Structured Languages SIG

DECUS U.S. CHAPTER

THE HEAP

STRUCTURED LANGUAGES

DECUS

DECUS MUMPS SIG

THIS IS THE FIRST

SUBSCRIPTION SERVICE ISSUE

Printed in the U.S.A.

STRUCTURED LANGUAGES/MUMPS

This first edition of the combined MUMPS and Languages and Tools Newsletter features the Languages and Tools SIG. Information on submitting material to "The Heap" can be found on page 1. To submit articles to the MUMPS vehicle, "The Tree", kindly send your material to:

Jim Bernard
Data Processing
Kettering Medical Center
3935 Southern Boulevard
Kettering, OH 45429

THE HEAP

Languages and Tools SIG Newsletter

Volume 7, Number 1 September 1983

From the Editor

This is the first paid issue of the "Heap". I have tried to include as much as possible so you will feel that you are getting the most for your subscription fee. You may want to show it to your friends who are not subscribing to get them interested. The majority of this newsletter comes from articles submitted by members of the LTSIG steering committee. We now have a core group that really wants to make the SIG a worthwhile entity within DECUS. It will take help from other people who are interested in Languages and Tools to keep the newsletter as good as it is this time. I would like to see articles from users of any language or tool. Applications of the C language package, how you are using CMS or MMS, why your company has chosen to use Pascal, etc. are all viable topics. I would like to have articles submitted on magnetic tape (800 or 1600 bpi, ANSI labled (VAX/VMS copy), or *Unix tar tape) in DSR (Digital Standard Runoff) format. This makes it easier for me to include them in the newsletter in a standard format. I will take paper copies as long as they are a good quality copy with margins similar to this newsletter (one inch left and right margins). In the future we hope to be able to utilize the DECUS VAX and the LN01 Laser printer to produce even better copy. With the subscription service in effect you should be seeing a change in the quality of newsletters.

In this newsletter, the members of the LTSIG steering committee have identified what they are responsible for and what they are doing in the real world. Kathy Hornbach has submitted the contents of her handouts from the Spring DECUS Symposium (Software Development Environment Plan and Creating a Software Development Environment Plan) as well as a report from the recent Softfair Conference. She has more for the next newsletter on how Lear Siegler is using languages and tools in a production environment. I have also included several of the Scribe reports from the 1983 Spring DECUS Symposium (see editors note before the articles). In addition, Shava Nerad's "Tailoring EDT for the Structured Language Programmer" is included. Let me know if these are the types of articles you would like to see in the future.

**UNIX is a Trademark of Western Electric Corporation*

With all that is in this newsletter, what will be coming next? Look for an article from Kathy Hornbach on the production of the "Tools Sources" booklet. Kathy also wants to create a dialog in the newsletter regarding the use and abuse of tools. You might want to get in touch with Kathy if you have or know of any software tools that we should include in her booklet.

Bits and Pieces

Several items have come in my mail that you might be interested in reading. If you have any short items that may be of interest to our readers, send them along and I will include them in this section in future newsletters.

- There is a company in Utah (Diser,Inc., 385 East 800 South, P.O. Box 70, Orem, Utah, 84057, (801) 227-2300) which is working with Modula-2. They are producing Wirth's Lilith computer system and are interested in hearing about any applications of Modula-2 or software tools developed with the Modula-2 language.

- 1983 Fall DECUS US Symposium will be held October 24-28, 1983, at the MGM Grand Hotel in Las Vegas, Nevada. Note that LTSIG is sponsering two pre-symposium seminars (Ada Programming Methodology by Ken Bowles and Software Engineering Techniques and Tools by John Barr).

- LTSIG library programs are maintaining their position in the top ten most frequently ordered library programs list. The C Language System, Pascal Compiler Version October 1981, and the Spring 1980 Pascal Symposium Tape were recent top ten favorites. Any comments??

- ANSI X3 is looking for members of a programming languages study group. Individuals involved with planning functions and those concerned with state-of-the-art development of languages would be most welcome and should contact Dr. JAN Lee, Chair, PL/SG, Virginia Tech, Department of Computer Science, Blacksburg, VA, 24001, (703) 961-6931.

- Those of you interested in PL/I may be interested in another ANSI X3 item. They have published an information bulletin to give the PL/I producer and usr communities an indication of the direction and current status of their current PL/I standards review work. You may obtain a copy of the document by sending ten dollars and a self-addressed mailing label to: PL/I-IB, X3 Secretariart, CBEMA, 311 First Street, NW, Suite 500, Washington, DC, 20001. This is your advance notice of the new standards effort. Don't be caught unaware like the COBOL and BASIC people seem to be at this time.

- Richard Coleman, Kajima International, Inc., 2100 North Central Road, Fort Lee, New Jersey, 07024, writes that he has been seeking information on Modula-2 that would run on a VAX under VMS to allow him to upgrade from FORTRAN mainly for commercial and project cost control applications. He would like to see a paper on *Ada vs. Modula-2 for the average user and the implications in a DEC environment of the heavy government support of Ada. Any comments??

- Chuck Evans, Director of Data Processing, Times Publishing Company, Times Square, Erie, Pennsylvania, 16534, writes that he is trying to start an RMS wish list. He says that "Too many ISG wishes are responded to with the answer, 'Well, that's really an RMS problem.', and from there, the wish disappears." He is planning a session for Las Vegas, and would like to get input from anyone who has problems with RMS.

- I have sent in my name and address to COSMIC, NASA's Software Dissemination Center, to get on their DEC user mailing list. Depending on what comes in the mail, I will include information about what they have in future newsletters.

- DIGITAL is currently looking at alternative languages for future development projects. Such languages as Modula-2 and Prolog are possible candidates. What do you think???

Job description of Languages and Tools SIG Chair

The job of the SIG Chair is predominantly that of coordinator of all the SIG's activties. That is, the chair is the primary point of contact between the SIG's members and the other organizations with which it must interact. These other organizations include DECUS US Chapter office staff, Digital, and any affiliated SIGs. Historically, the SIG chair has been the individual ultimately responsible for the operation of the SIG.

In large SIGs, it is clear that a single person cannot handle the entire job; other members of the SIG volunteer to take on some of the work, and the Chair (happily) delegates to them responsibilities appropriate to their expressed interests. As a large SIG continues to function, and its members increasingly offer to take on more of the work, the various areas of responsibility become institutionalized through the SIG's operating procedures. What remains, at least implicitly, in my view, is the expectation that the SIG Chair is the owner of any responsibility not explicitly claimed by some volunteer. In particular, the SIG Chair remains the individual responsible for

**ADA is a Trademark of the Department of Defense* submitting to the DECUS U. S. Chapter an annual budget for the SIG, and the information required to relicense the SIG for the next year.

For the Languages and Tools SIG, I'm happy to say, there are now a goodly number of volunteers for the numerous areas of responsibility we have collectively identified. The holders of those jobs have included herein their own job descriptions; from these you can see that a small SIG's Chair has a great deal of work he or she could do.

Biographical Sketch of the Current SIG Chair

James W. Livingston, Jr.
Measurex Corporation
One Results Way
Cupertino, CA 95014
(408) 255-1500

As may be said of many of us in this industry, I identify myself as a retread. After earning B.S. and M.S. degrees in Mathematics, I attended graduate school in Psychology, and received a Ph.D. in that discipline. In reterospect, I must acknowledge that my most enjoyable activities were building my experimental apparatus and programming the various computers on which I did my research.

After accepting a job offer at Livingston College of Rutgers University, I built my department a laboratory around a PDP-11/40, and served as its system manager, system programmer, and primary maintenance person. Some of you will recognize that there is the danger, in the situation I've described, that one's own research will be insufficient for the purposes of the tenure decision. Those who've not experienced the academic world from the junior faculty side may consider yourselves happily deprived.

As events materialized, I went to work for Digital Educational Services in 1976, after five years as an Assistant Professor of Psychology. At Digital, I taught courses in RSX-11M, IAS, and various other topics. As a Digital instructor, I received occasional job opportunities, one of which I finally took. I worked at Lawrence Berkeley Laboratories for a year, with responsibilities for VAX/VMS, DECNet, and Datatrieve. At the end of my year there, I joined Measurex, where I am today.

At Measurex, my responsibilities include supporting microprocessor development on MX's engineering VAXen, and writing system and communication software for MX products. My interests include language translators and other development tools, particularly those which support the development of software products.

Operating System Coordinator

As coordinator of the Operating System interest area of the LTSIG steering committee, I would like to introduce myself to the membership of our SIG. My name is Alan Rizzuto, I work at the following address:

Alan Rizzuto
Emc Controls, Inc.
P.O. Box 242
Cockeysville, M.D. 21030

As you can see from my address I work at EMC CONTROLS, INC. which is a company that produces distributed process control systems. My function is a software design engineer for special software projects in the Applications Engineering department. My past experience consists of eleven years in electronics, instrumentation, and computer hardware and four years in the software realm. My software background consists of Systems Engineering (ie. programming and debugging at executive level MACRO 11) and programming in higher level languages. My language background is in Fortran, Basic, Macro 11, C, Pascal, and 6809 Assembly.

The function I am serving on the steering committee for the LTSIG is O/S coordinator, which entails being the liaison of information between each of the Operating System coordinators and the LTSIG steering committee. Following is a list of the O/S coordinators and their respective positions:

IAS Coordinator	open
RSTS Coordinator	open
RSX-11M/M+ Coordinator	Alan Rizzuto Emc Controls, Inc. P.O. Box 242 Cockeysville, MD. 21030 (301) 667-4800
RT-11 Coordinator	open
UNIX Coordinator	open
VAX/VMS Coordinator	Louise M. Wholey Measurex Corporation One Results Way Cupertino, CA. 95014 (408) 255-1500 x4452

I am pleased to be able to serve the membership of this SIG and welcome all comments and questions. Also if anyone is interested in filling one of the above open positions, please contact me either by phone or letter.

Alan Rizzuto

Menu Coordinator

Greetings:

My name is Al Folsom, and I'm currently serving as the menu coordinator for the Languages and Tools SIG. For those of you who are not familiar with the menu process as conducted by other SIGs, the menu is a means by which the SIG membership can communicate it's desires to DEC and/or the SIG leadership. Action items for DEC or the SIG to address are suggested by the membership, and collated into a questionaire. This questionaire is then submitted to the SIG membership for voting, to determine the popularity of the items listed. The implication, of course, is that the more strongly the SIG expresses itself on any point, the more likely DEC will be to heed it's desires. There's power in numbers!

My responsibilities include soliciting the menu items, collating them and constructing the menu, collecting the menu responses and reducing the data collected for presentation at a symposium and in the newsletter. Ideally, this will all be done on a yearly cycle.

In addition to the menu, I would like to work on other methods of making the SIG more responsive to its membership. If anyone has suggestions in this regard, or items for the next menu, I would appreciate hearing from you. My work address is:

Alan L. Folsom, Jr.
Enertec Inc.
19 Jenkins Avenue
Lansdale, Pa. 19446

Thanks for your help.

