EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/PS 89-65 (CO)
15.12.1989

SOFTWARE DEVELOPMENT AND MAINTENANCE: AN APPROACH FOR A LARGE ACCELERATOR CONTROL SYSTEM

L. CASALEGNO, L. ORSINI, C.H. SICARD

Paper presented at the International Conference on
Accelerator and Large Experimental Physics
Control Systems
Vancouver, British Columbia, Canada
October 30 - November 3, 1989

Geneva, Switzerland

SOFTWARE DEVELOPMENT AND MAINTENANCE:

AN APPROACH FOR A LARGE ACCELERATOR CONTROL SYSTEM

L. CASALEGNO, L. ORSINI, C.H. SICARD
PS Division, CERN, CH - 1211 Geneva 23, Switzerland

Software maintenance costs are presently a large amount of the total life cycle cost of a software system. In case of large systems not only the costs of eliminating bugs, of fixing analysis and design errors, of introducing updates have to be taken into account, but also the coherence of the system as a whole while its parts are evolving mostly independently. The necessity of devising and supplying tools for helping maintenance and programmers in housekeeping and updating has been strongly felt in the case of the LEP preinjector control system. A set of utilities has been implemented to create a safe interface between programmers and files containing the control software. Through this interface consistent naming schemes, common compiling and object building procedures can be enforced so that development and maintenance stuff have not to bother about the details of executable code generation.

Procedures have been built to verify consistency, generate maintenance diagnostics and automatically update object and executable files taking into account multiple releases and versions. The tools and the techniques reported in the paper are of general usage in the Unix environment and have already been adopted for other projects.

1. Introduction

The CERN PS accelerator complex consists of several proton accelerators, two antiproton accumulators, two linacs, various transfer lines and the LEP preinjector. All these machines are controlled from a central room through a network of around 25 minicomputers and 100 microprocessors interfaced to the process hardware through CAMAC. This network includes front end process computers (FECs) which control subsystems of the accelerators [1]. The LEP preinjector is controlled by 2 FECs to which about 20 microcomputers are connected. The minicomputers are linked through the network to the computers driving the consoles of the main control room.

Each microcomputer (called SMACC) contains a MC68000 microprocessor, is located in a CAMAC crate and controls the hardware modules situated in the same crate. The software procedures written to execute the actions requested by the application programmes on the hardware devices have been called Control Modules. Control Modules as described in [2] are basically object oriented pieces of code composed of (i) a frame that performs the selection mechanism, data access and parameter checking, (ii) methods and, (iii) data specific to the Control Module. Control Module is the name of a general superclass including as subclasses Composite Variable Modules (CVMs) for beam variable control, Equipment Modules (EMs) for remote operations on equipment and Interface Modules (IMs) for remote hardware modules access [3].

The frame of the Control Module is invariant whereas methods and data have to be supplied by the Control Module developer. A sizable amount of different computers are involved in the process of software generation and execution for the LEP preinjector control, as depicted in Fig. 1, due mainly to compatibility requirements with the existing PS control system: IBM supports the Oracle DBMS, Vaxes running Ultrix are used for cross development, Norks Data minicomputers act as front end computers and concentrators, MC68000 based SMACCs supervise the equipment control. This configuration has required the creation of a development environment separated from the execution environment. The development environment supplies compilation, debug, test and maintenance facilities, whereas the execution or target environment guarantees the compatibility with the existing system and supplies real-time facilities.

During the development of the LEP preinjector control system we were faced with 4 kinds of problems with regard to Control Module development, upgrading and maintenance

1) Create an homogeneous development environment from the point of view of the developer,
2) maintain simultaneously different versions of the same frame to allow in parallel regular maintenance and new developments,
3) keep track of the files constituting each executable code in the microprocessors to easily rebuild the code in case of upgrading or bug fixing,
4) maintain simultaneously (i) a test application frame to be used on the development machine and (ii) the frame to be used in all target SMACC processors.

This paper describes the solutions adopted to cope with these requirements.

2. The Development Environment

The Unix operating system has been adopted as development environment after the decision of implementing the LPI control system using the C language. Generally speaking, the application programmers should be mainly concerned with their problem at hand rather than with the specificity of the development operating system and its utilities. On the other hand the complex development environment such as the one used for the LEP preinjector required a large amount of Unix facilities and a certain number of dedicated procedures built with standard "bricks" of Unix [4].

