STATISTICAL REPORT

User's guide to Express: A tool for building knowledge-based systems for statistical data analysis

by

Jan H. Aarseth and Ivar Heuch

Report no. 27
July 1996

Department of Mathematics
UNIVERSITY OF BERGEN
Bergen, Norway

Department of Mathematics
University of Bergen
N-5007 Bergen
Norway

ISSN 0333-1865

User's guide to Express: A tool for building knowledge-based systems for statistical data analysis

Jan H. Aarseth and Ivar Heuch

Statistical Report No. 27
July 1996

PREFACE

This guide describes the current version of the program system Express, which serves as a tool for building knowledge-based systems for statistical data analysis. The development of this program dates back to 1986, when Carlsen [1,2] constructed a system for comparing the location parameters of two samples. This system was designed specifically for use in combination with the statistical program package BMDP. The knowledge component was not separated physically from the main body of the system. Ottersen [3] developed a new version incorporating an exchangeable knowledge base. This made it easy to modify the knowledge included in the system and to change to a completely different domain of knowledge. In addition, it became possible to use other statistical software.

The first versions of Express were implemented in Fortran on computers with no facilities for advanced user interfaces. Aarseth [4,5] developed the system further with a DOS implementation for IBM-compatible personal computers. The program was still written in Fortran [6], but several assembler routines were included to improve the user interface. The improvement and revision process has continued until quite recently. The experiences made by Irgens [7] were particularly useful during this phase. She constructed a set of rules for model building in logistic regression within the framework of Express. Subsequently, a more extensive set of rules for logistic regression [8] has been completed.

It must be emphasized that the development of Express has basically been a research project. The particular solutions to practical problems selected by the system do not necessarily represent an optimal approach, taking all possible aspects of the data into account. The sets of rules which have been implemented must be regarded as relatively simple examples. The program system could certainly be made more user-friendly in many regards, but it was considered more important to try out various general possibilities, rather than spending an excessive amount of time on the refinement of user interfaces.

Chapters 1-3 of this guide are intended for users of the PC version of Express. The attention of beginning users is in particular directed to the sample sessions described in Section 2.7. Chapter 4 differs from the remainder of the guide in various respects. It deals with a UNIX version of Express running under the X Window System. This version was primarily developed to explore simulation as a tool for studying the underlying properties of the general analysis performed. At present there is a significant difference in speed between the PCs used for executing Express and the work stations. However, the recent development of faster PCs should also make it realistic to incorporate a simulation module into a future PC version of Express.

References:

[1] Carlsen, F. (1986) Express - An expert system which applies existing statistical packages. Thesis for the cand. scient. degree. Department of Mathematics, University of Bergen, Bergen (in Norwegian).

[2] Carlsen, F. and Heuch, I. (1986) Express - An expert system utilizing standard statistical packages. In: Compstat - Proceedings in computational statistics (F. De Antoni, N. Lauro and A. Rizzi, eds.), Physica-Verlag, Heidelberg, 265-270.

[3] Ottersen, G. (1988) Express-88. A statistical expert system shell. Thesis for the cand. scient degree. Department of Mathematics, University of Bergen, Bergen (in Norwegian).

[4] Aarseth, J.H. (1990) Express - A tool for construction of knowledge-based systems in statistics. Thesis for the cand. scient. degree. Department of Mathematics, University of Bergen, Bergen (in Norwegian).

[5] Heuch, I., Aarseth, J.H., Ottersen, G., and Carlsen, F. (1990) Adaptation of "Express" to the IBM PC: A tool for building knowledge-based statistical systems using existing packages. In: Compstat 1990 Software Catalogue, Dubrovnik, 13-14.

[6] The Microsoft Corporation (1987) Microsoft Fortran for the MS-DOS operating system.

[7] Irgens, Å. (1991) A knowledge-based system for logistic regression using Express. Thesis for the cand. scient. degree. Department of Mathematics, University of Bergen, Bergen (in Norwegian).

[8] Aarseth, J.H. and Heuch, I. (1996) Logistrule: A knowledge-based system for logistic regression. Statistical Report no. 28, University of Bergen, Bergen.

Chapter 1

INTRODUCTION

1.1 BASIC IDEAS 6
1.1.1 The purpose of Express 6

1.2 EXTERNAL REQUIREMENTS FOR USE OF EXPRESS 9
1.2.1 Amount of internal memory required 9
1.2.2 DOS version and video modes 9
1.2.3 External packages 9
1.2.4 Implementation of rules 10

1.3 INSTALLATION 11
1.3.1 Installation of the main program in Express 11
1.3.2 Installation of the libraries 12

1.4 BASIC PROGRAM ARCHITECTURE 13
1.4.1 Structure of programs in Express 13
1.4.2 The stack utilized by Express 15
1.4.3 Rules 15
1.4.4 The main menu and the data storage 17

1.1 BASIC IDEAS

1.1.1 The purpose of Express

Computers play an important role in contemporary applied statistics, and during the last few decades a great many new statistical computer packages have been released. Regarding the interaction between packages and users, there are two contrasting points of view concerning all the new possibilities that the users are able to take advantage of:

Some systems are so difficult to use, or they include so many different options, that it is impossible to have complete control over all the methods available. It seems reasonable that only a small group of highly specialized users will gain complete insight into these systems, whereas most ordinary users will not be able to take advantage of the new developments.

On the other hand, new systems have opened up vast opportunities for inexperienced users to perform their own statistical analyses. These users may be pleased to get quick answers to their original problems, but they do not inquire into the precision and justification of the results obtained. It appears that the power of the packages is almost dangerous to this kind of inexperienced users.

To reconcile these points of view, it is obvious that it would benefit the statistical community if the first restricted group of highly specialized users could formalize their knowledge and translate it into knowledge-based systems. This would offer a much safer approach to dealing with practical statistical problems for the second category of users referred to above.

The program system Express constitutes a tool for constructing knowledge-based statistical systems in data analysis which require repeated cycles of statistical analysis on a given data set. Express makes it possible to specify chains of rules in such a way that intermediate results determined by execution of standard program packages may affect the type of analysis to be carried out in subsequent steps. These intermediate results are assumed to be found in the ordinary output file produced by the packages involved, and they are stored in a special data base (working memory) maintained by Express. If the rules refer to statistical quantities which have not yet been found for the data set in question, the system will automatically generate control language for the appropriate external package, initiate execution of this package, scan the output, and store the relevant results.

At entry to Express, the user is presented with a menu providing a choice between different sets of rules applicable to various general statistical situations. For each set of rules, the user may decide to address different problems relating to the data, corresponding to separate levels of complexity in the analysis involved. The system then uses a flexible kind of backward chaining in order to solve the problem selected. Depending on system options, which may be

changed during the session, brief or comprehensive explanations are given of the sequence of rules invoked. If any rule refers to an unknown quantity that must be determined by an external statistical package, the user may watch how the system sets up suitable control language for this package, and how the output is subsequently scanned for the information desired, after the package has been executed by Express. At any time, the user may inquire about the values stored in the data base, or set or change particular values of this kind.

New sets of rules may be introduced by the knowledge engineer by writing particular subprograms returning values of unknown quantities to Express, in accordance with simple general specifications. In this way additional rules may be compiled and incorporated into the main program. In the applications considered until now, rules have been specified using Fortran, but in principle any language can be used which permits subprograms to be linked to the compiled version of Express.

As indicated above, there are different potential user categories of a system of this kind. The end user may apply Express to get an answer to a specific statistical problem. Chapter 2 provides a guide for such users. On the other hand, the operations performed by the knowledge engineer go far beyond those of the ordinary user. The knowledge engineer should determine, possibly in collaboration with other statisticians, the general structure of the analyses to be performed, whereas the end user takes advantage of the knowledge already stored in Express. Chapter 3 provides the necessary background for a knowledge engineer using Express.

At this point, it will be convenient to introduce a few technical terms which will be referred to frequently in this guide. At entry to Express, the user must select a general statistical area that should be considered. This is done in the "sets of rules menu". The different "sets of rules" that can be selected are completely independent. The process initiated when the system executes the analysis required, is called a "chaining of rules". This is because the analysis is broken down into relatively small parts, such that each part constitutes a separate "rule". Typically a "chaining of rules" aims at finding the values of several important statistical quantities. Adopting the terminology of general knowledge-based systems, we shall refer to each such quantity as a "slot". For example, a "slot" may be a measure of location in a distribution, a coefficient in a fitted model or any other numerical quantity. A "slot" may also represent the answer to a particular question raised during the "chaining of rules". Finally, a "slot" can be a block containing a more comprehensive set of information, such as a plot or a table.

"Slots" are defined in the "code file", in accordance with general specifications in Express. Each definition includes a name and a brief explanation, as well as an indication of which rule can determine the value in the "slot", and a specification of legal values. The actual "slot value" is stored in a separate file. In combination with the separate "rules", the "code file" constitutes the knowledge base of a particular "set of rules". As mentioned above, a "working memory" is used to keep track of the current state. The "working memory" consists of two

parts. The first part includes particular locations reserved for storing information about all current "slot values". This part of the working memory will be referred to as the "data base". The second part contains the special Express "stack" which is used to keep track of "rules" that must be completed in order to reach the final goal.

1.2 EXTERNAL REQUIREMENTS FOR USE OF EXPRESS

1.2.1 Amount of internal memory required

Considering the internal memory required by Express, the most critical situation occurs when Express is executing an external package. However, before Express initiates such an execution, the system removes all unneeded parts of itself from memory, leaving only 8K RAM. This means that the PC running Express must only have 8K memory more available than that required by any relevant external package. Taking into consideration that Express runs external packages in batch mode, which generally requires less memory than interactive mode, this does not impose any serious limitation. To execute the general part of Express, at least 330K RAM is needed, but to access all subprograms of the system, it is recommended that Express be run on a PC with at least 640K RAM.

1.2.2 DOS version and video modes

Express can only run under DOS version 3.0 or later. It cannot execute in sharing mode in OS/2. Express has not been adapted to Windows.

In practice, a colour monitor must be available to run version 2.0 of Express. With black and white monitors, some of the windows produced by Express may not be visible. Express includes a program for modifying the current video mode. This can be useful with black and white monitors which allow several shades of gray. To execute this program, simply type \EXPRESS\MODER on the command line. The program will then display the current state, and it is possible to change to a different video mode. In general, only modes between 0 and 3 and perhaps 7 (ordinary black and white) should be selected for use with Express, as the remaining numbers represent graphics modes rather than text modes.

1.2.3 External packages

For obvious reasons the packages used by Express to calculate the statistical quantities cannot be included with the program version distributed. Such packages must be acquired through ordinary commercial channels. If one is anxious to run Express in a certain situation but has no access to the particular package required, it may be possible to substitute another package performing similar computations, but some recoding will be needed in the code file (see Chapter 3). Version 2.0 of Express includes three relatively simple sets of rules for statistical analysis, and in addition a more comprehensive set of rules is available for logistic regression. These sets utilize the following packages:

Set of rules	External statistical package
Comparing location parameters for two samples	SAS version 6.3
Regression analysis with at most four independent variables	SAS version 6.3
Logistrule: Logistic regression with at most ten independent variables	BMDP version PC90
One-way ANOVA with up to six groups	BMDP version PC90

In the particular implementation of rules described here, it is assumed that SAS is available in the directory C:\SAS and that BMDP is stored in the directory C:\PC90. The selection of external packages is to some extent arbitrary in each case, and certain sets of rules might just as well have been based on other packages. Even interactive packages such as StatXact can be used within the framework of Express.

1.2.4 Implementation of rules

The main program of Express has been written in Version 4.1 of the Microsoft Fortran (and in certain parts, in assembler). This also applies to the sets of rules developed so far. Each set of rules consists mainly of a collection of calls to general utility routines written in Fortran, included in Express. During the systems development it was considered essential that it should be easy to incorporate future additions. This is achieved by a library of standard auxiliary routines which can be used in the construction of sets of rules. The main structure of any set of rules must be the same, and it is our hope that this library will facilitate the actual specification of rules. In some regards it is also an advantage that Express uses standard programming in the construction of new sets of rules, since it provides the knowledge engineer with great freedom. For a knowledge engineer well acquainted with the system and Fortran programming, it should not be too difficult to exploit the facilities provided.

Other Microsoft compilers, for languages such as Pascal, Basic and C, allow calls to be made to subroutines written in Microsoft Fortran. This makes it possible for the knowledge engineer to specify rules in other languages than Fortran.

1.3 INSTALLATION

1.3.1 Installation of the main program in Express

The main program and some relatively simple sets of rules are included on installation diskettes 1 to 4. In addition a diskette is supplied with the necessary libraries for constructing new sets of rules. Before the installation starts, one should make certain that no essential information is located in the directory C:\EXPRESS, as this directory will be overwritten. If the directory does not already exist, it will be created. To start the installation, simply type A:\INSTALL on the command line with diskette no. 1 in drive A and change the diskette when prompted. Express will then be installed automatically in the directory C:\EXPRESS. This position cannot be changed.

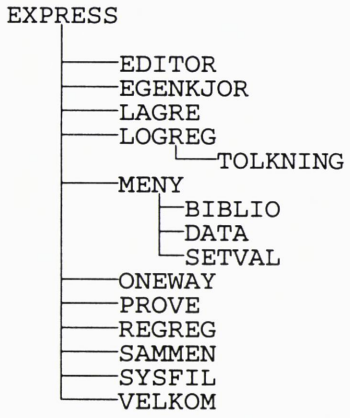


Figure 1.1 The organization of directories after installation of Express.

Figure 1.1 shows the directories created during the installation. Some of these directories include files needed in connection with particular sets of rules. Such files may be moved to other locations on the current disk or to another drive, provided that certain codes are changed in the system file (see Section 3.4.1). The directories containing files for the different sets of rules are:

Directory	Set of rules
ONEWAY	One-way ANOVA with up to six groups.
REGREG	Regression analysis with at most four independent variables.
SAMMEN	Comparison of location parameters for two samples.

LOGREG	Logistrule: Logistic regression with at most ten independent variables.
EGENKJOR	This is not a proper set of rules, but must be regarded as a special facility to allow the use of Express as an editor to set up commands and execute a particular external package. In the menu for selection of sets of rules, this alternative is called "Direct access to an external package" (see Figure 2.1). It should be noted that some knowledge about the file structure in Express is required to run this "set of rules".
PROVE	This is an extremely simple set of rules, merely intended to illustrate the structure of rules to new users.

Each directory in this list contributes at least six files to the set of rules in question. This includes a file collecting output from execution of external packages and a log file describing the analyses carried out. The code file includes various information about the statistical quantities to be determined. After a statistical analysis, the values of these quantities are stored in the data base. One file is used to store plots and tables and another one is used to record interrelations between quantities considered. The actual file names associated with each set of rules are listed in Appendix B.1. The EXE file containing the executable program for any particular set of rules is also located in the same directory. Some sets of rules include an additional help file with information which the knowledge engineer considers useful to the end user. Typically a quick introduction to the analysis is provided.

1.3.2 Installation of the libraries

To introduce a new set of rules, it is necessary to link the compiled file in the Fortran language (or other languages, see Section 1.2) to the library in Express. Files needed for this purpose are located on installation diskette no. 5. The files involved are:

File	**Description**
NYREGLER.LIB	Contains routines that can be called in any set of rules.
NYFELL.LIB	Contains particular routines common to the entire program system.
TILLEG.LIB	Version 2.0 of Express includes certain basic assembler routines for controlling the screen, which have been collected in this library.

These files are usually copied to a directory named C:\LIB.

1.4 BASIC PROGRAM ARCHITECTURE

1.4.1 Structure of programs in Express

In order to achieve as efficient use of the memory (RAM) as possible, Express has been divided into several distinct subprograms. A main concern during development was the wish to enable Express to execute external packages without exceeding the limit set by RAM. This was done by creating a shell, written in assembler code, that invokes the remaining parts of the system (Figure 1.2). This shell is the only part of Express resident in memory during the execution of an external package. It was also considered essential that the particular subprogram organizing data storage should be able to execute outside the regular main program. Hence, this subprogram is called directly from the outer shell.

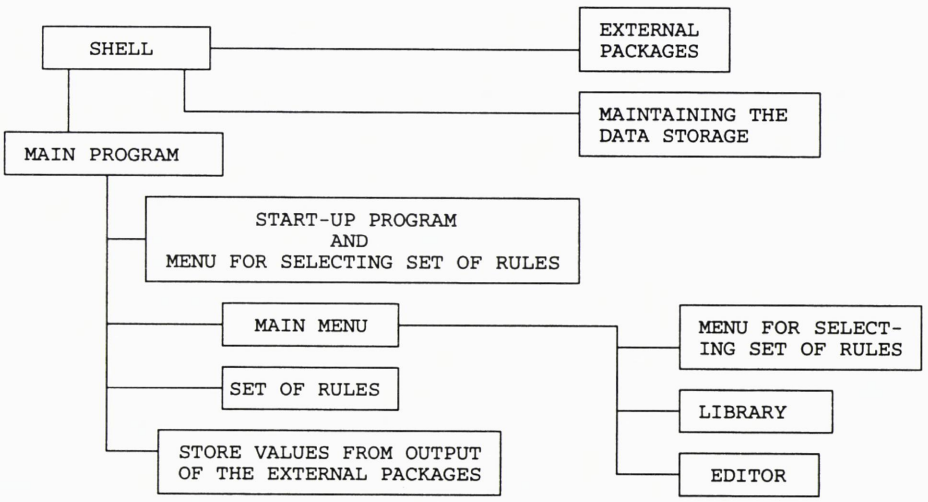


Figure 1.2 Relations between different parts of Express.

As indicated in Figure 1.3, the first action taken by Express is to execute the initiating programs, including the menu for selecting a set of rules. The next step is to activate the main menu, which offers a choice between several options relating to manipulation of statistical quantities and viewing the most important files in the system. More comprehensive information about these facilities is given in Chapter 2.

From the main menu it is possible to start a statistical analysis, corresponding to a chaining of rules. All rules pertaining to a general type of statistical analysis are connected through the main program belonging to that set of rules. When a rule is invoked, two different kinds of actions may be taken. On the one hand, it may turn out that all quantities needed to complete the rule have already been found or can be determined quite easily. This means that the rule can be dealt with immediately in its entirety, and the system can then proceed to the next

rule, unless the current rule was the last one in the analysis. On the other hand, it frequently turns out that the rule in question must execute another rule or an external package before the information becomes available which is needed in order to complete the current rule. To keep track of the rules which have not yet been completed at any particular moment, the system makes use of the stack in the working memory.

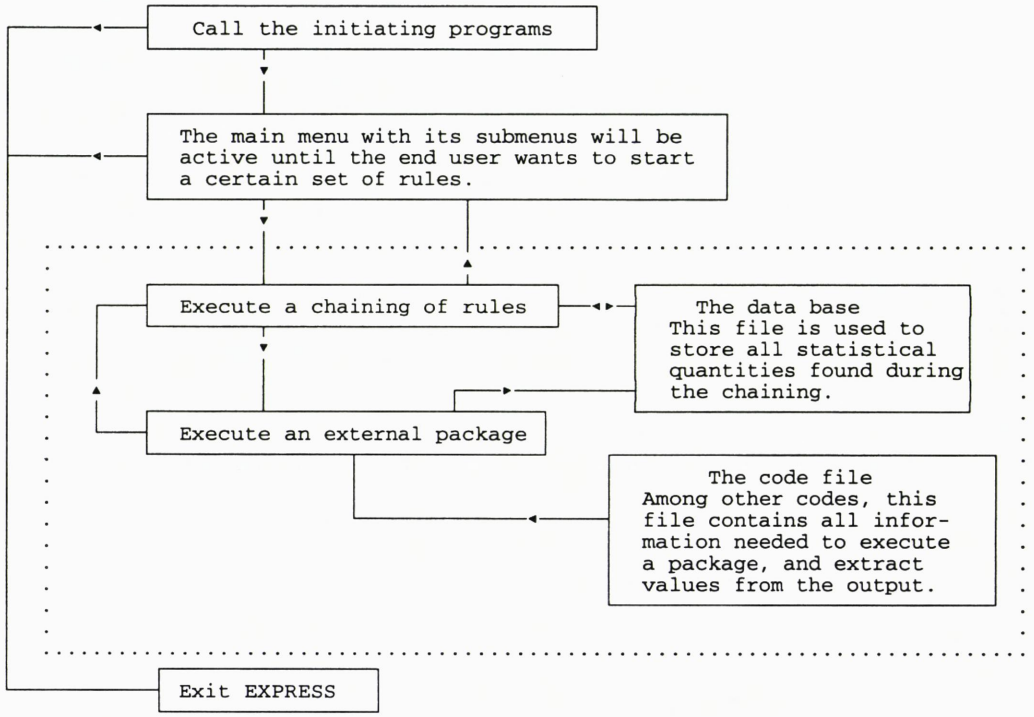


Figure 1.3 The general program flow during execution of Express.

The dotted line in Figure 1.3 indicates the action taken in a chaining of rules. The two most important files during this operation are the data base and the code file. The data base is used to store all quantities determined by rules or external packages. The code file includes definitions of all statistical quantities, represented by slots within the framework of Express. In addition, all information needed concerning initiation and execution of external packages is stored in this file. The first action taken when the system is about to execute a package is to generate a file with all necessary commands, written in a job control language adapted to the particular package. This file will then be used as input during execution, and the external package will generate some kind of output in a file. This output is scanned by Express in search of statistical quantities of interest, which are then stored in the data base for later use. After execution, Express continues with the same rule that led to the execution of the package, in the hope that the rule can now be completed. The process described above can be repeated several times before the final result is reached in a particular rule. At this time the main menu will be reactivated, and the user will have an opportunity to manipulate statistical quantities (slots) or to take a look at files describing the analysis performed.

1.4.2 The stack utilized by Express

The stack plays a very important role during a chaining of rules. The purpose of the stack is to keep track of all rules that remain to be completed at any time. Each possible rule connected with a general statistical problem is identified by a particular number which can be referred to in the stack. This number is put on top of the stack when the particular rule is activated, and the number will be removed from the stack when the rule has been completed. Other rules may be started in the meantime in order to determine unknown values, and the corresponding rule numbers are put on top of those already present on the stack. When a rule number is removed from the stack, the system reads the next number below. This number indicates which rule the system should return to.

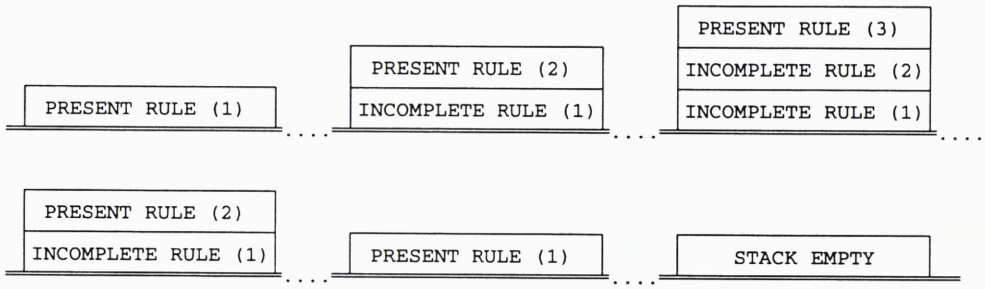


Figure 1.4 Illustrating the use of the stack in Express.

Figure 1.4 gives a simple example of how the stack may be used by Express during the chaining of rules. Rule no. 1 is activated first, and this rule then starts rule no. 2. Finally rule no. 3 is activated before the process is reversed. Rule no. 3 is completed, and it turns out that rule no. 2 can also be completed on the basis of information gathered so far. Thus rule no. 2 is removed from the stack, and in due turn the same happens to rule no. 1. The stack is now empty, and the chaining of rules has finished.

1.4.3 Rules

A set of rules is represented by several files, coded by the knowledge engineer, and a program written in Fortran (or possibly another compatible Microsoft language). Each particular rule is given by a subroutine in this program. The main program for the set of rules governs the flow between separate subroutines corresponding to different rules, utilizing the stack described above. When a return is made from a rule, an address will be supplied indicating where the program should continue. One possibility is that a new rule number must be read from the stack, which is usually the case when the rule invoked was carried out completely. This is how the stack decreases in size (see Figure 1.4). If Express leaves a rule temporarily before it is completed to start another rule, the stack will increase. This happens

if the currently active rule needs additional quantities computed by another rule before it can be completed.

Every rule starts in the same manner, adding its own rule number to the top of the stack. At the end of each rule, this number is removed from the stack (but only if the present rule has been carried out completely, not in connection with temporary exits). Between the initial and final parts of each rule, any statements can be included which perform calculations or make decisions concerning the analysis. These are mainly statements of the type

IF <logical statement> THEN <action 1> ELSE <action 2>

selecting various possibilities for different situations. Most of the executable statements in the rules will be calls to standard routines provided by Express, reading values from or writing values to the data base.

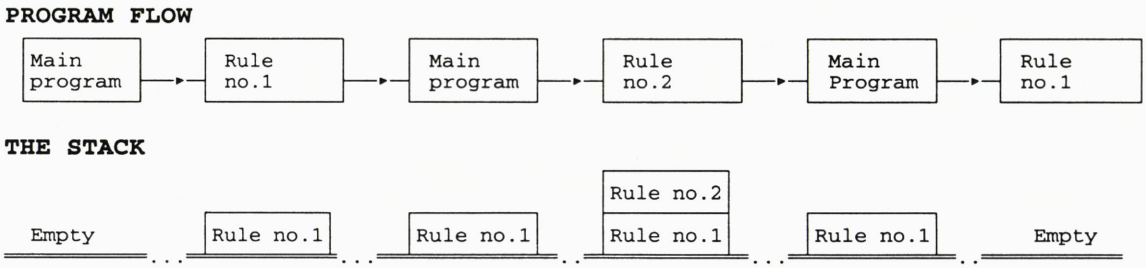


Figure 1.5 Illustrating the connection between the main program (the inference engine) and the different rules.

Figure 1.5 illustrates a simple chaining of rules, with an indication of how the control within the program changes between the different components. The chaining begins with the activation of rule no. 1 from the main program. When this rule is started, it pushes its rule number to the top of the stack. In search of a particular statistical quantity, this rule determines that rule no. 2 must be executed before it can be carried out completely. This is done by leaving rule no. 1 without removing the rule number from the stack. The main program will again be in charge of the chaining. Depending on the return address supplied by rule no. 1, rule no. 2 is activated. This rule also begins by pushing its rule number on the stack. Thus the stack now contains two rules, with the rule at the top being the active one. It turns out that rule no. 2 can be carried out completely. Before leaving control to the main program, rule no. 2 removes its rule number from the stack, leaving only rule no. 1. Let us now assume that rule no. 2 did not return any address indicating where the control should pass. This means that the main program must remove the next rule from the stack. This is rule no. 1, which can now be carried out completely. When the control returns to the main program, it will attempt to remove another rule number from the stack. But the stack is now empty, so the chaining is complete.

1.4.4 The main menu and the data storage

Express incorporates several facilities to assist in the understanding of the analysis being performed. During execution of a set of rules, essential information about the process is presented automatically on the screen. Thus all intermediate values considered are shown, with or without a comprehensive explanation, depending on the system settings. Any part of this information can be consulted by the user at any time in the log file. Facilities for viewing and manipulating values in slots are also available (see Chapter 2).

When a set of rules has been selected, Express collects information from the data base file about the number of variables needed in the present situation. Before the analysis can proceed, these variables must be selected from the particular data storage maintained by Express. A particular option in the main menu designated "DATA" gives the user an opportunity to read raw data into the data storage, and select variables for analysis in the current set of rules.

The data storage can include a large number of variables, not necessarily related to the active set of rules. When new variables are read into the storage, these are simply added to those already present. When variables are to be selected for a particular analysis, a list of those available will appear on the screen and a choice can be made. Before a variable can be analysed in a statistical package, it must be copied from the data storage to a file used as input to the package. This is taken care of by particular subroutines which can be invoked in the program representing a set of rules. Figure 1.6 shows the flow of data from an external ASCII file through the data storage, into files used as input to external packages.

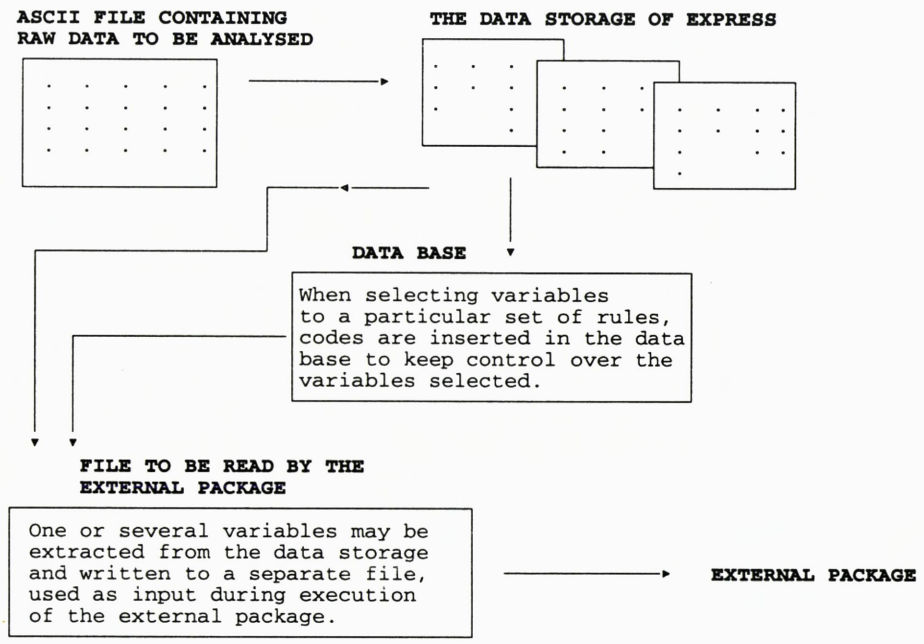


Figure 1.6 Connections between file containing raw data, the data storage in Express and the file read by the external package.