-Al Folsom

Symposia Coordinator

I am J.R. Westmoreland (James R.)., the symposia coordinator for the languages and tools SIG. My responsibility in this position is to make sure that the symposia are scheduled and set up according to the wishes of the SIG. I work for Utah Power & Light Company. I HAVE BEEN A MEMBER OF DECUS SINCE THE SIG WAS THE PASCAL SIG. My interests are compiler design and other language related topics.

If you need to contact me about a symposia related matter you can reach me by phone at: (O) 801-535-2387 (H) 801-262-5251 or write to:

ATTN: J.R. Westmoreland
Utah Power & Light
Systems & Computer Services rm 184
1407 W North Temple
Salt Lake City, Utah
84116

Member Services Coordinator
John R. Barr

The member services coordinator is responsible for the functioning of the activities of the LTSIG which provide services other than the menu and symposia. This includes user training (pre-symposia seminars), session notes, DECUS program library, and the newsletter. It is a catch-all grouping designed to keep the work load of the SIG chairman down to a reasonable level.

Following is a list of the current task assignments:

Librarian	James Triplett Intermetrics. Inc. 701 Concord Avenue Cambridge, MA 02138 (617) 661-1840
Session Notes	Mark Katz GTE Sylvania 77 A Street Needham, MA 02194 (617) 449-2000 x635

Newsletter Editor	John R. Barr Department of Computer Science University of Montana Missoula, MT 59812 (406) 243-2883
User Training	open

Please direct your questions about any of these positions to the appropriate person or to myself if you have any problems.

Productivity Tools Coordinator
Kathy Hornbach

I have just taken on the newly created position of Productivity Tools Coordinator for the Languages and Tools SIG. The title is loose enough that a good variety of things will fit underneath it; it seems to boil down to any tool/technique that improves the efficiency/quality of software production. This includes DEC products like CMS and MMS, and also productivity tools from other sources, like the Hughes/Berkley Software Tools. Examples of other tools that might fit under this heading are prettyprinters, static analysis tools, path coverage tools, configuration management tools, and requirements/design tools.

One of the first things we will be working on is putting together a "Tools Sources" booklet, that will list different tools available for DEC computers. If you have any questions or suggestions for this area of the SIG, feel free to write or call me at the below address.

When I am not working on DECUS activities, I am employed at Lear Siegler/Instrument Division. Our main product is aircraft instrumentation, including onboard navigation and flight management systems - we produce both the hardware and software. I am head of a group called "Software Tools, Database, and Standards", which corresponds quite closely with my DECUS role. We are fortunate to have most of DEC's software tools in-house (CMS, MMS, and all the VAX Information Architecture), along with other productivity tools from outside sources. These tools are in use by over 200 engineers developing realtime avionics and support software.

Kathy Hornbach
Lear Siegler/Instrument Div
4141 Eastern SE MS 121
Grand Rapids, MI 49508
(616)241-8800

Softfair Conference on Software Development Tools, Techniques, and Alternatives

Crystal City, VA
July 26 - 28, 1983

Kathy Hornbach
Lear Siegler/Instrument Division

The Softfair Conference brought together people doing state-of-the-art research in the areas of software development tools, methodologies, and environments. It was sponsored by the IEEE Computer Society, ACM Sigsoft, and the National Bureau of Standards. Over 500 people attended the conference. Well represented were:

- o the big computer companies - I saw several people each from IBM, DEC, HP, Apple.
- o the defense contractors - probably the largest overall percentage of attendees, not surprisingly.
- o the government - helped by the Washington, D.C location, there were many federal employees in attendance - both from the military, and from NSA, GSA and elsewhere.
- o the phone companies - quite a number of different phone companies sent representatives - evidence of the impact they hope to make in the software field
- o the universities - where much of this research is being done.
- o finally, and probably not surprisingly, the Japanese - (hoping to do to the software industry what they did to the automobile industry?) Not only were they there for observations; they gave many of the talks and had many of the exhibits.

There were two unique features to this conference that made it unlike any other, and especially valuable:

1. Big-screen demonstrations were given of several products. An ordinary VDT screen was enlarged with the help of some device, and projected onto a huge screen, capable of being seen by a room of several hundred. Hour-long, actual demonstrations of well-known (and not so well-known) products were given - a tremendously valuable tool in determining just how useful a given product is, and _much_ more revealing (often to the detriment of the product) than product brochures or sample outputs.

2. An exhibits area contained about three dozen booths, featuring both commercial and non-commercial products. Most were available for actual hands-on use by the attendees. This was an even more valuable experience for getting the "feel" for some of the tools.

The conference was organized in three parallel tracks - two concurrent papers, and a "big screen" demonstration. Almost all the sessions were of interest, which made for a lot of running back and forth between sessions. The papers were grouped as to topic; among the topics included in the conference were:

1. Verification, Validation, and Testing
2. Software Development Environments
3. Object-Oriented Programming
4. Software Development Management
5. Configuration Management and Maintenance
6. Rapid Prototyping and Application Generators
7. Syntax-Directed Editing
8. Debugging
9. Software Design
10. Software Development Data Management

In the area of Software Development Environments, there was much emphasis on providing tools to help automate the requirements and design phases. At least two companies were showing graphical tools to support the Yourdon Structured Analysis and Structured Design methodology - Boeing Computer Services demonstrated ARGUS, and Hughes/Ground Systems Group demonstrated ASAT and AIDES. The Hughes tools are currently being marketed; Boeing is considering marketing ARGUS.

Talks were given on static analysis and path coverage tools under the Verification, Validation and Testing sessions. Static analysis tools take source code as input, and perform various analyses on it. They can detect certain classes of bugs, such as mismatch in number or type of calling parameters, and variables that are used before being set. They also can provide extensive documentation about a program, such as calling trees and global variable cross-reference. Path coverage tools "instrument" the source (i.e. insert additional statements into the source without changing the logic), which is then compiled, linked and executed. The probe statments that were inserted put out data tracing execution. A final phase analyzes this probe data, and produces reports showing which portions of the code were and were not executed. General Research Corporation demonstrated a tool that performs both of these functions on Fortran and Cobol, called RXVP-80. Liverpool Data Research Associates gave a paper on a similar set of tools that work on a multitude of languages.

One session especially applicable to DEC customers was one called "Proto-Cycling: A New Method for Application Development Using Fourth Generation Languages", by P. Zajonc and K. McGowan from Information Builders, Inc. The basic thesis was that it is often better to give the user a quick prototype of the system, and let them find out what they do and do not like about it; rather than go through a long and expensive development cycle, only to find that what they told you they needed is really not what they want at all. This is now possible to do with fourth-generation languages. The application described in the paper used IBM equipment, but we have very successfully used Datatrieve, FMS, and DBMS for the same sort of thing.

DEC gave one paper at the conference, on the VAX Debugger and its internals.

E. Cristofor of Bell Labs gave a talk on the automatic generation of test drivers. The generation program takes as input a module that is to be unit tested, and outputs a driver for that module. The driver output contains initialization for the various input parameters and common variables, and displays the appropriate output results.

A panel discussion on "The Impact of Future Technologies" raised some interesting speculations. A wide range of possibilities were discussed; a consensus emerged in a few areas. These include:

- very, very high level, graphics oriented programming will become common as computing spreads to the masses. (Right now, the number of "programmers" is doubling every year, with the advent of personal computers)

- there will always be the need for more technical types to do the lower level work to support these very high level applications

- human factors studies/psychology are where most gains will come

I have covered only a small portion of what was presented at the conference. The conference proceedings (containing the papers presented at almost all the sessions) are available through the IEEE Computer Society, and probably through ACM as well. If you are interested in these topics, it would be a good investment.

As an editorial comment, the one thing that I wished I could have seen more of at a conference like this, is more <u>real</u> <u>world</u> tools. Many of the sessions given were on tools that were under development and had never actually been tried out; or tools that were in use only by their developers and maybe a couple others; or tools in use at universities but not in industry. While all the above mentioned are important, especially for the introduction of new ideas, nonetheless the type of tools that we are interested in getting in-house are well-tested, well-supported tools that have been in proven through use in REAL projects, by someone other than their developer. While some were present (such as DEC Debug, GRC RXVP-80, Hughes AIDES, among others), I had hoped to find more. Is it that, despite all the conference activity, no one is really marketing and using such tools? Or is it just that the practically minded people who use these tools in industry are busy working on their projects, and don't bother to give papers at conferences like these?

This is supposed to become an annual conference. I would recommend it to anyone who is interested in learning about leading edge techniques in the area of automated tools to support software development.

Selected Sections from the Lear Siegler/Instrument Division Software Development Environment Plan

Lear Siegler/Instrument Division
4141 Eastern SE MS 121
Grand Rapids, MI 49508

Presented at
DECUS Spring Symposium
St. Louis, MO
May 24, 1983
Kathy Hornbach

1.0 INTRODUCTION

A software development environment is a set of integrated, automated techniques for assistance in the software development process. A good software development environment can greatly increase both productivity and the quality of the final product. In fact, as projects get larger and more complex, a good software development environment becomes a neccessity.

Implementing a software development environment is much more difficult than simply walking out and spending a couple megabucks on state-of-the-art tools:

> "The task of creating effective environments is so difficult because it is tantamount to understanding the fundamental nature of the software processes. A specific environment does not merit the name unless it provides strong, uniform support for the entire process it is intended to facilitate; that is not possible unless the process is fully appreciated and understood". [1]

Before a really comprehensive software development environment can be implemented, we have to develop a much more refined software

[1] Osterweil, Leon; "Software Development Environments: The Next Five Years"; Computer; p. 36; April 1981

development methodology - to define exactly how we produce software; the specific steps taken and the specific documents produced. We are making good progress in this area - a number of people are attending seminars and studying the problem - and everyone is aware of the necessity.

This paper will describe some specific options and implementations of a software development environment in-house. This paper is divided into five sections:

1. The first section reviews the qualities of a good software development environment in more detail.

2. This is followed by a discussion of our current environment and the tools already in place.

3. The main part of the paper divides the software lifecycle into sections, and discusses the software development environment needed to support each phase, cumulatively building up a comprehensive environment.

4. Some general support tools, which are useful to both software development and management, are explored. Next-generation tools that we may be using in the near future are examined.

5. A summary is given of the topics covered, with specific recommendations on how to proceed with implementing a software development environment.

2.0 CHARACTERISTICS OF A GOOD ENVIRONMENT

A software development environment should aim for the following five characteristics: [3]

1. Breadth of scope and applicability - it must be able to support all phases of the software lifecycle; it must be able to support many different types of projects.

[3] Osterweil, op.cit., pp 37-40

2. User friendliness - an obvious requirement. This applies to both the individual tools and to the system as a whole - i.e. if it seems logical, cost effective and useful to do something on a computer, the software development environment should provide a tool to do it easily.

3. Reusability of components - the software development environment must be built of tool fragments rather than monolithic tools. Only fragments are flexible enough to provide an environment suitable for supporting different kinds of work and unexpected requirements.

4. Tight integration of tool capabilities - The tools must work together, with output from one tool being valid as input to another. Duplication of functions should be minimized.