This situation involves a certain mnemonic effort and is prone to errors, particularly for the occasional user. Besides, no file naming scheme can be enforced by using plain Unix commands. A consistent naming scheme is very important for automated maintenance.

Thus it has been decided to hide the Unix shell command level from the programmer and to build a menu driven facility including all the actions required to develop and test control modules and to build microcomputer images, i.e. the executable code to be loaded into the microprocessors. The menu is reported in Figure 2. Options are present for editing files, transferring from the IBM and compiling data tables, creating executable code, starting debug and test sessions, loading test sequences, displaying or printing manuals, choosing the frame version one wants to use in building microprocessor core images. For source code maintenance the source code control system sccs(1) [5] has been adopted. Source files can be edited directly from the Unix source code control system repository SCCS.

Test case sequences for covering all the code specific to a Control Module can be designed and maintained with the code itself so that if later modifications are executed the test sequences can be run in order to verify the absence of side effects. An on-line help facility is available to give information about each of the menu options.

For all the options in the menu the user has to supply at most a file name and all the syntax of the shell level commands is hidden. Besides, the menu acts as a filter and invalid answers are rejected. The occasional user makes no effort in having to remember complex commands and a coherent naming scheme for object code files, makefiles, debug files, has been easily established. The naming scheme allows to maintain the whole control system software through automated procedures of verification of the existence in each directory of the required files and of their generation date.

3. Maintaining the frame

The LEP preinjector control is composed of several instances of Control Modules written in different time and by different people. Even if the goal is to obtain a stable situation in which all the Control Modules loaded into the microprocessors contain the same version of the frame code, sometimes it happens that different versions coexist due to enhancements applied to new applications. Homogeneity is re-established normally during shutdowns by retrofitting last versions on old applications. Thus more than one version of the frame target code has to be available during normal operation.

In order to keep just one source of the code and to avoid including into the code itself conditional compilation statements, a maintenance environment has been set up. Figure 3 reports the user interface of the maintenance environment. The maintenance environment is based on the source code management system sccs(1) and a baselining facility written ad hoc. The source code management system sccs(1) has the ability to keep an indefinite number of versions of the same file and to retrieve any version the user asks for. The baselining facility is able to record the versions of each source file the frame is constituted of at a certain moment of time. The collection of all the source files and their version number at a certain time is called a baseline. By means of the information contained in a baseline, any frame version can be restored at any moment by generating a makefile containing for each source file the required version number. Any baseline can be listed or printed; a list of all existing baselines can be obtained too. Baselines can be deleted when no target microprocessor use it any more. A baseline can be also edited when a bug has been detected and it risks to affect the behaviour of all the frame versions present in the target microprocessors. The version of the file in which the bug has been found can be updated.

The version number of a file as generated by sccs(1) is composed of a release number and the number of the version in the release. When a file is updated the source code management system updates automatically the version in the release. When the number of modifications is large enough to require a drastic change to reference new versions, a change of release number can be executed. The release number of all the source files is changed and the version number in the release is set to 1. The baseline numbering scheme roughly follows the release numbering scheme.

It is also possible to build frame versions from the most recent version of all the files present in the source code management system repository SCCS both for target and development environment. A help facility is present also in this case to give details of each option in the menu. The scope of the maintenance environment is the directory in which it is loaded. Any directory can have this maintenance environment for maintaining different versions of the same software product provided the sccs(1) facility is used. As a matter of fact, the environment has already been used for maintaining code other than the frame of the control modules. sccs(1) forbids the access right to the files being edited on a user name basis. Unfortunately, our access rights to software code are based on a "per project" user name. Thus all programmers working on the frame code use the same user name and no file access control could be set up.

4. Building Makefiles for Target Microprocessors

The Oracle data base contains all the information needed to create the data structures and the list of method names associated with the properties of each Control Module [2, 6]. It has been decided to add to this operational data some maintenance information, too. Each method name declared in the data base has associated to it the full name of the file containing the corresponding code. Moreover, a special record in the data base is allocated to keep the names and locations of all files containing applications (normally real-time tasks or diagnostic tasks) to be included into a given microprocessor image.

Building a makefile [3] to generate an image for a microprocessor is not an easy task if one starts from scratch. Many files must always be present even if they are not directly referenced by the application at hand, the frame of the control module and its ancillary files have to be always loaded. Moreover maintaining these makefiles when changing release of general purpose files or frame related files can become very difficult. A method to generate makefiles from a common stub and information contained in the Oracle data base has been developed. The makefiles are built by means of two stubs.