Chapter 2

GUIDE FOR THE END USER

2.1 EXECUTING EXPRESS. THE MAIN MENU 20
2.1.1 Starting Express from the DOS command line 20
2.1.2 Selecting a particular set of rules 20
2.1.3 Main menu 21

2.2 DATA 23
2.2.1 Entering data into the data storage 24
2.2.2 Selecting data for analysis 25

2.3 EXECUTING A SET OF RULES 27
2.3.1 Basic problem menu 27
2.3.2 Chains of rules. The stack in Express 27
2.3.3 Exit from a chaining of rules 28
2.3.4 User prompts 28
2.3.5 Setting up commands for external packages 30
2.3.6 Extracting values from the output of a package 31

2.4 FILES 32

2.5 SLOTS 33

2.6 SYSTEM 36
2.6.1 System variables 36
2.6.2 Other possibilities 38

2.7 SAMPLE SESSIONS 39
2.7.1 One-way analysis of variance 39
2.7.2 Comparing location parameters for two samples 44
2.7.3 Regression analysis 48

2.1 EXECUTING EXPRESS. THE MAIN MENU

This chapter describes the Express interface to the end user. The main part of this chapter explains attributes of Express necessary to be able to conduct an analysis using Express. The last section in this chapter gives some sample sessions, making it easier to get to know the user interface by practical exercises.

2.1.1 Starting Express from the DOS command line

In order to use Express, type **\EXPRESS\EXPRESS** on the DOS command line. This command will initate execution of the file \EXPRESS\EXPRESS.COM. When the directory \EXPRESS is given explicitly in this command (or \EXPRESS is listed in the current PATH), the active directory at the time of execution can be arbitrary. The system also includes several executable EXE files in the directory \EXPRESS. No attempt should be made by the user to execute any of these files directly, except the special Express editor (stored in the file \EXPRESS\EDITOR\ED.EXE).

2.1.2 Selecting a particular set of rules

After introducing itself, the system will display a menu enabling the user to select the set of rules to be used in the following session (Figure 2.1). All sets of rules implemented in the current version of Express, possibly dealing with totally unrelated areas of application, will be shown. It should be noticed that it is not necessary to leave Express in order to switch to another set of rules during the session. This is because the menu involving different sets of rules can also be accessed from the main menu of Express appearing later.

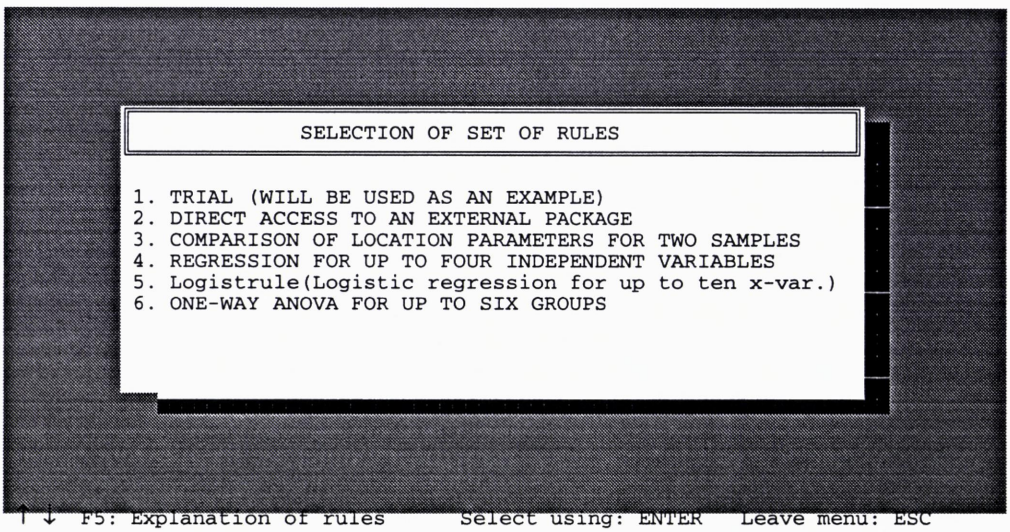


Figure 2.1 Menu for choosing sets of rules.

As in all menus of Express, the arrow keys can be used to highlight the item the user wants to select, and the actual selection is made by pressing the ENTER key. By pressing F5, an explanation will be shown of the set of rules being highlighted, if available. It is also possible to leave Express at this stage by pressing ESC.

We shall use regression analysis as an example in this chapter, so we select alternative no. 4 among the sets of rules shown in Figure 2.1.

2.1.3 Main menu

At this stage the main screen of Express will be displayed, with the main menu shown at the top. This will be the starting point for the different statistical analyses that can be carried out on the same data set, leading to results which will also be presented on the main screen. Moreover, the user will have an opportunity to study results more closely after a chain of rules has been completed.

The main screen of Express is shown in Figure 2.2. The different items in the main menu are presented in the upper window. To make a selection, simply use the arrow keys to highlight the desired alternative and then press ENTER. When the selection has been made, a corresponding pull-down menu, superimposed on the main screen, will appear. The table below provides a brief explanation of the different items in the separate menus. To return to the main menu from one of the pull-down menus, press the ESC key. There are two large windows below the menu at the top of the main screen. During the session, these windows display various kinds of information. The bottom line of the main screen always indicates which options are available to the user as a next step.

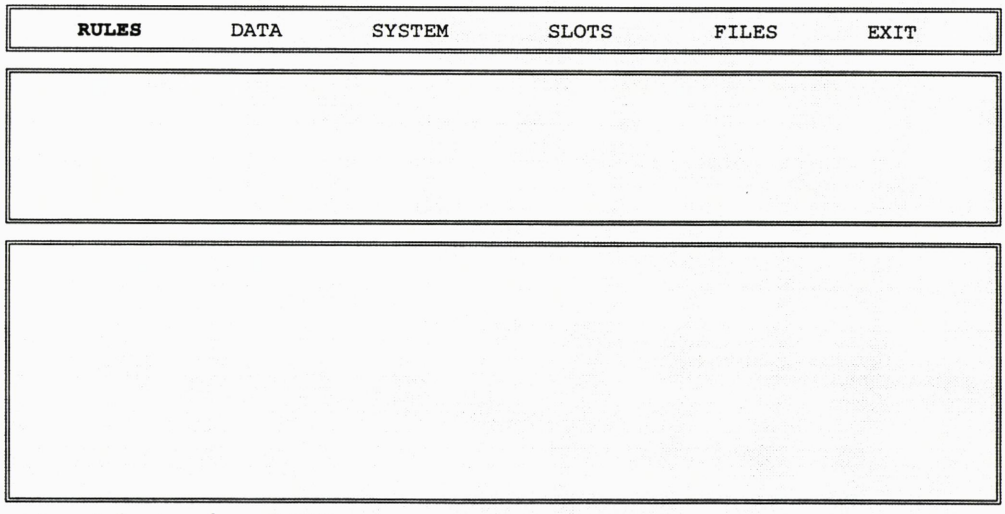


Figure 2.2 Main screen of Express.

Name	Function
RULES	If this item is selected, the rule menu will be displayed. This menu presents different alternative points of departure for a statistical analysis carried out by the particular set of rules selected previously.
DATA	Selection of this item enables the user to enter the data handling subprogram. It is also possible to specify which variables should be included in the subsequent analysis.
SYSTEM	Under this option, several system functions in Express may be accessed. Thus, particular system variables may be adjusted, and it is possible to enter the dictionary of Express which provides a general explanation of various terms. A general help function also appears under this heading, and furthermore, it is possible to return to the menu for selecting a particular set of rules. The editor of Express can be entered, and as a final alternative it is possible to exit temporarily to DOS.
SLOTS	The values in "slots" are the statistical quantities that are determined when rules are invoked, as well as certain other values set by the system, with essential information about the particular problem under consideration. Selecting this item, the user may access the information stored by Express in particular slots. Values in slots may be displayed or modified, or slots may be assigned the code for unknown values. The log file can also be reached from this menu.
FILES	Two files are of special interest after a particular set of rules has been started. The log file contains information about every step made by Express in the analysis until the current stage. Another file collects all output produced by the external program packages. Selecting the item "FILES", it is possible to examine and make copies of these output files. An explanation of the current set of rules may also be found under this option.
EXIT	The system offers two different ways of leaving Express. The user may wish to interrupt the execution of a particular set of rules, retaining all slot values obtained during the session. The session may later be resumed with the same values. Alternatively, the user may indicate that all information found previously should be erased before an exit is made.

2.2 DATA

Express maintains a separate data storage which includes all variables that can be referred to in the statistical analyses. The columns of the data storage represent the different variables and the rows the successive observations. The separate columns can contain different numbers of observations. Any observation may be represented by a particular missing value code interpreted internally by Express. All values in the data storage will normally be retained from one session to the next. New variables can easily be added to the storage from external files. Before invoking a set of rules, the user must specify which variables in the data storage will be considered in the current session.

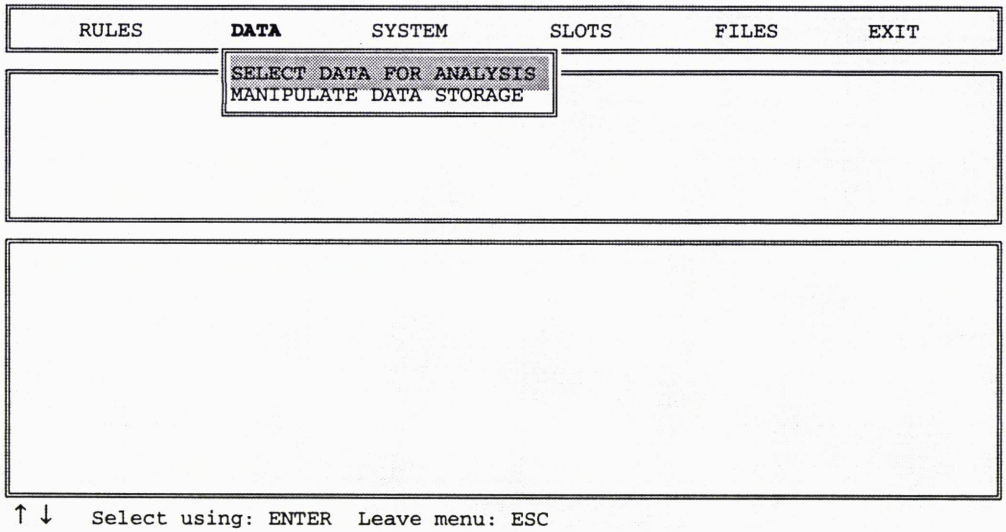


Figure 2.3 Menu for editing data.

The data storage can be manipulated through a particular data handling subprogram. To access this program, enter the "DATA" submenu from the main menu, highlight the second item shown and press ENTER. The alternatives offered for data handling are shown below.

Name	Description
F5-Add data	This alternative should be selected when variables are to be added to the data storage of Express.
F7-Erase data	To erase the data storage, simply activate this function.
F9-List data	This option makes it possible to list on the screen the data currently residing in the data storage.

2.2.1 Entering data into the data storage

Each variable to be entered into the data storage must be read from an external standard ASCII file, one at a time. Different variables may be read from the same or separate files. If a file of this kind includes more than a single variable, it is essential that the position of each variable should be known in advance. In this case the system will prompt the user for the left and right limits.

Procedure for reading a single variable into the data storage:

1. Select the option "Add data" (by pressing F5).
2. The system inquires about the name of the external source file. Type this name.
3. The program asks "Does this file include other variables?". If the response is "No", the system will start reading the data. In this case each number read will be recorded as a single observation. Different observations can be coded on the same or different records of the external file, but must be separated by at least one blank position. If, on the other hand, the variable of interest is only one among several variables on each record, the relevant location on each record must be specified in accordance with requests made by the program.
4. If the external file includes several variables, a decision must be made concerning missing observations. The system must first be instructed how to act if all positions between the leftmost and rightmost columns for a particular variable are blank. The user may decide to omit such values completely from the internal data storage, or assign the code for missing values. If a particular integer is used in the external file for representing a missing value, it is possible to change this to the special missing value code in Express. The system will guide the user through these specifications. If a value is omitted from the data storage, the subsequent values for the variable in question are moved up in the data set.
5. When Express has finished reading the observations from the external file, the variable must be assigned a name for later reference within Express.

Example

Suppose that we wish to read variables into the data storage of Express from the external file \OBSERV\REGR.DAT. This data set will be used to illustrate the set of rules for regression analysis. The variables in this file are arranged as shown in Figure 2.4, with the dependent variable occupying the first five positions. Assume that the value of this variable for the last observation by mistake has been placed on a separate record. The dependent variable is followed on each record by four independent variables, each occupying five positions. When the dependent variable is read into the data storage, we must answer "No" to the question inquiring whether blanks should be interpreted as missing values. Thus only 8 observations

are inserted into the data storage, and the last value of the dependent variable will be correctly aligned. Figure 2.5 shows the Express screen when the selections for the dependent variable have been completed. After assigning the name YVAR to this variable, we can repeat the same procedure for each independent variable, naming these variables X1VAR, X2VAR, X3VAR and X4VAR, respectively.

```
1   5    10   15   20   25   30
----+----+----+----+----+----+
    10    2   43   21    1
    19    3   49   22    2
    40    4   43   22    1
    50    4   49   21    2
    60    5   48   20    1
    40    4   46   22    2
    70    6   49   22    2
          3   42   21    1
    35
```

Figure 2.4 The file \OBSERV\REGR.DAT

```
 RULES    DATA    SYSTEM    SLOTS    FILES    EXIT

                THE DATA IN EXPRESS
EXPRESS utilises a main data storage, where variables may be
inserted, regardless of which rule is invoked. This data storage
may thus include several variables which have not been referred
to in the problem considered.

        READING OF NEW DATA

File containing data:   \observ\regr.dat

Does this file include other variables?                  YES   NO
In order to select the correct data values, please indicate
where data should be read on the file:
Leftmost column :  1
Rightmost column:  5
Should blank positions be interpreted as missing ?       YES   NO
Is an integer used as a "missing value" ?                YES   NO

Select using: ENTER
```

Figure 2.5 Storing one variable from the file \OBSERV\REGR.DAT into the data storage of Express.

2.2.2 Selecting data for analysis

A particular set of rules may impose restrictions on the number of variables that can be handled. In the example considered here, the rules for regression analysis can work at most with five variables: a single dependent variable which must always be specified, and from one to four independent variables.

After the data set has been entered into the storage as described Section 2.2.1, the user must indicate which variables should be included in the actual analysis. Highlight the item "Select data for analysis" shown in Figure 2.3 and press ENTER. It is then possible to start extracting variables from the storage using the keystrokes described below. Figure 2.6 illustrates this procedure for the set of rules for regression analysis.

1. At the bottom of the second window, the user will each time be presented with a short description of the variable that should be selected, formulated in general terms relating to the analysis performed. At the left side of the window, a list is displayed with at most four variables from the data storage. It should be noted that a brief list of this kind is shown even if the storage includes a greater number of variables. To search through the complete list with all the variables available, use the PgUp, PgDn and arrow keys. Each time a variable should be selected, highlight the corresponding variable name in the list on the left and press ENTER.
2. Now repeat this procedure as many times as there are variables to be included in the analysis. The system may prompt for a greater number of variables than the user wishes to include. In such cases, the selection process may be terminated by pressing the ESC key. When the selection of variables has been completed, the system will inquire whether the user wishes to proceed with the variables already specified, or whether changes should be made. However, if the user has not selected enough variables for the analysis, the system will return to the main menu.

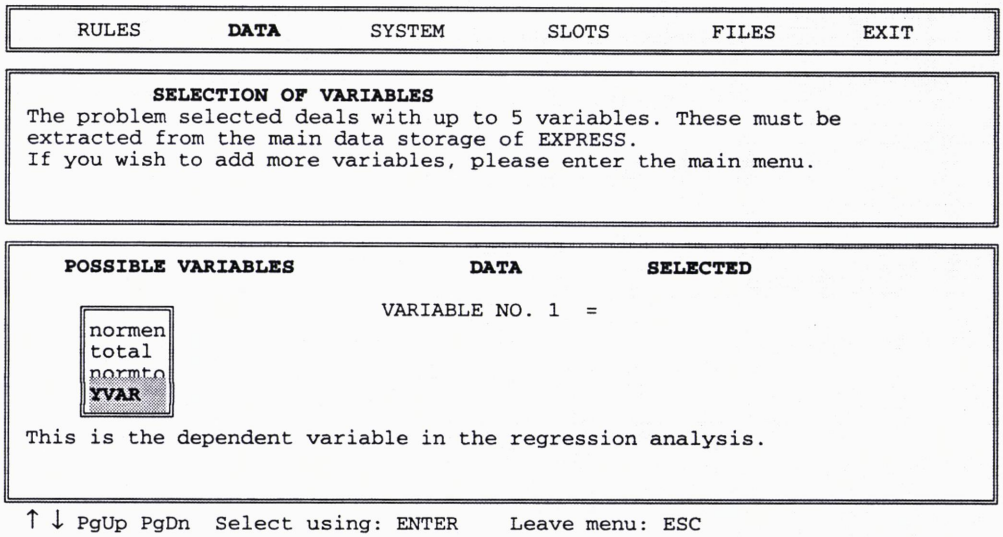


Figure 2.6 Selecting variables for regression analysis.

2.3 EXECUTING A SET OF RULES

2.3.1 Basic problem menu

For executing a set of rules, the alternative denoted by "RULES" should be selected from the main menu of Express. If a selection has already been made of variables to be analysed, the system will pass directly to the basic problem type menu (Figure 2.7). Otherwise, Express will automatically prompt for selection of variables, in the same way as described in Section 2.2.

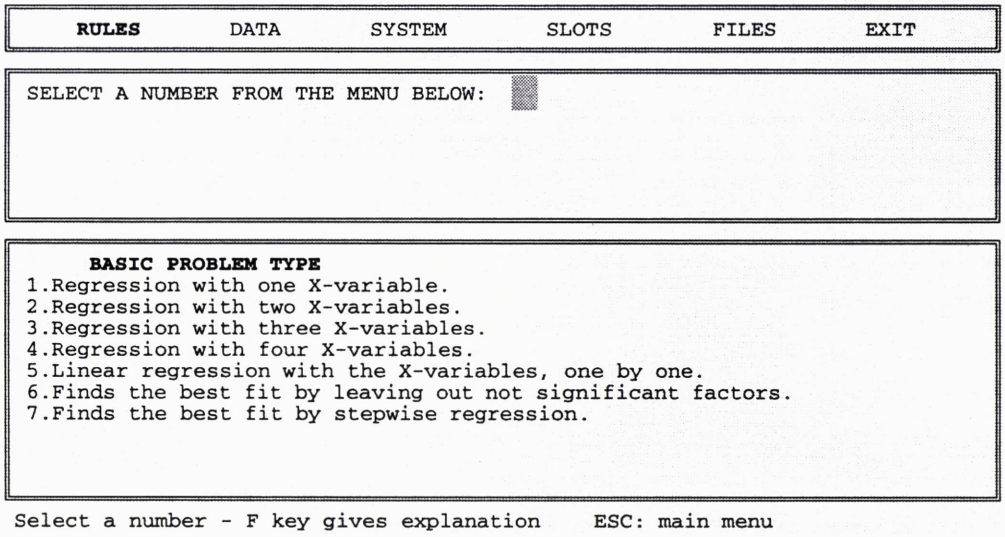


Figure 2.7 Menu for selection of basic problem type.

2.3.2 Chains of rules. The stack in Express

When a selection has been made in the basic problem menu, Express will immediately start processing the particular rule designed to solve this problem. Typically the aim of any rule is to determine a slot value. If this value has not already been ascertained, Express will make an attempt to find the correct value. This can be done in several ways, by simply following directives coded within the currently active rule, by starting other rules, by executing external packages, or even in special cases by prompting the user for the correct value. This procedure will be repeated, perhaps a large number of times, until the final conclusion to the basic problem selected has been reached. At any point during this process, the Express stack will show which rules with incomplete answers must still be processed before the main conclusion can be stated. The contents of the stack and a short explanation will usually be displayed on the screen, as shown in Figure 2.8. Various system settings determine how many details are presented to the user regarding the chaining of rules (see Section 2.6.1), but all intermediate results will be shown, except for certain plots which only appear under particular combinations of system settings.

Example

If we select problem no. 1 in the set of rules for regression, Express will perform a simple linear regression analysis with one independent variable (X-variable). Figure 2.8 gives a typical example of the information displayed on the main screen during execution of a set of rules.

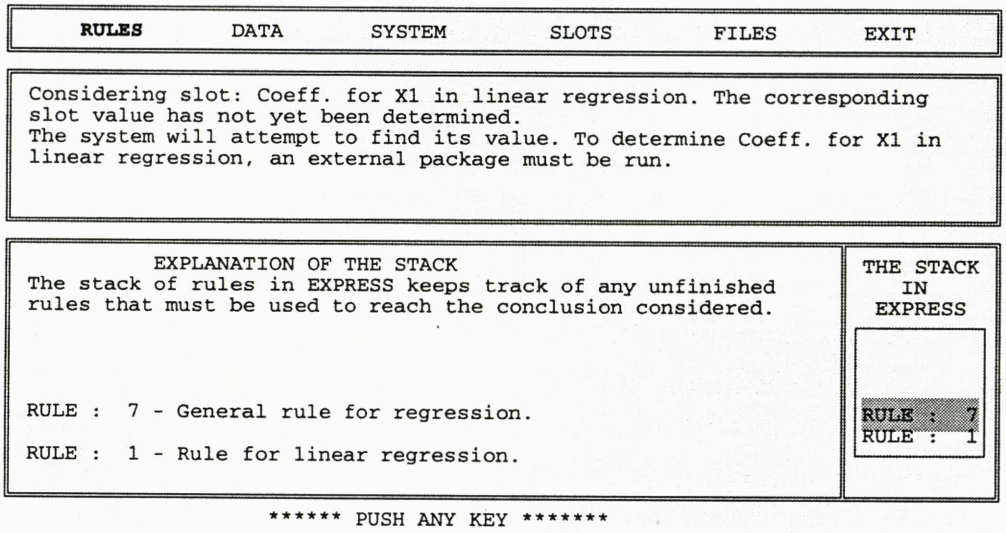


Figure 2.8 Executing a set of rules.

2.3.3 Exit from a chaining of rules

At almost any point during the execution of a set of rules, the user may press the ESC key and the system will respond, possibly with some delay, by terminating the chaining of rules. A return is then made to the main menu. All values assigned to slots will be preserved for a later restart, but the stack will be empty after the break.

In particular situations, the user may experience problems trying to restart the same rule. It is then recommended that the specification "New data to be analysed" should be made, even if the same variables are selected. Such problems represent errors in the construction of the particular set of rules applied and should be taken care of by the knowledge engineer.

2.3.4 User prompts

During the chaining of rules, Express will from time to time ask the user to provide additional information needed to carry out the analysis. In some situations the information may be essential to the basic interpretation of the structure of the problem considered. In other situations, Express simply gives the user an opportunity to influence the decisions made.

Example

In our basic example involving the set of rules for regression analysis, we assume that four variables were selected as potential independent variables before the chaining of rules was started, as indicated in Section 2.2. Suppose that problem no. 1, corresponding to an analysis with a single independent variable only, is selected. The system will then ask the user to specify which one among the four variables should be considered (Figure 2.9). The arrow keys can be used to highlight the variable of interest, and the selection is made using the ENTER key. For a problem involving the specification of several variables, this procedure must be repeated. Note that certain rules (e.g., those for regression analysis) will not accept selection of the same variable more than once.

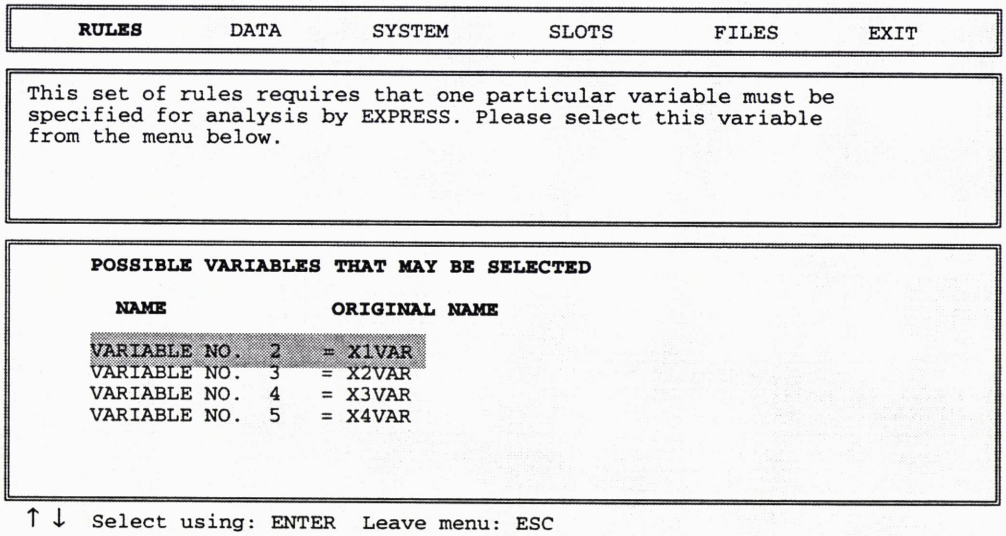


Figure 2.9 Select a variable among those available for this set of rules.

The list below indicates particular user prompts that may appear during a chaining of rules.

Name	Description
Select variable	Lets the user select one of the variables available for the current set of rules (see description above).
Assign value to a slot	For certain slots the system will give the user an opportunity to set the slot value before it makes an attempt to determine the value itself. If the user does not wish to assign a value to the slot, he can press ESC, and the system will, if possible, go on to determine the value in question. If, on the other hand, the user wishes to specify the slot value, the value supplied must be within the limits prescribed for this slot. These limits will be presented on the screen.

Fill in information	The user may in particular cases be required to specify part of the control language for external packages. In most situations, however, this prompt will appear as the result of an error or badly constructed sets of rules. Such problems should be rectified by the knowledge engineer.

2.3.5 Setting up commands for external packages

Express will frequently execute an external program package in order to determine a statistical quantity. In such cases the system sets up the commands (job control language) required, according to the rules which apply to that particular package. The commands are retrieved one by one from a special storage and are presented on the screen as they are generated. Each command may include several incomplete fields. To insert the appropriate specifications, the system goes through the information found so far and extracts the correct items. The specifications typically involve names of variables and data files, but other pieces of relevant information may also be inserted.

Depending on the system settings, output to the screen will be slow or fast when commands are displayed. In the slow mode it is possible to observe the system pointing to any incomplete fields in the commands. If the system is unable to supply the information required, the user will be asked for assistance. In this situation the chaining of rules will be terminated if no assistance is provided. Sometimes part of a command disappears from the screen during this process because Express has decided that the entire command can be formulated more briefly.

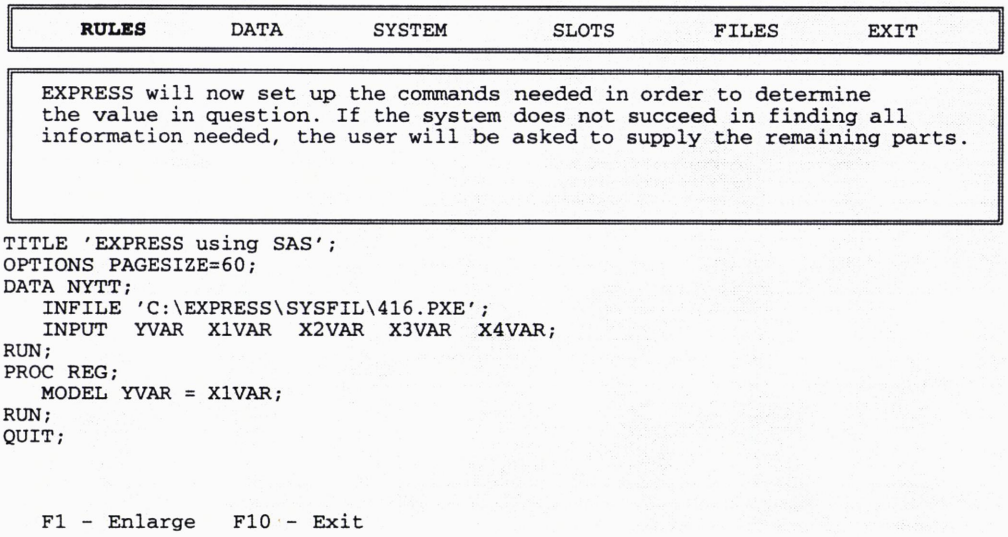


Figure 2.10 Screen shown after commands to external package have been completed.

