SLAC – PUB – 4197
January 1987
(M)

# An Environment for High Energy Physics Code Development*

DENNIS E. WISINSKI

*Stanford Linear Accelerator Center*
*Stanford University, Stanford, California, 94305*

## Abstract

As the size and complexity of high energy experiments increase there will be a greater need for better software tools and new programming environments. If these are not commercially available, then we must build them ourselves. This paper describes a prototype programming environment featuring a new type of file system, a "smart" editor, and integrated file management tools. This environment was constructed under the IBM VM/SP operating system. It uses the system interpreter, the system editor and the NOMAD2 relational database management system to create a software "shell" for the programmer. Some extensions to this environment are explored.

*Presented at the Conference on*
*Computing in High Energy Physics, Asilomar, CA, February 2-6, 1987*

* Work supported by the Department of Energy, contract DE – AC03 – 76SF00515.

## 1. Introduction

While many of the software development issues confronting high energy physicists are similar to those encountered in any large software shop, there are some significant differences. One of the persistent problem areas for those doing physics analysis is the lack of effective tools for managing the code that they write for their own use. This code is typically used in conjunction with a larger physics software package developed for a specific experiment. Because of the limited–length file names provided by most operating systems it is difficult to encode much useful information in the file names of these programs or their ancillary files. As a result, it is difficult to remember which programs or files do what. In general it is also difficult to group files together effectively. In those systems that permit tree-structured directories, you can put related groups of files into the same subdirectory, but what do you do when you want to include a file in two or more distinct groups? Usually you make copies for each subdirectory. But then you have to remember to copy the file to all the subdirectories each time that you change it. After a while such files inevitably get out of synchronization. Another frequent problem is compiling and linking a program in the correct "environment." The right virtual disks (or directories) must be used to assure that the correct source code is used. Different compiler optimization levels may be required depending on the source code. You have to make sure that the correct link–libraries are used, and so on.

In an attempt to address some of these problems, I have developed a small prototype file management system called FLEX (FiLe EXtension). It extends the capabilities of IBM's VM/SP CMS file system [1] by allowing one to attach additional attributes to files in the file system. I view this prototype not so much a solution to these problems as a tool for exploring extensions to current file systems, editors and software development environments.

We will look first at the goals and constraints that helped to shape the FLEX system, and then at the design of the FLEX system. Following that we examine

FLEX commands, extended commands and the FLEX editor. We will conclude by exploring some further developments suggested by ideas already incorporated in FLEX.

## 2. Goals and Constraints

There were a number of goals and constraints that guided the development of FLEX. The goals were:

- Provide users with a way to define their own file attributes.
- Provide them with tools to manipulate the files based on the values of these attributes.
- Allow for multivalued attributes.

Constraints that affected the design were:

- It should be easy to learn and to use.
- It should be possible for users to enable or disable all or part of FLEX.
- It should not hide the operating system (CMS) from the user's view.
- FLEX commands should use the same syntax as CMS commands.
- Command resolution should be handled the same way as other VM system products handle it.
- FLEX had to be layered on top of the operating system. I.e., there was to be no tampering with the native CMS file system.
- It had to be designed with the software tools on hand.
- Development time should be held to a minimum.

To stay within the bounds of the last two constraints, the NOMAD2 relational database management system from Dun & Bradstreet Computing Services was employed [2]. This was a software system that SLAC already owned and so it qualified as a "tool on hand." Having the entire range of functions associated

with a relational database system greatly simplified many of the design problems, allowing development time to be held to a minimum.

## 3. Design Overview

The basic strategy in the design of FLEX was to use the CMS file system in its unaltered form, and to keep track of the user-defined file attributes within a NOMAD2 database. (See Figure 1.) Commands issued by the user had to be tested to determine if they were "extended" commands, that is, commands that made use of the extended file attributes. To prescan all commands it was necessary to build a "shell" for the user. This shell program is written in REXX [3] which is the VM/SP system interpreter. To avoid the large overhead incurred each time the database is entered, the database is kept active at all times. This is accomplished by use of the NOMAD2 Programmer Interface (NPI2) [4]. A small assembler program containing a call to the NPI2 entry point, and naming the entry point of the shell is executed to set up the FLEX environment. In effect, the REXX shell program is a NOMAD2 "macro." This shell program consults the database each time a command is issued. It allows access to the system editor (Xedit), and controls the updating of CMS files. Figure 2 show a "normal" editing session and Figure 3 shows a FLEX editing session.