5. Use of a central database - Projects must keep track of a tremendous amount of details, facts and relationships. This necessitates a central database that tools can work out of and into.

These were the basic considerations used while putting together an implementation of a software development environment; keep them in mind as the various parts of the implementation are discussed.

3.0 BACKGROUND

We have an excellent starting point for implementing a sophisticated software development environment. This is so for three main reasons:

1. The VAX operating system, VMS, provides a strong, versatile, user-friendly foundation to build the tools on.

2. DEC offers a number of extremely versatile software packages that work together; these tools will form much of the basis for the software development environment.

3. The VAX is widely used by researchers for state-of-the-art work; a majority of software development environments and tools are implemented on the VAX before any other machine (e.g. RXVP).

With our DECnet links, Datatrieve Query Language, Forms Management System, UNIX-based software tools, RXVP, plus several other valuable tools on order, we have a good solid basis on which to build a software development environment. These tools will be the basic building blocks for much of what follows and will be explained here in more detail.

3.1 Software Engineering Database

As mentioned earlier, the keystone in any software development environment is a Software Engineering Database....this is a mechanism by which things (<u>any</u> and <u>every</u> things) can be stored, recalled, and correlated.

From Osterweil: [4]

> "The final, and perhaps most important characteristic of an environment is that it be coordinated and focused by access to a central repository of information. If software projects can be thought of as coordinated efforts to gain and disseminate a highly structured body of knowledge about a problem and its solution, the progress of the project will best be assured and facilitated by capturing structure and disseminating that body of knowledge as faithfully and effectively as possible. Those considerations imply the use of a data base and encompassing information management system as the centerpiece of any environment."

Specific proposals for database <u>contents</u> will be made later. The general capabilities of the database to be used for the software development environment will be outlined here.

DEC's database package, <u>DBMS</u>, will be used for the software engineering database. DBMS is a CODASYL standard network database; it is very similar to the IDMS database recently installed on the IBM, which is also a CODASYL standard. DBMS is currently on order.

[4] Osterweil, ibid. p. 40

DBMS is fully integrated with the rest of what DEC calls its "Information Architecture", which includes a Common Data Dictionary, Datatrieve Query Language, and Forms Management System. These integrated facilities simplify the writing of database programs and the performing of ad-hoc queries.

The Software Engineering Database will be the hub of the software development environment; most tools will work into and out of it. The power and flexibility offered by DBMS and related tools should greatly reduce the time necessary to make the software development environment operational.

Other tools that will be used as building blocks in the creation of a software development environment are :

o <u>DATATRIEVE</u> - An easy-to-learn, easy-to-use data query and report generation language. It will interface both with DBMS and FMS; implementing parts of the software development environment with Datatrieve will be very quick; it can also be used for ad-hoc querying by users. Datatrieve is also callable from DEC languages.

o <u>FMS (FORMS MANAGEMENT SYSTEM)</u> - provides a simple method for creating fill-in-the-blank forms on the terminal. These forms are useful any time data entry is to be performed, and for user interfaces. FMS works with Datatrieve and DBMS.

o <u>CMS (CODE MANAGEMENT SYSTEM)</u> - is a brand-new DEC tool, now on order, for doing version control of software. CMS is based on the highly acclaimed SCCS (Source Code Control System) on UNIX. It keeps track of software modules by version/mod, and allows recreation of any past version of the software. Reasons for changes are also saved. Modules can be specified as member of a group, i.e. a release. This means that it is possible to track every change and the reason for that change and to recreate any version of a software module or an entire release. And it does all this in less than three times the storage needed for the original source.

o <u>UNIX-like TOOLS</u> - The Software Tools User's Group created and distributed a set of tools based on those available under UNIX. (UNIX is the premier example of a software development environment). We have a set of over 75 of these tools on the VAX. They perform diverse functions, and give us a powerful subset of the UNIX capabilities. An example of some tools are:

- DC - desk calculator
- ENCRYPT - encrypt a file
- KWIC - keyword-in-context index
- UNIQ - strip adjacent repeated lines from a file
- FIND - find all occurrences of a string in a file (includes very complex pattern matching)
- WC - count of number of words in a file

o <u>RXVP</u> - A powerful debugging/testing aid that performs static error detection on the source; produces documentation on the source; and performs path coverage analysis during execution.

o <u>COMMON DATA DICTIONARY</u> - A uniform method for keeping track of data about data; it keeps track of all record formats and field meanings for all data used by DBMS and Datatrieve.

o <u>RUNOFF</u> - a text formatter that converts free-form input into a formatted document (this document was created with RUNOFF).

o <u>DEBUGGER</u> - for VAX native languages, it it an extremely sophisticated interactive, source oriented debugger.

o <u>MAIL</u> - an electronic mail system

4.0 <u>PROPOSAL FOR A SOFTWARE DEVELOPMENT ENVIRONMENT</u>

The software development environment will be described in pieces, based on the software lifecycle. The software development process is usually broken up into the following phases:

o Requirements specification

o Architectural design

o Detailed design

- o Code and debug
- o Testing and validation
- o Maintanence

The software development environment for each of these stages will be explored separately. General-purpose support tools will also be discussed.

The following sections will define each of these steps more rigorously. The software development methodology currently used here at LSI/ID for that phase will be discussed. Finally, the tools and environment that could be used to support the step will be explained. The software development environment will be built cumulatively, based on previous steps, until a complete environment is in place. This environment will continue to evolve as new techniques and tools become available that will both reduce the lifecycle cost of software and improve the quality of the final product.

:

:

<a section of ~25 pages came next, and very specifically described each phase of the software lifecycle, and the development environment that should be provided for it>

:

:

4.1 General Support Tools

The tools described in the previous sections were aimed at performing a specific task at a specific point in the software lifecycle. There are a number of general tools, however, that are useful for many tasks at many different times. Some of them have nothing to do with software development whatsoever; they just help people work more efficiently. Some of the general support tools recommended include:

1. GRAPHICS - An easy-to-use, interactive graphics package for producing line and bar charts.

2. OFFICE AUTOMATION TOOLS - for increasing the productivity of the "knowledge worker". Office automation tools include things like electronic mail, electronic file cabinets, automated calendar and meeting scheduling. These tools can be used by everyone, not just the software types.

3. TEXT FORMATTING - A text formatter (word processor) can be an extrememly sophisticated document-generating tool, going so far as to incorporate graphics, mathematics and different fonts.

4. MANAGEMENT TOOLS - Many management aids are provided by the software development environment and database, simply by keeping track of the status of all parts of the software development. Other automated tools are also available.

Each of these areas will now be explored in more depth.

:

:

<the tools were described in more detail, ~5 pages>

:

:

5.0 CONCLUSION AND WHAT TO DO NEXT

We have made tremendous progress in the last few years in providing a good software development environment for project use. But software systems have increased in size and complexity even more rapidly. To stay competitive and produce a quality product, we must take steps to implement a modern software development environment.

The software development environment presented here gives the general idea of what such an environment would be like. It is recommended that the following items be acted on:

1. Develop and formalize a complete software development methodology, covering all steps in the software lifecycle. This is important for the development of good software, as well as necessary for providing automated support. The Yourdan structured methodology now being investigated is the type of thing needed. This is not an

easy task, but it is a <u>very</u> important one.

2. Set up a continuing in-house training program in the software development methodology and the software development environment. The best tools in the world are useless if not used or used incorrectly.

3. Get a terminal on every desk. With all the work being done on line, from documention to mail to managment tools, working without a terminal will be akin to working without pencil and paper. How much work would get done if an engineer had to go to a pencil room every time he wanted to write something? Or even if he had to share his pencil with the other guy in the cubicle? This applies to most managers, also, who will be using the terminal for management reports and mail.

4. Take a long, hard look at Office Automation, and see how it would benefit engineering. We have so many of the pieces already in place (computer, networks, sophisticated users) that tremendous payoffs may be possible with very little investment.

5. Along with (4), consider getting in the phenomonal Xerox STAR or DEC VAXstation. With the amount of technical documentation done in engineering, a tool of this power could provide great gains in productivity.

6. Along with (4) and (5), work on integrating Word Processing very closely with any office automation or professional work station.

7. Continue to invest funds to buy these productivity tools, and the hardware that will be needed to support them.

8. Continue contacts, via seminars and symposiums, with people doing research or other users who may have already solved problems we are facing. Knowledge of RXVP, UNIX Software Tools, Xerox STAR, TELL-A-GRAF, CMS all came from attending conferences.

Spring DECUS Symposium
St. Louis, MO

Creating a Software Development Environment Plan

Kathy Hornbach
Lear Siegler/Instrument Div.
Grand Rapids, MI
May 24, 1983

What is a Software Development Environment?

A Software Development Environment (SDE) is a set of integrated, automated tools and techniques for assisting in the software development process that uniformly cover all aspects of the software lifecycle.

What is a Software Development Environment Plan?

Software Development Environment Plan is a strategy for providing a SDE for your particular installation. It analyzes where you are, where you want to be, what is needed to get there, and the steps that should be taken.

(It's a way to get all the fantastic tools we've been hearing about.)

Background

LSI develops:

- o Large real-time avionics systems
- o Programs are several hundred subroutines
- o Twelve or more people on a project
- o Large "support software" systems to support custom computers
- o Assemblers, linkers, environment simulators
- o Host-target dichotomy.

Advantages of Having a Plan

- o Helps integrate the tools into a coherent "whole" o Forces recognition of tools as a valid area of concern

o Power of eventual persuasion

o Produces a "shopping list" of tools

o Helps organize your thoughts

o Can force a formalization of your software development methodology

Contents of a SDE Plan

o Describe what a SDE is, and why it is important

o Outline your current environment, and any "building blocks" (e.g. FMS) that are already in place

o Describe the desired environment

- Describe tools in terms of the way your software development takes place (or should take place)

- Match up tools to each step in the process.

Outline of our Current SDE Plan

*What an SDE Is; Why it is Important
What has been accomplished since last Plan
*Underlying tools and philosophy
*Tools for software development
   *tools for requirements
   *tools for design
   *tools for implementation
   *tools for testing
Tools for document production
Tools for project management
Tools for office automation
Tools for information managment
Non-software issues
*Long Range Possibilities
*Summary and Reccomendation

* - included in original plan

Our Experience with SDE Plan

o First plan issued July '82

o Things that happened because of the plan: (Some were a direct result of plan; many were previous ideas that gained wider acceptance because of the plan).

- Switched to formalized software development methodology
- Obtained CMS, DBMS, CDD, VMSCAI, upgrade to DTR and FMS
- Wrote
    - Requirements Traceability Database
    - Requirements and design data dictionaries
    - Automated module documenting system
    - RUNOFF <--> IBM OS/6 translator.
    - Realtime test station/environment simulator

Other Results

o Tools are now included in annual budgetting process

o Users EXPECT tools and are willing to fund them

o Planned purchase of managerial workstations for department and division heads

Hints

o ADVERTISE your successful tools - put out memos saying how much time/money they have saved

o TELL people how using you were able to get something done for them in time (or ahead of time) because of TOOLX

o TEACH classes on tools and be there to answer questions; the best tools in the world won't help if they are unused

o CITE articles showing other firms' successes with tools that you would like to obtain

o GET users to help by telling them how their request could be fufilled sooner/better if you obtained a certain tool

o SOLICIT inputs from the people who will be using the tools - they will accept the plan much easier if they feel they helped shape it.

o TAKE advantage of free tools - a good portion of an environment can be gotten for media charge from universities, Uncle Sam, STUG, etc. Many of our most valuable tools were gratis.