When a set of commands has been completed, it can be examined on the screen before the external package is started (see Figure 2.10). The window showing the commands may be

enlarged by pressing the F1 key. If F1 is pressed once more, the window returns to normal size. In both modes the process can be continued by pressing F10 (indicating an "Exit" from the command window). The system now proceeds with the execution of the external program package. Express will indicate which package is about to be started. It should be noted that some packages may destroy the screen produced by Express during execution. However, Express will restore its own screen when the execution of the package has finished.

2.3.6 Extracting values from the output of a package

Each external package is assumed to generate an output file containing the values of interest to Express. Express scans this output, searching for useful results. The user can watch this process on the screen. The speed of the extraction process depends on system settings (Section 2.6.1). In the slow mode all quantites of interest and corresponding search keys will be highlighted. Figure 2.11 shows an example. The search operation does not only involve values needed at this particular stage, but other quantities of general interest as well. These are stored in the data base for later use. In particular situations certain values will be missing from the output, depending on other results during the execution of the package. A warning is given but the system will go on in the hope that the relevant conclusion can be reached despite the fact that some quantities are unknown. Problems of this kind may reflect errors in the coding of rules and should be reported to the knowledge engineer.

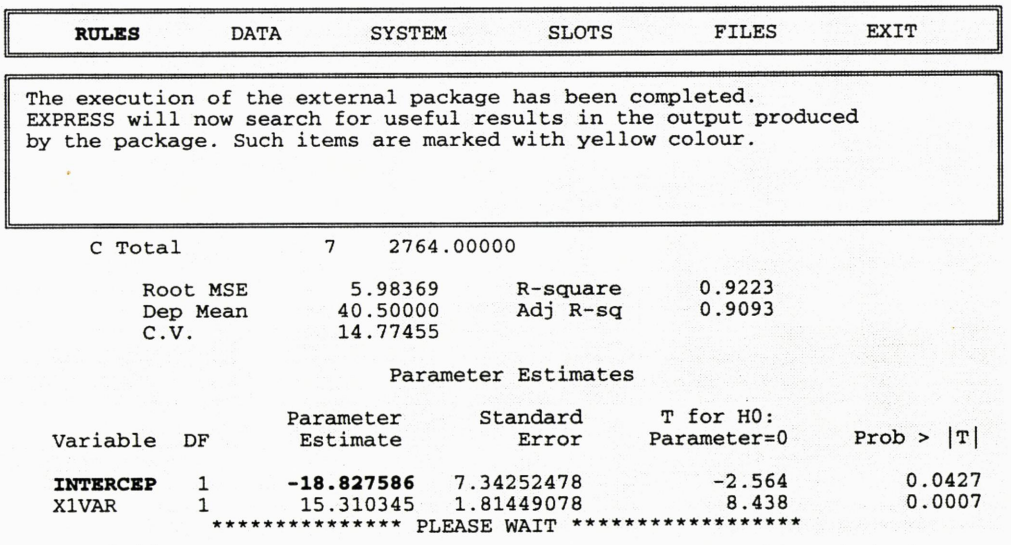


Figure 2.11 The main screen of Express during the search for useful results in the output from an external package. Items selected are highlighted and their values are stored for later use.

2.4 FILES

When Express returns to the main menu after a chain of rules has been executed, the user may wish to examine more closely the analysis carried out by the system. The log file keeps a complete record of the preceding chaining. This file as well as the file containing the output from the packages can be viewed on the screen or they can be stored in an external file. In this way the text can also be printed or retrieved in another document. The different options are shown in Figure 2.12. Note that it is also possible to examine a file containing explanations or help for the set of rules selected. This is the same information that can be accessed from the menu for the different sets of rules.

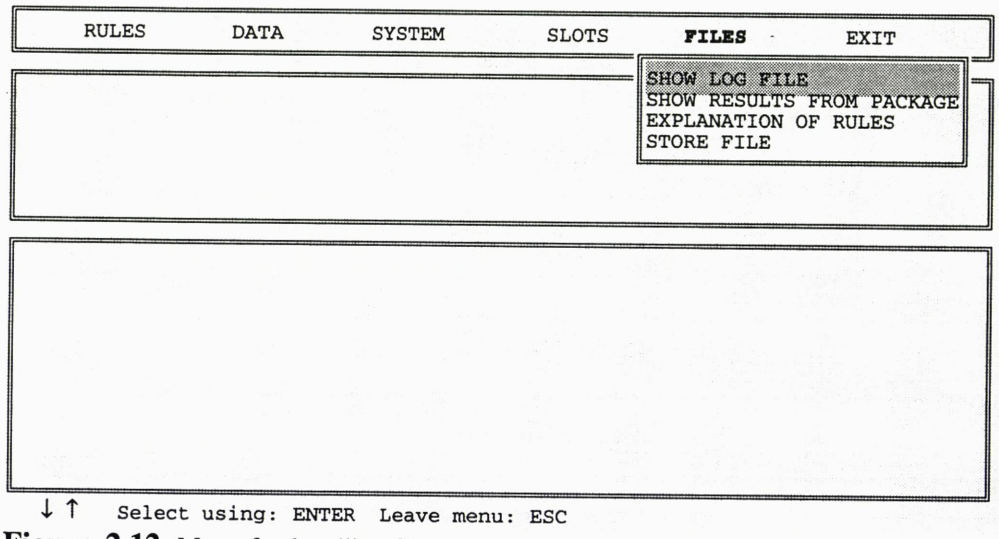


Figure 2.12 Menu for handling files with information.

In general, the log file contains the same information as was presented on the screen during the most recent chaining of rules. The only exceptions are search keys and statistical quantities extracted from results produced by the external packages. All output from the various executions of external packages will be stored successively in the special file used for this purpose. A row of asterisks is inserted to separate results from different executions.

If any of the first two items are selected in the pull-down menu in Figure 2.12, the corresponding file will be shown in the larger window of the main screen. The PgUp and PgDn keys and the arrow keys can be used to move the current position inside the file. The window can be enlarged by pressing F1 and reduced to normal size again by pressing F1 once more. In both situations the examination of the file can be terminated by pressing F10. An alternative is to end the presentation by erasing the contents of the file by pressing F7. Only results from subsequent analyses will then be included in this file. By selecting the last item in the pull-down menu, both the log file and the file with results from external packages can be copied to arbitrary external files. Express will prompt for file names.

2.5 SLOTS

In the terminology adopted in Express, all quantities referred to in the rules are regarded as slots. These can, for example, represent the actual values of regression coefficients, or can simply be the answers to questions concerning statistical significance of such coefficients. During the chaining of rules, the main aim of Express is to assign values to the relevant slots. In addition to the handling of slots by the system itself, however, Express also offers certain facilities for user manipulation of slots. The possibilities are shown in the table below. All items will appear in a pull-down menu when the option SLOTS is selected in the main menu (see Figure 2.13).

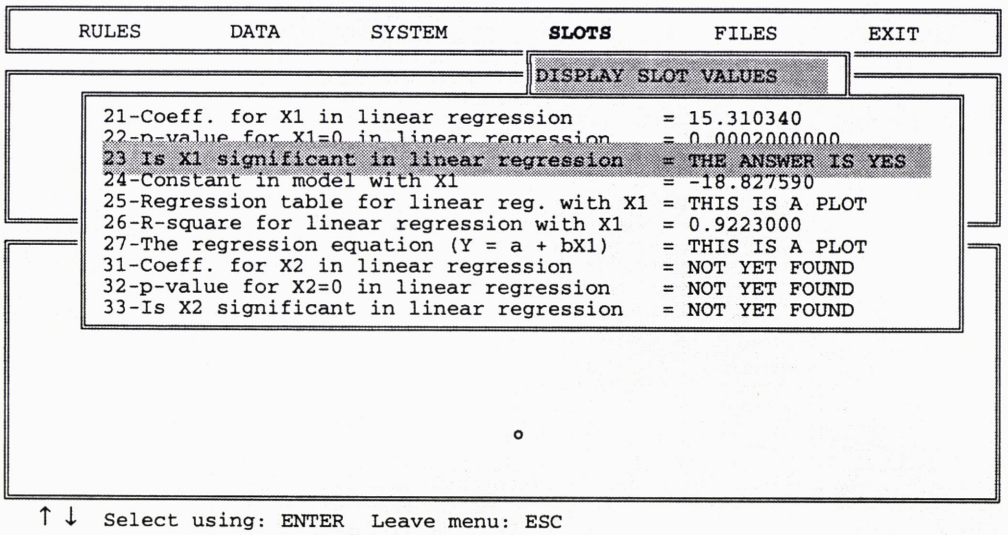


Figure 2.13 Selection of a slot for more detailed examination. The numbers to the left refer the position of slots in the code file and database, and are of no direct relevance to the end user.

Name	Description
DISPLAY SLOT VALUES	Regardless of the item selected in the pull-down menu, Express will present a list of relevant slots, providing a brief explanatory text for each one (Figure 2.13). The selection "Display slot values" generates a simple listing of corresponding values. When a particular slot has been selected from this list, Express will give a more detailed description of the slot.
ASSIGN SLOT VALUE	Almost any slot value can be modified by the end user. Exceptions are slots defined as blocks containing information displayed over several lines (i.e., plots or

tables) and certain special values coded in such a way that changes are not allowed. To modify a slot value, simply select the slot from the list and follow the instructions given (Figure 2.14). Express will not accept values outside the legal range presented on the screen. When a new value has been assigned to a slot in this way, all other slot values based on the former value will be regarded as unknown and must be determined again. Before the slot value is changed, Express will show the user the current state of the slot. At this stage, the user can decide to go on assigning a new value, or simply accept the value already stored by the system. This cycle may be repeated until the ESC key is pressed.

RELATIONSHIPS

During the chaining of rules, Express automatically records relationships between different slots. As an example, the system must determine the *p*-value for the coefficient of X1 in a linear regression analysis before it can be decided whether or not the coefficient is statistically significant. A diagrammatic overview of such relations can be examined by the user when the chaining has been completed. By selecting a particular slot, the user will be presented with a list of other slots whose values were needed in order to determine the current slot value. By pressing F5, it is possible to get a converse diagram, indicating which slot values were actually based on the slot currently selected in their computation. From both lists it is also possible to select immediately another slot to be considered in this way (see Figure 2.15).

SHOW LOG FILE

When a slot is selected under this option, Express searches the log file for the last reference to the slot, and displays that part of the log file on the screen.

SET SLOT TO UNKNOWN

If the user wishes to repeat parts of the chaining of rules, this can be done by assigning the code for unknown values to some of the slots and then restarting the chaining.

Figures 2.14 and 2.15 provide examples of how the value in a slot may be modified and how relationships between slots may be examined. In Figure 2.14 it should be noted that

conclusions are coded with the value 0 for "No" and 1 for "Yes". Arrows are inserted on the screen presenting relationships to clarify which slot values are based on other slots. The colour of the arrows indicates whether the values are set by the end user (yellow) or by the system (white).

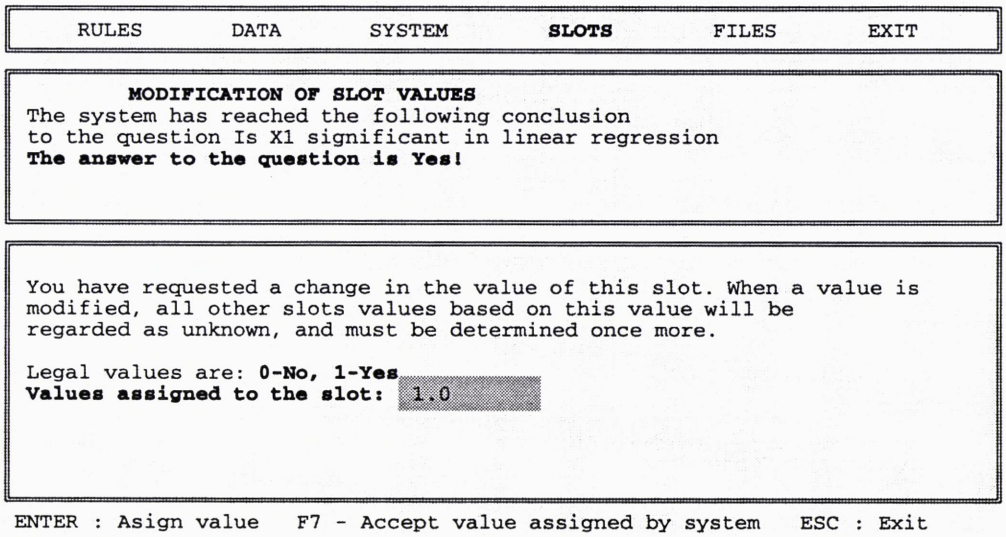


Figure 2.14 Screen during modification of value in a slot.

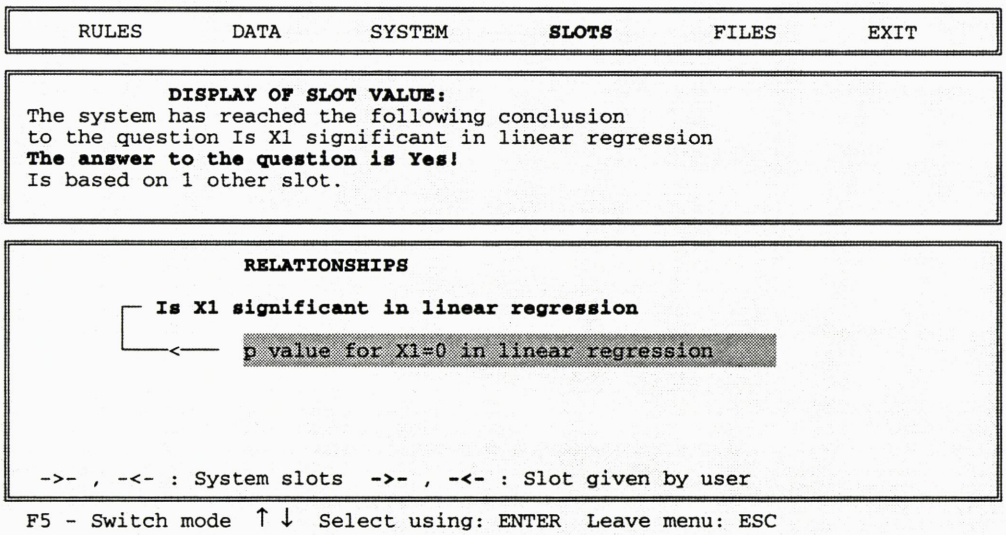


Figure 2.15 Screen displaying relationships between slots.

2.6 SYSTEM

The items in this submenu can be divided into two groups. We distinguish between system settings (or "system variables") and other options not directly connected with the chaining of rules.

2.6.1 System variables

The user has control over five system variables, given in the table below. Each variable affects the presentation during the chaining of rules, with two possible states, "on" and "off".

Name	Description
COMPREHENSIVE	During the chaining of rules, Express presents slot values as they are determined in the upper window of the main screen. If this system variable is turned on, a comprehensive explanation of each slot will be given in the lower window of the main screen at the same time. This explanation is also written to the log file. This assumes that the knowledge engineer has provided the system with a suitable set of explanations.
GRAPHICS	Certain slots may be defined as "blocks" represented by plots or tables occupying several lines. These slots will only be shown if this system variable has been turned on. Exceptions are particular plots or tables which have been defined in a special way to ensure that they are always displayed. Thus, in the set of rules performing regression analysis, we use a "block" to store the entire regression equation, written with symbols, since ordinary slots cannot include more than a single number. This slot should be obviously be presented regardless of the setting of the system variable.
STACK	As explained in Section 1.4.2, the Express stack keeps track of which rules must still be processed before the main conclusion can be stated. The contents of the stack can be displayed permanently on the screen by turning this system variable on. Figure 2.8 shows an example with the stack displayed in the lower right hand corner of the main screen. A brief explanation of the rules on the stack may also be presented, if the system variable "COMPREHENSIVE" is turned off. (Otherwise not enough space is left on the screen for these explanations.)

SLOW	This variable determines the speed of the presentation on the screen of the chaining of rules. The variable will usually be turned on, and Express will then pause in many situations to let the user follow the separate steps. If the system variable is turned off, Express will not stop to ask the user to confirm that the system should continue, as would otherwise frequently happen. Furthermore, Express will only pause immediately before the execution of an external package.
DEBUG	This system variable is only intended to be turned on as a tool for the knowledge engineer in debugging when new sets of rules are being tested.

To toggle between the different states, highlight the variable of interest and press ENTER. The current state is immediately changed to the opposite one (see Figure 2.16). When the variable SLOW is turned off, all other system variables will also be turned off automatically. This is the recommended setting for a chaining of rules which should proceed as fast as possible. Any system variable can of course be reset at any later stage in the analysis.

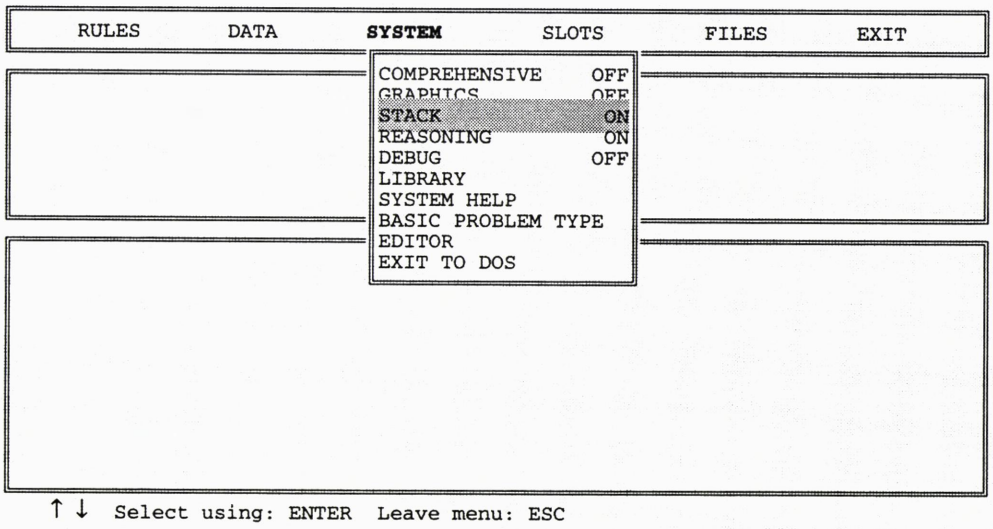


Figure 2.16 Items in the system menu.

2.6.2 Other possibilities

The remaining items in the SYSTEM submenu provide access to particular system facilities in Express.

Name	Description
DICTIONARY	Express maintains a special dictionary where users may insert key words (or brief sentences) with corresponding explanations. During a break in the chaining of rules, the dictionary may be consulted on any number of key words. Press F5 and the system prompts for the key word, and then responds with the explanation. To insert a new key word, press F6 and state the new word. Express will then prompt for the explanation. When the explanation has been inserted, press ENTER twice to register the new key word. To leave the dictionary, press F10.
SYSTEM HELP	This selection activates the general help function in Express. Use the arrow keys, PgUp and PgDn to move inside the text.
BASIC PROBLEM TYPE	This selection makes it possible to change to another set of rules. The menu in Figure 2.1 will appear on the screen, and a new set of rules can be selected. Slot values found in the previously active set of rules will be preserved for later use.
EDITOR	The particular Express editor is needed for constructing new sets of rules, or modifying old ones. It can also be used to examine or modify various system files. This is mainly a tool for the knowledge engineer and should be used with great care (see Chapter 3).
EXIT TO DOS	Selecting this item, an exit will be made to DOS without terminating the execution of Express. It should be noted that Express may occupy a sizable proportion of the total memory available, and thus large programs cannot be executed from DOS when a temporary exit is made. To return to Express from DOS, type "EXIT".

2.7 SAMPLE SESSIONS

This section describes three simple sample sessions with Express, based on the sets of rules for one-way analysis of variance, comparison of location parameters in two samples and regression analysis. These sessions provide a practical introduction to the system, in order to give the user the basic skills needed for other more complex applications. The reader is supposed to follow the instructions given below in boldface.

2.7.1 One-way analysis of variance

In this session we will compare the location parameters of three samples, consisting of 20 observations each. The data are read from an ordinary ASCII file supplied with Express. The samples have been generated from normal distributions, all with standard deviations equal to 1, and with means 0, 0 and 1, respectively. To run this example, BMDP must be available on the PC used, in the directory C:\PC90 (see Section 1.2.3). It should be emphasized that the sample session has been designed mainly to demonstrate the technical facilities offered by Express. More details about this set of rules are given in Appendix A.

1. START EXPRESS FROM THE DOS COMMAND LINE

Start Express from the DOS command line by typing

C:\EXPRESS> **EXPRESS**

To enter the menu for selection of a set of rules, **push any key**.

2. SELECT THE PROPER SET OF RULES

Use the arrow keys to **highlight** the item "6. ONE-WAY ANOVA FOR UP TO SIX GROUPS" and carry out the selection by pressing **ENTER** (see Figure 2.1).

3. ENTERING VARIABLES INTO THE DATA STORAGE

You have now reached the main menu of Express as displayed in Figure 2.2. The first step is to read the external data into the data storage. The file C:\EXPRESS\ONEWAY.DAT contains the three variables (samples) which are going to be analysed. Each variable includes data for 20 observations and occupies 10 positions on the records of the file. The first variable covers positions 1-10, the second 11-20 and the third 21-30. The three variables must be read separately into the data storage. To start entering the variables, **highlight** the item "DATA" in the main menu (see Figure 2.3) and press **ENTER**. Then **highlight** "MANIPULATE DATA STORAGE" (see Figure 2.3) and select by pressing **ENTER**. It is now possible to list

the data in the storage by pressing F9 and selecting one of the variables included. (If this is the first time Express is used, the storage will be empty.) Another option indicated is erasing the data storage (F7). We want to add new data so the key **F5** should be pressed.

As shown in Figure 2.5, we must first supply the name of the file containing the data. Thus write **c:\express\oneway.dat**. Furthermore, as this file includes three variables, the response YES must be **highlighted** to the question "Does this file include other variables?". Again, carry out the actual selection by pressing **ENTER**. The next two fields to be filled in indicate the leftmost and rightmost position occupied by the variable. The first variable is located between positions **1** and **10**, so these numbers are specified (and **ENTER** is pressed each time). The questions concerning missing values should both be answered by NO. Again this is done by **highlighting** NO and then pressing **ENTER**. Express now reads data from the external file into the data storage. Only the five first values are shown on the screen. Finally, a variable name must be given. The name can occupy up to 6 positions and must not include any blank spaces. For example, the first variable may be called **ONEW1**.

The procedure described must then be repeated for the two remaining variables, beginning with **F5** pressed to indicate that more data should be added to the storage. The only difference now is that the variables occur in other positions on the file (11-20 for the second variable and 21-30 for the third). These variables may be given the names **ONEW2** and **ONEW3**. Now try to list one of the variables by first pressing **F9**, **highlighting** the variable of interest and pressing **ENTER**. Only the first seven values are shown, but it is possible to move up and down in the data set using arrow keys. To finish with a particular variable, press **F10**. To return to the main menu, press **ESC**.

4. SELECT VARIABLES TO BE ANALYSED

The next step is to select variables for the actual analysis. Note that all instructions given so far deal with the data storage only. This storage is general and can be reached from any set of rules. The following specifications connect the data with the particular problem considered in one-way analysis of variance. Again **highlight** "DATA" and press **ENTER**. Choose the item "SELECT DATA FOR ANALYSIS" by pressing **ENTER** (see Figure 2.3). To get access to the menu for selection of variables (Figure 2.6), press **ENTER** once more. **Highlight** the variable ONEW1 and press **ENTER**. Repeat this procedure for ONEW2 and ONEW3. To stop the selection of variables when the system prompts for variable no. 4, press **ESC** and then **ENTER**. (If an incorrect selection has been made, press ESC twice to restart the variable selection.) During a short interval, nothing happens on the screen while the system prepares internally for the analysis. Then **press any key** to return to the main menu.

5. SHOW AN EXPLANATION OF THE CURRENT SET OF RULES

Before the analysis is started, take a look at the explanation file for this set of rules. First **highlight** the option "FILES" in the main menu and press **ENTER.** Then select the item

"EXPLANATION OF RULES". Now the map shown in Figure A.1 (in Appendix A) is displayed. Use arrow keys, PgUp and PgDn to move around in this explanation file. To exit, press **F10**.

6. LIST THE SLOTS FOR THE CURRENT SET OF RULES

Also take a look at the slots associated with the current set of rules. **Highlight** "SLOTS" in the main menu and press **ENTER** twice. A list of the different slots will appear on the screen. Use PgUp and PgDn to move inside the list. If a particular slot is **highlighted** and selected by pressing **ENTER**, information about the slot value and how it was determined will appear on the screen. As the analysis has not yet been started, only a short message will appear to the effect that the slot value has not been found. Now **press any key** (except the ESC key) to return to the list of slots. **Press F5** to list the values of all slots. (The slot number is also shown on the left side.) Again we notice that no values have been found at this stage. To return to the main menu, **press ESC** twice.

7. SET SYSTEM VARIABLES

Before the actual analysis starts, check that all system variables have been properly set. **Highlight** the option "SYSTEM" and press **ENTER.** Suppose that we want Express to give as much information as possible in the course of the analysis. Then all system variables should be set to ON, except for the "DEBUG" variable which must be OFF. To change one of the settings, simply **highlight** the proper variable and press **ENTER**. (Only the first five lines in the SYSTEM submenu represent system variables. The lines below provide access to other parts of Express.) **Press ESC** to leave this submenu.

8. EXECUTE THE ANALYSIS

We are now ready to start the analysis (the chaining of rules). First **highlight** the option "RULES" and press **ENTER**. For this set of rules, three different items are shown in the basic problem menu. We want to investigate if there are any differences between the location parameters of the three variables selected, thus **press 1**. The analysis is started by pressing **ENTER**. Express will now use the upper window to indicate which rules are activated and which slots should be found. The lower window of the main screen is used to give additional explanations to various slots. During the subsequent analysis, we move from one step to another by following the instructions appearing in the bottom line.

The stack appears in the lower right hand corner of the screen. At the moment, only rule no. 1 has been put on the stack. Press **ENTER**. The text in the upper window now confirms that the main rule for the analysis of variance has been activated. Press **any key.** Express indicates that the rule refers to the slot with information on whether location parameters differ. Of course no information has yet been generated. Continue by pressing **any key**. To determine the correct slot value, Express considers the slot indicating whether all variables

are normally distributed. No value has been assigned to this slot either, but the user is given the opportunity to specify a suitable value (1 if all variables are assumed in advance to be normal, 0 otherwise). Press **ESC** to indicate that Express should generate the information without user interaction. The system responds by indicating that it will proceed on its own, and after any key has been pressed, it activates rule no. 2, for deciding whether all variables are normally distributed. Note how this rule is added to the stack, with the active rule at the top shown in green.

Press **any key** to proceed. Rule no. 2 refers to the slot indicating whether the first variable x1 is normally distributed. The user is again offered the possibility of specifying the corresponding slot value. Press **ESC** to show that this is not wanted. The system indicates that it will determine the slot value and goes on to activate rule no. 3, which is a general rule for making a decision on normality for any particular variable. To assist in decisions about normality, a histogram is generated for x1 by this rule. The diagram is considered as a slot with unknown "value", so the external package BMDP must be executed to generate the histogram. The next step involves setting up control language for BMDP, with the file and variable names inserted from the data base. A flashing yellow arrow shows where such substitutions are made. When the command stream has been completed, **F10** must be pressed to continue. In any case, Express informs the user that the program BMDP 2D is about to be executed. During the actual execution, the display changes to show standard information from BMDP. Depending on the computer, the execution may proceed so quickly that details cannot be distinguished. Express regains control and goes through the output file, extracting various relevant pieces of information in addition to the histogram. Each time any information is found which should be inserted into the data base, the program slows down. Search keys and relevant information are marked in yellow. The information extracted in this situation includes values of basic statistics such as skewness, kurtosis, mean values, etc. In addition, the *p*-value and test statistic for the Shapiro-Wilk test for normality are found.