## 4. Database Design

The FLEX database was designed using NOMAD2. A relational database design was chosen for its simplicity and flexibility [5]. Each primary entity in FLEX is represented by a table (a NOMAD2 master) in the database. The tables for primary entities are CLASSES, MINIDISKS, FILES, and XCOMMANDS (extended commands). CLASSES is actually a metaentity. The instances in this table are just the other entities in the database. It is necessary to keep track of the MINIDISKS on which the files reside since the FLEX editor and other extended commands need to be able to access minidisks on which files of interest are located, even if they are not normally attached to the user's virtual machine.

FILES contains the fileids of all of the files known to FLEX. The names of all extended commands known to FLEX are stored in XCOMMANDS.

In addition to the tables corresponding to the primary entities, there are the tables DISK_TRAITS, DISK_TAGGING, FILE_TRAITS, FILE_TAGGING, and PREREQUISITES. Instances in the DISK_TRAITS table show the *names* of any disk attributes the user has defined. Likewise, FILE_TRAITS keeps track of the names of user–defined file attributes. The two TAGGING tables are where the *actual values* of attributes belonging to disks (or files) are stored. The PREREQUISITES table is a collection of all conditions that must be satisfied before extended commands may be executed. This table arrangement allows multivalued attributes (traits) for disks and files, and multiple prerequisites for commands. (See Figure 4.)

## 5. FLEX Commands

One may use NOMAD2 commands to maintain the database, but a small set of commands with simple syntax is provided for most FLEX maintenance. These commands are ADD, REMOVE, MODIFY, and SHOW. All of these commands operate on any of the objects that are listed in the CLASSES master (disks, files, commands, etc.). ADD, REMOVE and MODIFY are used to add, remove and change files, disks and commands that are in the database. SHOW is used to examine database contents. Examples of command syntax are:

```
ADD writelock TO FILES
ADD writelock yes TO FILE profile exec dzw191
```

The first command adds an attribute field to all files. The second sets the value of the attribute WRITELOCK to YES for the file named PROFILE EXEC on the disk named DZW191.

## 6. Extended Commands

Any command that makes use of extended attributes for disks or files is called

an extended command (XCMD). FLEX provides two methods for designing extended commands. Extended commands may be written in NOMAD2 procedural language or may be written in REXX using imbedded NOMAD2 statements. This method is necessary for complex commands, such as the command that invokes the FLEX editor. Writing procedures is unacceptable to most users, so an alternative method of extending commands is provided. This consists of establishing "prerequisites" that must be met before a command can be executed. For example, suppose that the user has created a file attribute called PROTECTED that can have a value of YES or NO. If the value is YES, then any action that tries to alter or erase the file must be prohibited. To create an extended ERASE function, the user merely adds the name of the function to the COMMANDS table, selects a parsing template from a small set that has been provided, and adds a condition to the PREREQUISITE table for the command ERASE. The condition is an expression which must evaluate to a logical TRUE before the command will be executed. In this case the condition is PROTECTED = NO. Extended commands can be enabled and disabled individually or as a class. The diagram in Figure 5 shows how commands are processed.

## 7. FLEX Editor

FLEX provides a "smart" editor for editing files that have extended attributes. This editor will (if necessary) link and access the disk on which the file being edited is stored, and release and detach that disk when editing is finished. It enforces any prerequisites that are indicated for the file. The actual editing is done by the system editor (Xedit). If there is a profile attribute for that file, an Xedit profile of the same name will be run before editor control is turned over to the user. In addition, the editor will restore any editing attributes such as position of the current line, cursor position, etc., if any of those editing attributes have been attached to the file.

