file in a specified domain. ACCESS_FILE_IN_DOMAIN opens files for READ access only.

CONTEXT_COLLECT_PARTS.FOR

This file contains a routine which collects the names of all files in the current context which match a given file specification. The file specification may include the normal VMS wildcard characters. This routine will collect the names of files which are not PLAID part files as well as those that are.

CONTEXT_DISPLAY_PARTS.FOR

A routine used to display the list of files generated by CONTEXT_COLLECT_PARTS.

CONTEXT_LIST_PARTS.FOR

A routine that collects and displays all filenames in the current context which match a given file specification. CONTEXT_LIST_PARTS uses CONTEXT_COLLECT_PARTS to generate the list of files and CONTEXT_DISPLAY_PARTS to display the list.

CREATE_FILE.FOR

CREATE_FILE was modified to recognize and create the five new file types that have been added to the Multi-User PLAID.

The new file-types are:

TRE -- tree files generated by DMC.
PDF -- Primitive description files.
CDF -- COG description files.
TDF -- Target description files.
DDF -- Display description files.

CREATE_PARTS_LIST_FILE.FOR

Creates an indexed file for collecting a list of filenames. This is one of a suite of routines which include:

- FIND_PARTS_LIST_ENTRY.FOR
- READ_PARTS_LIST_ENTRY.FOR
- REWRITE_PARTS_LIST_ENTRY.FOR
- WRITE_PARTS_LIST_ENTRY.FOR

DIR_MANIP_FAST.FOR

A new implementation of DIR_MANIP which uses direct calls to system services to collect file names rather than spawning a DCL "DIRECTORY" command. This implementation is a little faster than the original one.

DOMAIN_IN_CONTEXT.FOR

A function which determines whether or not a given domain is in the current context. This is essentially an adaptation of the "List Context Structure" (LCSF) command.

FIND_PARTS_LIST_ENTRY.FOR

A function which, given a part name, performs a lookup in the current parts-list file to retrieve information on that file. See CREATE_PARTS_LIST_FILE for names of related routines.

FORTRAN_REGX.FOR

A collection of routines to provide convenient FORTRAN access to C routines "fsearch", "fselect", "regcomp" and "regexp". These routines are as follows:

MATCH_FILE -- Called by KBASE_DESC_CONTEXT and KBASE_DESC_GLOBAL to provide an interface to C routine "fsearch". Provides information for K-Base description reports.

SELECT_FILE -- Called by KBASE_DESC_CONTEXT and KBASE_DESC_GLOBAL to provide an interface to C routine "fselect". Selects files for processing by MATCH_FILE.

GET_FIELD -- Access routine to return the contents of a named field from a K-Base description file.

APPEND_FIELD -- Appends a new attribute field to an extant K-Base description file.

REPLACE_FIELD -- Changes the value of an attribute in an extant K-Base description file.

CHAR_TO_ASCIZ -- Converts a FORTRAN character string to a C style ASCIZ character string.

LINE_TO_ASCIZ -- Converts a FORTRAN character string to a C style ASCIZ character string terminated by a newline.

GET_TEXT_STRING -- Reads a record from a K-Base description file and returns the information as a C-style ASCIZ character string.

PUT_TEXT_STRING -- Writes a C-style ASCIZ character string to a K_Base description file.

FMATCH.C

A function which searches a K-Base description file for aN attribute name matching a UNIX-style regular expression. The function return value is the number of matches encountered.

FSEARCH.C

A function which searches a K-Base description file for a attribute name matching a UNIX-style regular expression. Attribute/value pairs which meet matching criteria are written to the specified output channel.

GET_ASSOC_FILENAME.FOR

A function which, given a PLAID filename returns the logical value .TRUE. if there is an associated filename and false if there is not. If .TRUE. is returned, the associated filename is also returned via the second parameter. This function is used to perform the mapping from part names to description file names AND description file names to part names; a bidirectional association.

NOTE: this function does NOT check to see if the associated part file exists. It just determines whether or not the given part type has an associated file type by performing a table lookup.