Rule no. 3 is then reactivated. The histogram is displayed on the screen by Express as additional support for the user. Press **F10** to proceed from viewing the plot. It is now possible for the user to interrupt the chaining of rules (by pressing **ESC**) to make his own decision about the normality issue. However, the default rule refers to the *p*-value in question, and with the given data set, normality is accepted for the variable x1. Rule no. 3 is removed from the stack and rule no. 2 for all three variables is reactivated. This rule now refers to the slot indicating whether the next variable x2 is normally distributed, and so on. For simplicity, indicate to Express that the user has decided that x2 as well as x3 are normally distributed. (Express would reach the same conclusions in this data set.) In each case, press **ENTER** to show that the user will supply a value. The cursor moves into the red field where the appropriate specification can be typed. Press **ENTER** to go on. Rule no. 2 is now completed, with the conclusion that all variables are normally distributed, and only rule no. 1 remains on the stack.

The further analysis proceeds with rule no. 4, to decide whether the variables possess identical

variances. Express executes BMDP 7D, which gives the *p*-value for the Levene test, as well as other useful results for one-way analysis of variance. The final conclusion is drawn by rule no. 1 on the basis of the results for the *F*-test:

> The system has reached the following conclusion
> to the question "Do location parameters differ?":
> **The answer to the question is Yes!**

9. EXPLORE A SUMMARY OF THE ANALYSIS

After this conclusion has been presented, the main menu will once more be active. A summary of the analysis can be seen by displaying the log file on the screen. **Highlight** "FILES" and press **ENTER** twice. The log file is shown in the lower window of the main screen. To enlarge the window, press **F1**. The arrow keys and PgUp and PgDn can be used to move around in the file. Press **F10** to exit from the log file. It is also possible to review the output from the external packages executed during the session by selecting the option "SHOW RESULTS FROM PACKAGE". Both the log file and the file containing output from packages may be copied to external files specified by the user, by selecting the option "STORE FILE" from the submenu of "FILES". In this way the results can easily be printed.

10. EXPLORE SLOT VALUES AND RELATIONSHIPS

Now take a new look at the list of slots, by **highlighting** "SLOTS" and pressing **ENTER** twice. Also press **F5** to see the short summary of the slot values. To get a more comprehensive explanation, **highlight** the slot of interest and press **ENTER**. To return to the list of slots, **press any key**.

Finally, explore the relationship between slots. **Highlight** "RELATIONSHIPS" in the submenu for "SLOTS". In the list of slots shown, move ahead using the **PgDn** key until, for example, the slot "Is x1 normally distributed?" appears. **Highlight** this line and press **ENTER**. The lower window of the main screen now shows relations between the different slots (see Figure 2.15). A white arrow leads from the lower slot "p-value for x1 - Test for normality" (shown on a black background) to the selected upper slot, indicating that the *p*-value was used to decide whether x1 was normally distributed. Press **F5**. The direction of the arrow is reversed, and it now points to the slot indicating whether *all* three variables are normally distributed. This is the only slot based immediately on the normality of the first variable. Press **ENTER**. The slot for overall normality is moved to the upper field, and the arrow shows that the basic decision whether location parameters differ is based on the normality slot. Press **F5** to determine which slot values the overall normality depends on. Three arrows now lead from the normality for the separate variables to the overall normality. The different colouring in white and yellow indicates that the normality for x1 was decided on by the system, while the corresponding decisions for x2 and x3 were made by the user. Now press, for example, **F5** to return to the previous state and then press **ENTER** and **F5**,

so that relationships with the basic decision about location parameters are shown.

To exit to the main menu, press **ESC**.

11. ERASE A SLOT VALUE

To demonstrate the possibilities of Express, we erase the value of a slot, e.g. that for normality of x1. Select "SLOTS" and then **highlight** "SET SLOT TO UNKNOWN". The list of slots appears in a submenu. Move down to the slot marked "Is x1 normally distributed?" and select by **highlighting** and pressing **ENTER**. The upper Express window shows the current information about the slot. In the lower window, the user is given the opportunity to change his mind about deleting the information. If one wishes to continue erasing the slot value, move the yellow colouring from the option "NO" to the option "YES" using an arrow key and press **ENTER**. Express indicates that the information has been deleted.

Press **ESC** and then move into the option "DISPLAY SLOT VALUES". Use **F5** to display actual values along with slot names. It will be seen that not only the slot connected with normality of x1 carries the message "NOT YET FOUND", but so does any slot with a value which was previously based on the normality of x1. For example, the main decision about location parameters reached before is no longer valid.

12. END THE SESSION AND EXIT FROM EXPRESS

To end this sample session, **highlight** "EXIT" and press **ENTER**. If you want to leave temporarily, select the option "PAUSE". In this case all slot values will be saved for later use. If the option "QUIT" is selected, all results are erased.

The complete log file summarizing this sample session is shown in Appendix A.

2.7.2 Comparing location parameters for two samples

In this session we compare the location parameters of two samples. The data, derived from a study of Parkinson's disease, were reproduced from Section 12.3 of the Minitab Handbook [1]. The purpose is to test whether an operation affects the speaking ability of the patients. To run this set of rules, SAS must be installed on the PC in the directory C:\SAS (see Section 1.2.3).

1. START EXPRESS FROM THE DOS COMMAND LINE

Start Express from the DOS command line by typing

C:\EXPRESS> **EXPRESS**

To enter the menu for selection of a set of rules, **push any key**.

2. SELECT THE PROPER SET OF RULES

Use the arrow keys to **highlight** the item "3. COMPARISON OF LOCATION PARAMETERS FOR TWO SAMPLES". Carry out the selection by pressing **ENTER** (see Figure 2.1).

3. ENTERING VARIABLES INTO THE DATA STORAGE

The file C:\EXPRESS\PARKINSO.DAT contains the two samples that should be compared. The first variable, including 14 scores of speaking ability for patients who did not have an operation, covers positions 1-5. The second variable, including 8 scores for patients who had an operation, covers positions 6-10.

The procedure for entering these variables (samples) into the data storage is identical to that described in step 3 in the preceding section. Simply exchange the file name and use the correct values for the leftmost and rightmost positions. The two variables may for instance be named PARK_1 and PARK_2.

Note that the display of variables in the data storage only includes four names at a time. If the storage already contains three variables (say, from the previous sample session), adding two more makes it necessary to move ahead in the list using the PgDn key.

4. SELECT VARIABLES TO BE ANALYSED

The next step is to select the variables to be analysed. **Highlight** "DATA" and press **ENTER**. Now select the item "SELECT DATA FOR ANALYSIS" by pressing **ENTER** (see Figure 2.3). As shown in Figure 2.6, it is now possible to make a selection among the variables in the data storage. **Highlight** the variable PARK_1 and press **ENTER**. Repeat this procedure for PARK_2. Again it may be necessary to use PgDn and PgUp keys to move within the list of variables. To end the selection, press **ENTER**. Finally **press any key** to return to the main menu.

5. SHOW AN EXPLANATION OF THE CURRENT SET OF RULES

Select "FILES" in the main menu by **highlighting** and pressing **ENTER.** Then select the item "EXPLANATION OF RULES". A short explanation of the strategy used in the analysis is presented. Use arrow keys and PgUp and PgDn to move around in the explanation file. To exit, press **F10**.

6. LIST THE SLOTS FOR THE CURRENT SET OF RULES

Take a look at the slots associated with the current set of rules. **Highlight** "SLOTS" in the main menu and press **ENTER** twice. A list of the different slots appears. Use PgUp and PgDn to move around in the list. By **highlighting** a particular slot and making a selection by pressing **ENTER**, the slot value and information indicating how it was determined will be shown. As the analysis has not yet been started, only a brief message indicates that the slot value has not been found. Now **press any key** (except the ESC key) to return to the list of slots. **Press F5** to list the values connected to each slot. Once more we notice that no values have been found. To return to the main menu, **press ESC** twice.

7. SET SYSTEM VARIABLES

Highlight the option "SYSTEM" and press **ENTER.** We still want to adjust the system variables so as to give as much information as possible during the analysis. This is achieved by setting all variables to ON except for the "DEBUG" variable which must be OFF. **Press ESC** to leave this submenu.

8. EXECUTE THE ANALYSIS

Although the strategy is not very complicated, the complete execution takes a relatively long time. The analysis may be interrupted by pressing ESC at almost any time. If an EXIT is made from Express using the option "PAUSE", the session may be restarted later by making the appropriate selection in the menu for problem types. All slot values from the previous run are then retained.

First **highlight** "RULES" and then press **ENTER**. For the current set of rules, there are five items in the basic problem menu. We wish to investigate whether there is any difference between the underlying location parameters of the two variables. Hence **press 1** to make the appropriate selection. The analysis is then started by pressing **ENTER**.

The main rule no. 20 is started first. The strategy can handle different kinds of analysis, and the next rule activated, no. 21, determines whether the comparison should be made between observations in two different samples, paired observations, or count data in two categories in a contingency table. Express can handle this problem through default procedures, but if possible, the decision should obviously be made by the user. When the corresponding slot is considered by Express, press **ENTER** to indicate that a user selection will be made, and then specify the number **1** (for two independent samples). Rule no. 21 is subsequently removed from the stack.

The next rule activated, no. 23, performs trimming of the data, if necessary. This rule invokes rule no. 24, which determines whether a sample includes outliers. The definition is based on the interquartile range for the sample. To determine this value, a SAS run is carried out,

sorting the data according to sample and applying PROC UNIVARIATE to determine simple sample statistics. These values are used to define lower and upper limits for trimming, by moving away from the lower and upper quartiles by a distance of 2 "steps", where a "step" is defined as 1.5 times the interquartile range. In this data set, rule no. 24 leads to the conclusion that no trimming is necessary for either variable.

Rule no. 1 is then started to determine whether both variables are normally distributed. This is carried out by activating rule no. 10 in turn for each variable. SAS is executed to obtain various statistics and plots, using PROC UNIVARIATE, PROC CHART, PROC RANK and PROC PLOT. The plots, shown separately by Express, may be considered by the user for making an independent decision about normality, but the default decision is based on the sample skewness and kurtosis. Both statistics are standardized by dividing with their asymptotical standard error, valid under the assumption of normality. By treating the ratio as a standard normal variable, approximate *p*-values are found for the separate tests for normality based on skewness or kurtosis. Each *p*-value is classified as consistent with the normality assumption if it exceeds 0.15 and inconsistent if it is less than 0.05. Intermediate *p*-values are regarded as inconclusive. By considering the nine combinations of results from skewness and kurtosis, the overall conclusion is summarized in a "normal category", ranging from 0 to 5, with 0 representing results inconsistent with normality both for skewness and kurtosis, and 5 corresponding to consistent result for skewness as well as kurtosis.

Both the "normal category" and *p*-values for normality determined through SAS may be considered informally by the user in assessing the issue concerning normality. In the current data set, both variables turn out to belong to "normal category" 4, so no normality assumption is made using the default rule. Express then goes on to consider the possibility of log-normal distributions for the two variables, by similar procedures. The system also checks whether there are any negative observations. The hypothesis of lognormality turns out to be reasonable for the first variable but not for the second one, because of the incorrect kind of asymmetry indicated by a negative skewness. Finally, among several non-parametric test results found, the Wilcoxon test is used by the strategy to produce the final conclusion:

> The system has reached the following conclusion
> to the question "Are location parameters identical?":
> **The answer to the question is No!**

9. EXPLORE A SUMMARY OF THE ANALYSIS

Look at log file and output from SAS as described in step 9 in the preceding sample session.

10. EXPLORE SLOT VALUES AND RELATIONSHIPS

Follow the procedures in step 10 in the first sample session. For example, begin by considering the main slot "Are location parameters identical?", and go through the slots used

to decide on this main question.

11. END THE SESSION AND EXIT FROM EXPRESS

To end this sample session, **highlight** "EXIT" and press **ENTER**. If you want to retain the results found, select "PAUSE".

2.7.3 Regression analysis

This set of rules functions mainly as a front-end to the REG procedure in SAS, by fitting different regression models specified by the user. The file C:\EXPRESS\REGRES.DAT contains a data set with one dependent variable and four independent variables. This is Hald's data set used extensively as an example in [2]. Start reading the five variables into the data storage as described in step 3 in the first sample session. The positions of the variables on the file are: 1-6 for the first independent variable (which may be named x1, say), 7-12 for the second independent variable (x2), 13-18 for the third independent variable (x3), 19-24 for the fourth independent variable (x4) and 25-30 for dependent variable (Y). The same variables may be selected for analysis (see step 4 in first sample session). However, notice that the set of rules enquires about the *dependent* variable first, and then the independent variables.

As in the preceding sample sessions, the analysis can be activated by **highlighting** "RULES" and pressing **ENTER**. For the regression set of rules, the basic problem menu includes 7 options. Try several of these. Under option 5, for example, Express generates one set of commands for each independent variable and fits a simple linear regression. The resulting regression equations will be presented.

This data set is interesting in several regards because of the strong but not complete linear relationships among the independent variables. For example, all independent variables make at least marginally significant contributions in separate regression analyses. In an analysis including all four variables, however, only x1 has a p-value which suggests anything at all approaching significance.

Under the option SLOTS, it will be observed that Express maintains separate slots in connection with all possible models for the estimates and p-values associated with a particular independent variable.

References:

[1] Ryan B.F. and Joiner, B.L. (1994) Minitab handbook. 3rd ed. Duxbury Press, Belmont, Calif.

[2] Draper, N.R. and Smith, H. (1981) Applied regression analysis. 2nd ed. John Wiley, New York.

Chapter 3

GUIDE FOR THE KNOWLEDGE ENGINEER

3.1 INTRODUCTION 52
3.1.1 Files in Express 53
3.1.2 The Express editor 56

3.2 SELECTING A STATISTICAL STRATEGY FOR A SET OF RULES 62

3.3 SLOTS BELONGING TO A SET OF RULES 64

3.4 EXTERNAL PACKAGES NEEDED IN A SET OF RULES 65
3.4.1 Editing the system file 65
Basic information about sets of rules (65); Names of files for sets of rules (66); Names and other information on files (67); Statistical program packages (68); Subdivision of records in various files (68); Information on files which are specific to each set of rules (69); Files for data storage (70)
3.4.2 Packages executed by Express 70

3.5 WRITING PROGRAM CODE FOR A SET OF RULES 73
3.5.1 Utility routines provided by Express for use in rules 73
3.5.2 The main program for a set of rules 82
3.5.3 Subroutines representing rules 86
3.5.4 Compiling and linking the program for a set of rules 91

3.6 FILES FOR A SET OF RULES 92
3.6.1 Initiation of files and installation of a new set of rules 92
3.6.2 Editing the code file 93
Text and codes for slots (94); Commands for packages (96); Additional information for commands (97); Search keys (105); Codes for execution of packages (111); Menu for basic problem type (115); Particular text for this set of rules (116)
3.6.3 Editing the data base 117
3.6.4 Debugging 120

3.1 INTRODUCTION

In contrast to the end user, the knowledge engineer - in this chapter simply referred to as the user - must have a deeper understanding of the structure of Express. To modify a set of rules or to create a new set, some familiarity is required with the different files in the system. The user should also have a basic understanding of the construction of Fortran programs.

To create the knowledge base associated with a new set of rules, it is in general necessary to go through the following stages:

A decision should be made about the kind of problems the set of rules should deal with, and an outline of the structure of the analysis should be worked out, including an initial strategy map.

When a decision has been made regarding the basic structure, a list should be set up including the statistical quantities needed to execute the analysis. These are the "slots" referred to in the general description of Express.

A decision should be made as to external packages to be used in the calculation of statistical results. These packages must be prepared for execution under Express.

On the basis of the outline of the analysis, a set of Fortran subprograms must be constructed, incorporating the different parts of the analysis as separate rules. The basic structure of each subprogram must comply with the general conventions adopted in Express. A particular main program linking the separate rules must also be written. All subprograms can take advantage of a special library of utility routines supplied by Express.

Express provides a separate program to initiate files needed in the construction of a new knowledge base. The knowledge engineer must execute this program before proceeding to the last item on this list.

The last and often most time consuming part of this process is to insert the correct information into the code file for the particular knowledge base.

Each stage in this process will be described in detail below. In general we recommend that

the user should take a close look at knowledge bases which are already available before any new ones are constructed. In the subsequent presentation we provide some examples derived from the knowledge bases for regression analysis and for comparison of location parameters. Appendix A should be consulted for a relatively simple complete example involving one-way analysis of variance.

The list above may give the impression that the various stages of the process can be carried out successively as isolated steps. It must be pointed out, however, that the different parts of this process interact, and problems occurring at later stages must to some extent be taken into account in the earlier phases. In particular, when the program is constructed for a certain knowledge base, insertion of the corresponding information into the code file must be carried out in parallel. For this reason it is suggested that this chapter should be read in its entirety before any attempt is made to construct a new knowledge base.

3.1.1 Files in Express

During execution, several files with different kinds of information are opened by Express. The most important ones are listed below. The number given after each file name is the particular file number used by Fortran. A list of actual file names in each implementation is given in Appendix B.

Name	Description
The system file (5)	This file contains general basic information needed by the system. Immediately after Express has been started, information is extracted from the system file about other files used by the system and by the different sets of rules. The text associated with the menu for selecting a set of rules is also located in this file, with a code indicating the currently active set. Furthermore, basic information about external packages is stored in the system file, as well as information about files utilized by the data storage. The system file is described in detail in Section 3.4.1. **Direct file with record length 80.**
The code file (15)	Each knowledge base must be assigned a separate code file. This file includes definitions of all slots considered in the analysis. It also provides Express with more detailed information needed for execution of external packages. This includes generalized control language specific to such packages and search keys needed for extracting the relevant quantities from the output produced. Moreover, the code file

contains text for the menu used to select a particular problem within any set of rules. Space is also allocated to storage of a more comprehensive explanation of rules and slots. The structure of the code file is explained in Section 3.6.2.

Direct file with record length 100.

The data base (11)

Each knowledge base must include a particular "data base", which is used for storage of slot values as they are being determined by the currently active set of rules. There is a close connection between the code file and the data base. Each slot defined in the code file is represented by a separate record in the data base. When Express or the user assigns a value to a slot, this value will be stored for later use in the data base in accordance with special conventions. The data base is also important in the internal bookkeeping of relations between slots, and in recording the last reference to any particular slot in the log file. Details of the data base are given in Section 3.6.3.

Direct file with record length 70.

The following files are not so important to the user as the three fundamental files referred to above, but a description can be helpful in the basic understanding of the structure of Express.

File containing commands for a package (18)

When an external package is about to be executed, commands are extracted successively from the code file, and after some modification, if needed, the commands are written to this file. This text will be used as input to the external package.

Sequential file.

File containing output (17)

This file will contain the output after an execution of an external package. Express uses this file in the search for relevant quantities determined by the external package. (The file only includes the results from the last execution and is not identical to the file that can be displayed on the screen by the end user.)

Sequential file.

File collecting output (27)

This is the file that Express displays to the end user when an inquiry is made about the output from packages (see Section 2.4). All results from package executions are collected in this file.

Direct file with record length 80.

File containing plots and tables (16)

When a slot is defined as a block (i.e., a plot or a table), the corresponding "value" cannot be stored in the data base (as the data base only accepts numbers). In such cases Express stores an address in the data base which points to a particular record in the present file,

where the actual plot or table is stored.

Direct file with record length 80.

Log file (28)

During a chaining of rules, Express writes information about all steps taken to the log file. This file can be examined from the main menu, and it plays an important part in the understanding of an analysis that has been carried out.

Direct file with record length 80.

Text file (33)

File including general text for presentation during execution of Express. (This file does not include names of slots and any other text relating to a particular set of rules. This information is located in the code file).

Direct file with record length 80.

Help file (35)

The contents of this file are displayed when the end user asks for general help.

Direct file with record length 70.

File containing the stack and slots (32)

This file serves two different purposes. First, it is used by Express to store the stack during a chaining of rules. Second, it is used to record which slots are used by the current set of rules (considering record numbers in the code file and the data base). Whenever the end user selects a new set of rules, the system must regenerate the list of slots. This list is used when slots are displayed under the options associated with the item "SLOTS" in the main menu (see Section 2.5).

Direct file with record length 6.

File containing definition of windows (22)

Express can write information on the screen in many different windows. The upper left corner and the lower right corner of all such windows are defined in this file. The file also includes information about the colours used in different windows.

Direct file with record length 8.

Relationship file (23)

Express maintains a list of logical relations between the different slots considered in the current set of rules. For each slot which has been assigned a value, the data base includes a reference to an address in this file where information of such relationships is stored.

Direct file with record length 8.

Other files

In addition to the files referred to above, several files are used to maintain the data storage of Express.

Direct files with record length 80.

Several of the files in Express, including the system file, the code file and the data base, have been set up as direct files in Fortran. Direct files are random-access files with records that can be read and written in any order. The records are numbered sequentially, starting with 1, and all records have the same length. This contrasts with ordinary sequential files, with records arranged in the order in which they were written. The Express editor can only handle direct files. Thus, all files needed in the construction of a new set of rules are assumed to be direct. However, the editor includes an option for creating a direct copy of a sequential file and vice versa. The record length must be specified before any file can be edited. If the user wants to edit an arbitrary file, the file name and the number of records must also be given (see Figure 3.3).

In the following explanation of the coding of rules in Express, the close connection between the code file and the data base plays an important role. The first section of the code file defines the slots considered in the analysis. For example, if a slot is defined in record no. 30 in the code file, then record no. 30 in the data base will store information about this slot during the analysis. Several codes are inserted into the data base to keep track of how the slot value was found and to indicate its relationship to other slots. The slot value itself is also stored in the data base.

However, this is not the only connection between the code file and the data base. When an external package is about to be executed, all commands used as input to the package must be stored in the code file. Such commands frequently need additional information, extracted from the data base, in order to be complete. This also applies to search keys representing particular results in the output produced by the package. In this connection, a special feature of the data base is often taken advantage of. This is the code inserted at position 5 in each record in the data base. This code differs from zero only if the corresponding slot has been assigned a value, and the actual non-zero code indicates how the slot value was determined (see Section 3.6.3).

3.1.2 The Express editor

The Express editor is used to write essential information into the files associated with a set of rules. This section does not give any detailed account of the actual information stored in these files but provides a general introduction to the use of the editor.

The editor can be started as an independent program from the command line in DOS by typing C:\EXPRESS\EDITOR\ED. Alternatively, it can be started from the main Express menu under the "SYSTEM" option. If it is impossible to start the editor from inside Express, the computer probably has not enough memory available to execute both programs at the same time.

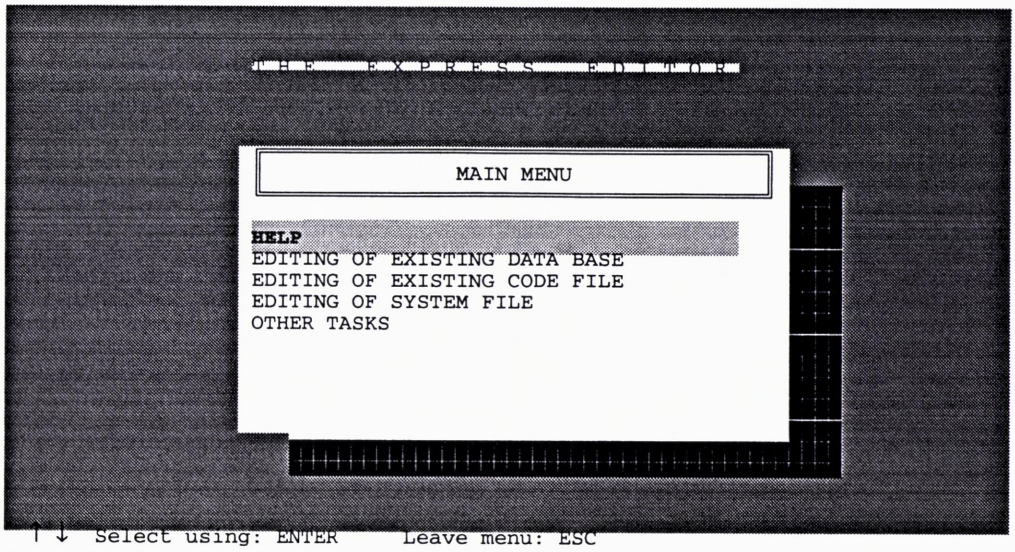


Figure 3.1 The main menu of the EXPRESS editor.

The main menu of the editor appears first on the screen (Figure 3.1). As in other menus of Express, a particular item is selected by highlighting the corresponding text and pressing ENTER. The main menu offers a choice between editing the data base or the code file for the active set of rules, or the general system file. In addition, the item designated "OTHER TASKS" offers several facilities described below. A "HELP" function is also available. The kind of help presented depends on the current situation when the request is being made. When a file is edited, help can be obtained by pressing the function key F1. Thus if F1 is pressed when the data base is edited, instructions for handling this particular file will appear on the screen.

After a particular file has been selected in the editor, the different records will be shown on the screen. The cursor indicates the current position where editing will take place. During editing, the record number corresponding to this position will be displayed in the bottom right corner of the editor. Each record is divided into a certain number of fields, with codes for different kinds of information. As an example, Figure 3.2 illustrates editing of the particular section of the code file which contains control languge for external packages. The complete file structure is described in Section 3.6.2. Of the three fields, the last one contains the commands, with an indication of which parts which must be changed or completed before the external package can actually be executed. The percentage sign is used for this purpose. The first field of each record includes an address (a record number) to another section of the code file where the information needed to complete the command can be found. The code "1" indicates that the command in question is already complete. The middle field is used to indicate whether parts of the command can be deleted. How records are divided into fields differs between files. In addition, the system and code files are subdivided into several successive sections, each with a different separation of records into fields. The editor will keep track of the subdivision into fields for standard files selected by the user from a menu in Express. For other files, the editor prompts for positions separating fields before the editing

starts.

To protect the information stored, the editor will by default not allow changes to be made to any file. For actual editing of the text in the current record, the particular F2 or F3 keys must be pressed. Changes may then be made in the current field or the entire current record, respectively.

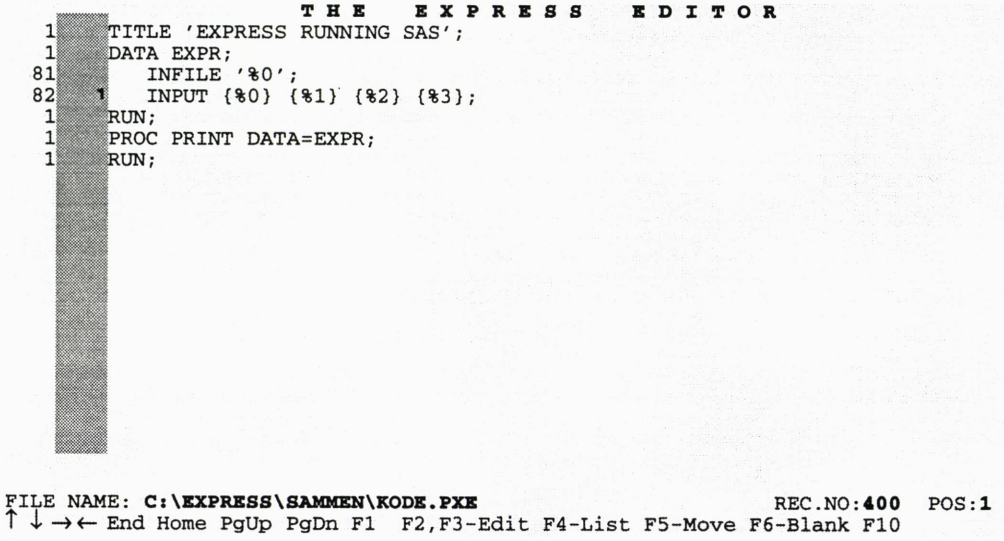


Figure 3.2 The editor screen during editing of a file.

The list below describes the commands available for editing.

Command	Description
< left arrow >	Moves the cursor one position to the left.
< right arrow >	Moves the cursor one position to the right.
< up arrow >	Moves the cursor one record up, unless the cursor is positioned at the first record shown on the screen. (To move to records not currently shown on the screen, use PgUp, PgDn or F4.)
< down arrow >	Moves the cursor one record down, unless the cursor is positioned at the last record shown on the screen. (To move to records not currently shown on the screen, use PgUp, PgDn or F4.)
< PgUp >	Moves the cursor to the top of the previous page.
< PgDn >	Moves the cursor to the top of the next page.

< F1 >	Provides help in the present situation.
< F2 >	Makes it possible to modify the contents of the field in which the cursor is located.
< F3 >	Allows editing of all fields in a record, one by one.
< F4 >	Moves the cursor to a specified record number and changes the screen so that this record is the first one shown. When this key is pressed, the editor prompts for the record number.
< F5 >	Makes it possible to move records from one location in the file to another. The editor will inquire about the record numbers of the first and last lines to be moved, in addition to the record number where the lines should be inserted.
< F6 >	Erases the contents of records specified by the user. The user will be prompted for the first and last records to be erased.
< F10 >	Exit from the present file.
< Ins >	This function has an effect only during actual editing when changes may be made to the file. It switches between INSET mode and TYPEOVER mode. The current mode is displayed in the upper right corner during editing. **Insert mode:** Additional characters are inserted at the location of the cursor. Existing characters are pushed to the right. **Typeover mode:** Characters are replaced at the location of the cursor.