## 8. Status

The FLEX prototype has undergone several changes since its inception. The original impetus for the project was the idea of providing the user with a method of attaching attributes to files. Originally FLEX was to have only its own set of commands to manipulate the files — an editor, a file list facility, and few commands like COPY. As work progressed, it became clear that a merely static set of attributes, while useful for grouping files, was not nearly satisfying enough. If one could extend file attributes, then why not commands also? But writing extended commands would prove difficult most users since the commands are usually written in NOMAD2 procedural language. The key to useful command extension proved to be the concept of "prerequisites" — simple logical conditions that must be satisfied before the command (for example, a CMS command) can be issued. The prototype schema presently incorporates all of the above ideas. Some built-in commands are in place (ADD, SHOW). Some are partially built and working, but not yet complete (editor). Others have not yet been started (e.g., a full-screen file manipulation facility). The NOMAD2 database editor and other NOMAD2 commands currently substitute for incomplete commands.

## 9. Future Directions

After dealing with extended entities for a time, one begins to think of them as new objects rather than "extended" objects. They have their own properties and are manipulated by their own set of operators. The step to an object–based programming environment is a small one. Certainly this is not a new idea. Implementations of such systems have been around for some time, the most famous probably being Xerox's Smalltalk [6]. A similar approach in a VM environment is Smallworld [7]. But these environments are likely to seem strange to high energy physicists. So in the near future I think that FLEX will incorporate only a couple more new objects – probably something like "environments" or "configurations." Feedback from high energy physics users will determine what happens after that.

## Acknowledgements

I would like to thank Karen Heidenreich and Terry Schalk for their comments and suggestions throughout the development of this prototype and also Frank Rothacker for his NOMAD2 interface program which plays a key role in the FLEX system. This work was supported by the Department of Energy under Contract DE-AC03-76SF00515.

## References

[1] Virtual Machine / System Product Introduction, IBM Systems Library, GC19-6200.

[2] C.J. Date, Database — A Primer, Addison-Wesley, Reading, Massachusetts (1984) 115–132.

[3] M.F. Cowlishaw, The REXX Language, Prentice Hall, Englewood Cliffs, New Jersey (1985).

[4] NOMAD2 Reference Manual (vol. 2), D&B Computing Services, Wilton, Connecticut (1985) 16-1–16-26 .

[5] C.J. Date, An Introduction to Database Systems (vol. 1) 4th ed., Addison-Wesley, Reading, Massachusetts (1986)

[6] A. Goldberg, Introducing the Smalltalk-80 System, Byte 6(8) August, 1981 14–26.

[7] M.A. Laff, B. Hailpern, SW2 — An Object–based Programming Environment, in Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, (Seattle, Washington, June 1985), ACM, New York (1985) 1–11.