The first stub is common to all makefiles and contains the name of the files to be always loaded into the microprocessors with their correct release and version number and the commands for the linkage and image generation phase. Moreover a backup repository for SMACC images has been set up in the FECs to which the microprocessors are connected [6]. This stub provides the commands to send the generated images from the development machine into the FEC repository. The second stub is dynamically created through the information in the data base. When a new image is built, first a procedure extracts from the data base the full names of all the files containing code specific to the given microprocessor. Then this stub is linked to the first one and the executable makefile is produced and executed.

5. Maintaining source code for two different targets

The task of maintaining the source code for many different target systems is more general than the scope of the LEP preinjector project. However, for the sake of completeness of the maintenance issue, some comments about the method used to cope with this problem are reported. As far as possible, no conditional compilation was used: having the same code in both environments guarantees better test conditions. When incompatibility cases arose, a subset of the C language was chosen so that the code would run on both machines. Instructions present in the language that might have given problems when porting the code like "field" [7] were banned. If one behaves carefully a very portable code can be produced. No compromises with the speed issue had to be made and no extraordinary manpower efforts had to be allocated. Plans for moving the present frame to a VME environment have been already set in place. No sizable problems are foreseen for this porting.

6. Future developments

The number of SMACCs in the LEP preinjector control system is rather large and almost no SMACC contains exactly the same core image. Thus many makefiles have to be created and maintained. A maintenance makefile will enable to selectively execute the makefiles that will find the target image out of date. Updating of this overall makefile will be made through the information contained in the data base.

The possibility to access the Oracle data base from a remote job through SQL-NET will allow to include data table generation into the makefiles. The whole PS control system, including the LEP preinjector is being upgraded by means of the inclusion of a network of workstations. The availability of these powerful machines will allow to build a more sophisticated CASE support. An extended investigation will be made in order to evaluate the possibility to include the present development and maintenance scheme into the new CASE tools.

7. Conclusions

A maintenance environment for widely distributed executables produced from the same source code was set up rather successfully hiding the plain Unix commands under a layer of menu-driven facilities. The intrinsic structure of the Unix operating system and C shell command language, with the availability of a large amount of bricks with which building larger utilities fitted quite well the design requirements of such an environment. The menu driven facility for Control Module building can be considered as a canvas for any application development environment because it basically contains all the actions necessary to obtain a software product:

(i) write and maintain the data part of the software product,
(ii) create and maintain the code part of the software product,
(iii) debug and test the software product,
(iv) create the executables,
(v) document the software product,
(vi) help the user in navigating through the different option by means of a context sensitive help facility.

The acceptance of the development environment by the programmers was quite positive and the general remark is of an enhancement of reliability and a reduced loss of time for occasional users. sccs(1) and baselining were found very useful for the follow up of the modifications and the upgrade of the frame code. Tests of new ideas could be performed safely without the need of keeping multiple copies of the same file; wrong modifications were easily discarded. The Control Module frame maintenance environment is now the only method used to update the Control Module frame code. This experience is a good encouragement to develop this approach of software workbench which will integrate the new workstation tools.

Acknowledgements

The authors would like to thank Ana Paula Pereira for her helpful participation in the early phase of this project and for the production of the backbone of the development environment.

REFERENCES

[1] L. Casalegno et al., Distributed application software architecture applied to the LEP preinjector controls, presented at the 7th IFAC workshop on Distributed Control Systems. Mayschoss/Bad Neuenhar, Fed. Rep. of Germany, September 30 - October 2, 1986.

[2] L.Casalegno et al., Building software modules for driving hardware controlling physical variables: an object oriented approach, presented at the First International Conference on Accelerators and Large Experimental Physics Control Systems, Vilars sur Ollon, September 28 - October 2, 1987, CERN/PS 87-81 (CO).