When a particular field is being edited, it is sometimes useful to know the total number of positions currently occupied. This number is shown in the upper righthand corner. When the text in the field has been modfied, blank positions after the last non-blank position are not included in this count.

The option "OTHER TASKS" in the main editor menu offers several other facilities for file handling, listed below.

Facilities	**Description**
Editing of other files	This function makes use of the information stored in the system file to edit particular other files. (The selection of files is made by the knowledge engineer). The user can in addition select an arbitrary file for editing (see Figure 3.3). In that case, the exact record length and an

upper bound for the number of records must also be specified. The next step is to divide the records of the file into separate fields. The first record of the file will be shown on the screen, and the user can apply the arrow and ENTER keys to select limits for different fields. Thus, if a file should be divided into two fields, the first covering the positions 1 to 10 and the second the positions 11 to 80, the cursor should be moved to position 11 and the ENTER key pressed. Now the first field has been selected and cannot be changed. To complete the subdivision into fields, move the cursor to position 81 and press ENTER. Finally, to proceed to the actual editing, press F10. If a file has a record length of, say, 80 characters and we are only interested in the first 40 positions, we may select several fields between positions 1 and 40 and then press F10. The editing of the first 40 positions on each record can then start.

Records to be copied

This facility makes it possible to copy one or more records from a source file to another file (possibly the same one as the source). The editor will prompt for file names, record lengths and records to be copied (Figure 3.4).

Convert a direct file to a sequential file

The editor will prompt for information about the names of the direct and sequential files and the record length of the direct file.

Convert a sequential file to a direct file

This facility is similar to the previous one, but copying is carried out in the reverse order. If a line in the sequential file exceeds the record length of the direct file, the characters beyond the record length limit will not be included in the direct file.

Produce a printable file

Every non-empty record in a direct file, with the corresponding record number, is copied to a printable sequential file named by the user. This facility is of particular use for debugging purposes when a new code file is being created.

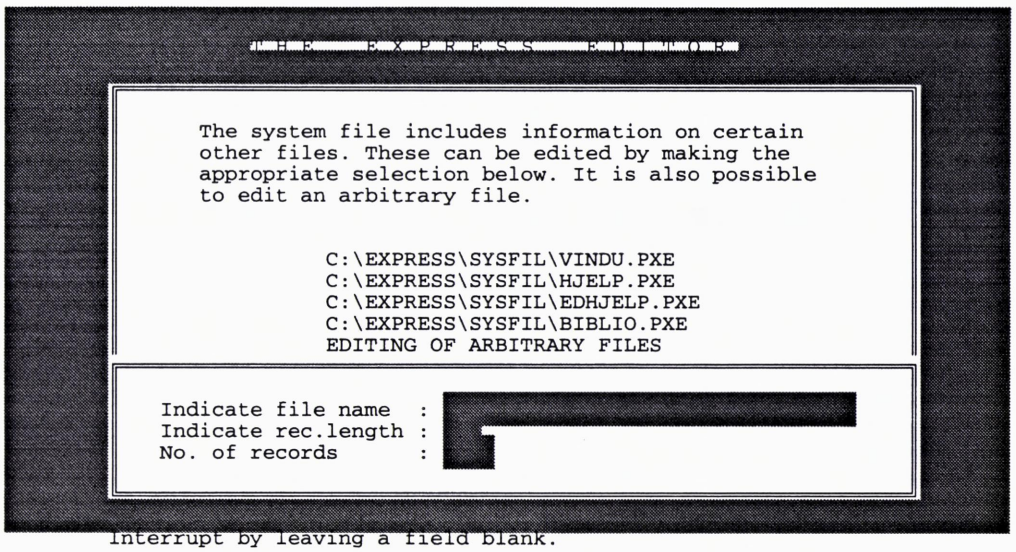


Figure 3.3 Screen for selecting another file to be edited.

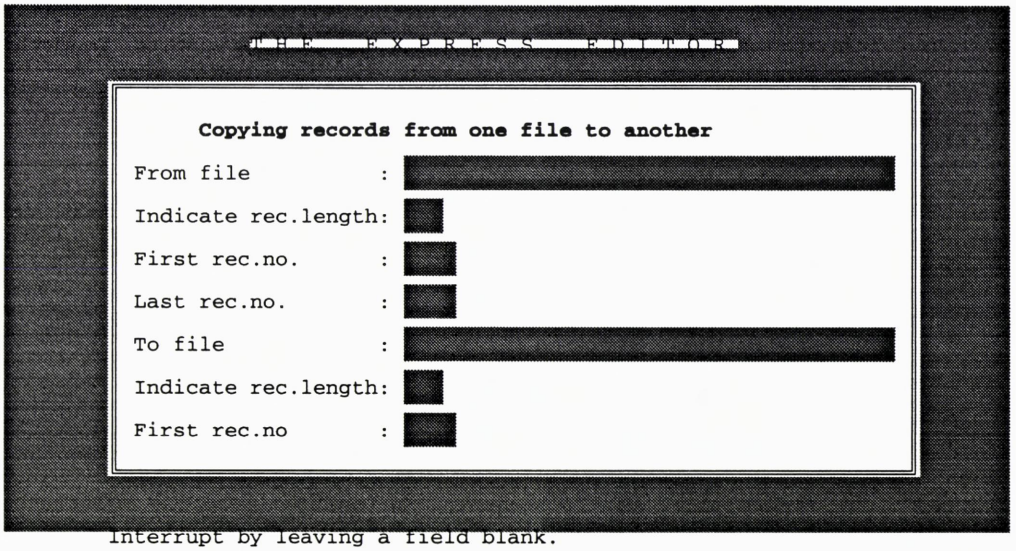


Figure 3.4 Screen for copying records from one file to another.

3.2 SELECTING A STATISTICAL STRATEGY FOR A SET OF RULES

The first step in creating a new set of rules is to decide on the general kind of analysis that should be performed. Consider as an example the set of rules implemented in Express for comparing locations parameters of two variables. In this situation the structure of the analysis can be outlined by a strategy map as shown in Figure 3.5.

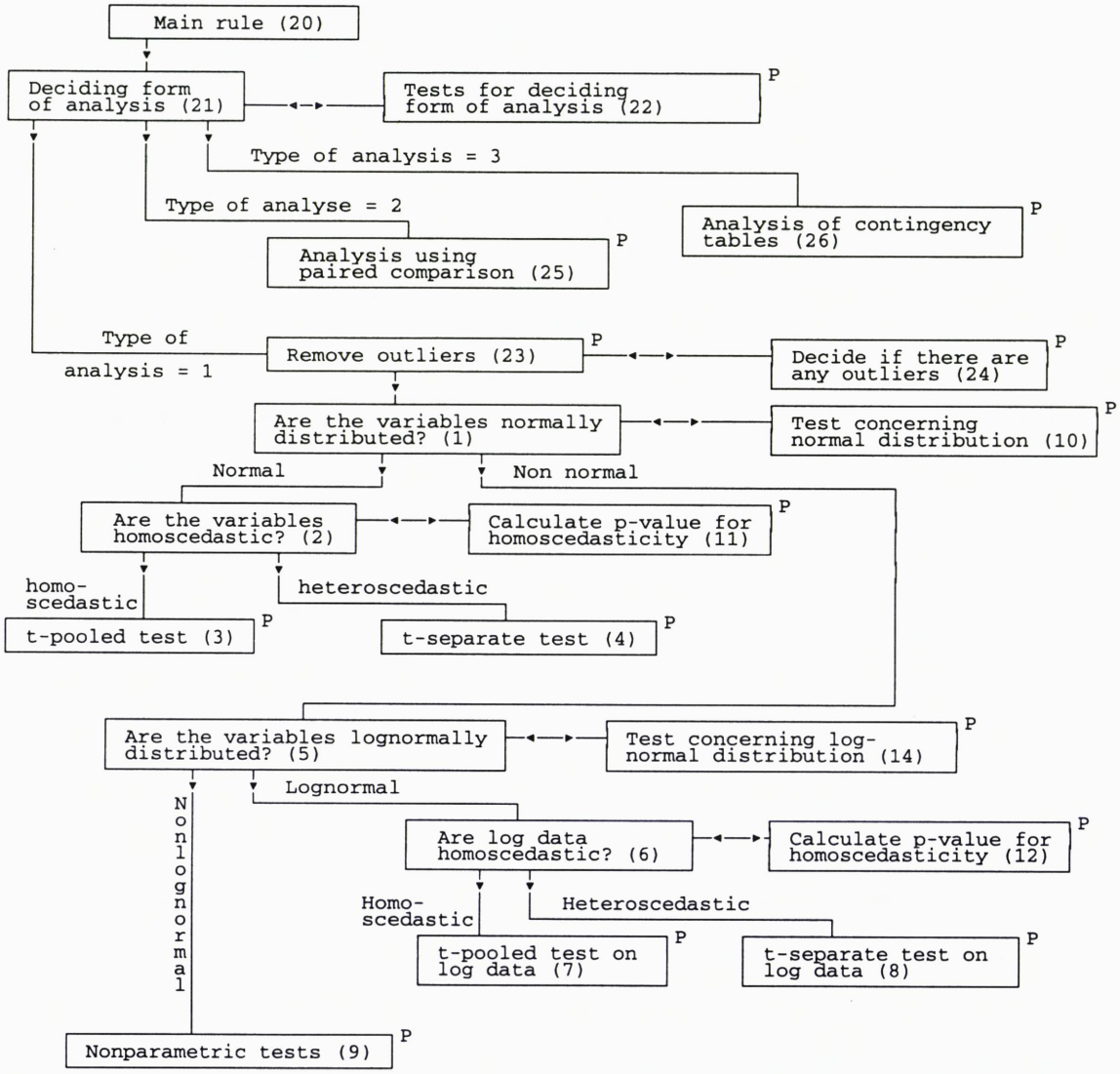


Figure 3.5 Describing the strategy for deciding whether two variables have identical location parameters.

In this case, the outline actually includes the complete set of rules. Each box in Figure 3.5 represents a single rule, with the rule number given in brackets. A letter "P" in the upper right corner indicates that an external package may be executed from the rule considered. In most situations, it is not so easy to set up an outline of the analysis in advance, but it is recommended that sufficient time be spent on developing as detailed a map as possible. It is technically possible in Express to let one rule be responsible for several distinct parts of the

analysis, but this is not normally advisable. It is preferable at this stage to attempt to break the analysis down into logically separate parts.

3.3 SLOTS BELONGING TO A SET OF RULES

Slots represent reserved storage space in the data base in Express, for statistical quantities or for conclusions reached during an analysis. All slots must be defined in the code file before a new set of rules can be executed. Within the set of rules, particular numerical values, or more generally information given by pieces of text, can be assigned to a slot in different ways. Express distinguishes between seven slot types, depending on the kind of information stored and the procedure prescribed for obtaining the information. The different types are described in Section 3.6.2 in connection with the editing of the code file. Each slot should be assigned a short descriptive name, occupying at most 40 positions.

Most slots are associated with a particular variable or a group of variables in the statistical analysis. For example, slots representing statistical results such as the number of observations, the skewness coefficient, the kurtosis, the histogram and the normal plot must be defined for each of the two variables in the set of rules comparing location parameters. Each variable is also assigned a separate slot for the response to questions of the following kind: "is the variable normally distributed?" or "do the sample values for this variable include outliers?" On the other hand, slots representing general conclusions concerning equality of variances over variables, or the main issue of identical location parameters, are defined once only, for the complete data set.

In situations of this kind, several similar computations must be carried out successively to determine slot values for different variables. It is not necessary to write separate rules for each variable in such situations. For example, only one rule is used in the location parameter set for deciding on normality of any variable. When the rule is activated, it receives a particular number indicating which variable is referred to. In this connection the slots in the data base must be organized in a special way. The sequence of slots for the first variable can, for example, start at record no. 100 in the data base. Assume that all records up to and including record no. 199 are used for this variable. Records 200-299 should then contain the corresponding slots for the second variable, in the same order. A rule which is able to deal with either variable determines in each case the location of the relevant slot by referring to a particular "jump code" (equal to 100 in this example). The slot number is computed in a straightforward manner, taking into account the variable number. If it is not feasible to assign slot locations in this regular way, it is of course possible to write a separate rule for every operation performed on each variable. With three or more variables, however, the testing and maintenance of rules becomes much easier with a regular slot structure.

3.4 EXTERNAL PACKAGES NEEDED IN A SET OF RULES

3.4.1 Editing the system file

The system file is divided into several sections on the basis of the information stored in the different records. Furthermore, each record is usually subdivided into distinct fields, with different specifications. The field lengths are shown in brackets in the explanation below. To start editing the system file, select the corresponding item in the main menu of the Express editor. A secondary menu will be displayed and the user may select a particular section of the system file for editing (Figure 3.6).

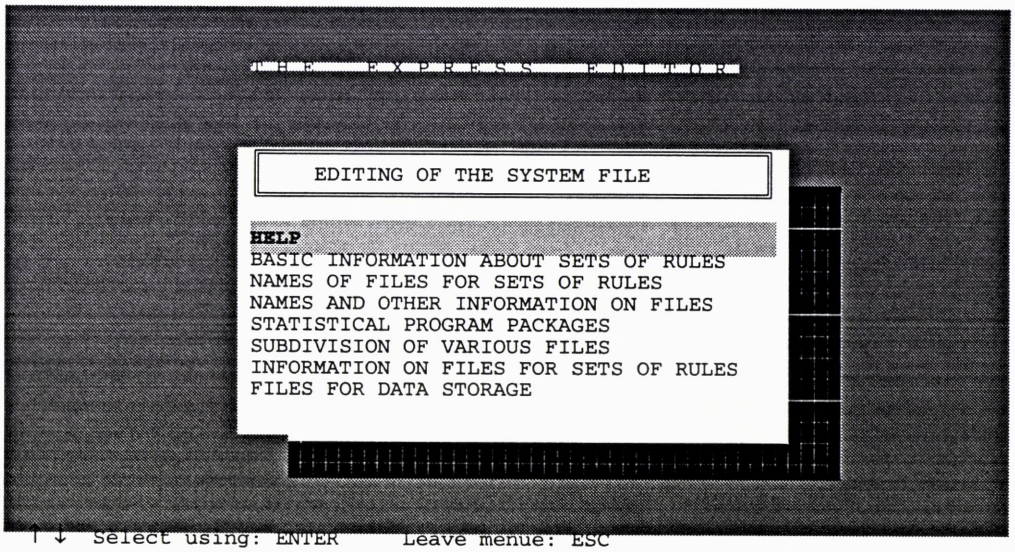


Figure 3.6 Menu for selecting a part of the system file for editing.

We will first give a comprehensive explanation of each item in this list.

Basic information about sets of rules

The first 8 records in this section include permanent specifications of how records are divided into fields in various subsequent parts of the system file. This information is needed by the editor to display the correct screen during editing. Record no. 10 defines the name and the record length of the system file itself, in addition to the number of records. **THESE CODES MUST NOT BE CHANGED.**

Figure 3.7 shows records no. 11-16 in the standard version of Express. Each record provides basic information about a set of rules. It should be noted that the presentation in the figures of this guide differs slightly from the display in the editor. Rather than using different colours to separate fields, as the editor does, we will use solid vertical lines. Also note that the record

number, included on the left side of the figures (outside the box), is not actually part of the file.

11	6	60	1.TRIAL SET OF RULES	141	1	151
12		60	2.EXECUTING AN EXTERNAL PACKAGE	325	1	321
13		60	3.COMPARING LOCATION FOR TWO SAMPLES	696	2	690
14		60	4.REGRESSION FOR UP TO FOUR INDEPENDENT VARIABLES	1065	5	1031
15		60	5.LOGISTIC REGRESSION FOR TWO INDEPENDENT VARIABLES	2373	3	2352
16		60	6.ONE-WAY ANALYSIS OF VARIANCE	1311	6	1350

Figure 3.7 Contents of the first section of the system file.

Field 1 (4 positions, right adjusted): This field is empty except for record no. 11. The number inserted at this position keeps track of how many sets of rules have been included in the system (at most 10). This number is automatically adjusted when a new set of rules is added by the separate installation program (see Section 3.6.1).
Field 2 (4 positions, right adjusted): Indicates the length of the name associated with the set of rules, which will be specified in field no. 3.
Field 3 (60 positions, left adjusted): This field contains the name of the set of rules. As for fields 1 and 2, this information will be inserted automatically during the installation procedure for a new set of rules. In contrast, the next three fields must be filled in by the user.
Field 4 (4 positions, right adjusted): The code file can include a brief explanation (at most one line) of each rule in a particular set. Field no. 4 should give the record number of the first record containing an explanation.
Field 5 (4 positions, right adjusted): Gives the maximum number of variables that the set of rules can accomodate. (This field is not active in the current version of Express).
Field 6 (4 positions, right adjusted): This field gives the address in the code file (the record number) of a short explanation of each variable that must be selected. For example, in the selection of variables for the set of rules for regression analysis, a message is given when the dependent variable should be selected, and another message when each independent variable is selected.

Names of files for sets of rules

The records belonging to this section of the system file consist of a single field. However, the section itself is divided into two subsections (records 21-30 and 31-40) with different kinds of information:

Records 21-30: Each record provides the name of the main executable file representing a set of rules. The order of the sets corresponds to that used in the previous section (records 11-20) of the system file. The file name associated with a particular set of rules must be specified by the user during the installation of the set.
Records 31-40: A more comprehensive explanation of a set of rules can be stored in a direct file (with record length 70), which can be consulted by the end user during execution. If such

a file has been prepared for a particular set of rules, its name should be inserted in the appropriate record in this section. Otherwise the record should be left blank.

Figure 3.8 illustrates the use of this part of the system file. Record no. 21 gives the name of the executable file for the first set of rules, record no. 22 for the second set of rules, and so one. An explanation file is only available for the third set of rules in record no. 33. If an explanation file should become available later, the corresponding name can be inserted into the appropriate record at that stage.

```
21 C:\EXPRESS\PROVE\REG.EXE
22 C:\EXPRESS\EGENKJOR\REG.EXE
23 C:\EXPRESS\SAMMEN\FIRREG.EXE
24 C:\EXPRESS\REGREG\REG.EXE
25 C:\EXPRESS\LOGREG\REG.EXE
26 C:\EXPRESS\ONEWAY\REG.EXE
 .
 .
31
32
33 C:\EXPRESS\SAMMEN\SAMTO.RFA
34
35 C:\EXPRESS\LOGREG\FORKLAR.PXE
36 C:\EXPRESS\ONEWAY\ONEWAY.RFA
```

Figure 3.8 Second section of the system file, with two subsections.

Names and other information on files

The records in this section of the system file are divided into 11 fields. The section consists of three subsections.

Record 50: Figure 3.3 shows the screen when an arbitrary file is selected for editing. A menu with at most four file names will appear. Record no. 50 in the system file is used to specify which file names will be displayed. The specifications refer to the more general definitions of files in records 59-80 below. The first four fields of record no. 50 should contain at most four of these record numbers. The actual file names displayed will then be read from the corresponding records.

Records 51-58: This space is reserved for internal use by Express to indicate the location of several essential files referred to during execution of an external package. The contents of these records should not be changed.

Records 59-80: These records provide basic information used for handling certain important files in Express. Each file is represented by text occupying two records in the system file. In the first record, field no. 1 should specify the record length. Fields 2 and 3 contain the first and last record numbers, respectively. Field no. 11 indicates the file name. The second record assigned to a particular file is used to store limits for the different fields in the records of the file. This information is used by the editor. The first field in the second record should indicate the total number of fields in question. The subsequent fields in the second record define the left limits of fields. Thus, if the first field in the external file covers positions 1-5, the number 1 must be given in field no. 2 and the number 6 in field no. 3. The upper limit for the last

field is specified in the same way, by an additional number equal to the rightmost position of the last field plus one.

Statistical program packages

Records no. 81-100 of the system file contain information needed for executing external software under Express. Each record represents an external program which can be started separately. Express will in each case use a particular BAT file containing DOS commands to initiate the execution.

Field 1 (47 positions, left adjusted): This field indicates the name of the BAT file.
Field 2 (4 positions, left adjusted): When a variable is read into the Express data storage, it may include missing values which are marked in a particular way by Express. However, when a data file adapted to an external package is generated, the particular missing value code for this package must be inserted. This code must be stored in field no. 2, left adjusted.
Field 3 (9 positions): Not in use.
Field 4 (20 positions, left adjusted): This field gives a brief name associated with the external package. This name will be presented to the end user when the package is executed during a chaining of rules.

Figure 3.9 shows the definitions needed to run SAS, Minitab, several of the programs included in the BMDP package and StatXact.

81	C:\EXPRESS\SYSFIL\SAS.BAT	.		SAS.
82	C:\EXPRESS\SYSFIL\MINITAB.BAT	*		MINITAB.
83	C:\EXPRESS\SYSFIL\BMDP5.BAT	*		BMDP 5D.
84	C:\EXPRESS\SYSFIL\BMDP4F.BAT	*		BMDP 4F.
85	C:\EXPRESS\SYSFIL\BMDPLR.BAT	*		BMDP LR.
86	C:\EXPRESS\SYSFIL\BMDP2D.BAT	*		BMDP 2D.
87	C:\EXPRESS\SYSFIL\BMDP1D.BAT	*		BMDP 1D.
88	C:\EXPRESS\SYSFIL\BMDP3D.BAT	*		BMDP 3D.
89	C:\EXPRESS\SYSFIL\STAT.BAT	*		STATXACT.

Figure 3.9 Codes for defining packages to run under EXPRESS.

Subdivision of records in various files

This section of the system file is used to divide the records in the data base, the code file and certain other files into separate fields. A single record is used to define the fields of each file which has a common subdivision for all records. For the code file, however, a separate record is needed for each section of the file (in the same way as for the system file). As before, the first field indicates the number of fields to follow, and subsequent fields define the field limits in the file in question. This information should not be changed. The specifications are given in record 101 for the data base, in records 102-108 for the code file, in record 110 for the relationship file, and in record 111 for the file containing the stack and slots.

Information on files which are specific to each set of rules

Each set of rules must have its own data base, code file, relationship file, file for plots and tables, file collecting output, and log file. This section of the system file is used to store information about these files, starting in record no. 131. Record no. 130 is used in general to record which set of rules is currently active at any time. Records 131-145 belong to the first set of rules, records 146-160 to the second set and so on, each set occupying a block of 15 consecutive records. The first record in each block is used to define the data base. Field no. 1 defines the record length, whereas fields 2 and 3 indicate the first and last record numbers. The file name is located in the final field.

This is the common layout of the file definitions in this section of the system file. The second record in each block provides the general definitions for the code file. However, additional specifications are needed for this file, covering records 3-8 in each block. Each record is associated with a particular section of the code file (as defined in Section 3.6.2). Thus the third record in each block gives the first and last record numbers of the particular section of the code file which defines slots. The fourth record specifies the boundaries of the section dealing with execution of packages. The fifth record indicates where commands should be stored for external packages in the code file, and the sixth record sets the boundaries for the section with additional information for incomplete commands. The seventh record in each block shows the location of search keys in the code file, and the eighth record sets limits for text needed in the presentation of the set of rules. The section of the code file concerned with basic problems for which a chaining of rules can be started, occupies the remainder of the file. Thus no specifications should be given in the system file for this section.

The next set of definitions in the block in the system file pertain to the relationship file. The following record is used internally by Express. Then definitions are given successively for the plot/table file, the output file and the log file. The specification of record numbers and lengths will be inserted by the installation program for a new set of rules. The number of records can be modified by the user if required BUT RECORD LENGTHS MUST NOT BE CHANGED. Figure 3.10 shows the codes for the "TRIAL" set of rules (the first set) and the set comparing location parameters for two variables (the third set). Note that the number of records may differ between similar files belonging to distinct sets of rules.

```
130     6
131    70     1    30 C:\EXPRESS\PROVE\GDB.PXE
132   100     1    30 C:\EXPRESS\PROVE\KODE.PXE
133     1    30
134    31    40
135    41    80
136    81   100
137   101   120
138   121   200
139     8     1  1000 C:\EXPRESS\PROVE\CONECT.PXE
140     6     1  1000 C:\EXPRESS\SYSFIL\TABTO.PXE
141    80     1   110 C:\EXPRESS\PROVE\PLOT.PXE
142    80     1   500 C:\EXPRESS\PROVE\OUTPUT.PXE
143    80     1  1000 C:\EXPRESS\PROVE\LOG.PXE
  .
  .

161    70     1   300 C:\EXPRESS\SAMMEN\GDB.PXE
162   100     1   550 C:\EXPRESS\SAMMEN\KODE.PXE
163     1   300
164   301   320
165   321   550
166   551   570
167   571   650
168   651  1000
169     8     1  1000 C:\EXPRESS\SAMMEN\CONECT.PXE
170     6     1  1000 C:\EXPRESS\SYSFIL\TABTO.PXE
171    80     1  1000 C:\EXPRESS\SAMMEN\PLOT.PXE
172    80     1  1000 C:\EXPRESS\SAMMEN\OUTPUT.PXE
173    80     1  9999 C:\EXPRESS\SAMMEN\LOG.PXE
  .
  .
```

Figure 3.10 Information, about files connected to a particular set of rules, stored in the system file.

Files for data storage

The final section of the system file contains definitions of files used in the data storage. The first record (record number 300) in this section specifies two numbers, the total number of files in the data storage and the the number of variables included in the storage at the current time. If the end user indicates that the data storage should be erased, the only action taken is to set the latter number equal to zero.

3.4.2 Packages executed by Express

We will describe in more detail the preparations needed for running external packages under Express. The set of rules considered as an example, comparing location parameters for two variables, relies on SAS only but previous versions of this set also used Minitab. Figure 3.9 shows the definitions needed in the system file for these particular packages. A period is used to indicate a missing value in SAS, whereas the corresponding symbol is an asterisk in Minitab. Now consider the BAT files with commands initiating the correct program executions. Two separate tasks must be handled by these files:

- **First,** the package must be executed using the instructions (written in the particular language prescribed for this package) which are stored in the file C:\EXPRESS\SYSFIL\STYR.PXE .

- **Second,** the output from the package must be copied to the file C:\EXPRESS\SYSFIL\UTSKRIFT.PXE .

Examples

Figure 3.11 shows the commands in the BAT file adapted to SAS (version 6.3):

```
CD\SAS
SAS.EXE C:\EXPRESS\SYSFIL\STYR.PXE  >C:\EXPRESS\SYSFIL\BOSS.PXE
COPY C:\SAS\STYR.LST C:\EXPRESS\SYSFIL\UTSKRIFT.PXE
CD\EXPRESS
```

Figure 3.11 Contents of the file C:\EXPRESS\SYSFIL\SAS.BAT

First the active directory is changed to the directory containing the SAS package. The next command executes SAS, using the correct file as input. A special feature of this command is the redirection of all output which would normally go to the screen to the particular auxiliary file C:\EXPRESS\SYSFIL\BOSS.PXE. This is done to avoid disrupting the Express screen, since running SAS in batch mode normally produces some messages on the screen. With other packages such as Minitab, it is not always possible to retain the Express screen during execution of the external software. In such situations, Express will restore its screen after the package execution. The next command in the BAT file copies the ordinary output, generated by SAS in the file C:\SAS\STYR.LST, to the general file in which Express is supposed to find the output. Finally, the active directory is changed to C:\EXPRESS.

To execute Minitab (version 7.1), particular care is needed to set up a proper run stream (Figure 3.12). This version of Minitab does not allow the program to run in ordinary batch mode with direct options for naming input and output files. Instead we must rely on both redirection operators in DOS, "<" for input and ">" for output. Thus, in the second command in Figure 3.12, the input is taken from the file C:\EXPRESS\SYSFIL\BAT.PXE and the output is sent to the file C:\EXPRESS\UTSKRIFT.PXE.

```
CD\MINITAB
MTB71_S <C:\EXPRESS\SYSFIL\BAT.PXE  >C:\EXPRESS\SYSFIL\UTSKRIFT.PXE
CD\EXPRESS
```

Figure 3.12 Contents of the file C:\EXPRESS\SYSFIL\MINITAB.BAT

In this situation the BAT file does not refer explicitly to the standard file C:\EXPRESS\SYSFIL\STYR.PXE containing the internal commands. This is because Minitab must be started before the program can be given the name of the file containing Minitab

commands. For this reason, an intermediate file C:\EXPRESS\SYSFIL\BAT.PXE is used which only includes the Minitab command EXECUTE 'C:\EXPRESS\SYSFIL\STYR.PXE'.

As a final example we list one of the BAT files used for executing BMDP (Figure 3.13).

```
D:
CD\BMDP
ERASE LIST
BMDP 1D IN=C:\EXPRESS\SYSFIL\STYR.PXE OUT=LIST
COPY LIST C:\EXPRESS\SYSFIL\UTSKRIFT.PXE
C:
```

Figure 3.13 Contents of the file C:\EXPRESS\SYSFIL\BMDP1D.BAT

In our implementation, this package was located on the D: disk. Hence we first change the active drive and the directory. BMDP will normally append the output from a particular run to any existing text in the output file. In order to store only the most recent output, the contents of this file are first erased. Then the particular program 1D in the BMDP package is executed with appropriate input and output files, before the output is copied to the designated output file in Express.