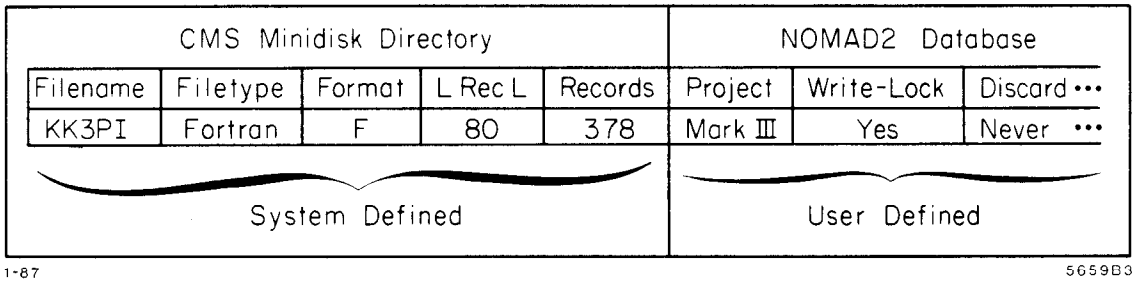


**Figure 1. File Attributes**

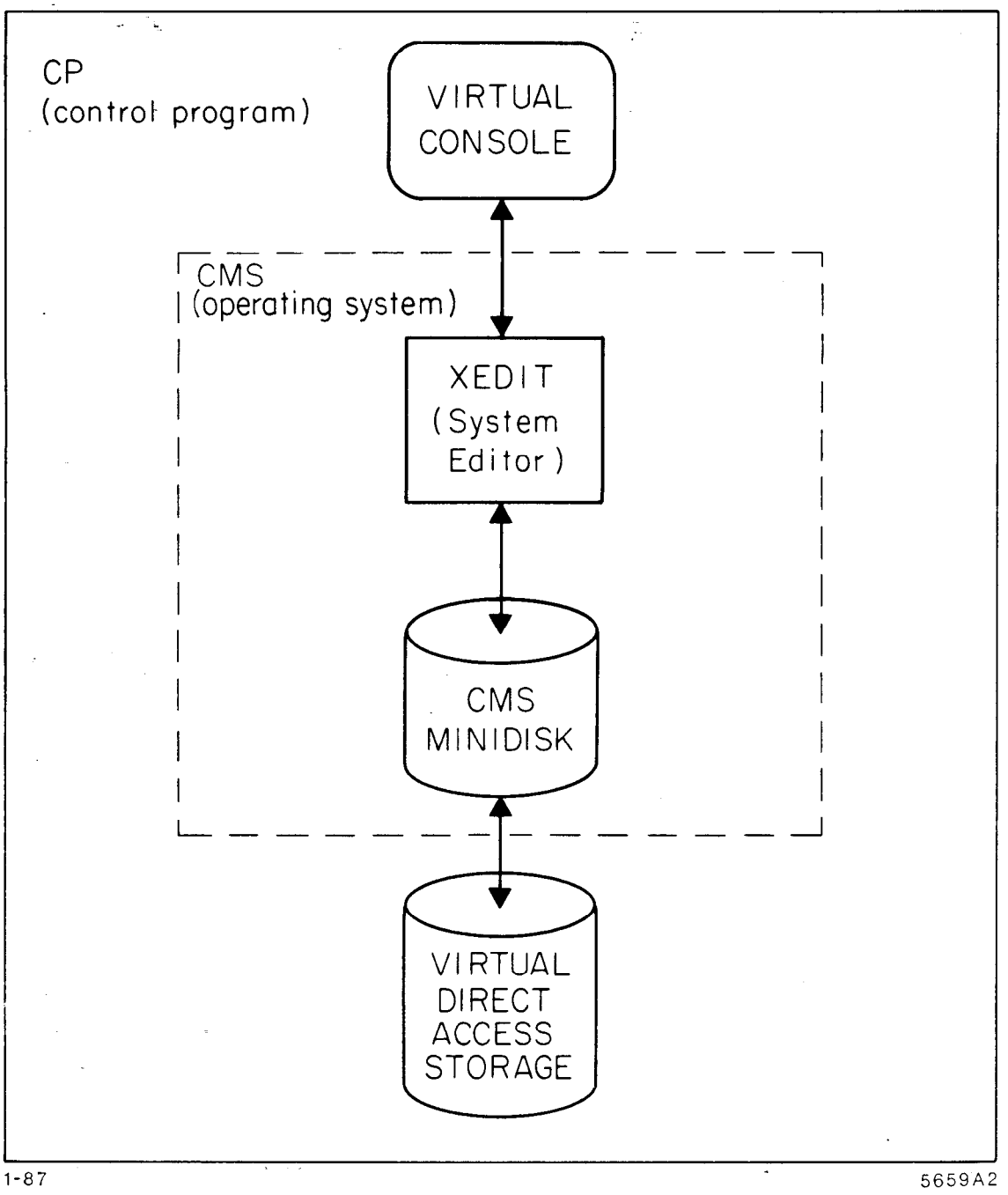


Figure 2. Normal Editing