[3] L.Casalegno et al., Control Module Handbook, PS/CO/Note 88-007, 1988

[4] B.W.Kernighan, R.Pike, The UNIX Programming Environment, Prentice-Hall Software Series,1984

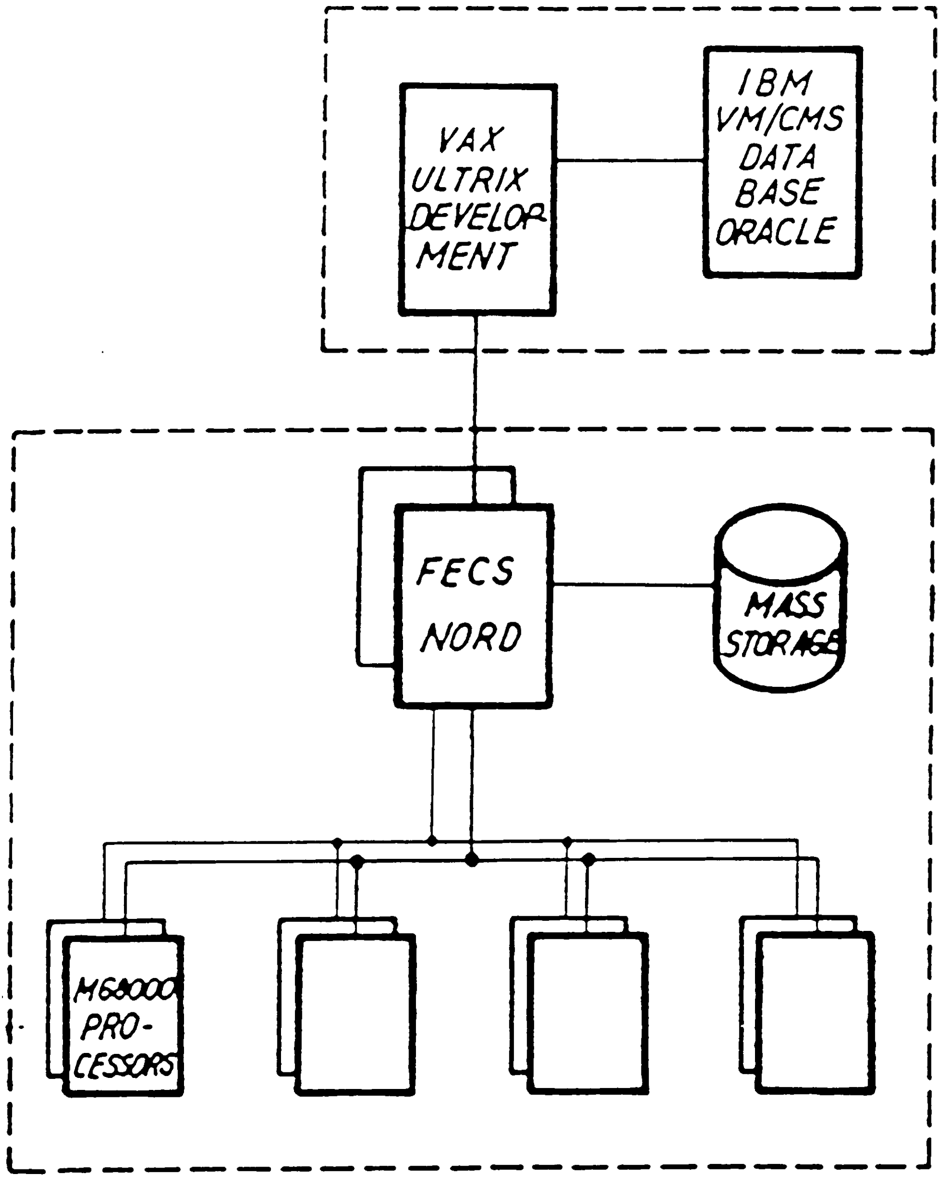
[5] Eric Allman, An Introduction to the Source Code Control System, Project Ingres, University of California at Berkeley.