Shopping List

o Sophisticated text formatter/typesetter

o Automated support for structured analysis and structured design

- Complexity measurement for Fortran-77
- Global data flow analysis for Fortran-77
- Conferencing
- Execution profiler for Fortran-77

1983 SPRING SYMPOSIUM REPORTS

The following session reports were produced by the newly initiated scribe service sponsered by the Publications Committee and spearheaded by the one and only Ralph Stamerjohn. These reports will give our readers who can't make it to a DECUS Symposia the chance the see what happened in a timely manner. To say the least, Ralph did a super job of organizing the scribes and producing the tapes that went out to all of the newsletter editors. DECUS hopes to continue this service in the future. You may want to send letters of thanks to Ralph:

Ralph Stamerjohn
Monsanto, MS: V2B
800 N. Lindbergh
St. Louis, MO 63167

I am sure he and his employer will be happy to see that people appreciate all the hard work Ralph has been putting into DECUS.

An Overview of the VAX Languages Environment

Bill Keating
Digital Equipment Corporation
Nashua, NH

Art McClinton, Session Chairperson
Mitre Corporation
McLean, VA

Reported by Omar El-Ghazzawy, DECUS Scribe Service

The presentation on the VAX Language Environment at the 1983 Spring DECUS U.S. Symposium was given by Bill Keating of DIGITAL. The discussion gave a broad overview of the VAX language environment and its major components. The languages and tools supported were enumerated, as well as planned future offerings.

The VAX is designed to provide a 'common language' environment, which means that VAX languages adhere to set standards. These standards include:

1. A symbolic debugger interface

2. Use of a symbolic traceback facility

3. Use of the Common Run Time Library

4. Conformance to VAX calling standards

5. Common handling of exceptions

6. Use of VAX-11 RMS.

Any new language offerings should conform to these standards.

In terms of planned future products, ADA and RPG were mentioned in the high level language category. Emphasis was placed on improving the tools associated with programming productivity in future releases. Suggested improvements were:

1. Full debugging support for all languages

2. Increased graphics and I/O capabilities for all languages

3. A tighter coupling between the debug - edit - compile process.

In addition, several futuristic tools are under development by DIGITAL. These are as follows:

1. (2) Project management

2. (3) Specification writing, read and review

3. (1) Design

4. (2) Language-oriented editing

5. (2) Overall configuration management

6. (3) Test management

7. (1) Test generation

8. (3) Dynamic Analysis Performance

9. (2) Static Source Analysis

10. (2) Generalized string processor generator

The numbers indicated above range from 1 (least development) to 3 (most development) and are designed to give an indication of the product's position in the development pipeline.

ADA Language System

Martin Wolfe
U.S. Army
Fort Monmouth, NJ

Peter Beck, Session Chairperson
U.S. Army
Dover, NJ

Reported by Jeff Stapleton, DECUS Scribe Service

ALS, the ADA Language System, is meant to be a complete program environment. Its host is a VAX/VMS and its target is the Intel 8086. This system will be presented informally to the Army in August of 1983. The formal presentation is scheduled for July of 1984, and, at that time, it will have full bootstrapping capabilities. From there, it will undergo testing and physical configuration auditing.

The ALS environment is divided into three main components. The user will view only the top level, the Command Language. Below that is the Tool Set, which is supported by the actual Data Base.

The Data Base, the lowest level, is basically a tree structure similar to UNIX, MULTICS, and VMS. It is accessible by either files or directory paths. Both use ADA names, being 20 characters or less. The names may be addressed absolutely or relatively. An advanced capacity of the Data Base is supporting extensibility, and both attributes and associations can be done dynamically by the user. Associations are name lists of related nodes in a tree.

Softech, the developer of the ALS environment, has had to change the normal view of updating files. Files can be changed in one of two ways. Revision is when an old file is updated, and consequently the old version is automatically obsolete. A variation is two different versions of the same file, in which both versions are to be kept concurrently. A variation is incorporated into the tree by linking the two versions with a variation header node. This can be thought of as a directory by "and" nodes. Choosing the final variation can be considered "or" nodes, since both of the variations will not ultimately be kept. Softech's philosophy of data management is to put the variations as low as possible in the tree.

The access control follows the normal lock and key paradigm. There is a small wrinkle added, however. The user needs two keys, a user identification or a team identification, and a tool identification.

A tool is any executable program or a Command Language procedure. An executable program can be added anywhere in the Data Base by a user without any additional coding. Command Language is the primary interface between the user and the system. The commands are very "ADA"-like in their syntax. To keep matters simple, any CL command that has ADA syntax also keeps the same ADA semantics. Commas and parentheses may be omitted, but this is suggested only for online CRT use. For documentation purposes, it is strongly recommended that all commas and parentheses be used for readability.

Substitutors are also tools. A substitutor is a cross between a Macro and CL variables. There are predefined substitutors used for control purposes, which allow quick implementation of simple tools. For complex tools, ADA must be used. Common sequences can also be used as single commands.

An interesting fact brought up during the questioning was that ALS does not support piping. Unlike UNIX, output from one procedure cannot be piped as input into another procedure. Softech had considered this feature but turned it down since it is still in the development stage. Another piece of data brought out involved the File Administrator. Softech is currently keeping ALS portable, but as this is adding a large amount of overhead, they are rethinking this standard and it may possibly change. There was nothing definite about this, however.

Overview of PASCAL for RSX users

Michael Ross
Digital Equipment Corporation
Nashua, NH

John R. Barr, Session Chairperson
University of Montana
Missoula, MT

Reported by Laura Havlin, DECUS Scribe Service

Michael Ross, as a representative for DIGITAL, talked about the PDP-11 PASCAL/RSX. This system is based on the "OMSI PASCAL-2" compiler acquired by DIGITAL. PDP-11 PASCAL/RSX is an implementation of the Pascal language that accepts programs compatible with Level 0 of the ISO Specification for the Computer Programming Language Pascal (Draft International Standard 7185). PDP-11 PASCAL/RSX is a multipass optimizing compiler that provides all standard Pascal data types and statements, as well as extensions.

The data types, control statements, and predeclared functions and procedures provided by Pascal include (1) INTEGER, REAL, CHAR, BOOLEAN, enumerated, and subrange data types, (2) ARRAY, RECORD, SET, and FILE structured data types, (3) FOR, REPEAT, WHILE repetitive control statements, (4) CASE, IF-THEN-ELSE conditional statements, (4) BEGIN...END compound statement, (5) GOTO statement, (6) READ, WRITE, READLN, and WRITELN input and output procedures.

The extensions for PASCAL/RSX include: (1) support for RSX File Control System (FCS) sequential files, with fixed or variable length records; (2) sequential access to fixed or variable length records, and both direct and sequential access to fixed length records; (3) EXTERNAL and FORWARD procedure and function directives; (4) OTHERWISE clause in case statements; (5) REM operator to supply the remainder in division operations; (6) global, local, and external attributes on variables and subprograms; (7) MODULE reserved word for separate and independent compilation; (8) static and automatic allocation attributes; (9) %INCLUDE directive to allow multiple compilation units to access the same program text; (10) binary, hexadecimal, and octal constants; (11) exponentiation operator; (12) dollar sign ($) and underscore () characters in identifiers; (13) value initialization in declaration section at program level; and (14) predefined procedures CLOSE, DATE, OPEN, TIME, and HALT.

PDP-11 PASCAL/RSX programs can be compiled in interactive mode or with an indirect command file. The PDP-11 PASCAL/RSX compiler performs optimizations designed to improve execution speed, including constant folding, constant conversion, constant pooling, and global register assignment. Optional compiler instructions and input and output file attributes are specified by compiler switches that: (1) perform run-time checks of array bounds, case selectors, pointers, string bounds, and subrange bounds; (2) limit the number of error messages printed and allow compilation to continue regardless of the number of errors; (3) cause code to be generated that provides source line numbers at run-time; (4) determine whether to generate EIS, FPP, or FIS instructions; (5) produce an optional, machine-code listing; (6) identify the Pascal compiler version number; (7) print warning-level messages that identify the use of PDP-11 Pascal extensions; (8) provide automatic spooling of the listing file; (9) maintain qualifier settings for subsequent compilations in interactive mode or in the same indirect command file.

After compilation, the RSX task builder is used to produce an executable image file, and to provide support for both relocatable and resident object libraries. Task builder options create checkpointable tasks, identify the use of floating hardware, provide Online Debugging Support (ODT) and allow simultaneous execution of multiple versions of a single task.

PDP-11 PASCAL/RSX allows use of the PDP-11 FORTRAN standard calling sequence, permitting Pascal programs to communicate with FORTRAN callable system routines for realtime applications. However, routines written in FORTRAN cannot be called from PDP-11 PASCAL/RSX. In addition, PDP-11 PASCAL/RSX programs can call RSX system services for process-control operations, system devices, and special peripheral access.

The PDP-11 PASCAL/RSX is expected to be available for order during the latter half of this year.

Programming in PASCAL for RSX Users

Mike Ross
Digital Equipment Corporation
Nashua, NH

John Barr, Session Chairperson
University of Montana
Missoula, MT

Reported by Marty Olevitch, DECUS Scribe Service

Mike Ross of the Technical Languages and Environments Group of Digital Equipment gave a description of the PASCAL language as it relates to the RSX environment. PDP-11 PASCAL provides a program development environment well adapted to RSX-11M and RSX-11M+. Its compiler accepts the normal RSX command line syntax and switches for both MCR and DCL, including indirect command files. The compiler can be tailored to the needs of the user at installation time. Access to the RSX executive directives is supplied through the SEQ11 procedure call interface at run time, the compiled object modules may be placed in libraries with LBR, and OTS can be overlayed or placed in a resident library.

Strong points of the PDP-11 PASCAL compiler are that it will accept almost any number of identifiers and constants as well as any number of procedures and functions in a module. It is comparable in speed to the other PDP-11 compilers. The compiler allows data sharing between separately compiled modules via the GLOBAL and EXTERNAL variable declarations, as well as overlaid PSECT. In addition, it provides upward mobility with VAX PASCAL V2 at the source language level, an enhanced machine code listing and run-time error trace for easier debugging, and produces FPP, EIS, or FIS code for floating point operations.

Optimization features of this compiler include the elimination of common subexpressions, global register allocation, jump/branch resolution, the folding of constant expressions, and deletion of areas of inaccesible code. Also included is flow boolean optimization, meaning that a boolean expression will not be evaluated in full if it is discovered to be true or false prior to the complete evaluation.