The commands needed for execution of other external packages will in most situations follow the same pattern. Minor modifications might be needed, depending on the rules prescribed for starting the package in each case. Sufficient space is allocated in the system file for up to 20 different external packages to run under Express.

3.5 WRITING PROGRAM CODE FOR A SET OF RULES

A set of rules consists of a Fortran program, reflecting the chosen statistical strategy, in addition to information given in the code file and other files associated with the rules. Because of the strong links between the program code and the code file, the development of these major components of the set of rules should proceed in parallel. The organization of the main Fortran program is described in Section 3.5.2. The different rules will normally be represented by separate Fortran subroutines, as explained in Section 3.5.3. Both the main program and the subroutines should make extensive use of general utility routines provided by Express for technical purposes. These routines are described in Section 3.5.1.

3.5.1 Utility routines provided by Express for use in rules

The list below gives an explanation of the various routines available. The type of each argument passed to or from a routine is given in brackets, where "I" stands for an integer variable, "R" for a real (floating point) variable, "L" for a logical variable and "S" for a string (a character variable). For strings, a number following the "S" indicates the length.

Routine	Format and description
INITRE	**Format:** CALL INITRE() The call to this routine must be the first executable statement in the main program of any set of rules. This routine automatically opens all files needed in the execution of the set of rules.
STENGRE	**Format:** CALL STENGRE() This routine closes all files opened by the currently active set of rules. The call to this routine must be the last executable statement in the main program.
HENTPAR SENDPAR	**Format:** CALL HENTPAR(DATANO,RECORD,CODEREC,JUMP) **Format:** CALL SENDPAR(DATANO,RECORD,CODEREC,JUMP) When a connection has been established to the correct files, certain basic arguments must be retrieved from the main outside program in Express by means of the routine HENTPAR. Similarly, when the program for the set of rules terminates, the same arguments must be passed to the main Express program through the routine SENDPAR. The arguments will at any time be stored in the system file (by SENDPAR and the outher shell of Express). These arguments are: DATANO(I) This argument can in principle be used for different purposes, as a parameter affecting decisions being made by the rules. Its

main usage has been to record which variable is currently being analysed, as in the set of rules comparing location parameters. In the rules for regression analysis, however, the argument DATANO is used to record which model is analysed.

RECORD(I) This argument shows the current record number being handled in the data base. The argument is not significant when the program for a set of rules is entered. However, it is essential to pass this record number to the main Express program when a temporary exit is made to execute an external package.

CODEREC(I) If this argument is non-zero when the main program in Express regains control, it indicates that the set of rules has been left temporarily to provide space for the execution of an external package. Different non-zero values of this argument indicate which package and streams of commands should be executed.

JUMP(I) As explained in Section 3.3, this is the "jump code" used to calculate the correct record numbers in the current application of the active rule. This facility makes it possible to utilize the same rule to determine similar quantities pertaining to different variables.

PAASTB **Format:** CALL PAASTB(RULENO,DATANO)

This routine pushes a particular rule onto the top of the Express stack.

RULENO(I) The number of the rule.

DATANO(I) The current value of the quantity DATANO (see HENTPAR above) is stored on the stack until the rule is popped off.

LESSTB **Format:** CALL LESSTB(RULENO,DATANO,FINISH)

This routine pops off the top rule on the Express stack.

RULENO(I) Returns the number of the rule popped off the stack.

DATANO(I) Returns the value of DATANO (see HENTPAR above) which was current when the rule was pushed onto the top of the stack.

FINISH(L) This is a logical argument returning the value TRUE if the stack is empty, FALSE otherwise.

LESGDB **Format:** CALL LESGDB(RECORD,VALUE,RULENO,TARGET,FOUND)

This is perhaps the most important utility routine provided by Express. If a particular slot is specified, Express will return the present status of the slot in question, as recorded in the data base. If the slot value has already been found, it is displayed on the screen. In contrast, if the value is still unknown, the end user is given a corresponding message. To make it possible to determine the slot value, LESGDB also returns the number associated with the rule that should be started next. (This number is found in the code file in connection

with the definition of the current slot). This routine automatic checks whether it is the user or the system that have decided the value of the considered slot, and a message about this is given to the end user. If the slot is defined as a plot/table it will simply be displayed on the screen and naturally no value will be returned.

RECORD(I) The slot number.

VALUE(R) Returns the value of the slot if it has been found already.

RULENO(I) Returns the number of the rule to be started in order to determine the slot value.

TARGET(I) When a call is made to LESGDB to check on a slot value, this value is often needed for determining the value of another slot. On entry to LESGDB, the argument TARGET should indicate which slot, if any, depends on the present slot value in this way. The slot number specified is used to record relationships between slots. If the argument TARGET is equal to zero, no relationship is recorded.

FOUND(L) This argument is equal to TRUE on return from LESGDB if the slot value had already been determined, and is FALSE otherwise.

Example

Figure 3.14 shows the typical use of LESGDB. Depending on the value returned by the routine, we can continue to the correct address.

```
RECORD = 25 + (DATANO - 1)*JUMP
TARGET = 46 + (DATANO - 1)*JUMP
CALL LESGDB(RECORD,VALUE,RULENO,TARGET,FOUND)
IF ( .NOT. FOUND ) THEN
   ADDRESS = 1
   GOTO 999
ELSE
   IF (INT(VALUE) .EQ. 1) THEN
      ADDRESS = 2
   ELSE
      ADDRESS = 3
   ENDIF
ENDIF
```

Figure 3.14 One way to use the LESGDB routine.

This example also illustrates how the "jump code" can be used to set up the correct slot address. The value of the argument DATANO determines which one of the slots used for similar purposes should be considered.

TILGDB **Format:** CALL TILGDB(RECORD,VALUE,DISPLAY,TARGET)

This routine is used to record the value of a slot in the data base, provided that this value was determined inside the Fortran program itself and not by any external package. If the slot considered are a plot/table, this routine may not

be used. Instead the routine PLOTIN should be called (see below).
The arguments RECORD, VALUE and TARGET have the same function as in the routine LESGDB.

DISPLAY(L) This argument indicates whether the value being recorded in the data base should at the same time be displayed on the screen. In that case, this argument must have the value TRUE.

Example

In the set of rules comparing location parameters for two variables, the standard error of the coefficient of skewness is determined under the assumption that the variables are normally distributed. This quantity is defined as the square root of the following expression:

$$s^2 = \frac{6(n-2)}{(n+1)(n+3)}$$

In Fortran code this will be:

```
DISPLAY = .TRUE.
RECORD = 100
TARGET = 105
SDVALUE = SQRT(6*(NUM-2))/((NUM+1)*(NUM+3))
CALL TILGDB(RECORD,SDVALUE,DISPLAY,TARGET)
```

Figure 3.15 Making use of the TILGDB routine.

KODER

Format: CALL KODER(RECORD,CODE,VALUE,USERVA)

This routine reads the value of a slot from the data base, as well as certain codes associated with the slot. In contrary to LESGDB, this routine does not display any of the results on the screen. Nighter will there be any recording of connections between different slots when this routine is called.

RECORD(I) Specifies the record number of the slot to be considered.

CODE(I) The records in the data base include a separate field with a code indicating how the slot value was determined (located in position 5 on each record; see Section 3.6.3). This code is returned through the argument CODE, with the following possible values:

0 - Not yet found.
1 - Value extracted from the output of an external package.
2 - Value assigned by the program for the set of rules.
3 - Value specified by the end user.
5 - This slot contains a plot or a table which has been found.
8 - The slot contains the name of one of the variables used in

the analysis.
9 - Express was unable to determine this value.

VALUE(R) Returns the slot value assigned by the system.

USERVA(R) Returns the slot value assigned by the user.

LNKODER **Format:** CALL LNKODER(RECORD,CODE,VALUE,USERVA)

This routine serves the same purpose as KODER, but the arguments VALUE and USERVA represent strings of length 20 rather than real numbers (these strings are simply obtained by reading the numbers stored in the data base as character using the Fortran A20 format).

PARTEX **Format:** CALL PARTEX(RECORD,TEXT,LENGTH)

Each slot is assigned a separate record in the code file, indicating the type of the slot. This record also includes a brief name (occupying at most 40 positions) associated with the slot. The routine PARTEX returns this name.

RECORD(I) Indicates which slot should be considered.

TEXT(S40) Returns the name of the slot.

LENGTH(I) Returns the length of the name.

PLOTIN **Format:** CALL PLOTIN(RECORD,LINES,NUMBER)

Usually, plots and tables are extracted by Express from the output produced by an external package. Express takes the necessary action to store this information in the plot file, and to insert the corresponding access information into the data base. If a plot or a table in particular cases is generated inside the Fortran program representing the rule itself, the routine PLOTIN can be applied to store this information. It is only required that the plot or table is composed of an array of strings.

RECORD(I) The record number of the slot corresponding to the plot or table.

LINES(S80) This is the array of strings of length 80 containing the plot or table which should be stored. The number of strings must not exceed 50.

NUMBER(I) An integer between 1 and 50, indicating the total number of lines in the plot or table.

UTNULL **Format:** CALL UTNULL(RECORD,FIRST)

The end user may terminate a chaining of rules in Express at almost any time. It may sometimes be difficult to restart the set of rules in such situations, since certain rules may utilize particular "auxiliary slots" which should be assigned the code for unknown values before an exit is made. Under normal circumstances, this is taken care of by the rule itself, but interrupts given by the end user require special handling. The routine UTNULL makes it possible

to specify up to 12 slots which will be assigned the code for unknown values if the end user interrupts the chaining. A separate call to this routine is needed for each additional slot that should be treated in this way, usual such a call will be initiated at the same time as the slot is about to be used. It is also possible to erase the complete list used for this purpose before a slot value is stored.

RECORD(I) Slot number to be inserted in the list indicating which slots should be assigned the code for an unknown value when an interrupt is given by the end user.

FIRST(L) Must be equal to FALSE if the specified slot should be added to the already existing list. The value TRUE indicates that the previous list should be erased before the slot number is inserted.

ANTUT **Format:** CALL ANTUT(MAXI,MINI,CODE,NUMBER)

This routine returns information about how many variables can be used and how many are already in use in the currently active set of rules. For example, in the set of rules for regression analysis, at least two variables must be selected (the dependent and one independent variable), but it is possible to work with up to five variables in all (one dependent and four independent variables).

MAXI(I) Returns the maximum number of variables that can be selected in this set of rules.

MINI(I) Returns the minimum number of variables required.

CODE(I) This argument shows whether the variables have already been selected or not. The value zero indicates that selection has not yet taken place.

NUMBER(I) Indicates the number of variables that the end user has selected for the current session.

NOKDAT **Format:** CALL NOKDAT(NUMBER,ENOUGH,MAXI)

This routine is used to make certain that a sufficient number of variables have been selected in the current session before a particular analysis is carried out. Thus, in the example involving regression analysis, suppose that the end user has indicated that he wants to carry out regression analysis with three independent variables. The rule performing this analysis must first check that four variables have actually been selected (i.e., one dependent and three independent variables).

NUMBER(I) On Entry this indicates the number of variables needed to continue.

ENOUGH(L) Is equal to TRUE if at least NUMBER variables were selected for the current session.

MAXI(I) Gives the number of variables selected by the end user for the session.

HENTOPP **Format:** CALL HENTOPP(VARNO,NAME,OBS,MISSIN)

When the end user selects variables from the data storage for use in the current session, these will be assigned the numbers 1, 2, 3, etc. Each variable is associated with one record in the data base (starting at record 11; see Section 3.6.3), where the variable name, the number of observations and the number of missing values are stored. The routine HENTOPP will return this information for a specific variable.

VARNO(I) Gives the number of the variable to be considered.

NAME(S6) Returns the name of the variable.

OBS(I) Returns the number of observations.

MISSIN(I) Indicates how many observations have missing values.

HNTNR **Format:** CALL HNTNR(START,END,VARNO)

This routine will prompt the end user for specification of a particular variable number. An interval of admissible variable numbers can be specified when the routine is called. Consider once more as an example the set of rules for regression analysis. Suppose that the end user has selected one dependent and four independent variables, but only wants to carry out linear regression with one of the independent variables. This is an option offered in the basic problem menu, as shown in Figure 2.7. The rule for linear regression will then apply the routine HNTNR to let the end user select a particular one of the independent variables. Of course the dependent variable cannot be selected at this point, so the end user must only be presented with the variables numbered 2 to 5 as alternatives. This is achieved through the START and END arguments. If record no. 10 in the data base (used to store the number associated with the variable selected) already contains a non-zero entry, no action is taken.

START(I) Indicates on entry the number assigned to the first admissible variable.

END(I) Indicates on entry the number assigned to the last admissible variable.

VARNO(I) Returns the number of the variable selected by the end user. If this number is zero, no variable was selected.

SKR10 **Format:** CALL SKR10(DATANO)

Writes the number DATANO(I) to record no. 10 in the data base. This may be useful when we want a specified record to store the number of the variable currently being considered. For example, when a stream of commands is being set up for an external package, the variable name must often be inserted in particular commands. The number associated with the present variable can then be read from record 10 in the data base, and it is easy to determine the variable name as well.

VARFIL **Format:** CALL VARFIL(VARNO,RECORD,CODEREC,ALWAYS)

Two routines are available in Express, VARFIL and ENFIL, to produce data files which should read by external packages. One of these two files will usually be called before an external package is to be executed. The routine VARFIL generates a file with one column for each original variable.

VARNO(I) This is an array of integers of unspecified dimension, where the first integer informs the routine how many variables are to be used, and the following integers list the numbers associated with the separate variables. Thus, to generate a file containing the dependent variable and the first two independent variables in our regression example, the first element of VARNO must be equal to 3, followed by the three elements 1, 2 and 3.

RECORD(I) When the data file has been completed, VARFIL will store the name of the file in the data base. (The file name is given by Express). Usually we will assign space for this information at the beginning of the data base, following the records used to store information about the variables selected. For example, in the set of rules for regression we must use records 11-15 for recording information about the variables selected. (Some records may be empty if less than five variables were selected). We can then use as many records as required, starting at record 16, to store the necessary file names. There are two ways of indicating the correct address of the record that should be used in the data base. Assume that we want to store the file name in record 16, say. Then the argument RECORD can be assigned this value, and the name will automatically be stored there. Another indirect way of specifying the address is by assigning the argument RECORD either of the values 1 and 2. These numbers refer to two particular fields in the section containing codes for executions of packages in the code file (see Section 3.6.2). If RECORD is equal to 1, the record number contained in the first field is used, otherwise the record number in the second field is selected.

CODEREC(I) Each set of rules will usually involve several different executions of external packages (e.g., ten executions of SAS and three executions of Minitab), numbered 1, 2, 3, etc. The argument CODEREC indicates which execution the data belongs to. This information is needed in order to find the record used to store information about the file and to insert appropriate missing value codes depending on the package. Note that a file read in a SAS session may not be used in a Minitab session, because the two software packages apply different codes for missing values.

ALWAYS(L) Under normal circumstances, with ALWAYS equal to FALSE, the data file is not regenerated if it already exists (have been generated for use in another execution of an external package). This happens only if the argument ALWAYS is equal to TRUE.

ENFIL **Format:** CALL ENFIL(VARNO,RECORD,CODEREC,ALWAYS)
This routine is almost identical to VARFIL. The only difference is that ENFIL generates a data file with only two columns of numbers. The first column contains the separate variables stacked upon each other, whereas the second column contains a corresponding grouping variable generated by Express. The arguments are identical to those in VARFIL. However, ENFIL also generates two column names which are inserted into the same record as the file name.

In addition to the standard routines listed above, it is sometimes useful to take advantage of other routines which are normally used internally by Express itself. These are:

TREET **Format:** CALL TREET(TARGET,RECORD)
This routine can be used to record relationships between slots (see LESGDB).

TARGET(I) The slot value at this location has been determined on the basis of the value in the slot RECORD (in addition to other information, possibly).

RECORD(I) The slot referred to in the determination of the slot value of TARGET.

NYPOST **Format:** CALL NYPOST(RECORD,NEWVAL,SYSUSE,NEWCOD,WHICH)
The data base includes four different fields (or more precise five, but the first code contains only the record number) for storage of codes for a particular slot (see Section 3.6.3). In addition, two fields are reserved for storage of slot values, one for the value assigned by the system, and another one for the value assigned by the end user. The routine NYPOST can change the contents of these fields. Usually the code indicating the status of the slot, located in field no. 2 on the record (position no. 5; see KODER above) is the only one that should be modified in this way. **This routine must be used with great caution.**

RECORD(I) Indicates which slot the changes should be made for.

NEWVAL(R) Contains the value that should be assigned to the slot.

SYSUSE(I) If SYSUSE is equal to 1, the system value in the slot will be changed to NEWVAL. Otherwise the user specified slot value will be changed.

NEWCOD(I) Contains a new code to be inserted in one of the fields containing codes for the slot.

WHICH(I) Gives the number of the field to be changed (1, 2, 3 or 4), where the field we might be interested in changing is the first.

NYKODE **Format:** CALL NYKODE(RECORD,NEWCOD,WHICH)
This routine functions in the same way as NYPOST, with the exception that NYKODE can only change fields containing codes.

3.5.2 The main program for a set of rules

The main program responsible for the administration of the analysis forms an essential part of any set of rules. When a chaining of rules is started, the main program executes the first rule required. This rule will either be completed before the main program takes control once more, or a temporary exit is made from the rule. In both cases, the main program must retrieve a return address (statement number) in order to continue with the correct call to another rule. The basic structure of the main program for a particular set of rules is shown below:

```
      START
        OPEN NECESSARY FILES
        GET ARGUMENTS FROM THE MAIN PROGRAM IN EXPRESS
1       GET NO. OF RULE TO BE STARTED FROM THE STACK
10      GO TO STATEMENT BELOW CORRESPONDING TO THIS RULE NO.
100     CALL RULE NO. 1
             GO TO RETURN ADDRESS
200     CALL RULE NO. 2
             GO TO RETURN ADDRESS
          .
          .
          .
900     CALL RULE NO. 9
             GO TO RETURN ADDRESS
1000    EXECUTE AN EXTERNAL PACKAGE
999     SEND ARGUMENTS TO THE MAIN PROGRAM IN EXPRESS
        CLOSE NECESSARY FILES
      END
```

To give a more complete explanation, we consider an example. Figure 3.16 shows some parts of the main program in the set of rules comparing location parameters for two variables. After the necessary definitions of Fortran variables, we proceed with opening the files required during execution (CALL INITRE). As described in chapter 1, only the outer Express shell is active when an external package is executed. For this reason, during a chaining of rules, a temporary exit must sometimes be made from the main program in the set of rules, in order to leave sufficient memory space for the package. This structure requires some kind of communication between the main program in Express and the main program in the set of rules. This is taken care of by the routines HENTPAR and SENDPAR. At the beginning of the program for the set of rules, the arguments are obtained from the main program in Express by means of the routine HENTPAR. Similarly, before the program for the set of rules terminates, the arguments are returned by the routine SENDPAR. The next step taken in the program for the set of rules is to determine from the Express stack which rule should be activated. The LESSTB routine is marked with address 1 in the description above, and a jump

to this address will be necessary when a new rule is to be popped off the stack. If the stack is empty, the argument FINISH will be returned with the value TRUE, and it is time to return to the main program in Express.

```
$STORAGE:2
      PROGRAM REGLER
C
C  Set of rules for comparing location parameters for two variables.
C
      INTEGER*2 RULENO,DATANO,CODEREC,RECORD,JUMP,ADDRESS
      LOGICAL FINISH
C
C Initiate files, and get arguments from the main program.
C
      CALL INITRE()
      CALL HENTPAR(DATANO,RECORD,CODEREC,JUMP)
C
C  Pops the first rule off the stack.
C
1     CALL LESSTB(RULENO,DATANO,FINISH)
      IF (FINISH) THEN
         CODEREC = 0
         GOTO 999
      ENDIF
C
C  Mechanism for indirect addressing.
C
10    CONTINUE
      GOTO(101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
     $111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121,
     $122, 123, 124, 125, 126, 1000, 1000, 1000, 1000, 1000,
     $1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
     $1000, 1000,1000,1000,1000) RULENO
C
C  Main rule.
C
120   RULENO = 20
      CALL HOVREG(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
C
C  Deciding form of analysis.
C
121   RULENO = 21
      CALL REG21(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (125,126,123,10)ADDRESS
C
C  Tests for deciding form of analysis.
C
122   RULENO = 22
      CALL REG22(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
      .
      .
      .
C Are the variables normally distributed?
C
101   RULENO=1
      CALL REG1(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (10,102,105)ADDRESS
C
C  Have the variables equal variances?
C
102   RULENO=2
      CALL REG2(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (10,103,104)ADDRESS
C
C  t-pooled test.
103   RULENO=3
      CALL REG3(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
C
C  t-separate test.
104   RULENO=4
      CALL REG4(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
      .
      .
      .
C
C  Executes an external package.
1000  CODEREC=RULENO-30
999   CONTINUE
      CALL SENDPAR(DATANO,RECORD,CODEREC,JUMP)
      CALL STENGRE()
      END
```

Figure 3.16 The main program in the set of rules comparing location parameters.

The argument CODEREC, passed to the Express main program through the routine SENDPAR, indicates whether or not an external package is to be executed when the main program terminates in a set of rules. This is why CODEREC must be assigned the value zero, to finish the chaining of rules rather than executing a package, when the stack is empty and we want to return to the main menu of Express.

The GOTO statement at address 10 in the main program for a set of rules is essential in selecting the correct rule to be started next. As described previously, the LESGDB routine may return with a message indicating that the slot value of interest has not yet been found. In that case, the routine usually returns a rule number (RULENO) to show which rule the analysis should proceed with. This rule number will then be passed back to the main program of the set of rules. A transfer in the main program to address 10, with RULENO indicating the number of the rule to be executed next, will lead to a new jump to the appropriate routine call. Consider, for example, the call of the rule REG21 in Figure 3.16, corresponding to rule no. 21. This is the rule deciding which form of analysis should be selected. When a return is made from this rule, it is possible to jump to four different addresses in the program, depending on the value of the argument ADRESS. If ADRESS is equal to 1, 2 or 3, the program flow will continue at addresses 125, 126 or 123, respectively. These addresses represent different kinds of specific analyses (see Figure 3.5). If ADRESS is equal to 4, the address of the next statement is determined in an indirect way. A jump is first made to statement 10. In this case the argument RULENO returned by REG21 will indicate the number of the rule which should continue the chaining. As shown in Figure 3.5, rule number 22 is the only rule that can be started from rule number 21, except for rules concerned with specific analyses. Thus the argument RULENO must in this case contain the value 22. (This number will be returned when LESGDB is used inside the rule.) The 22nd number in the GOTO statement at address 10 is 122, which gives the address of the desired call. It may be surprising that the first rule in Figure 3.5 is numbered 20, the second 21, etc., and that rule number 1 only occurs later in the analysis. This is because this particular set of rules was modified and developed further after the first version was constructed. To understand how the chaining is carried out in general, it can be useful to compare the strategy map in Figure 3.5 with the main program in Figure 3.16.

When all calls to the relevant routines have been defined in the main program for a set of rules, some statements are needed for execution of external packages. As indicated above, if the argument CODEREC has a non-zero value when the program for a set of rules terminates, Express should execute an external package. In that case we simply jump to the end of the program at address 1000, assign the correct value to CODEREC and return to the main program of Express. In a particular set of rules there may be several different ways of running external packages. For example, one set of commands may be needed for computing values related to problems concerning normality of a variable, another set of commands may be used for finding the *p*-value for homoscedasticity, and still another set may be needed for performing a *t*-test. The different sets of commands are numbered 1, 2, 3, etc. The argument CODEREC must contain the number (1,2,3...) corresponding to the package execution we

want. In the main program for comparing location parameters, we have regarded the different situations requiring a package execution as rules with rule numbers 31, 32, 33, 34,... . This means that we must let CODEREC = RULENO - 30 to get the correct value.

In Figure 3.17 an attempt has been made to show the flow between the main program and the different routines corresponding to rules. The main program is entered at statement 1, and control passes to statement 10 to jump to the correct subroutine call. In this example, rule no. 1 is activated first, but it is assumed that the execution of this rule cannot be completed. Following a temporary exit from rule no. 1, a return is made to the main program with information about where to continue. Control first passes back to statement no. 10, and then to the statement for rule no. 2, as indicated by rule no. 1. The chaining will now continue in the same way until the last rule has been removed from the stack.

As described above, the rule from which a temporary exit is made can alternatively indicate that a jump should be made to statement 1000, resulting in the execution of an external package. If this happens, it is always triggered by the need to determine another unknown slot value. When the package execution has been carried out, the top rule on the stack will be restarted and the process will continue at the same point, or at least within the same rule, as before the package execution.

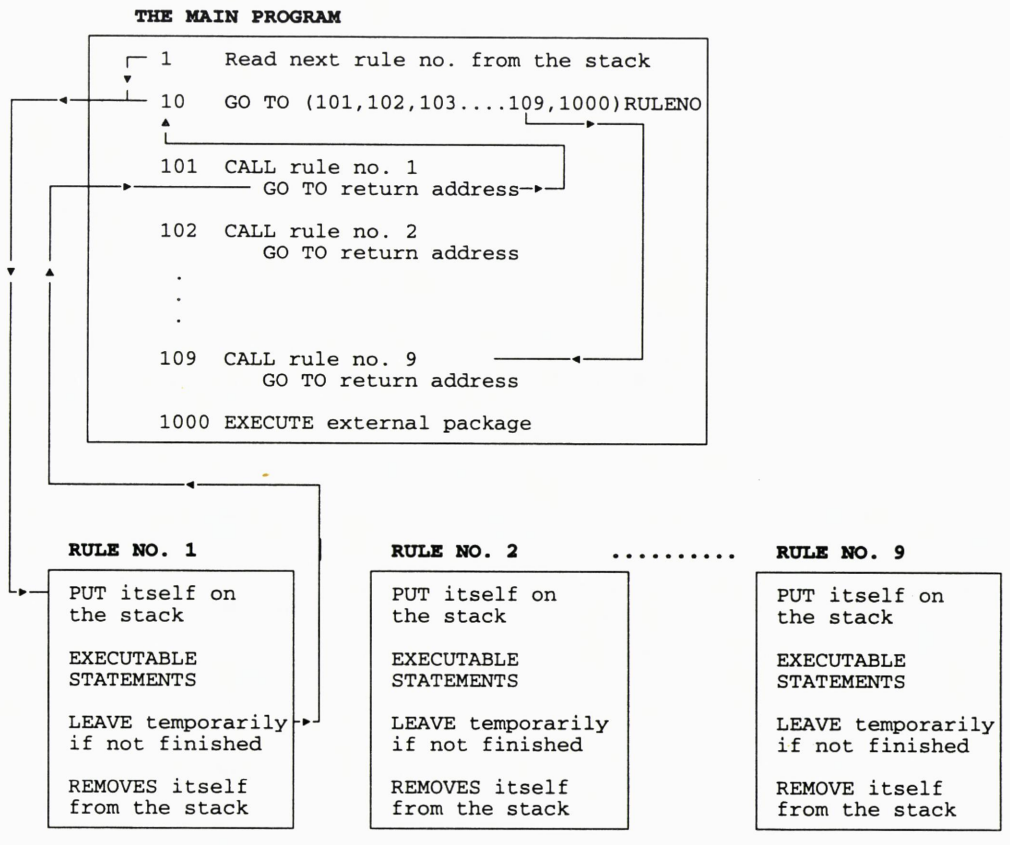


Figure 3.17 Description of the first steps in a chaining of rules.

3.5.3 Subroutines representing rules

The first part of a rule includes general definitions and codes for putting the rule number onto the stack. This is followed by a main part with executable statements and finally a part showing how an exit can be made from the rule. One possibility is to leave the currently active rule on a temporary basis if it is decided that results from other rules must be found first (perhaps including the execution of an external package). The second way of leaving a rule applies when the rule has been carried out completely. In this case, the rule must be popped off the stack, in contrast to the case when the rule is left temporarily. A subroutine representing a particular rule will thus have the following general structure:

```
      START
         PUT ITS OWN RULE NO. ON TOP OF THE STACK
         EXECUTABLE STATEMENTS
         GO TO 999 IF RULE HAS NOT BEEN COMPLETED
         REMOVE ITSELF FROM THE STACK
999      CONTINUE
      END
```

Four parameters are usual used for each subroutine constituting a rule, but if necessary other may be used as well. The four arguments that must be passed are: RULENO, DATANO, RECORD and ADDRESS. Since the main program in the set of rules must at any moment know the number of the active rule, the argument RULENO must be passed to the rule and updated. The argument DATANO, usual keeping track of the variable considered at the present time must also be updated if a change inside a rule is made regarding the variable considered. As described above, each rule returns an address, giving the answer to where to continue the chaining of rules. This address in passed through the argument ADDRESS. The last argument RECORD shall contain the number of the last slot considered in the rule. This argument is not as important as the other three. It is only used to get access to the correct slot name, which will be displayed for the end user if a rule is temporary exited to execute an external package which main aim is to find the value of this particular slot.