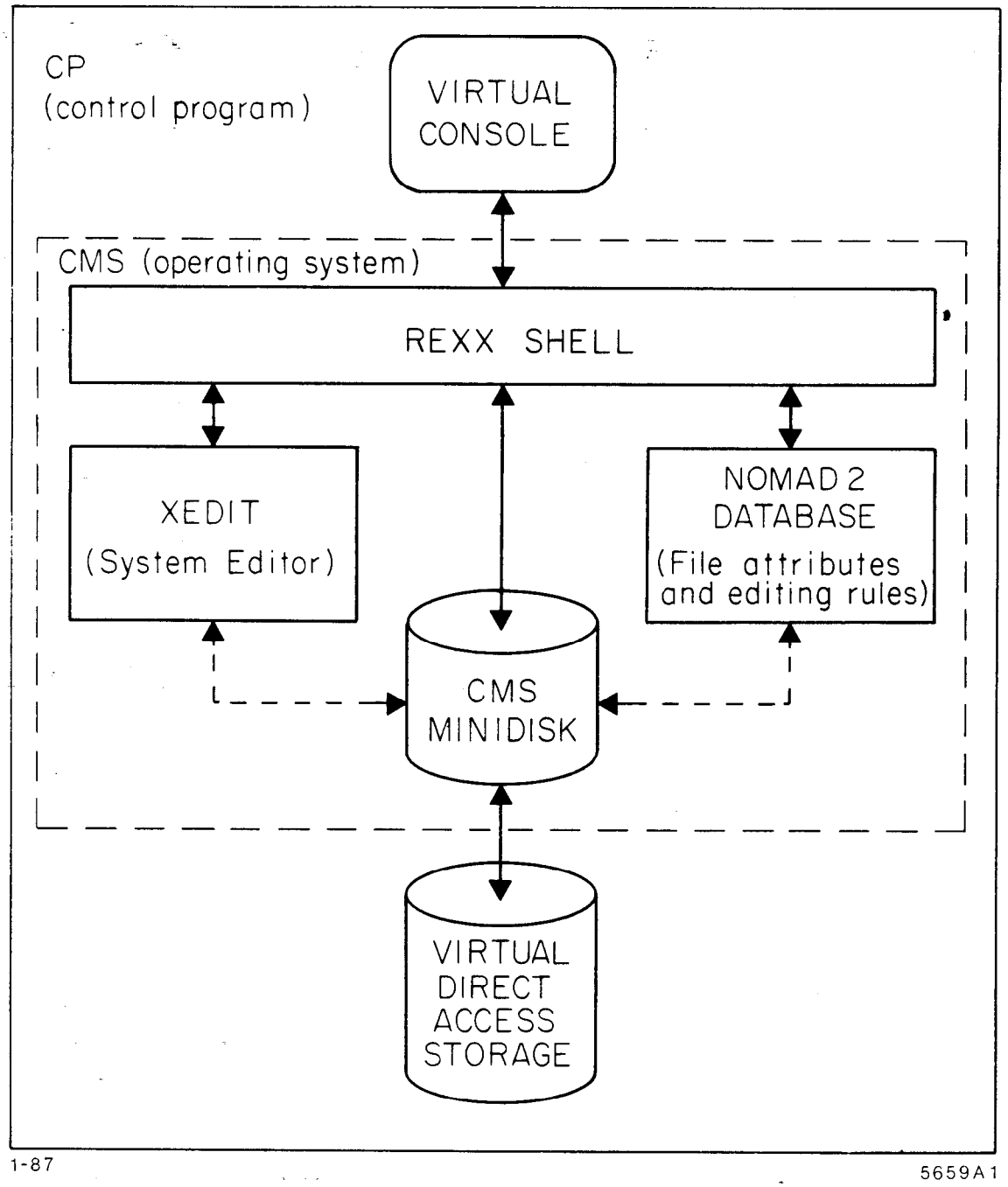


Figure 3. FLEX Editing

| FILENAME | FILETYPE | PHYSICS-TYPE |
|---|---|---|
| TRACKFIT | FORTRAN | PSI |