Ross emphasized the reliability of this PASCAL, noting that the compiler and OTS have undergone over 3,000 PASCAL tests. The compiler is self-compiling and usable on any machine. Incorporated is the PDP-11 workfile system. Complete documentation such as the Language Reference, User's Guide, and Installation Guide are readily available, and information intended to anticipate user problems is provided.

PDP-11 FORTRAN-77 Futures

George Hetrick
Digital Equipment Corporation
Nashua, NH

Gary Maxwell, Session Chairperson
U.S. Geological Survey
Menlo Park, CA

Reported by Jeff Stapleton, DECUS Scribe Service

The PDP-11 FORTRAN-77 DEBUG, currently being field tested, was announced at the PDP-11 FORTRAN-77 Future session. DEBUG is a fully symbolic debugger for PDP-11s. This means the symbolic name can be used instead of the octal representation. It has full support of overlaid programs, and DEBUG supports both FORTRAN-77 and MACRO-11. DEBUG runs as two tasks and supports all FORTRAN-77 data types. It requires 2K words in the user task and 28K words in the user's background task. DEBUG for RSX-11M, RSX-11M-PLUS, RSTS/E, and VAX/VMS will be available in October of 1983.

Commands are separated into four categories. These are: (1) Display Control Commands, (2) Command Logging Commands, (3) Execution Commands, and (4) Convenience Commands.

The question and answer part of the session revealed some interesting information: debugging a program with subroutines residing in a library is allowed if the program was compiled with the debugger switch on. This does result in a larger object file, but the object code remains the same size. Watch points are not available in the first version. During the conversation, it was revealed that is considered a good idea to always provide a return break. It was also mentioned that the number of break points you can set depends upon the amount of memory you have left. Since break points are dynamically stored, it depends on the amount of memory available to you. George Hetrick mentioned that he has set as many as 50 break points without any problems!

Building and Using Resident Libraries

S. Reid Madsen
Weidner Communications
Provo, UT

Allan J. Bennett, Session Chairperson
Clark Equipment Company
Battle Creek, MI

Reported by Marc Caffee, DECUS Scribe Service

The RSX taskbuilder permits three types of shared regions: shared commons, shared libraries, and dynamically allocated regions. Several types of shared libraries were discussed, as follows:

1. Resident library - a library that requires a dedicated address window in your task.

2. Cluster library - a set of libraries mapped through one window (this type of library can be very slow).

3. Supervisor mode library (RSX-11M-PLUS only) - the library mapped uses supervisor I space while the task is mapped in D space.

The purpose in using a library is to reduce the physical address space required for a task by putting routines used by many tasks into a common area (library). By doing this a net gain in physical address space is made. Using shared libraries also reduces disk swapping, since the space requirements for tasks is less than that required by tasks which do not use shared libraries. This not only increases throughput, but can possibly forestall the purchase of additional memory.

An example of an existing library is the FCS library. During the SYSGEN procedure you can link a number of tasks to FCSRES. Additionally, some of the layered products (e.g. FORTRAN) have the ability to build resident libraries. In addition to the above libraries the user can write his own libraries. In order to establish a library you must: define the library contents, build the library, install the library, and finally link the task to the library.

A user built library can contain either a MACRO-11 source file or an overlay descriptor file. MACRO-11 source files are used by specifying the source file's global entry points. If an ODL file is used, the file contains the names of the object modules to be included in the library.

After the contents of the library have been established, TKB is used to build the library. By using different TKB commands you can build overlaid libraries, cluster libraries, and supervisor libraries. In building a memory resident overlay a potential pitfall occurs in referencing entry points. If an entry point is not referenced in the root segment it should be defined using the .GBLREF option.

In building cluster libraries there are several things to remember.

1. All libraries except the first must be memory-resident overlays.

2. User task vectors indirectly resolve all interlibrary references.

3. Revectored entry point symbols must not appear in the "Upstream" symbol definition file. A .GBLXCL option must appear for each symbol being revectored to a "Downstream" library.

4. All libraries must be built PIC or built for the same address.

5. Trap or asynchronous entry into a library is not allowed.

Building a supervisor mode library is similar to building other libraries.

Once a library has been built, the next step is installation. One way to do this is to use VMR to create a new bootable system image. One strategy is to install each library in a partition which has the same name as the task. To get the size for the partition to be created look at the memory allocation map.

You are now ready to link to the resident library which you have created. For a non-clustered and non-supervisor-mode library use either the LIBR or RESLIB TKB options. For clustered libraries use the CLSTR keyword. Supervisor-mode libraries are specified by SUPLIB or RESSUP.

In a test to check performance three different situations were compared:

1. No libraries

2. Cluster library in which both libraries were overlaid

3. Cluster library where the first library was not overlaid

Cases 1. and 2. had essentially the same performance, while case 3 was slower by about a factor of 9.

<<<NOTE TO THE EDITOR: A complete written version of this session will probably appear in MultiTasker in the near future. Included in this version will be illustrative examples, etc.>>>

FORTRAN/RT Tutorial - EIA, FIS and FPU

Ron Trellue
Sandia National Laboratories
Albuquerque, NM

David E. Ellis, Major, USAF
Air Force Weapons Laboratory
Kirtland AFB, NM

Jack Crowell, Session Chairperson
Los Alamos National Laboratory
Los Alamos, NM

Reported by Gavin Perry, DECUS Scribe Service

This tutorial is a repeat of the one presented at Anaheim (Fall 82) DECUS. New material added consists of benchmarks run for the various processors and FORTRAN options. If you've ever wondered what NHD, EIS, FIS, and FPU are, and whether you want in-line or threaded code, get this tutorial from the proceedings. Here are the basics.

The NHD, EIS, etc. are processor options. You should use the compiler and library that are optimized for the hardware that you have on your system. To find out what you have run the SHOW CONFIG monitor command in RT-11.

The FORTRAN compiler can be built to produce either inline or threaded code. Inline code will usually run faster, but it depends on the application. Inline code generates a stream of MACRO assembly language code with fewer library calls. Inline FPU code is not possible with the present compiler but FORTRAN-77 will have this feature if (when) it becomes available for RT-11. The timing benchmark tests show that under RSX (which has F77) the program that uses a lot of number crunching in floating point will run twice as fast. If you don't have arithmetic hardware, or if you want to be able to use the Fortran debugger then threaded code must be used. Threaded code consists of a string of subroutine calls to the Fortran Object Time System (OTS) library that is hardware independent. The OTS routines are then selected (when you build the library) to optimize for the hardware available. A threaded object module can be linked with any of the five types of OTS libraries to produce runnable code for any PDP-11. Because of all the subroutine calls threaded code tends to run slower but it may produce a smaller .SAV file allowing larger applications to run, and the Fortran debugger requires threaded code to run so it is probably the best way to go for student applications.

For each application it is important to know what types of operations will be executed, what type of hardware is available and how to build the software to optimize the application for your needs whether they are running speed, ease of use or size of application. FORTRAN-77 with its ability to produce in-line FPU code will be an important enhancement for those applications which are floating point compute bound.

VAX-11 PASCAL I/O Tutorial

Joyce Spencer
Digital Equipment Corporation
Nashua, NH

John R. Barr, Session Chairperson
University of Montana
Missoula, MT

Reported by Marty Olevitch, DECUS Scribe Service

Joyce Spencer of DIGITAL Technical Languages and Environments Group gave an introductory tutorial on the input/output facilities of the VAX-11 PASCAL language. She asserted that, as opposed to other versions of PASCAL, this implementation has quite adequate I/O capabilities, with access to many of the features supported by RMS. Her talk centered on PASCAL predeclared procedures and functions that allow creation and manipulation of files, and internal I/O.

Ms. Spencer first went over the fundamental RMS concepts of how RMS records are organized into RMS files, and the access methods that are used on those files. The general rule is that one RMS record is equivalent to one PASCAL file component (with the exception of text files). Fixed or variable length formats are available. RMS files are collections of records arranged in a specific order. There are three basic organizations, and one is specified at the time the file is created. In sequential organization, the file components are arranged in a physical sequence, in the order that they were written to the file. Files with relative organization have components that are a given number of fixed-length cells. The order is their physical sequence, but a cell may either contain data or not. A file component may be referred to by its cell number, as well as sequentially. With indexed organization, the file components are ordered according to some keys that exist in every component.

Access methods are the way files are stored and retrieved, and again there are three methods. They are distinct from the file's organization, and are specified each time a file is opened. In sequential accessing, the files are processed in order from first to last, either according to physical sequence, cell number, or ascending order of the primary key. Direct accessing involves specifying a particular cell number which is then processed. The keyed access method can be used on files with indexed organization, which are processed according to the specified key.

I/O in PASCAL is done with a set of predeclared procedures. They are OPEN and CLOSE, sequential access routines, what Ms. Spencer called miscellaneous routines, text routines, direct access, keyed access, and terminal I/O routines. All of the above procedures have a large number of parameters that may be chosen by the user. A new feature of version two is the ERROR parameter. It allows one to specify an action to be taken upon an error signal, and may be used with all of the I/O commands.

System Programming in VAX-11 PASCAL V2

Joyce Spencer
Digital Equipment Corporation
Nashua, NH

John Barr, Session Chairperson
University of Montana
Missoula, MT

Reported by Omar El-Ghazzawy, DECUS Scribe Service

Joyce Spencer of DIGITAL covered new features in VAX-11 PASCAL V2 that make the language suitable for systems programing. These features include new data types, attributes, I/O functionality separate compilation, and much more.

System programs are a class of programs which create and synchronize processes, access devices and files, and share data and code. A language suitable for such tasks requires a flexible grammar, and a solid integration into the VAX "common language" environment. Such a language is VAX-11 PASCAL V2.

The new data types In VAX-11 PASCAL V2 include: UNSIGNED, DOUBLE (G-float format), QUADRUPLE, and VARYING CHAR.

These new attributes permit greater control over the behavior of the compiler, data items, routines and compilation units. Separate compilation of PROGRAMs and MODULEs. The attributes ENVIRONMENT and INHERIT can be used to share variables, routines, constants and types, thus greatly simplifying the compilation process. In terms of I/O functionality, the principle additions are:

1. ISMA support

2. ERROR := CONTINUE parameter for I/O procedures

3. Several new predeclared routines: STATUS, TRUNCATE, UNLOCK, and UFB

4. An enhanced OPEN with the new parameters ORGANIZATION, DISPOSITION and USERACTION

5. An enhanced CLOSE with the addition of the DISPOSITION and USERACTION parameters

Some other new features include new builtin routines for STRING and UNSIGNED variables, nonpositional parameters and the inclusion of default parameters.