GET_DESC_FILENAME.FOR

A function which, given a PLAID filename returns the logical value .TRUE. if there is an associated filename and false if there is not. If .TRUE. is returned, the associated filename is also returned via the second parameter. This function is used to perform the mapping from part names to description file names AND description file names to part names; a bidirectional association.

NOTE: this function does NOT check to see if the associated part file exists. It just determines whether or not the given part type has an associated file type by performing a table lookup.

GET_FILE_DATES.FOR

A routine which returns the creation date, backup date, and last revision date for the last file opened by ACCESS_FILE or PM9CM.

NOTE: this routine should be called immediately after the file open operation because subsequent open operations will over- write the information stored in the /RMS_FILE_ID/ common block.

GET_FILE_IFS.FOR

A routine which returns the VMS Internal File Specifier/ID of the last file opened by ACCESS_FILE or PM9CM. The IFS is used by the routine IFS_REOPEN to perform a very fast file open operation.

NOTE: this routine should be called immediately after the file open operation because subsequent open operations will over- write the information stored in the /RMS_FILE_ID/ common block.

GET_FILE_PROT.FOR

A routine which returns the VMS file protection attributes for the last file opened by ACCESS_FILE or PM9CM. See the "VAX Record Management Services Manual" section on XABPRO for the specifics of the returned protection vector.

NOTE: this routine should be called immediately after the file open operation because subsequent open operations will over- write the information stored in the /RMS_FILE_ID/ common block.

GET_NEXT_FIELD_NAME.FOR

A function which, given a comma-delimited list of Kbase field names, sequentially returns both the first element of the list (the LISP "(CAR list)") and the list minus the first element (the LISP "(CDR list")). The return value of the function is STS_CONTINUE if the get was successful or STS_SUCCESS if the list has been exhausted.

GET_NEXT_FILE_SPEC.FOR

A function which, given a comma-delimited list of file names, returns the first file name in the list (the LISP "(CAR list)") and the list minus the first file name (the LISP "(CDR list)"). The return value of the function is STS_CONTINUE if the get was successful or STS_SUCCESS if the list has been exhausted. GLOBAL_COLLECT_PARTS.FOR

A routine used to construct a list of all files in a given subtree of the Multi-User system which match a specified file name expression (using DCL wildcards). This routine is used by GLOBAL_LIST_PARTS and KBASE_DESC_PARTS to collect file names for further processing.

GLOBAL_DET_LEVEL.FOR

A function which determines the part "occurrence level" of each file listed in a parts-list file generated by GLOBAL_COLLECT_PARTS. The occurrence level is essential to the correct determination of which files hide other files in the Multi-User environment.

GLOBAL_DISPLAY_PARTS.FOR

A routine which simply displays the list of files generated by GLOBAL_LIST_PARTS.

GLOBAL_LIST_PARTS.FOR

The routine which implements the Multi-User "FIND/GLOBAL ..." command. This routine collects all occurrences of a given file specification in an entire subtree of the Multi-User context structure by performing a top-down (pre-order) traversal of the subtree.

KBASE_DESCRIBE_PARTS.FOR

A routine which, given a parts-list file, a list of field-names, and a list of content strings, determines which description files contain a field-name and content string match. This is the report generation program for the Multi-User commands "DESCRIBE/GLOBAL ..." and "DESCRIBE/CONTEXT ...".

KBASE_DESC_CONTEXT.FOR

The routine which implements the Multi-User "DESCRIBE/GLOBAL ..." command. This program creates a parts-list file, collects all the files which match a given file specification, and generates a report for each file whose description file contains a given field-name/field-content match.

KBASE_DESC_GLOBAL.FOR

The routine which implements the Multi-User "DESCRIBE/CONTEXT ..." command. This program creates a parts-list file, collects all the files which match a given file specification, and generates a report for each file whose description file contains a given field-name/field-content match.

KBASE_FIND_CONTEXT.FOR

The routine which implements the Multi-User "FIND/CONTEXT ..." command. This program uses CONTEXT_LIST_PARTS to collect and display the file names which match a given file specification within a given Multi-User context.

KBASE_FIND_GLOBAL.FOR

The routine which implements the Multi-User "FIND/global ..." command. This program uses GLOBAL_LIST_PARTS to collect and display the file names which match a given file specification within a given subtree of the Multi-User environment.