| | | PSI" |
| | | F |

1-87

5659A4

Figure 4. Multivalued Attributes

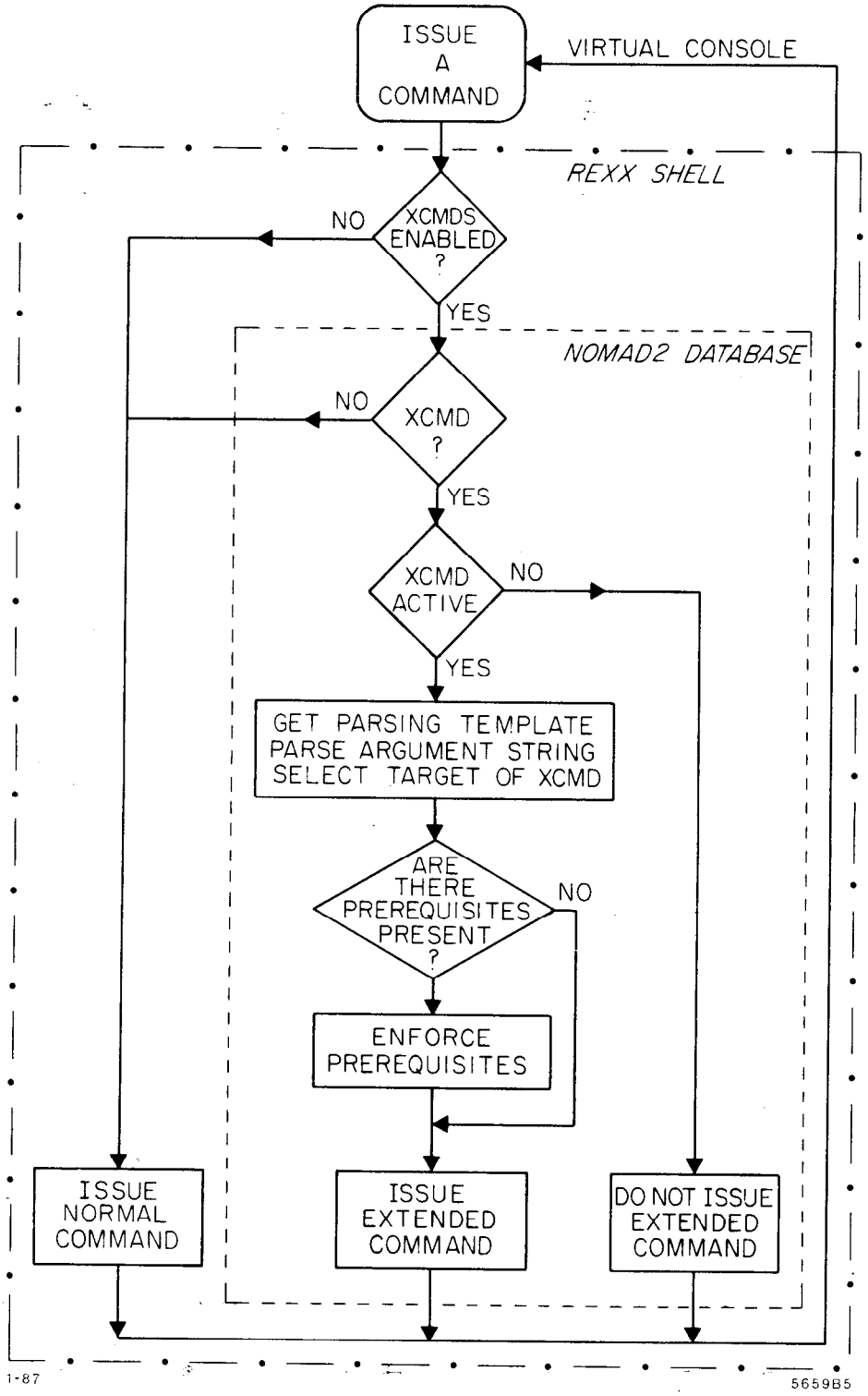


Figure 5. FLEX Command Processing