VAX-11 PASCAL V2 is well integrated into the VAX/VMS environment. It supports the VAX-11 Procedure Calling Standard, the VAX-11 Symbolic Debugger and access to system supplied routines such as RMS, RTL and system services. When calling system supplied routines, the procedure names take the form:

```
facility-code$procedure-name or procedure-name
```

To obtain system status codes, the following construct can be used:

```
[ INHERIT ('SYS$LIBRARY:STARLET') ]
```

A summary of the question and answer session follows:

Q. Are environment files going to provide for LIB$ functions ?

A. This is a good idea which may be provided later on.

Q. What about the bug encountered when you open a file unsuccessfully the first time?

A. This has been fixed and will be made available in the next maintenance release of VMS.

Q. In the STARLET environment file are types associated with the routines defined?

A. We are thinking of doing so in the next release.

Q. How about opening a sequential file in append mode?

A. It is on our list.

Q. What about a data type that is an external constant?

A. This is also on our list.

Tailoring EDT for the Structured Language Programmer

Shava Nerad
Varian Associates
Gloucester, MA

James W. Livingston, Session Chairperson
Measurex Corporation
Cupertino, CA

Reported by Margaret Watters, DECUS Scribe Service

Shava Nerad and R. Mark Chilenskas wrote a paper entitled, "Tailoring EDT for the Structured Language Programmer". Nerad, who spoke before a standing room only crowd, highlighted the methods that she has found helpful when editing her programs using EDT. Many members of the audience were pleased to discover that there are convenient ways to edit their programs using EDT.

The problem with EDT is that it defines words, sentences, paragraphs, and pages within the context of the English language; therefore when a programmer tries to delete a single entity of the language in which he is writing, he often ends up deleting an entire line section. Nerad and Chilenskas have found methods to redefine the delimiters of the word, sentence, paragraph, and page functions. When Nerad complained about the poor documentation for setting up a command file to do this, there was a murmer of agreement from the audience.

One member of the audience suggested that instead of using Nerad's method to move up and down the initializing file, to gold the arrow keys. Several members of the audience agreed that this was a better method.

<<<Editors note: the following paper expands on the session report above.>>>

Shava Nerad
Varian Associates, Gloucester, Ma.

R Mark Chilenskas
Computer Corp of America, Cambridge, Ma.

ABSTRACT

This paper will discuss EDT features that are desireable to the structured language programmer, present the authors' own EDTINI.EDT file as a model for the reader, and present a tailored layout of the EDT keypad mode functions.

EDITOR FLEXIBILITY

The importance of the editor to a programmer's work should not be underestimated. A majority of any programmer's time is spent writing code in the editor. This makes the editor one of the most important software tools the programmer will use.

For the structured languages programmer, the language that he manipulates in his day to day work is one of rigid syntax and regular, formalized layout. Though the details of each language will be different, I will use Pascal as the language of example for this paper.

EDT, the VMS full screen capable editor, is an underutilized resource at most VAX sites. The ease of use and flexibility of the editor make it a tool that is in some ways superior to more powerful, less friendly editors such as TECO. The structured languages programmer's code relies on formatting for readability. The entities word, paragraph and page may differ from standard text and between languages. Many features of EDT can be used to tailor the editor to a special code manipulation tool, rather than using the default initialization functions of EDT, which are gauged toward general word processing.

TRAVERSING ENTITIES

Entities are significant pieces of text that may be regarded as logical entities in their own right. Entities, defined by delimiters, allow the editor to match patterns to either move the reference pointer, or to delete/replace a particular range of characters (word, paragraph...). The standard entities of computer languages are, by level of complexity:

1. atoms or tokens

2. statements or s-expressions

3. modules

An atom or token is the base, unreducible unit of meaning in the language. This token-parsed Pascal statement divides into twelve tokens:

```
writeln ( MyFile , ' This is a test ' ) ;
```

Tokens can be combined into s-expressions or statements, usually delimited by reserved words or other structure markers, such as the semicolon in Pascal. The 'writeln' example above shows a Pascal statement which defines the action of writing the string literal 'This is a test' to the file Myfile. Complex statements can be formed according to the specific nesting syntax of the language, allowing subroutines, procedures, functions, declaration sections, et al, to be created as complex statements.

A module is an independant compilation unit, producing its own symbol table, requiring independant declarations (if required).

Tokens can be thought of as replacing the words and punctuation of a natural language. Simple statements are similar to sentences, complex statements are similar to paragraphs. A module might be considered to be like a chapter of a book, or article of a journal.

In composing and editing code, these entities are important to finding one's 'place' in the file. The ability to find the next significant paragraph/complex statement, to neatly delete the next significant word--these are actions defined by the context of the language in which you work, be it English, LISP, APL, or Pascal.

In programming Pascal, the important tokens of the language are: program, module, procedure, function, begin, end, ord, *, =, :=, for, while, record, ;, (,) , { , }, [,] , and so on. Some of these (program, ;, end, }...) are delimiters of statements or modules. Each of these characteristic elements of the language takes on the importance one would accord WORD, SENTENCE, PARAGRAPH, or PAGE when referring to the language English. While the default initialization in EDT only defines the space, the carriage return, and the tab characters as valid delimeters of WORDS, in the Pascal statement:

```
Function MakeWaves(var FirstLocation: array[1..8]of char;
    counter:integer; AFlag,BFlag: boolean):real;extern;
```

consists of seven words, whereas the significant tokens in the statement are around thirty. By defining the entity WORD in EDT as including the delimiters (,), :, ;, and so on, we can more realistically represent the realities of the language in which we are working--not English, but Pascal. Under EDT default initialization, a DELETE WORD BACK from the end of the statement used in the example above would delete 'boolean):real;extern'. This is almost certainly more than was meant to be deleted by the programmer.

Similarly 'for', 'case', 'if', 'repeat' and other single statement oriented tokens would be defined as similar to SENTENCE delimiters, and 'function' and 'procedure' would be defined as PARAGRAPH entities. The tokens 'program' and 'module' would be equated to PAGE delimiters. Unfortunately, in EDT only single character delimeters can be defined for WORD and SENTENCE delimiters. Although PARAGRAPH and PAGE entity definitions will take a string literal, they will only accept one delimiter definition for either entity.

EDITING STRUCTURES

In addition to moving through a program, structures must be entered and changed. These structures are expressed by keywords and indentation. Ideally, the editor should help the programmer deal with both with minimum effort and drudgery.

Indentation is used to group statements together. All the statements inside a given s-expression (say a BEGIN/END group) are indented to the same level. When entering statements there should be no reason for the programmer to laboriously tab and space to the proper starting column. Rather, there should be a simple way to move to the next line at the proper indentation level. The indentation level should change when a BEGIN or END statement is encountered. In addition, it is useful to automatically shift text right or left when a BEGIN is added or deleted to the program.

EDT provides simple techniques to set the current indentation level and tab-shift text. Adding a few key definitions lets the programmer start at the right place on the next line. (See key 3, key gold 3, and key 21 in the example)

Another useful tool is abbreviation. Every time a FOR loop occurs in Pascal, a group of keywords and new lines is entered:

```
FOR := TO DO
    BEGIN

    END;
```

The same can be said for CASE, WHILE, etc. It is possible to enter these skeletal structures with single commands rather than by typing all the characters every time. This could not only save time but serve as a reminder of the syntax for the s-expression. The latter is not an important consideration in a small language like Pascal, but it could be useful when learning the many new control structures in ADA.

As most of our programming is done in Pascal, we have not added abbreviations for structures. Since structures are simple and not very long, we did not think the time saved would be worth the effort of remembering all the abbreviations. Other programmers could of course decide otherwise.

However, another way to use abbreviations has been helpful. Many programmers like others to use long descriptive names, but very few like to type them in. Thus they often use abbreviations for the names, sacrificing clarity to escape drudgery. However, you can use named buffers to provide a technique to expand token abbreviations into fully spelled out words. The definitions to do this are provided in our init file (V, gold V). You would not want to implement statement structure abbreviations in exactly this way as they must be:

1. indented to the proper level and,

2. the cursor should be left at one of the "blanks" in the structure, not at the end.

These problems can be taken care of by combining EDT commands in a macro with text insertion.

IMPLEMENTATION NOTES

A complete EDTINI.EDT type file may be chosen from DCL (e.g. edit/edt/command=workdir:pascalini.edt). We include the line "edt :== 'edit/edt/command=tools:edtini.edt'" in our LOGIN.COM's. By the use of EDT "macros", an environment appropriate to the current task, (see included buffers RNOMODE and TEXTMODE in example) may be initialized without exiting the editor.

The EDTINI.EDT file in the example files is commented. This is not a legal option in EDTINI.EDT files. If you plan to use the file, all comments delimited with "(*" and "*)" should be removed.

Several of the example files include representations of control characters in the format 'X'. In the implementation of these files, the SPECINS command should be used to insert these non-printing characters in your text.

A personalized HELP file is included in the example. Using this redefinition greatly reduces the general utility of the HELP facility supplied with EDT, but is sufficient for the experienced user who may try to learn a new keyboard.

CONCLUSIONS

EDT is not a perfect editor for structured languages. It is primarily designed to handle lines, words, sentences and other structures of written English. Programming languages have other basic entities, namely tokens, s-expressions and modules. However EDT provides reasonably powerful customization tools which lets the user redefine entities and new functions to handle the new entities.

Tokens are easy to describe in EDT because they are delimited by a single character. S-expressions and modules, however, are delimited by strings. EDT will only permit a single string as delimiter for the PARAGRAPH and PAGE entities, not a choice between a group of strings like FOR, IF, CASE etc. Thus s-expressions and modules are not easily described to EDT. This makes it harder to move over structures than would be desirable.

The macro and define key capabilities of EDT allow programmers to perform common operations like moving to the next line with at the proper indentation and abbreviations with single commands. This customization is a powerful tool which can make common operations quicker and let the programmer concentrate on the code rather than its format. This in turn leads to faster programming with fewer mistakes.