KBASE_FORMAT_CLOSE.FOR

The routine which implements the Multi-User "FORMAT/CLOSE ..." command. This procedure simply closes the report file used by the "DESCRIBE/GLOBAL ..." and "DESCRIBE/CONTEXT ..." commands.

KBASE_FORMAT_OPEN.FOR

The routine which implements the Multi-User "FORMAT/OPEN ..." command. This procedure opens the specified report files and stores information about the fields desired in the report. The list of field-names is maintained for use by KBASE_DESCRIBE_PARTS to determine which field contents are written to the report.

KBASE_FORMAT_SHOW.FOR

The routine which implements the Multi-User "FORM/SHOW ..." command which simply shows the user the name of the report file and the field-names which will be reported upon.

KBASE_INIT.FOR

A routine which simply initializes the Kbase report generation environment.

MU9INQ.FOR

A suite of subroutines for obtaining information about the current process from the Multi-User system. The subroutines in this suite are:

Mu9Inquire

A routine which acquires information about the current process from the Multi-User Monitor process and stores

Mu9AskMulti

A routine which performs an inter-process communication with the Multi-User Monitor (a detached process) to collect all relevant information about the calling process.

Mu9GetAccess

A routine which returns the Multi-User priviledge list. The priviledge for a given user determines which Multi-User commands that user may perform. Note: always call Mu9Inquire before calling this routine.

Mu9GetAccount

A routine which returns the Multi-User account name for the invoking process. Note: always call Mu9Inquire before calling this routine.

Mu9GetProject

A routine which returns the current Multi-User project for the invoking user. Note: always call Mu9Inquire before calling this routine.

Mu9GetUic

A routine which returns the VMS UIC of the invoking user. Note: always call Mu9Inquire before calling this routine.

PUT_FRT_TO_DIR_FAST.FOR

A routine which adds foreign references to the parts-list file so that they will appear in the search listings.

READ_PARTS_LIST_ENTRY.FOR

A function which reads the next sequential entry from a specified parts-list (indexed) file.

REGERROR.C

Routine called to handle errors in regular expression compilation.

REGEXP.C

Contains routines used to match UNIX-style regular expressions against character strings. Contains routines "regcomp" and "regexec", and routines which they reference.

regcomp Compiles a character string containing a regular expression into an internal representation of that regular expression.

regexec Matches a compiled regular expression from "regcomp" against a text string.

REGSUB.C

Used to perform substitution of strings based on regular expression searches. Provided for future use.

REWRITE_PARTS_LIST_ENTRY.FOR

A function which modifies the contents of a specified record in a parts-list file.

WRITE_PARTS_LIST_ENTRY.FOR

A function which writes an entry into a parts-list file.

NEW COMMON BLOCKS

KBASE.H
REGEXP.H
REGMAGIC.H

MODIFIED COMMON BLOCKS

ACCACTION_INI.H
ACCEXTEN.H
ACCEXTEN_INI.H
ACCPARMS.H
ACCPARMS_INI.H
ACCSEARCH_INI.H

MODIFIED PROGRAMS

COMMAND_PROC.FOR
GET_PARM.FOR
INITIALIZE_USER.FOR
LIST_CSF.FOR
PARSEDR.FOR
PUT_FILE_IFS.FOR
REOPEN_FILE.FOR
IFS_REOPEN.MAR
WAIT_M.FOR

MODIFIED COMMAND PROCEDURES

MULTIUTIL.CMP
MULTIUTIL.LIB

Bibliography for K-Base

Bobrow, Daniel G., and Allan Collins, Representation and Understanding, New York: Academic Press, 1975 [UTSA lib: BF311.R388]

Bobrow, Daniel G., Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd, "GUS, A Frame-Driven Dialog System", Artificial Intelligence 8(2), 1977 [UTSA: Q335.A785]

Brachman, Ronald J., "On the Epistemological Status of Semantic Networks", in Associative Networks - Representation and Use of Knowledge by Computers, edited by Nicholas V. Findler, New York: Academic Press, 1979 [see also: Findler 1979]

Brachman, Ronald J., and James G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", Cognitive Science 9, pp 171-216, 1985