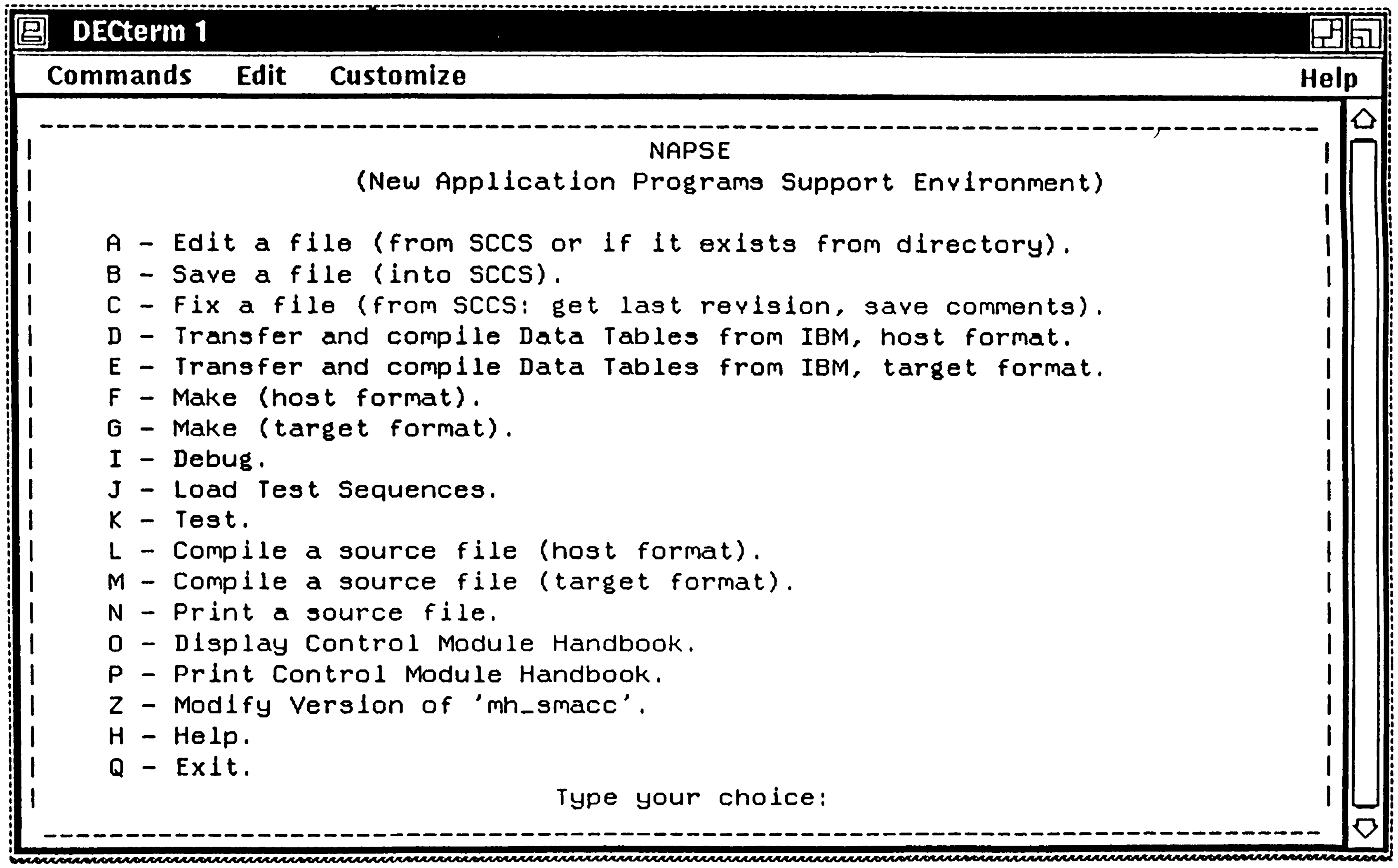
[6] L.Casalegno et al, Process equipment data organisation in CERN PS Controls, this conference

[7] B.W.Kernighan, D.M.Ritchie, The C programming language, Prentice Hall Software Series, 1978