Files included:

edtini.edt	:	the edt initialization file
help.edt	:	keypad layout learning aide
rnomode.edt	:	useful definitions for DSR tasks
textmode.edt	:	useful definitions for word processing

EDTINI.EDT

```
include tools:rnomode.edt =rnomode
include tools:textmode.edt =textmode
            (* macros to allow further tailoring of the editor,      *)
            (* according to the current task                         *)

include tools:help.edt =help
            (* this brings the associated help file into the buffer *)
            (* "help", so that a personalized keypad help display   *)
            (* can be produced.                                      *)

find =main 1
            (* returns the editor's attention to your main buffer.  *)
            (* otherwise, you would remain in buffer "help"         *)

set term vt100    (* or as appropriate *)

set tab 4     (* my preference     *)

define key 10 as "ext change =help."
            (* the standard keypad HELP key now displays our "help" *)

define key gold 10 as "^?'ctrl-'."
            (* Insert control character, using the prompt, "ctrl-"  *)
            (* instead of the standard SPECINS.  I find it easier   *)
            (* to remember "ctrl-z" than the ASCII decimal          *)
            (* equivalent numbers, from long experience with other  *)
            (* editors that use control characters extensively.     *)

define key 11 as "CUTSR=dotdotk."
            (* Cut Select Range into buffer Dotdotk.                *)
            (* "dotdotk" is an arbitrary buffer name, lifted from   *)
            (* TECO, that we will use as the standard deletion      *)
            (* buffer.                                              *)

define key gold 11 as "CUTSR=?'Buffer name:  '."
            (* Cut Select Range into specified buffer, prompt for   *)
            (* buffer name using prompt "Buffer name:  ".           *)

define key 17 as "PASTE=DOTDOTK."
            (* undelete key.  This will undelete words, lines, or   *)
            (* select ranges, but not single characters.  I have    *)
            (* found that it is very frustrating to have a deleted  *)
            (* paragraph disappear when you thoughtlessly delete a  *)
            (* single character.                                    *)
```

```
define key gold 17 as "PASTE=?'Buffer name:  '."
            (* the complement of Gold 11, this key Pastes in a       *)
            (* specified buffer, prompted by "Buffer name:  ".       *)
            (* Note that the named buffer need not have been         *)
            (* created by Gold 11, but may be buffers created by     *)
            (* Includes, etc. such as the "help" buffer.             *)

define key 7 as "ADV."
            (* FORWARD.  Sets the editor looking forward in the      *)
            (* text for the entity to be manipulated.                *)

define key gold 7 as "BR."
            (* TOP.  Sets the cursor to the first character of the   *)
            (* file.  Note that this is the opposite pairing from    *)
            (* the standard EDT keypad, which pairs FORWARD with     *)
            (* BOTTOM, and BACK with BOTTOM.  This is a matter of    *)
            (* personal preference.                                  *)

define key 8 as "BACK."
            (* sets the editor looking backward in the text for the *)
            (* entity to be manipulated.                             *)

define key gold 8 as "ER."
            (* BOTTOM.  Sets the cursor to the [EOB] marker          *)

define key 9 as "''."
            (* FIND NEXT.  Find next entity as set by gold 9         *)

define key gold 9 as "'?'Search for:  ''."
            (* SET FIND.  Set entity for FIND NEXT, search through   *)
            (* text for its first occurance.  Note that the editor   *)
            (* is not able to search for the character "'", since    *)
            (* that is its string delimiter character.               *)

define key 18 as "CUTSR=DOTDOTK PASTE=REPLBUF."
            (* Select the FIND string, cut to delete buffer, then    *)
            (* paste in the string in the replace buffer, REPLBUF    *)
            (* REPLBUF is usually filled by using gold 18.           *)

define key gold 18 as "SEL I?'Replace with: '^ZCUTSR=REPLBUF."
                    (* ^Z is specins *)
            (* Cut the selected string, which is prompted for with   *)
            (* the prompt "Replace with:  ", into the replace        *)
            (* buffer, called REPLBUF.                               *)

define key 4 as "C."    (* set operating entity to CHARACTER      *)

define key gold 4 as "DC."     (* delete one CHARACTER               *)

define key 5 as "W."    (* set operating entity to WORD              *)

define key gold 5 as "CUTW=DOTDOTK."
            (* Delete one WORD.  Note that deleted WORDs are put in *)
            (* DOTDOTK, the delete buffer, for possible undeletion   *)

define key 6 as "L."    (* set operating entity to LINE              *)
define key gold 6 as "CUTL=DOTDOTK."
          (* Delete one LINE.  Note that deleted LINEs are put in *)
          (* DOTDOTK, the delete buffer, for possible undeletion  *)

define key 19 as "EL." (* set cursor at END OF LINE        *)

define key gold 19 as "CUTEL=DOTDOTK."
          (* Delete from current cursor position to the end of    *)
          (* the current line.  This fractional line is placed in *)
          (* DOTDOTK, the delete buffer, for possible undeletion. *)

define key 1 as "?'Command:  '."
          (* to use line mode commands from keypad mode           *)

define key gold 1 as "DEFK."
          (* to define keys while in keypad mode                  *)

define key 2 as "4"
          (* arbitrary count specifier. (e.g. {2 gold-6} deletes  *)
          (* 4 characters, {2 6} moves the cursor 4 lines)  The   *)
          (* same function may be accomplished by hitting {gold}  *)
          (* and the keyboard (not keypad) number 4, followed by  *)
          (* the desired operator (line, delw, etc.)              *)

define key gold 2 as "+EPAR FILLBPAR."
          (* Fill Paragraph.  Find the end of the paragraph, do   *)
          (* a fill to the beginning of the paragraph.  Note that *)
          (* this may produce undesireable results if you are set *)
          (* in the BACK direction, rather than ADV.              *)

define key 3 as "TC."
          (* Tab Compute.  Of particular interest to structured   *)
          (* programmers, this function sets the indentation to   *)
          (* the current tab position divided by the set tab      *)
          (* number.  By using ENTER rather than RETURN, one can  *)
          (* go automatically to the set indent, rather than      *)
          (* having to strike {RETURN, TAB,...TAB} to get to the  *)
          (* desired column.  Associated with this function are   *)
          (* {^E}/Indent-tabstop and {^R}/Indent+tabstop.         *)

define key gold 3 as "TADJSR."
          (* tab adjust select range.  By using {gold}({"-"}){digit} *)
          (* {gold} {key 3}, one can adjust a select range back   *)
          (* or forward by an arbitrary number of tab stops.      *)
          (* This is an invaluable aid to the programmer who has  *)
          (* to bracket a range of code with one more structure   *)
          (* loop, or conversely, take out a loop that is nested  *)
          (* around a range of code.  It preserves standard loop  *)
          (* indenting.                                           *)

define key 0 as "20L."
          (* move cursor 20 lines.  Since programmers seldom work *)
          (* in Paragraphs, this simply moves the cursor about    *)
          (* one screen in the desired direction.                 *)

define key gold 0 as "TOP."
          (* move cursor's current line to the top of the screen, *)
```

```
            (* scrolling text appropriately.  Since programmers      *)
            (* seldom work in pages terminated by <FF>'s, this       *)
            (* replaces the PAGE function nicely.                    *)

define key 16 as "SEL."
            (* set SELECT mode.  The current cursor position will    *)
            (* be the beginning of a select range.                   *)

define key gold 16 as "SR."
            (* Cancel Select mode, move cursor back to original      *)
            (* position where Select was issued.                     *)

define key 21 as "I<CR>^ZTAB."          (* <cr> & ^z are specins *)
            (* Enter is defined as INSERT <carriage return>, then    *)
            (* Insert TAB, as defined by default (tab 1) or TAB      *)
            (* select key.  See TAB ADJUST/gold 18                   *)

define key gold 21 as "I<CR>^Z-C."      (* <cr> & ^z are specins *)
            (* Open a new, blank line at the current one             *)

define key control B as "EXT CHA LAST."
define key gold control B as "EXT CHA=?'Buffer:  '."
            (* These allow switching between buffers.  ^B goes to    *)
            (* the buffer you edited prior to this one, Gold ^B      *)
    (* lets you specify a particular buffer.                 *)

define key control D as "+DC."
    (* Sometimes it is inconvenient to go to the keypad      *)
    (* just to delete a couple of characters                 *)

define key control E as "I^A^Z-30TADJL -L TD TAB -C +'^A' +DC."
                         (* ^a's & ^z are specins *)
    (* This is a key to controlling indentation.  It moves   *)
    (* you to the left one indentation level.  Thus if the   *)
    (* next line is an END; you would type ENTER followed    *)
    (* by ^E to move you left.                               *)

define key control H as "-DC-CUNDC+EL."
    (* Everyone transposes characters from time to time.     *)
    (* This lets you BACKARROW to the mistyped characters    *)
    (* and reverse them easily, ending at the end of the     *)
    (* line so you can continue typing.                      *)

define key control J as "-CUTW=DOTDOTK."
    (* Deletes a word backward but puts it in the special    *)
    (* undelete buffer.                                      *)

define key control K as "-c +EW CHGC-W +EW."
define key control L as "BW +CHGCC +EW."
    (* These keys capitalize the full prior word or first    *)
    (* character of the prior word respectively.             *)

define key control R as "I^A^Z-30TADJL -L TI TAB -C +' A' +DC."
                               (* ^a's & ^z are specins *)
    (* The complement of ^E, ^R moves you one indentation    *)
    (* level to the right                                    *)
```

```
define key gold control I as "-BL TAB +EL."
    (* Sometimes a line is not indented properly, usually  *)
    (* because it was "wrapped" from the prior line.  This *)
    (* indents the line properly.                          *)

define key control V as "PASTE=?'Abbr:  '."
define key gold control V as "SEL I?'Expanded:  '^Z CUTSR=?'  Abbr:  '."
                           (* ^Z is specins *)
    (* These are the "token abbreviations".  Gold ^V puts  *)
    (* the expanded text in a buffer called by the         *)
    (* abbreviated name.  ^V puts the expanded form of an  *)
    (* abbreviation in your buffer.                        *)

set ent w ' <LF><VT><FF><CR>();[],
    (* This defines tokens for Pascal.                     *)

set ent sen ;.}
    (* This defines the end of s-expressions.  It lets you *)
    (* move forward to the end of an s-expression, but is  *)
    (* not convenient for moving backward.                 *)

set ent par <CR><CR>  (* EDT DEFAULT *)

set ent page <CR>end
    (* If you do not indent the END statement terminating  *)
    (* modules and procedures, this defines modules for    *)
    (* Pascal.                                             *)

set wrap 78
set cursor 0:21
set lines 20
set mode change
```

HELP.EDT

```
   --------------------------------------- ctrl-B     Move to prior buffer
   |  GOLD    | CTRL-    | CUT B  |  GET B | Gctrl-B    Move to named buffer
   |   PF1    |   PF2    |  PF3   |   PF4  | ctrl-D     DelC under cursor
   |  GOLD    | (HELP)   | DEL B  |  UNDEL | ctrl-H     TransposeC, @EOL
   |-------------------------------------- | ctrl-J     Back DelW
   |   TOP    | BOTTOM   |SET FIND|SET REPL| ctrl-K     Change case, priorW
   |    7     |    8     |   9    |    -   | ctrl-L     Capitalize priorW
   | FORWARD  | BACKWARD |  FIND  |  REPL  | Gold-tab   JustifyL 1 tab ->
   |-------------------------------------- | Gctrl-V    Set abbrev
   |  DEL C   | DEL W    | DEL L  |DEL@EOL | ctrl-V     Insert abbrev string
   |    4     |    5     |   6    |    ,   | ctrl-E     Indent back 1 tabstop
   |    C     |    W     |   L    |  @EOL  | ctrl-R     Indent forw 1 tabstop
   |-------------------------------------- | backspace  TransposeC, @EOL
   | DEF KEY  |   FILL   |PRETTYPR|  OPEN  | linefeed   Back DelW
   |    1     |    2     |   3    |  LINE  |
   | COMMAND  | (ARG=4)  |SET IND |        |
   |-----------------------------|  ENTER  |
   |     1/2 PAGE        | RESET  |        | PRESS ctrl-B TO RETURN
   |         0           ←—|   .     | ALIGN  |
   |_______PAGE__________|_SELECT_|__TEXT__|
```

RNOMODE.EDT

```
set wrap 78
set ent word '       _<CR>'
set ent par '<CR> '

set ent page '<CR>.ch'
    (* pages are DSR chapters.                              *)

define key control U as 'i^&\*^Z -2C.'
           (* This is an example of a structure abbreviation.      *)
    (* DSR expects underlined words and phrases to be       *)
    (* enclosed by ^&  \, which I can never remember.  This *)
    (* puts a blank underline in the buffer, which I can    *)
    (* then fill in.                                        *)
```