Let us once more look at some examples from the set of rules comparing location parameters. The rule listed in Figure 3.18 is rule number 1 in this set, and its purpose is to check for normality for the two variables involved. At entry, it pushes itself onto the stack, as all rules must do. Then, using the LESGDB routine, the rule inquires into the normality for the first variable only. If this information has not yet been found, the argument FOUND will be FALSE, and RULENO will contain the value 10, which is the rule to be started in order to decide on normality for separate variables (see Figure 3.5). The correct value of RULENO is found in the code file. The argument ADDRESS is given the value 1 if the slot value is unknown. A shown in Figure 3.16, the statement after the call to the rule REGEN is a jump statement depending on the value of ADDRESS, to one of the statements 10, 102 or 103. Thus, if a return is made to the main program with ADDRESS equal to 1, control passes to statement number 10, where the value of the RULENO argument determines that control should pass to the call to rule number 10 in statement 110.

When rule number 10 has been completed, the answer to the question about normality for the first variable will be known. Rule REGEN will then be restarted. Since the first call to LESGDB will produce a definite decision concerning the normality of this variable, the rule will continue to examine the normality of the second variable. This is done in exactly the same manner as for the first variable. We notice that the routine SKR10 is used to write the variable number in question (DATANO) to the data base. This is necessary in order to indicate to rule number 10 which variable should be considered. This number will of course be available also through the argument DATANO. The main reason the number of the variable is written to the database is that when control languages for external languages shall be set up the number of the variable must be contained in the data base.

When decisions have been made about normality for both variables, the rule REGEN must return the correct address to the main program. If both variables are normally distributed, we must continue with rule number 2 (see Figure 3.5) for deciding whether the variables have equal variances. If the conditions for normality do not hold, rule number 5 is started, to determine whether the variables are lognormally distributed. The corresponding values of the argument ADDRESS are 2 and 3, which lead to the direct addresses 102 and 105 in the main program.

```
      SUBROUTINE REGEN(RULENO,DATANO,RECORD,ADDRESS)
      INTEGER*2 RULENO,DATANO,RECORD,ADDRESS
      REAL VALUE1,VALUE2
      LOGICAL FINISH,FOUND
C
C  Is variable 1 normally  distributed?
C
      CALL PAASTB(RULENO,DATANO)
      RECORD=34
      DATANO=1
      CALL LESGDB(RECORD,VALUE1,RULENO,51,FOUND)
      IF (.NOT. FOUND) THEN
         CALL SKR10(DATANO)
         ADDRESS = 1
         GOTO 999
      ENDIF
C
C  Is variable 2 normally distributed?
C
      RECORD=134
      DATANO=2
      CALL LESGDB(RECORD,VALUE2,RULENO,51,FOUND)
      IF (.NOT. FOUND) THEN
         CALL SKR10(DATANO)
         ADDRESS = 1
         GOTO 999
      ENDIF
C
C  Gives the correct address to the main program, depending on whether both
C  variables are normally distributed or not.
C
      IF(INT(VALUE1) .EQ. 0 .OR. INT(VALUE2) .EQ. 0)   THEN
         ADDRESS = 3
      ELSE
         ADDRESS = 2
      ENDIF
      CALL LESSTB(RULENO,DATANO,FINISH)
999   CONTINUE
      END
```

Figure 3.18 Rule checking the distribution of the two variables in the set comparing location parameters.

Figure 3.19 shows another rule, REGTRE, from the set comparing location parameters. The purpose of this rule is first to determine the *p*-value from a *t*-test with pooled variances, and second to draw the actual conclusion regarding the rejection of the null hypothesis of identical population means. REGTRE has the same basic structure as the rule REGEN described above, but REGTRE may initiate the execution of an external package if the value of record (slot)

37 is unknown. The argument RULENO will now indicate which set of commands should be run. This argument must be a value above 30 (since the "rules" for executing packages start at number 31), which in this particular case is 33. This number will be returned by the LESGDB subroutine. As described in section 3.5.1 this routine first searches the data base to decide if the value of the slot is already found. If not LESGDB will inquire the knowledge based where a rule number, 33 in this case, will be found. Before the external package can be executed, a data file must be created for the package with the two variables to be compared. In this case the argument DATANO is set equal to 3. This code indicates that the analysis should involve both variables (in contrast to the situations with DATANO equal to 1 or 2). To generate the file including both variables, we utilize the routine ENFIL. For this purpose, we first set the argument VARNO(1) equal to 2 (the number of variables to be included in the file). The two arguments VARNO(2) and VARNR(3) indicate the numbers assigned to the particular variables selected, in this case 1 and 2. Then a call is made to ENFIL to generate the file, before a temporary exit is made to the main program for package execution. Subsequently the rule will be restarted, and it will then determine the final conclusion of the test on the basis of the *p*-value. The routine TILGDB is used to write the this conclusion into the data base in record number 51.

```
      SUBROUTINE REGTRE(RULENO,DATANO,RECORD,ADDRESS)
      INTEGER*2 RULENO,DATANO,RECORD,ADDRESS,VARNO(3)
      REAL VALUE1
      LOGICAL FINISH,UNTRUE,FOUND,TRUEEX
      TRUEEX = .TRUE.
      UNTRUE = .FALSE.
C
C  Finds results of t-pooled test.
C
      CALL PAASTB(RULENO,DATANO)
      RECORD=37
      DATANO=3
      CALL LESGDB(RECORD,VALUE1,RULENO,51,FOUND)
      IF ( .NOT. FOUND ) THEN
         VARNO(1) = 2
         VARNO(2) = 1
         VARNO(3) = 2
         CALL ENFIL(VARNO,1,RULENO - 30,TRUEEX)
         CALL SKR10(DATANO)
         ADDRESS = 2
         GOTO 999
      ENDIF
      RECORD=51
      IF (VALUE1 .LT. .10) THEN
         CALL TILGDB(RECORD,0.0,UNTRUE,0)
      ELSE
         CALL TILGDB(RECORD,1.0,UNTRUE,0)
      ENDIF
      CALL LESSTB(RULENO,DATANO,FINISH)
      ADDRESS = 1
999   CONTINUE
```

Figure 3.19 Rule for executing a t-test.

Example

Consider the main program in the set of rules performing regression analysis, shown in Figure 3.20. This set includes only nine routines representing different rules, and only five different sets of commands for executing a package. This makes the main program quite easy to understand. The rules REGEN, REGTO, REGTRE and REGFIR involve regression analysis with one, two, three or four independent variables, respectively. The rule REGFEM handles all independent variables separately in succession and carries out a linear regression analysis for each. The routine REGSEKS starts with all independent variables available, performs

regression analysis and checks for significance of the different variables in the model. The model is then refitted without the non significant variables. This process is continued until all independent variables included in the model are significant. The routine REGREG represents the rule that actually executes a particular regression analysis. All other rules referred to above simply make the appropriate preparations for the analysis to be performed by REGREG. When REGREG has finished, the rule that invoked REGREG will again be in charge, carrying out the subsequent analysis. The two last rules, REGATT and REGNI, are used to perform stepwise regression analysis as defined under the stepwise option offered by SAS (in PROC REG).

```
$STORAGE:2
      PROGRAM REGRESJON
C
C
      INTEGER*2 RULENO,DATANO,CODEREC,RECORD,ADDRESS,JUMP,IDATANO
      LOGICAL FINISH
C
      CALL INITRE()
      CALL HENTPAR(DATANO,RECORD,CODEREC,JUMP)
1     CALL LESSTB(RULENO,IDATANO,FINISH)
      IF (FINISH) THEN
         CODEREC = 0
         GOTO 999
      ENDIF
C
10    CONTINUE
      GOTO(100,200,300,400,500,600,700,800,900,1000,1000,1000,
     $1000,1000) RULENO
C
100   RULENO = 1
      CALL REGEN(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,10)ADDRESS
C
200   RULENO = 2
      CALL REGTO(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,10)ADDRESS
C
300   RULENO = 3
      CALL REGTRE(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,10)ADDRESS
C
400   RULENO = 4
      CALL REGFIR(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,10)ADDRESS
C
500   RULENO = 5
      CALL REGFEM(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,100)ADDRESS
C
600   RULENO = 6
      CALL REGSEKS(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,100,200,300,400)ADDRESS
C
700   RULENO = 7
      CALL REGREG(RULENO,DATANO,RECORD,JUMP,ADDRESS)
      GOTO (1,10)ADDRESS
C
800   RULENO = 8
      CALL REGATT(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
C
900   RULENO = 9
      CALL REGNI(RULENO,DATANO,RECORD,ADDRESS)
      GOTO (1,10)ADDRESS
C
C     SETTER I GANG ET PROGRAM
C
1000  CODEREC = RULENO-9
999   CONTINUE
      CALL SENDPAR(DATANO,RECORD,CODEREC,JUMP)
      CALL STENGRE()
      END
```

Figure 3.20 The main program in the set of rules performing regression.

In contrast to the rules comparing location parameters, the regression set of rules is not restricted to a fixed number of variables in each session. Thus it is possible for the end user to go through one session with only two independent variables and then restart the same session selecting, say, four independent variables. Such situations require special care in the

construction of rules. Let us suppose that the end user has selected one dependent and four independent variables in a particular session. The end user then requests that a regression analysis be carried out with two independent variables only. Thus the routine REGTO must be started. Figure 3.21 shows the source code of this routine. When the rule has pushed its own rule number onto the stack, the first step taken is to check whether a sufficient number of variables are available. This particular rule requires three variables, so the routine NOKDAT is called (with the first argument equal to 3), and if on return the argument NOK is false, the chaining is interrupted. The next step is to prompt the end user to select which pair of independent variables among those available should be considered. Since this rule may be interrupted on a temporary basis and later restarted, an auxiliary slot (record 395) must be set equal to 0 if this selection has not been made and to 1 if the selection has taken place. In addition, four auxiliary slots (records 391, 392, 393 and 394) are needed for indicating whether the independent variables 1, 2, 3 or 4 have been selected in the regression analysis.

```
      SUBROUTINE REGTO(RULNO,DATANO,RECORD,JUMP,ADDRESS)
      INTEGER*2 RULNO,DATANO,RECORD,JUMP,ADDRESS,LOKKE,HELP,DATEN,DATTO,MAKSI
      REAL VALUE,SYST,MAAL
      LOGICAL FINISH,CORRECT,UCORECT,NOK,FOUND
C
      CORRECT = .TRUE.
      UNCORECT = .FALSE.
      CALL PAASTB(RULNO,DATANO)
      CALL NOKDAT(3,NOK,MAKSI)
      IF (.NOT. NOK) GOTO 777
      CALL KODER(395,KODE,VALUE,SYST)
      IF (KODE .EQ. 0) THEN
         CALL UTNULL(395,CORRECT)
         CALL NULLDAT()
         DATANO = 0
         CALL SKR10(DATANO)
         CALL HNTNR(2,5,DATEN)
         IF (DATEN .EQ. 0) GOTO 777
         HELP = 389 + DATEN
         CALL NYKODE(HELP,1,1)
11       DATANO = 0
         CALL SKR10(DATANO)
         CALL HNTNR(2,5,DATTO)
         IF (DATEN .EQ. DATTO) GOTO 11
         IF (DATTO .EQ. 0) GOTO 777
         HELP = 389 + DATTO
         CALL NYKODE(HELP,1,1)
         CALL NYKODE(395,1,1)
         IF (MIN0(DATEN,DATTO) .EQ. 2) THEN
            DATANO = DATEN + DATTO - 3
         ELSE
            DATANO = DATEN + DATTO - 2
         ENDIF
      ENDIF
      JUMP = 15
      RECORD = 69 + (DATANO - 2)*JUMP
      MAAL = RECORD
      CALL NYPOST(404,MAAL,1,1,1)
      CALL LESGDB(RECORD,VALUE,RULNO,255,FOUND)
      IF ( .NOT. FOUND ) THEN
         SYST = 2
         CALL TILGDB(400,SYST,UNCORECT,0)
         IRECORD = 60  + (DATANO - 2)*JUMP
         CALL NYPOST(401,IRECORD,1,0,0)
         CALL SKR10(DATANO)
         ADDRESS = 2
         GOTO 999
      ENDIF
777   CALL LESSTB(RULNO,DATANO,FINISH)
      ADDRESS = 1
      CALL NYKODE(395,0,1)
999   CONTINUE
      END
```

Figure 3.21 Rule for regression with two independent variables.

As shown in Figure 3.21, the routine KODER is used to determine the current value of the auxiliary slot indicating whether variables have been selected. If this value is equal to 0, the end user will be prompted for selection. The first step is to call the routine UTNULL with the argument RECORD equal to 395. This indicates that slot number 395 will be assigned the

value 0 (for unknown) if the end user should interrupt the chaining of rules. Then the routine NULLDAT is used for erasing any information in records 391 to 394 (by assigning the value 0 to the code in position 5). Now the selection of variables can start using the routine HNTNR. This routine will, depending of the selection made by the end user, return a number between 2 and 5. When variables have been selected, the routine NYKODE is used to record this selection in the corresponding auxiliary slots. Finally, when both independent variables have been selected, the rule constructs a special argument value DATANO corresponding precisely to regression analysis with the variables in question. On the basis of this number, the correct slot (RECORD) is found which we will investigate further. This particular slot shall if found contain the regression equation for the specified model. If it is not found (and LESGDB returns with the argument FOUND equal to FALSE), the rule executing the general regression analyisis must be started. Before this is done, the auxiliary slot in record 400 is assigned the value 2, indicating the number of loops (equal to the number of independent variables) that the REGREG rule must go through. Another auxiliary slot is used to pass the number of the first record the REGREG routine shall investigate. This is a slot defined as the first coefficient in the regression analysis. In this example it is quite obvious how auxiliary slots must be introduced to pass essential information from one rule to another.

3.5.4 Compiling and linking the program for a set of rules

The source file (or files) of the program for a set of rules must compiled and linked with the libraries containing the necessary routines for Express. If we use the Microsoft Fortran, version 4.1, the command starting the compilation can be:

FL /FPc /c <*name of file to be compiled*>

This will produce an object code file with the same name as the source file, but with the file name extension OBJ. To create an executable program (in an EXE file), we must link the object file to the necessary libraries with the following command:

LINK <*Object filename*>,<*Executable name*>,,\lib\nyregler \lib\nyfell \lib\tilleg

This is based on the assumption that the libraries have been copied to the directory c:\lib.

3.6 FILES FOR A SET OF RULES

3.6.1 Initiation of files and installation of a new set of rules

When an executable file has been generated representing the actual rules, corresponding information must be inserted into the code file for the set of rules. Certain codes must also be set in the system file and the data base. Express provides a particular program for installing a new set of rules and initiating files needed by this set. Before this program is executed, it is recommended that a new directory should be created for these files. Then let this directory be the active one and simply specify file names without paths when the installation program inquires about files. It should be noted that the corresponding EXE file with program code must already exist when a new set of rules is installed. This program must not necessarily represent the final version of the rules, and debugging and further development can still be carried out. It will often be convenient to start inserting information into the code file long before the rule program has been fully developed. In such situations it may be necessary to refer to a dummy program for the rules. In principle, any existing EXE file may be specified if the correct name is inserted later into the system file (see Section 3.4.1). By contrast, when the installation program prompts for names of other files to be created, such as the data base and the code file, these names will not be accepted if the corresponding files already exist.

To execute the installation program, simply type \EXPRESS\NYSETT on the DOS command line. On entry, the system will state which number will be associated with the new set of rules to be installed. The next step is to assign a name to this set which will be presented to the end user in the menu for selecting a set of rules. The name should describe the statistical area dealt with. The installation program then prompts for the name of the EXE file containing the actual rules. These specifications will be inserted into the first two sections of the system file. The names of other files associated with the set of rules are inserted into a particular subsequent section of the same file (see Section 3.4.1). These files are:

Files	Description
The data base	This is the file in which most of the information collected by Express is stored. During installation, the user must assign a particular name to the data base, and furthermore, he must specify how many records the data base should include.
The code file	This is the most important file in the installation of a set of rules. The installation program will prompt the user for information to be included in this file, such as the number of streams of commands for external

packages, the number of possible commands for external packages, and the number of possible search keys to be used. The numbers specified need not be completely accurate and it may be advisable to reserve some additional space. These numbers can be changed later (see Figure 3.10 in the section "Editing the system file").

Other files — The user must assign separate names to the file for storage of plots and tables, the file collecting output from external packages, the log file, and finally, the file recording relationships between slots.

The new set of rules will now be included in the general Express menu for selecting such sets (shown in Figure 2.1). To edit the files belonging to the new set, start Express and select precisely this set in the set of rules menu. Now the editor can be entered. If errors occur during the installation so that the set of rules in question does not appear on the menu, any new files created must be erased and the installation restarted.

3.6.2 Editing the code file

If the option designated "Editing of existing code file" is selected in the main menu of the Express editor, the code file of the currently active set of rules will be opened for editing. If the user wants to edit the code file associated with another set of rules, the appropriate set should first be made the active one in the set of rules menu, and the editor must then be restarted. Each section of the code file is represented by a separate item in the editing menu (Figure 3.22). The information stored in each such section is described below.

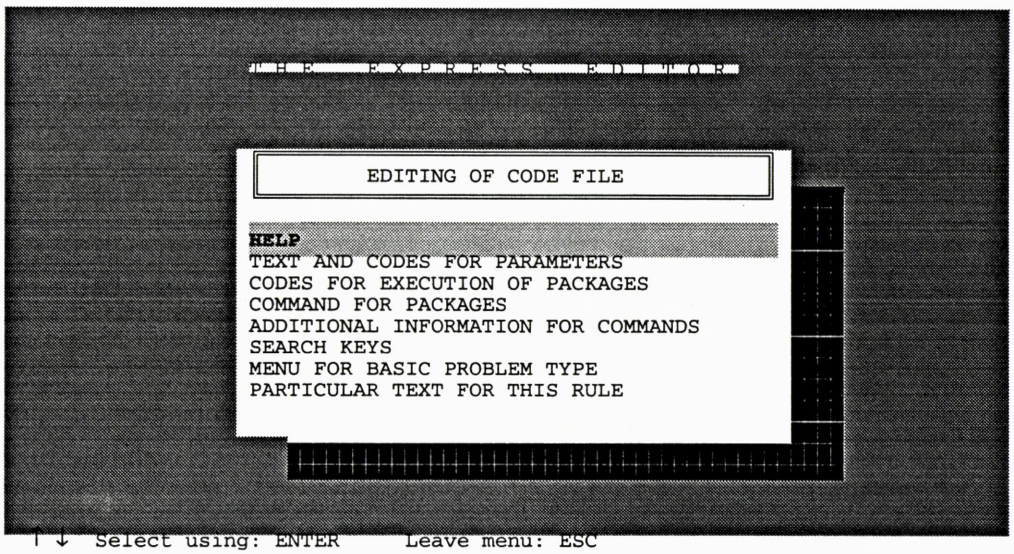


Figure 3.22 Menu for selecting part to edit in the system file.

Text and codes for slots

This section is used to define all slots considered in the analysis. Each record represents a separate slot and is divided into 10 fields allocated to particular types of information:

Field 1 (4 positions, right adjusted): The first field should always be coded with the integer 1 (which simply indicates that the definition of the slot covers a single record).
Field 2 (4 positions, right adjusted): In field no. 3 a name will be assigned to the slot as a string. Field no. 2 should indicate the length of this string (or rather, the number of characters plus 1).
Field 3 (40 positions, left adjusted): Is used to define a slot name, consisting of at most 40 characters.
Field 4 (2 positions, right adjusted): Express distinguishes between different types of slots. Field no. 4 indicates the type of the slot currently being considered. The admissible slot types are:

Type	Description
1	The slot contains a number which can be determined by computation in the rule program or by an external package.
2	The slot contains the response to a question which can only be answered by yes or no. This will often represent a conclusion reached by a relatively complex chain of reasoning. The slot value is determined by the rule program or an external package.
3	As for type 1, the slot contains a number, but in this case the end user will be requested to provide the correct slot value before the system makes any attempt to determine the value.
4	As for type 2, the slot represents a response answered by yes or no, but as for type 3, the end user will first be prompted for the slot value.
5	The slot represents a block of information (a plot or a table). Slots of this type will not be considered in the analysis if the system variable GRAPHICS is turned off (see Section 2.6.1).
6	As for type 5, the slot represents a block of information, but the block will always be displayed, regardless of the state of the system variable GRAPHICS.
7	The slot is similar to type 6, but in addition the block of information will be written to the log file.

This section of the code file may also include records which have been left empty or are used for other purposes, provided that field no. 4 is coded with the number 0 or is blank. It is recommended that the code -1 should be inserted into field no. 4 in the first record following the definition of the last regular slot. This code is interpreted by Express as a termination sign when Express displays a list of slots on the screen. As shown in the example involving

regression analysis, a set of rules may also incorporate certain auxiliary slots. These should not be included in the list displayed by Express. To make such auxiliary slots invisible to the end user, they should be defined after the termination code -1.

Field 5 (4 positions, right adjusted): This field must contain the number of the rule to be started in order to find the value of the current slot. This number will be returned by the RULENO argument in the LESGDB routine (Section 3.5.1) if the slot value is unknown when LESGDB is called.

Field 6 (4 positions, right adjusted): A more comprehensive explanation of each slot may be stored at the end of the code file (in the section corresponding to the heading "Particular text for this rule"). Field no. 6 must contain the record number where the explanation is located. If no explanation is available, the the field should be empty.

Field 7 (1 position): Express makes a distinction between integers and real (floating point) numbers when slot values are displayed. Field no. 7 must be contain the number 1 if the slot value should be treated as an integer and 0 if it represents a real number.

Field 8 (4 positions, right adjusted): The last three fields are used to control legal values in a slot. Field no. 8 indicates the lower limit. In this connection, the particular code -999 represents minus infinity.

Field 9 (4 positions, right adjusted): Indicates the upper limit of the legal values. The particular code 9999 represents plus infinity.

Fields 8 and 9 can also be coded in a particular way so that the user will be unable to change the slot value or to assign the code for unknown values to the slot in question. These particular codes are:

Field 8	Field 9	Description
-777	7777	This selection turns off the opportunity for the end user to assign the code for unknown values to this slot.
-888	8888	Under this option, the user is not allowed to change the present value of the slot.
-777	8888	Neither of the two functions will be available to the end user.

Field 10 (13 positions, left adjusted): This field is used to store a string with information about the admissible values for the slot. The text will be displayed to the end user.

Example

Figure 3.23 provides some examples of slot definitions in the code file. The sequence of regular slots known to the end user is terminated by the code -1 in field no. 4. All subsequent slots are regarded as auxiliary slots, maintained internally by the set of rules. It should be noted that field no. 5 does not always indicate the number of a rule to be started. This is because certain slot values will usually be found at an earlier stage by rules activated in order

to determine other slot values. The rule number must be supplied in field no. 5 only for slots which are referred to explicitly by the routine LESGDB.

	1	2	3	4	5	6	7	8	9	10
21	1	35	Coeff. for X1 in linear regression	1	10		0	-888	8888	No changes
22	1	38	p-value for X1=0 in linear regression	1	10		0	0	1	[0,1]
23	1	40	Is X1 significant in linear regression?	2				0	1	{0-No, 1-Yes}
24	1	26	Constant in model with X1	1			0	-999	9999	All values
25	1	40	Regressiontable for linear reg. with X1	5						
26	1	39	R-square for linear regression with X1	1			0			
27	1	39	The regression equation (Y = a + bX1)	7	7					
31	1	35	Coeff. for X2 in linear regression	1	10		0	-888	8888	No changes
32	1	38	p-value for X2=0 in linear regression	1	10		0	0	1	[0,1]
33	1	40	Is X2 significant in linear regression?	2				0	1	{0-No, 1-Yes}
34	1	26	Constant in model with X2	1			0	-999	9999	All values
35	1	40	Regressiontable for linear reg. with X2	5						
36	1	39	R-square for linear regression with X2	1			0			
37	1	39	The regression equation (Y = a + bX2)	7	7					
.										
.										
.										
270	1	33	Coeff. for the third X-variable	1			0	-888	8888	No changes
271	1	34	p-value for the third X-variable	1			0	-888	8888	No changes
272	1	32	Coeff. for the fourth X-variable	1			0	-888	8888	No changes
273	1	33	p-value for the fourth X-variable	1			0	-888	8888	No changes
274	1	22	Constant in the model	1	14		0	-888	8888	No changes
275				-1						
391	1		Is X1 in use?							
392	1		Is X2 in use?							
393	1		Is X3 in use?							
394	1		Is X4 in use?							
.										

Figure 3.23 Some slot definitions in the code file for the set of rules executing regression analysis. The leftmost column shows record numbers. Field numbers are indicated at the top.

Commands for packages

This is the next section inserted into the code file. It must include all commands used in executing external packages. The appropriate set of commands will be submitted by Express to the package in question and interpreted according to the rules which apply to each particular external software system. Each record in this section of the code file represents a separate command. By means of a special coding system, additional information may be inserted, and certain parts of the commands may also be deleted. Further directives needed in such cases are given by the next section in the code file.

The different records are grouped in a natural way to form streams of commands transmitted to the external package at the same time. The same general command streams may be used in several different situations. The specifications in the subsequent section of the code file entitled "Codes for execution of packages" indicate which records in the current section should actually be included in any particular command stream.

The three fields in each record of the current section are:

Field 1 (4 positions, right adjusted): If the command given in field no. 3 is already complete, field no. 1 must contain the integer 1. If this is not the case, the number given in field no. 1 must be an address (record number) in the next section. At this address, information must be supplied about the text that should be inserted into the empty space in

the command.

Field 2 (4 positions, right adjusted): This field may contain one out of three possible codes. The codes 1 and 2 indicate that we are dealing with a command from which certain parts can be deleted. On the other hand, if the complete command should always be retained, the code must be equal to 0 (or the field can be empty). As an example, consider the following command:

```
INPUT %0 %1 %2;
```

This is a SAS command for reading three variables. Each percentage sign indicates that additional information must be inserted, in this case specifying separate variable names. If we want to use this general command in situations when the number of variables is not fixed in advance, it may be necessary to omit one or two of the variables from the command. For example, suppose that two commands should appear in the following way when they have been completed:

```
i) INPUT YVAR X1VAR X2VAR;
ii) INPUT YVAR X1VAR;
```

This means that the reference to the last variable has been dropped in the second command. To decide which parts of the general command should be removed during this process, Express can check for the state of a particular slot for each part. Any part of a general command that can potentially be removed, should be included in curly brackets in the general specification. Thus the general command inserted into the code file must appear as follows:

```
INPUT %0 {%1} {%2};
```

Here the two parts at the right may be omitted when the command is completed. The description of the next section explains the difference between codes 1 and 2. The length of field no. 2 is 4 and the code must be right adjusted.

Field 3 (72 positions, left adjusted): This field contains the actual general command, perhaps in an incomplete form. A command can include several unknown parts. Each such part must be marked by the code *%<number>*, where the *<number>* is an integer that must be added to the address given in field no. 1. For example, if field no. 1 contains the address 200 and *<number>* is equal to 0, the additional information will be in record 200. In contrast, if *<number>* is equal to 2, additional information will be found in record 202.

Additional information for commands

As indicated above, this section contains information used to complete particular commands given to external packages. Each record indicates to Express how the system should react when a particular missing piece of information is encountered in a command. The section

includes references to records in specified files which can be read in order to find the information needed. Basically, two different operations may be carried out when a record is completed by including additional information. First, names of variables and files may be inserted. Records in the current section contain addresses (record numbers), mainly in the data base, where such information is stored. Second, it may be indicated in the current section that the state of a particular slot, usually an auxiliary slot, should be checked. A particular code (the integer in position 5) in the data base indicates whether the slot value has been found or not. The system may then decide, on the basis of this information, whether an unknown part of a command should be deleted.

Field 1 (4 positions, right adjusted): The first three fields are used to find the address (record and file number) where additional information is stored. Field no. 2 indicates the file number, whereas field no. 3 gives the record number. Field no. 1 can be used indirectly to add a certain number of records to the address given in the subsequent fields. The number given in field no. 1 will be regarded as an address in the data base. In practice, field no. 1 should either be empty or contain the number 10. In the latter case, Express will find the current variable number (stored in record 10 in the data base by the standard routine SKR10) and add this number to the record number specified in field 3. This will provide the final address for the additional information required. (If field no. 1 indicates a record different from no. 10 in the data base, it should be noted that the number read must be located in positions 6 and 7.).

Field 2 (4 positions, right adjusted): Contains the number assigned to the file to be considered. In practice, only the data base (file no. 11) has been used for this purpose in the applications considered so far. It is not recommended that this practice should be changed, although in principle any file number listed Section 3.1.1 can be specified.

Field 3 (4 positions, right adjusted): Gives the record number where additional information can be found (possibly modified as described under field no. 1).