Figure Captions

Fig. 1


VAX ULTRIX DEVELOPMENT
IBM VM/CMS DATA BASE ORACLE
FECS NORD
MASS STORAGE
M68000 PROCESSORS

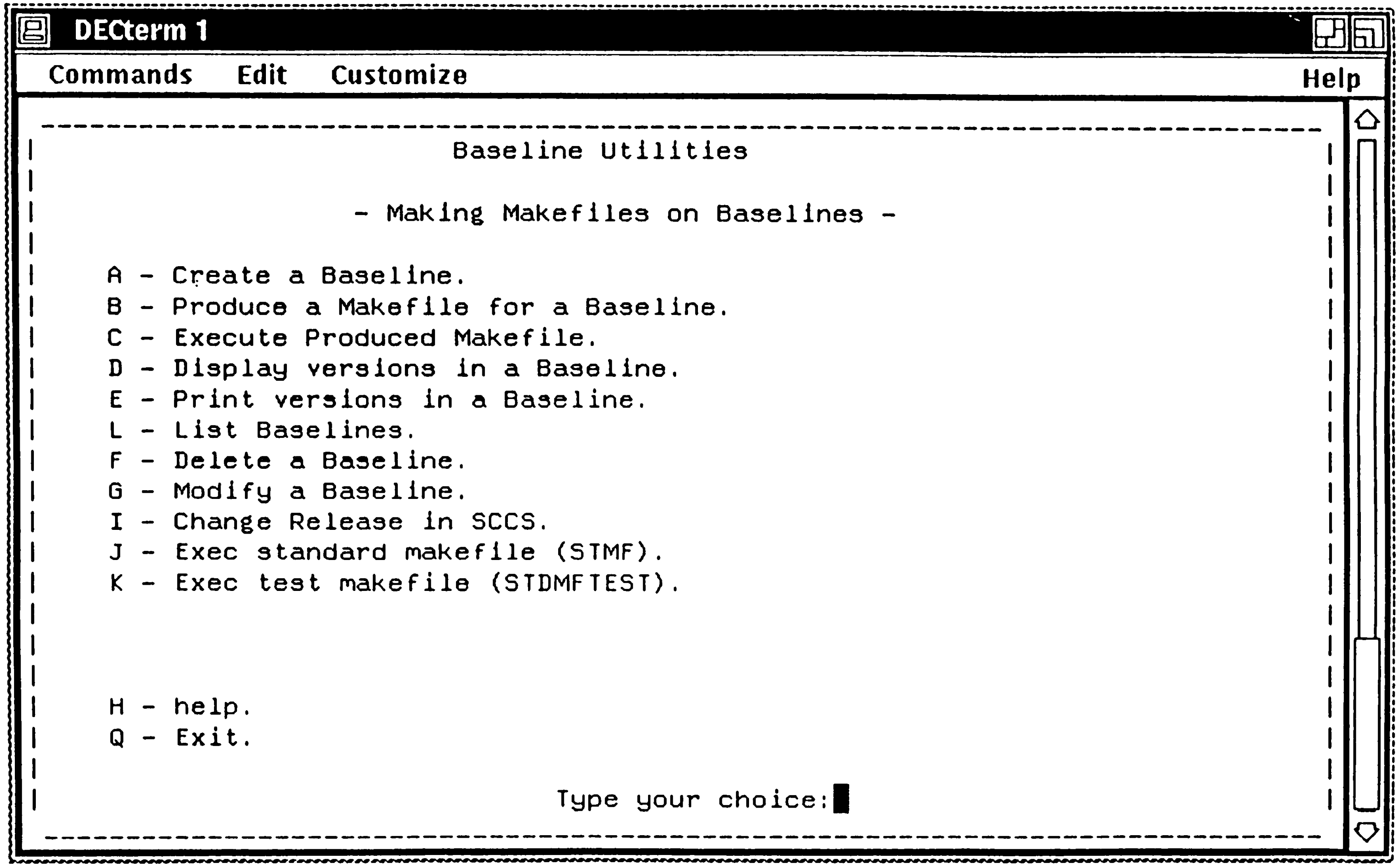


```
                         NAPSE
        (New Application Programs Support Environment)

  A - Edit a file (from SCCS or if it exists from directory).
  B - Save a file (into SCCS).
  C - Fix a file (from SCCS: get last revision, save comments).
  D - Transfer and compile Data Tables from IBM, host format.
  E - Transfer and compile Data Tables from IBM, target format.
  F - Make (host format).
  G - Make (target format).
  I - Debug.
  J - Load Test Sequences.
  K - Test.
  L - Compile a source file (host format).
  M - Compile a source file (target format).
  N - Print a source file.
  O - Display Control Module Handbook.
  P - Print Control Module Handbook.
  Z - Modify Version of 'mh_smacc'.
  H - Help.
  Q - Exit.
                    Type your choice:
```

Fig. 2

Fig. 3.



```
                     Baseline Utilities

              - Making Makefiles on Baselines -

    A - Create a Baseline.
    B - Produce a Makefile for a Baseline.
    C - Execute Produced Makefile.
    D - Display versions in a Baseline.
    E - Print versions in a Baseline.
    L - List Baselines.
    F - Delete a Baseline.
    G - Modify a Baseline.
    I - Change Release in SCCS.
    J - Exec standard makefile (STMF).
    K - Exec test makefile (STDMFTEST).



    H - help.
    Q - Exit.

                         Type your choice:█
```