TEXTMODE.EDT

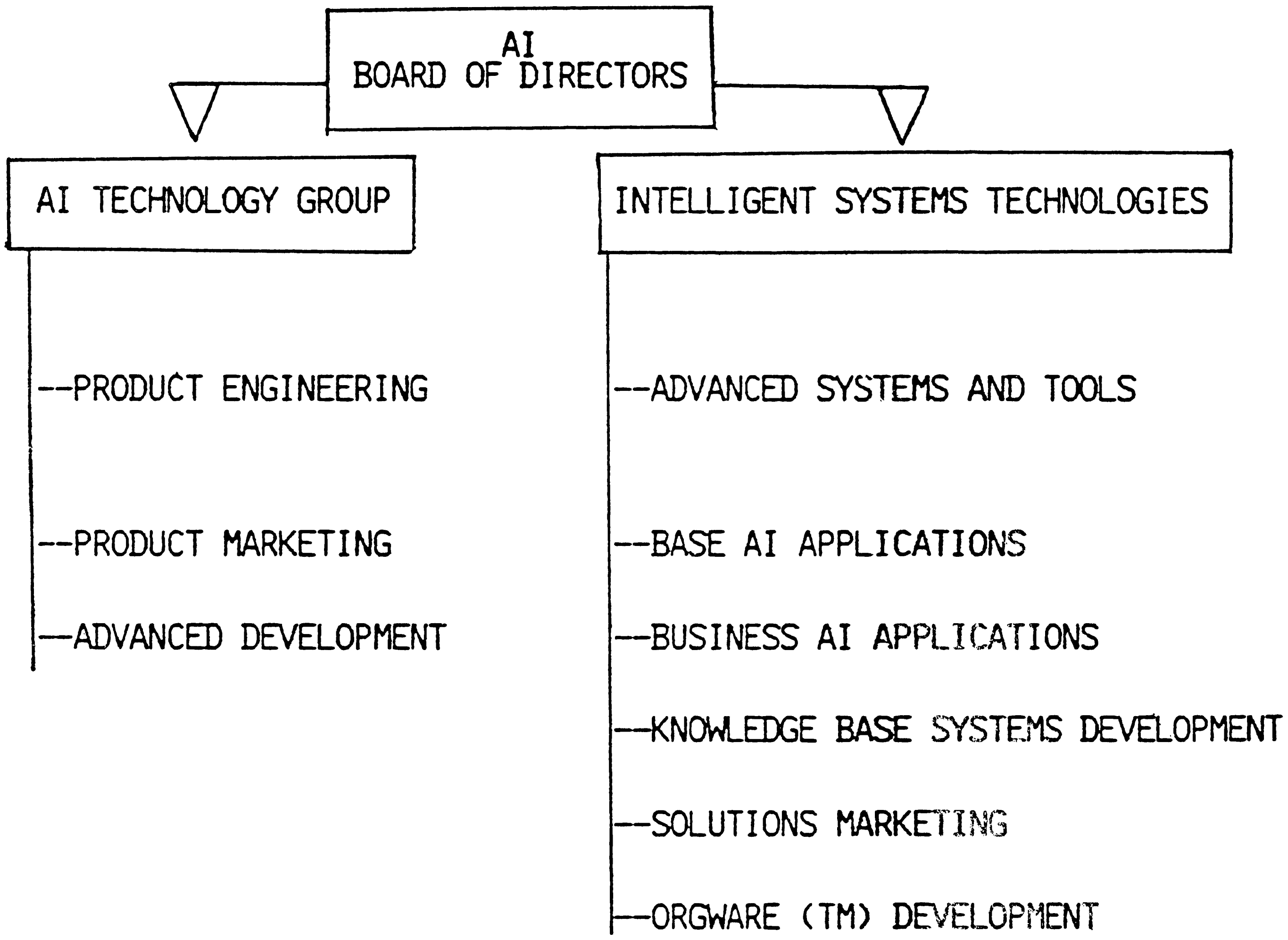
```
set wrap 78
set ent word '      <CR>'
set ent par '<CR> '
set ent page '<CR>.se'
define key control  V as "-L PAR +L."
define key gold     V as "SEL -L PAR -EL."
define key gold control V as "+BPAR +EL SEL +EPAR +L."
```

Artificial Intelligence Technology Center

The Artificial Intelligence Technology Center at DIGITAL is currently developing tools to support AI products and will be developing additional AI products in the future. Art Beane gave me the following slides which have been used at several presentations about what the Artificial Intelligence technology Center is attempting to accomplish. Currently, Common Lisp and Interlisp are being developed through a third party software program in conjuction with Rutgers University and the USC/ISI (University of Southern California/Information Sciences Institute). They are also interested in LOGO, PROLOG, and Interlisp compatability software. The intent of the center will be to produce the richest variety of software for VAX systems that can be used to develop the next generation of AI products. The LTSIG hopes to maintain a line of communication with the AI Technology Center. Users of Lisp based systems are encouraged to contact the LTSIG if they are interested in aiding our efforts.

DIGITAL EQUIPMENT CORPORATION

ARTIFICIAL INTELLIGENCE TECHNOLOGY CENTER


AI BOARD OF DIRECTORS
AI TECHNOLOGY GROUP
--PRODUCT ENGINEERING
--PRODUCT MARKETING
--ADVANCED DEVELOPMENT
INTELLIGENT SYSTEMS TECHNOLOGIES
--ADVANCED SYSTEMS AND TOOLS
--BASE AI APPLICATIONS
--BUSINESS AI APPLICATIONS
--KNOWLEDGE BASE SYSTEMS DEVELOPMENT
--SOLUTIONS MARKETING
--ORGWARE (TM) DEVELOPMENT

ARTIFICIAL INTELLIGENCE TECHNOLOGY GROUP

ADVANCED DEVELOPMENT

o NATURAL LANGUAGE INTERFACES

o KNOWLEDGE REPRESENTATION

o INTELLIGENT VLSI/CAD TOOLS

o INTELLIGENT DIAGNOSTIC TOOL

o GENERIC HARDWARE DIAGNOSTIC TOOL

o SYMPTOM-DIRECTED DIAGNOSIS

o SECURITY MONITOR

o EXPERT SYSTEM FOR PROGRAM DEBUGGING

o INTELLIGENT EDITOR INTERFACE

TECHNOLOGY TRANSFER

- CONSULTATION WITH RESEARCHERS
- EXTERNAL RESEARCH PROGRAM
- MICROELECTRONICS AND COMPUTER CORPORATION

COMMON LISP GOALS

o COMMONALITY	—	AMONG FOUR DIALECTS
o PORTABILITY	—	TO YET OTHER MACHINES
o POWER	--	TO ENABLE BUILDING OF LARGE SYSTEMS
o EXPRESSIVENESS	—	FOR EASE OF READING/WRITING
o COMPATIBILITY	—	WITH OLDER LISP DIALECTS (MACLISP AND INTERLISP)
o STABILITY	—	ONCE DEFINED
o EFFICIENCY	—	SUBJECT TO THE OTHER GOALS
o CONSISTENCY	—	BETWEEN COMPILER AND INTERPRETER, ESPECIALLY VARIABLE SCOPING

HOW GOALS ARE MET

- PORTABILITY.
  - —AVOIDANCE OF MACHINE-DEPENDENT FEATURES
- POWER
  - —MANY BUILT-IN OPERATIONS (LARGE LANGUAGE)
- EXPRESSIVENESS
  - —GENERIC OPERATORS
  - —USER-DEFINED TYPES
- EFFICIENCY
  - —DECLARATIONS
- CONSISTENCY
  - —VARIABLE SCOPING CHANGES

COMMON LISP IMPLEMENTATIONS

o SPICE LISP (CMU)

o VAX COMMON LISP: VMS, UNIX (DEC, CMU)

--SHARES CODE WITH SPICE LISP

o EXTENDED TOPS-20 (RUTGERS)

--SHARES CODE WITH SPICE LISP

o LISP MACHINE LISP (SYMBOLICS, MIT)

--SUPERSET, BUT WILL RUN COMMON LISP
--RUNNING NOW WITH FULL ENVIRONMENT
--NEEDS CONVERSION
--SYMBOLICS SUPPORT (BUNDLED)

o VAX NIL

--SUPERSET, BUT WILL RUN COMMON LISP
--FOR VAX/VMS, MAY MOVE TO UNIX LATER

OPS BACKGROUND

o OPS4 DEVELOPED IN THE LATE 70'S BY CHARLES FORGY AT CARNEGIE-MELLON UNIVERSITY (CMU)

o INTRODUCED TO DIGITAL BY JOHN MCDERMOTT OF CMU AS THE IMPLEMENTATION LANGUAGE FOR THE EXPERT VAX CONFIGURATOR (R1) PROGRAM

o OPS5 DEVELOPED BY FORGY SHORTLY AFTER R1 DELIVERED

o MCDERMOTT CONVERTED R1 TO RUN USING OPS5 (BOTH OPS4 AND OPS5 RAN IN MACLISP ON DEC-10S AND DEC-20S)

o DUE TO THE INCREASING MEMORY REQUIREMENTS FOR R1, OPS5 CONVERTED TO FRANZLISP ON VAX

o OPS5 COMPILER IMPLEMENTED IN BLISS-32 (VMS) IN EARLY 1982

o OPS7 CURRENTLY BEING DEFINED BY FORGY

OPS5

o DIGITAL'S MAJOR INTERNAL ARTIFICIAL INTELLIGENCE APPLICATIONS LANGUAGE

o A RULE-BASED (PRODUCTION SYSTEM) LANGUAGE

o CONSISTS OF A COMPILER AND AN INTERPRETER, BOTH WRITTEN IN BLISS FOR EXECUTION ON VAX/VMS

o USED BY THE XCON, ISA, AND XSEL PROJECTS IN THE INTELLIGENT SYSTEMS TECHNOLOGIES GROUP

THIRD PARTY SOFTWARE PROGRAM

o INITIAL EMPHASIS ON TOOLS

--COMMON LISP-20 (RUTGERS)

--INTERLISP/VMS; INTERLISP/UNIX (USC/ISI)

--LOGO, PROLOG

--INTERLISP COMPATIBILITY SOFTWARE

o APPLICATIONS FROM SOFTWARE HOUSES

o GOAL: RICHEST POSSIBLE VARIETY OF SOFTWARE FOR VAX SYSTEMS

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
ONE IRON WAY, MRO2-1/C11
MARLBORO, MASSACHUSETTS 01752

Bulk Rate
U.S. Postage
PAID
Fitchburg, MA
Permit No.
21

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: ____________

Name: ____________

Company: ____________

Address: ____________

____

State/Country: ____________

Zip/Postal Code: ____________

Mail to: DECUS - ATT: Subscription Service
One Iron Way, MRO2-1/C11
Marlboro, Massachusetts 01752 USA