Brooks, Rodney A., "Symbolic Reasoning among 3-D Models and 2-D Images", Artificial Intelligence 17, August 1981. Also in Computer Vision, edited by J. Michael Brady, North-Holland, Amsterdam, 1981

Bundy, Alan, "Will it Reach the Top? Prediction in the Mechanics World", Artificial Intelligence 10, 1978, pp 129-146

Cox, Brad j., "Message/Object Programming: An Evolutionary Change in Programming Technology", IEEE Software, Jan 1984, pp 50-61

Date, C. J., An Introduction to Database Systems, Second Edition, pub. Addison-Wesley 1977

Findler, Nicholas V. (editor), Associative Networks - Representation and Use of Knowledge by Computers, New York: Academic Press, 1979 [UTSA: Q360.A87]

Funt, V. Brian, "Problem-Solving with Diagrammatic Representations", Artificial Intelligence 13(3), 1980

Goldberg, Adele and David Robson, Smalltalk-80: The Language and Its Implementation, Copyright 1983 Xerox Corp., Pub. Addision-Wesley 1983

Hailpern, Brent, "Multiparadigm Languages", IEEE Software, January 1986, pp. 6-9

Hewitt, Carl E., P, Bishop, & R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence", IJCAI 3, 1973, pp 235-245 [UTSA: N/A]

Hewitt, Carl E. "Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence 8(3), 1977

Kahn, Kenneth, and Anthony G. Gorry, "Mechanizing Temporal Knowledge", Artificial Intelligence 9(1), 1977

Jorgensen, Charles, William Hamel, and Charles Weisbin, "Autonomous Robot Navigation", Byte Magazine, Jan. 1986, pp 223-235

Kuipers, Benjamin, "A Frame for frames: Representing Knowledge for Recognition", in Representation and Understanding, edited by Daniel G. Bobrow and Allan Collins, New York: Academic Press, 1975 [UTSA: BF311.R388]

Lee, Kunwoo, and David C. Gossard, "A Hierarchical Data Structure for Representing Assemblies: Part 1", Computer-Aided Design 17(1), pp 15-19, 1985

Lee, Kunwoo, and Guy Andrews, "Inference of the Positions of Components in an Assembly: Part 2", Computer-Aided Design 17(1), pp 15-19, 1985

Levesque, Hector and John Mylopoulos, "A Procedural Semantics for Semantic Networks", in Associative Networks - Representation and Use of Knowledge by Computers, edited by Nicholas V. Findler, New York: Academic Press, 1979, pp 93-119

Mackworth, Alan K., "Interpreting Pictures of Polyhedral Scenes", Artificial Intelligence 4(2), 1973

Pascoe, Geoffrey A., "Elements of Object-Oriented Programming", Byte Magazine, Aug. 1989, pp 139 ff

Schubert Lenhart K., Randolph G. Goebel, and Nicholas J. Cercone, "The Structure and Organization of a Semantic Net for Comprehension and Inference", Associative Networks, 1979, New York: Academic Press, pp 121 ff

Stefik, Mark J., Daniel G. Bobrow, and Kenneth M. Kahn, "Integrating Access-Oriented Programming into a Multiparadigm Environment", IEEE Software, January 1986, pages 10-18

Stroustrup, Bjarne, The C++ Programming Language, Addison-Wesley, 1986

Winograd, Terry, "Frame Representations and the Declarative/Procedural Controversy", in Representation and Understanding, edited by Daniel G. Bobrow and Allan Collins, New York: Academic Press, 1975

Winston, Morton E., Roger Chaffin, and Douglas Herrmann, "A Taxonomy of Part-Whole Relations", Cognitive Science 11, pp 417-444, 1987 [UTSA]

Winston, Patrick, "Learning by Creating and Justifying Transfer Frames", Artificial Intelligence 10(2), 1978

Winston, Patrick, Artificial Intelligence, 2nd Ed., Reading MA: Addison-Wesley, 1984

Woods, William A., "What's in a Link: Foundations for Semantic Networks", in Representation and Understanding, edited by Daniel G. Bobrow and Allan Collins, New York: Academic Press, 1975