Field 4 (15 positions, left adjusted): When the information is read from the specified record, a particular integer is read first, with a code that must differ from 0 to indicate that the correct information has been found. The additional information is read into a string. Field no. 4 supplies a Fortran format for this read operation involving two variables. For example, the format (4X,I1,A6) indicates that a jump should be made to position no. 5 on the record, a one digit code should be read at this location, and finally a string consisting of six characters should be read from positions 6-11. If the number read from position no. 5 is equal to 0 (indicating that the additional information searched for has not yet been determined by Express), two possible actions may be taken. Which one is selected, depends on the code in field no. 2 in the previous section containing the commands, as shown by the list below:

Code	Description
0	A zero indicates that the missing part of the command must be inserted. If the information cannot be found at the address considered, the system should ask the

	end user for assistance. In this situation field no. 7 may contain the address (record number) of a text formulated as a question that can be given to the end user. If the end user is unable to answer this question, the chaining of rules will be terminated.
1	This indicates that the missing part of the command can be deleted if the information that should be included is unavailable.
2	This code is similar to code 1, but any additional information found is inserted into the command only in certain circumstances, depending on the contents of a second record in the file considered (normally the data base). The corresponding address (record number) must be given in field no. 5 (in the current section), with the appropriate format in field no. 6. Only two possible codes 0 and 1 are allowed at the second location. With code 1, the same rules apply as above (when the first code read is equal to 1). With the code 0 at the second location, the incomplete part of the command will be deleted in any case, even if the additional information needed was found at the first address.

Field 5 (4 positions, right adjusted): If field no. 4. contains the code 2, field no. 5 should indicate the record number of the second location to be checked.
Field 6 (15 positions, left adjusted): Contains the format to be used in reading the code at the second location, defined in the same general way as the format in field no. 4. If field no. 6 is left blank, the same format (or the first part of this format) is used as specified in field no. 4.
Field 7 (4 positions, right adjusted): Contains an address in the code file to the question which is given to the user if the system itself is unable to find the information required.

Example with comparison of location parameters

We consider two relatively simple sets of commands for execution of the external package SAS. They form minor parts of the much larger collection of commands associated with the set of rules comparing location parameters for two variables. In the first situation, SAS reads the complete data set from an ordinary ASCII file, generates a corresponding SAS data set, and then sorts this data set according to the magnitude of the observations. In the second situation, SAS simply reads an ASCII file with only one of the original variables and creates the corresponding SAS data set.

Figure 3.24 shows parts of the section containing the actual commands and also the first few records in the section with additional information. Suppose that the first records in the data base are as shown in Figure 3.25. The records in this part of the data base do not contain the same kind of information, and the subdivision into fields may differ from record to record. As usual, record no. 10 contains the current variable number, in this situation 1. The next pair of records indicate that the two variables X1VAR and X2VAR should be compared. The first field in each record gives an address showing where the actual variable is located in the data storage. The next field contains a code indicating whether the particular variable has been

selected by the user or not (with values 1 and 0, respectively). In this set of rules, both variables must be selected in order to carry through the analysis. Following the field containing the variable name, the last two fields show the number of observations and the number of missing values, respectively.

	1	2	3
.			
355	1		TITLE 'Execution of SAS';
356	1		OPTIONS PAGESIZE=60 ;
357	1		DATA EXPR;
358	556		INFILE '%0';
359	553		INPUT %0 %1 ;
360	1		RUN;
361	1		PROC SORT DATA=EXPR;
362	553		BY %0;
363	1		RUN;
.			
428	1		TITLE 'Execution of SAS';
429	1		OPTIONS PAGESIZE=60 ;
430	1		DATA NYTT;
431	551		INFILE '%0' ;
432	552		INPUT %0 ;
433	1		RUN;
.			

	1	2	3	4	5	6	7
551	10	11	12	(4X,I1,A40)			600
552	10	11	10	(4X,I1,A6)			600
553		11	15	(4X,I1,40X,A6)			600
554		11	15	(4X,I1,46X,A6)			600
555		11	233	(4X,I1,9X,A6)			600
556		11	15	(4X,I1,A40)			600
557		11	16	(4X,I1,A40)			600
.							

Figure 3.24 Parts of the contents of the two sections in the code file concerning commands for external packages, in the set of rules comparing location parameters.

The next four records in the data base include information about data files generated by Express for use by external packages. The second field in any such record, as indicated in Figure 3.25, contains the integer 6 if the file has already been generated, 0 otherwise. The next field contains the file name (assigned by Express). These files may be generated using either of the routines ENFIL or VARFIL (Section 3.5.1). The routine ENFIL produces a file with variables stacked on top of each other, with an additional column for a grouping variable. Thus the file will include two columns. The names associated with these columns are inserted after the file name in the data base. These names occupy the positions 41-46 and 47-52. The VARFIL routine generates a file containing as many columns as there are variables, with the variable names included in the list of variables starting at record 11. In this set of rules, the data file specified in record 15 in the data base was generated by ENFIL and the file specified in record 16 by VARFIL. The files referred to in records 13 and 14 include the observations for each separate variable.

```
 .
 .
10    10  1
11     5| 1|X1VAR        8      0
12     6| 1|X2VAR        8      0
13       | 6|C:\EXPRESS\SYSFIL\313.PXE
14       | 6|C:\EXPRESS\SYSFIL\314.PXE
15       | 6|C:\EXPRESS\SYSFIL\315.PXE                    VARDAT|GRUPER
16       | 6|C:\EXPRESS\SYSFIL\316.PXE
```

Figure 3.25 The first records in the data base for the set of rule comparing location parameters for two variable.

Our aim is now to insert the correct information from the data base into the incomplete parts of the commands shown in Figure 3.24. Consider the commands starting at record 355. The first three commands are already complete. In the command at record number 358, the correct file name is missing. The address for additional information is 556. According to record 556, the additional information is located in file 11 (the data base) at record number 15. The information should be read with the format (4X,I1,A40). As shown by Figure 3.25, the integer read at record 15 is equal to 6, and the string will contain the file name C:\EXPRESS\SYSFIL\315.PXE. This is the file containing one column with the two original variables and one column with a grouping variable. Since the integer read differs from 0, this file name will be inserted into the command:

INFILE 'C:\EXPRESS\SYSFIL\315.PXE';

The next command in record 359 includes two undefined parts, representing names of "variables" that should be read from the data file. As for the previous command, no part of the command can be omitted. The address for additional information is 553. For the first incomplete part we must read information from this record (because <*number*> equals 0 in %<*number*>). This enables us to read the first name listed after the file name in the data base (at record 15), and the name VARDAT is returned. For the second unknown part, the address for additional information will be 554 = 553 + 1 (since <*number*> equals 1). In view of the format shown at record 554, the second name GRUPER following the file name in record 15 in the data base will be returned. The complete command has now been found:

INPUT VARDAT GRUPER;

This is followed by two complete commands and then another incomplete command (BY %0;). The unknown part will here be exchanged with the same information used in the first unknown part of the INPUT command.

The sequence of commands starting at record no. 428 is rather similar but illustrates the use of field no. 1 in the additional information section, referring to the current variable number. In this case the analysis wil be carried out on one of the two possible variables only. Which variable should be considered, is determined by the number stored in record 10 in the data base. The address with specifications about reading additional information for the INFILE command is 551. This record indicates that information should be read from the data base (file no. 11). The record number specified for the information is 12, but since field no. 1 in

record no. 551 contains the number 10, Express will automatically add the current variable number to 12. In this situation the current variable number is 1, which leads to the address 13 in the data base. The information stored in record no. 13 is the file name C:\EXPRESS\SYSFIL\313.PXE, which will be inserted into the command in record no. 431. The following INPUT statement in record no. 432 also utilizes the variable number to obtain to the correct address. The address for additional information given in record 552 is record no. 10 (in the data base), added to the current variable number 1. Thus the actual address will be record 11, containing the name of the first variable. On the other hand, if the current variable number had been equal to 2, the second variable name would have been inserted.

Example with regression analysis

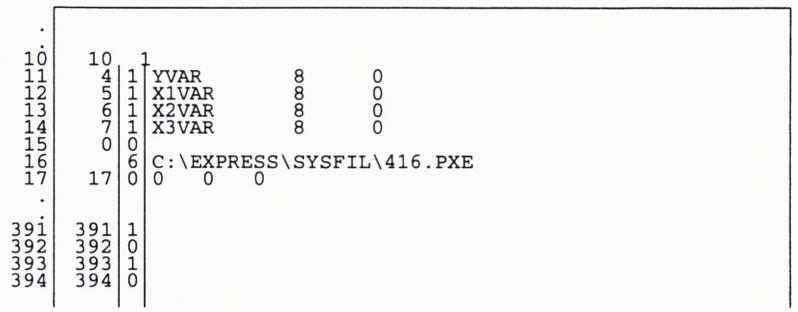
Consider the set of rules for regression analysis. Figure 3.26 shows all the commands to SAS, with additional information, needed in this set of rules. After some general specifications, the stream of commands first includes a SAS "data step", creating the SAS data set NYTT. This data set is used implicitly in a "proc step" carrying out the actual regression analysis. Which variables should be included, is specified in a MODEL statement. It should be noted that the two different MODEL statements shown in Figure 3.26 are used in different situations. The first one is used by the set of rules in ordinary regression analysis and the second one in a stepwise analysis. The correct choice is made on the basis of the specifications in the section of the code file named "Codes for execution of packages". All other commands in Figure 3.26 are included in both situations.

```
        1    2                          3
521     1        TITLE 'Execution of SAS';
522     1        OPTIONS PAGESIZE=60;
523     1        DATA NYTT;
524   721           INFILE '%0';
525   722    1      INPUT %0  %1  {%2} {%3} {%4};
526     1        RUN;
527     1        PROC REG;
528   722    2      MODEL %0 = {%1} {%2} {%3} {%4};
529   722    1      MODEL %0 = {%1} {%2} {%3} {%4} / SELECTION=STEPWISE ;
530     1        RUN;
531     1        QUIT;
```

```
        1    2    3        4            5              6              7
721          11   16  (4X,I1,A)
722          11   11  (4X,I1,A6)
723          11   12  (4X,I1,A6)          391
724          11   13  (4X,I1,A6)          392
725          11   14  (4X,I1,A6)          393
726          11   15  (4X,I1,A6)          394
```

Figure 3.26 The contents of the two sections concerning commands for external packages, in the code file for the set of rules performing regression analysis.

Suppose that the end user has specified, during the process selecting variables from the data storage, that one dependent and three independent variables should be included in general in the data analysis. The records 11-14 in the data base, shown in Figure 3.27, represent the separate variables. Record no. 15 does not specify any variable in this situation, since no fourth independent variable has been introduced.



10	10	1			
11	4	1	YVAR	8	0
12	5	1	X1VAR	8	0
13	6	1	X2VAR	8	0
14	7	1	X3VAR	8	0
15	0	0			
16		6	C:\EXPRESS\SYSFIL\416.PXE		
17	17	0	0 0 0		
391	391	1			
392	392	0			
393	393	1			
394	394	0			

Figure 3.27 Contents of the first records in the data base for the set of rules performing regression analysis.

In principle, several different regression analyses may be carried out with this general selection of variables. Suppose that the end user actually wants to carry out a regression analysis involving only two of the independent variables, for example fitting this model:

$$YVAR = \alpha + \beta X1VAR + \gamma X3VAR \qquad (3.1)$$

This can be done by selecting the option "Regression with two X-variables" in the basic problem type menu (Figure 2.7). The system will then ask the user to specify which independent variables should be considered. The response is recorded in the auxiliary slots in records 391-394 in the data base, with a code 1 for those variables which have been selected.

Consider the corresponding commands to SAS generated by Express. The first three commands in the records 521-523 are already complete. By a procedure similar to that described in the previous example, the INFILE command in record no. 524 will be completed with the file name located in record no. 16 in the data base. The next command in record no. 525 has this form:

```
INPUT %0  %1  {%2} {%3} {%4};
```

Parts of this command may possibly be deleted, as indicated by the code 1 in field no. 2. However, the set of rules requires at least two variables to perform any regression analysis at all. For this reason, only the last three items in the INPUT command may be deleted. The additional information needed is found in the records 722-726 in the code file. The first record, concerning the dependent variable, reads the variable name from record 11 in the data base. The remaining records with additional information will also be used by subsequent commands in the command stream. This explains the additional codes appearing in these records. Since the INPUT command being considered has the code 1 in the field no. 2, these additional codes are ignored.

The substitution process completing the INPUT command first obtains the variable name located in record 11 in the data base, and then inserts this name into the first unknown part

of the command. The second unknown part is handled in the same way using the name from record no. 12. However, each one of the records 13-15 may carry the message that no corresponding variable has been selected, which in turn implies that that particular part of the command should be deleted. With the actual specifications shown in Figure 3.27, this applies to record no. 15 only (since records 13 and 14 both have a non-zero code in field no. 2). Thus, the last unknown part is deleted, and the final command appears in this form:

```
INPUT YVAR X1VAR X2VAR X3VAR;
```

The two SAS commands in the records 526 and 527 are left unchanged. Consider the command in record no. 528:

```
MODEL %0 = {%1} {%2} {%3} {%4};
```

The model specification transferred to SAS must include a dependent variable, corresponding to the first unknown part of this command. However, any of the remaining parts of the MODEL statement, representing independent variables, can be deleted. This contrasts with the INPUT statement above, which had to include all variables that could be considered in the present session, in order to ensure that they would be available in the SAS data set used. Now assume as before that the model in (3.1) should be fitted. In this case it is clearly not sufficient to consider the records 11-15 in the data base in order to decide which parts of the MODEL command should be retained. If we had relied on this information only, all three variables X1VAR, X2VAR, X3VAR would have been included.

To establish the correct model, we must make use of the auxiliary slots 391-394. The definitions of the separate slots are included in Figure 3.23, with a one-to-one correspondence to the four potential independent variables. Because of the user specification indicating that only X1VAR and X3VAR should be included in the model, field no. 2 in records 391 and 393 contains the value 1, whereas field no. 2 in records 392 and 394 contains the value 0.

The system must be instructed to check records 391-394 before the additional information is inserted into the command. This is done by assigning the value 2 to field no. 2 in record no. 528 in the code file. This code implies that the system should not automatically insert the additional information if it has been found. Consider the additional information specified by record number 723 in the code file. As before, record no. 12 in the data base indicates that the information needed is the variable name X1VAR. Field no. 5 in record no. 723 states that record no. 391 in the data base must be checked before the variable name can be inserted into the command. In this case, field no. 6, assumed to contain the format for reading the code in record no. 391, is empty, so the read operation simply uses the first part (4X,I1) of the format given in field no. 4. Thus the integer 1 is read from position no. 5 in record no. 391 in the data base, which indicates that the additional information should in fact be inserted. In contrast, the code 0 in record no. 392 in the data base results in the removal of the part of the command involving the variable X2VAR. In this way the final MODEL statement

becomes

```
MODEL YVAR = X1VAR X3VAR;
```

Summary of sections in the code file with specifications of commands for external software:

Section defining the commands:

Field no. 1 contains a right adjusted address used for completing the command (or the code 1 for already complete commands). This address refers to a record number in the next section. Field no. 2 either contains the code 0, indicating that no part of the command may be dropped, or one of the integers 1 or 2. These codes indicate that part of the command can be removed. Field no. 3 specifies the command itself, possibly including several incomplete parts. Each such part must be marked by a symbol *%<number>*, where *<number>* is an integer that will be added to the address given in field no. 1.

Section defining additional information for commands:

Fields 1, 2 and 3 are used to give the address (file number and record number) for information to be inserted into the command. Field no. 4 contains the format for reading an integer and a string. The string will be inserted into the command if the integer differs from 0. This is done automatically, except when the second field in the section defining the command contains the code 2. In that case, field no. 5 gives the address of a record with a special code, for which a zero value indicates that the particular incomplete part of the command should be dropped. Field no. 6 gives the format for reading this particular code. Field no. 7 should contain an address to a question that may be given the end user if the system is unable to insert information into a command that cannot be omitted.

Search keys

When an external package has been executed, Express will scan the output in search of relevant results. The present section of the code file specifies this process in detail. In the extraction of results from an output file, Express must first identify particular locations close to the results of interest. This is accomplished in each case by means of a particular "search key", which is a string that can be recognized in the output file. When such a key has been detected, the actual results of interest in the neighbourhood of the key will be extracted on the basis of specifications given by other codes in this section.

Each record in this section represents a particular search key, with an associated piece of information which is stored in a slot in the data base when it has been extracted from the output file. In the standard situation this information is assumed to follow immediately after the search key in the output file, but this specification may be modified in various ways.

When the first search key is to be found, the search is started from the top of the output file considered. After this have been found and the search for the second key shall be initiate, the search usual will begin after the first search key. But as noted below there are may be situations where the new search starts after the result extracted and not after the search key (see field 9). If the end of the output file is reach before the search key have been found, the search will continue on the top of the file. But if the end is reach once more, still without finding the search key, Express will conclude that the file does not contain this item. The search for the next search key will then start at the top of the output file.

Field 1 (10 positions, left adjusted): It is possible that a string searched for may occur several times in the output file. To avoid ambiguity, it is possible to search for an auxiliary string first, before the search for the main key starts. If this is desirable, the auxiliary string should be inserted into field no. 1.

Field 2 (2 positions, right adjusted): In stead of specifying an auxiliary string, it is possible to avoid ambiguity by counting the number of occurrences of the search key in the file. Thus, if an output file includes a particular search key five times and we wish to stop at the second occurrence, field no. 2 must contain the integer 2 (right adjusted). It should be noted that the search and the counting of occurrences do not start at the top of the file, but at the point immediately following the search key found most recently. If a search key does not appear in the file as many times as specified in field no. 2, the current search key is ignored. As a special case, if field no. 2 contains the code -1, the last occurrence of the key in the output file will be searched for.

Field 3 (12 positions, left adjusted): This field must contain the actual search key. If a search is carried out for the name of one of the variables selected by the end user for analysis, field no. 3 must be empty. (A search for a variable name is specified in field no. 8.) No distinction is made between upper case or lower case letters in the specification of search keys in this field.

Field 4 (2 positions, right adjusted): This field can be used in two different ways:

a) If a number (or a variable name, see field no. 8) should be extracted from the output file, it may happen that the string of interest does not follow immediately after the search key. It is then possible to skip several strings in the output file before the extraction is made. A string is in this situation defined as a sequence of characters, not including any blank positions. For example, if two strings should be skipped, field no. 4 must contain the integer 2.

b) When a number is extracted, it is also possible to specify the particular positions covered by the number in the output. Field no. 4 must then give the number of lines from the search key to the line containing the number of interest. It is also possible to extract a block from the output, e.g. a plot or a table. In the same way as described above, field no. 4 should

indicate the number of lines from the search key to the beginning of the block. The interpretation of field no. 4 depends on the specifications in other fields as described below.
Field 5 (4 positions, right adjusted): Gives the record number (slot) indicating where the result extracted should be stored in the data base.
Field 6 (2 positions, right adjusted): A particular search key may not always be of interest when a set of rules is executed. For example, in the general set of rules for regression analysis, the actual number of variables can vary, although search keys should in principle be defined for the maximum number of variables that can be accomodated. To avoid listing all possible sets of search keys, we can make use of field no. 6 to indicate whether a particular key should be considered or not:

Code	Description
0	This value indicates that the search key should always be used.
1	With this code, a record number must be supplied in field no. 7. Express will then consider the corresponding record in the data base, and check if the associated slot value has already been determined or not. The search key will be considered in the extraction process only if the slot value is known.
2	In this case the record number given in field no. 7 must represent one of the records in the code file containing additional information for a command for external packages. The purpose is to ignore the search key if a particular incomplete part of a command was deleted during the generation of control language for the external package. As explained in the description of the two preceding sections, such a removal of an incomplete part can be made to dependent on whether or not a certain slot value in the data base is know or not. With the code 2 in field no. 6, no search will be made for the current key if the slot value in the record indicated by field no. 7 is unknown. Note that only the first of the two possible record numbers, given in the section for additional information, will be checked before a decision is made to ignore the search key.

Field 7 (4 positions, right adjusted): Indicates which record in the data base or the code file should be checked to determine whether a search key should be ignored (see description of field no. 6).
Field 8 (2 positions, right adjusted): Field no. 3 is normally used to specify the search key, but in certain situations this is not possible. In particular, it may be convenient to search for one of the variable names used. These names may change from one chaining of rules to another, and thus special keys must be applied for this kind of search. In this connection it should be noted that a variable name in Express can consist of at most 6 characters. The possible codes in field no. 8 are:

Code	Description
0	This field must be equal to zero or be empty if the ordinary search key in field no. 3 should be used.
1,2,3..	When an integer is specified within the range of the variable numbers used, Express will search for the variable name corresponding to the given integer. (The name is determined in the data base at the record number given by the integer specified plus 10.)
-1	With this code the search key in field no. 3 is found first, but rather than extracting a value or a block from the output, a string is extracted. This string will then be compared to the names of the variables in use. If a matching name is found, the number of the corresponding variable is stored in the slot specified (by record number) in field no. 5. If no variable name matches the string extracted, the string will be stored at the slot in question, and the slot will be assigned the code 8.
-2	This is an alternative to the code -1. A search is still made for a string matching a variable name, but field no. 5 is now assumed to specify the number of the first among a row of auxiliary slots, one for each possible variable. If the string is recognized as a variable name, the auxiliary slot corresponding to this variable will be marked as containing a known value and assigned the slot value 1.0. No action is taken with regard to the auxiliary slots representing the remaining variables. Suppose as an example that there are four variables, and that one of the variable names occurs in the output. Assume that the auxiliary slots are represented by the records 301-304. Thus record no. 301 corresponds to the first variable, with a name specified in record 11 in the data base. Record 302 corresponds to the variable specified in record number 12 in the data base, etc. The number 301 must be inserted in advance into field no. 5. Now suppose that the string extracted from the output turns out to be identical to the name of the second variable. Record no. 302 is then assigned the value 1.0, whereas the other auxiliary slots remain unchanged.
-3	This code leads to the same action as the code -1, except when the string extracted does not match any variable name. In that case the search process in the output file will be terminated.
-4	This code leads to the same action as the code -2, but if the string extracted does not match any variable name, the search process is terminated.

Field 9 (2 positions, right adjusted): This field, in combination with fields 4, 11 and 12, is used if information should be extracted from particular columns in the output file. As indicated above, field no. 4 is used to specify the number of lines in the output file from the search key to the line including the information of interest. Fields 11 and 12 define the left and right limits of the columns which may include the result to be extracted. Field no. 9 specifies how these limits should be interpreted, and how the search should proceed when the desired result has been extracted. The alternatives available are:

Code	Description
0	With the code 0 or an empty field, the option for extracting information from particular columns is not used.
1	With this code, the leftmost and rightmost limits for the information extracted (specified in fields 11 and 12) are interpreted in relative sense, with the end of the search key as the starting point. Thus the limits can be negative or positive. The left limit indicates the number of positions to be moved across, after the search key has been found, to arrive at the position where the information of interest begins. When the information has been extracted, before the search process is restarted, the current position in the output file will be at the last search key.
2	Similar to the code 1, except for the current position in the output file when a piece of information has been extracted. With the code 2, the search process will restart at the point following the information extracted and not at the previous search key.
3	The leftmost and rightmost limits will in this case be determined relatively to the left margin of the file. This implies that the limits specified in fields 11 and 12 must both be numbers between 1 and 80. The current position in the file when information has been extracted is at the search key.
4	Similar to code 3, but the current position in the file when information has been extracted is immediately after the result found.

Field 10 (2 positions, right adjusted): This field specifies whether the result to be extracted from the output file is a block of information. If field no. 10 contains a non-zero number, the system will automatically extract a block consisting of a certain number of lines, starting at the position located immediately following the search key. As described above, field no. 4 can contain a number indicating how many lines there are between the search key and the block. A block can consist of any sequence of successive positions on each record. Fields 11 and 12 can be used to specify the left and the right limits defining the block, counting from the left margin of each record in the file.

Field 11 (3 positions, right adjusted): Gives the left limit for extracting information. This field should be empty or equal to zero, except when a block of information is extracted or when the relevant information is restricted to particular positions on each record.

Field 12 (3 positions, right adjusted): Gives the right limit for extracting information.

Examples

Figure 3.28 shows some of the search keys used in the set of rules comparing location parameters. The codes in record no. 571 will lead to the extraction of the number following the first occurrence of the search key "Mean". The value extracted will be assigned to the slot at record number 18. (This record number may be changed, as explained in the description of the next section.) The search key located in record 575 has the integer 2 in field no. 4.

This is because the value of interest does not occur immediately after the search key in the output file. Two strings will be skipped before the value is extracted. The last two search keys in Figure 3.28 define blocks that should be extracted from the output. Field no. 10 indicates how many lines should be extracted, whereas fields 11 and 12 give the left and right limits for the blocks.

	1	2	3	4	5	6	7	8	9	10	11	12
571			Mean	0	18							
572			Std Dev	0	24							
573			Skewness	0	25							
574			Kurtosis	0	27							
575			:Normal	2	19							
576			FREQUENCY	0	31					21	1	80
577			Plot of	1	30					19	1	80

Figure 3.28 Some of the search keys used in the set of rules comparing location parameters for two variables.

Figure 3.29 shows certain search keys from the set of rules performing regression analysis. These keys illustrate a more advanced search technique than the example in Figure 3.28. The added flexibility is needed in connection with output files from stepwise regression analysis. The output produced by SAS will then include results pertaining to several models, with the final model at the end of the file. Hence the first search key must locate the last model considered by SAS. In fact, the search key at record no. 965 has the code -1 in field no. 2, which indicates that the last occurrence of the search key should be found. Then the extraction of the different quantities in the regression model can start.

	1	2	3	4	5	6	7	8	9	10	11	12
965		-1	INTERCEP	0	274							
966			INTERCEP	1	390			-4	1		-8	20
967			INTERCEP	6	266							
968			INTERCEP	10	267							
969			INTERCEP	2	390			-4	1		-8	20
970			INTERCEP	12	268							
971			INTERCEP	16	269							
972			INTERCEP	3	390			-4	1		-8	20
973			INTERCEP	18	270							
974			INTERCEP	22	271							
975			INTERCEP	4	390			-4	1		-8	20
976			INTERCEP	24	272							
977			INTERCEP	28	273							

Figure 3.29 Some of the definitions of search keys in the set of rules for regression analysis.

The first search key extracts the intercept in the regression model, as the first character string after the search key. Figure 3.30 displays the relevant part of the output produced by SAS in a typical example. The INTERCEP string shown here is the last one of its kind in the file, so the value extracted will be 81.5. All subsequent search keys are also identical to the string INTERCEP. This means that the position of all values extracted are defined relatively to the same string in the output.

For the next search key, a string will be read and then compared with all variable names in the hope of finding a match (since fields 8 and 9 have the codes -4 and 1, respectively). In view of the left and right limits specified in fields 11 and 12, the string returned is 'X1VAR'. This corresponds to the first independent variable, which is the second variable in use (see Figure 3.27). Since field no. 4 contains the record number 390, the auxiliary slot in record

no. 391 (390 + variable number - 1) is assigned the value 1.0.

```
-------------------------------------------------------------------------------
Step 2   Variable X3VAR Entered     R-square = 0.95344803   C(p) =   5.20580302

                DF         Sum of Squares       Mean Square          F   Prob>F

Regression       2          2635.33035714    1317.66517857      51.20   0.0005
Error            5           128.66964286      25.73392857
Total            7          2764.00000000

                Parameter        Standard          Type II
Variable         Estimate           Error  Sum of Squares          F   Prob>F

INTERCEP       81.50000000     55.18316362     56.13169014       2.18   0.1997
X1VAR          15.47321429      1.54086381   2595.00777650     100.84   0.0002
X3VAR          -4.72321429      2.58132705     86.15794335       3.35   0.1268

Bounds on condition number:      1.003348,      4.013393
-------------------------------------------------------------------------------
```

Figure 3.30 A part of the output file from SAS executing stepwise regression.

Now the system has determined which variable is the first independent one in the model generated by the stepwise regression. The next pair of search keys will lead to the extraction of the regression coefficient (in this case 15.47321429) and the *p*-value (equal to 0.0002) for this variable. The same procedure is repeated for the next independent variable, in this case X3VAR. As shown by Figure 3.30, no other independent variables are included in the final model determined by SAS. However, Express is not yet aware of this fact, so the program attempts to go through the procedure once more. This time the string extracted does not match any of the variable names in use, and the search and extraction process is terminated (since the code in field no. 8 is equal to -4). When the system returns to the chaining, the rules will determine on the basis of the auxiliary slots in records 391-394 which variables were actually included in the final model.

Codes for execution of packages

The present section of the code file determines the actual composition of each stream of commands when an external package is executed. Each record represents a particular command stream, with codes indicating which commands and search keys defined in the previous sections should be included. This is necessary because a particular set of rules may require external packages to be executed several times. Thus the set of rules for regression analysis incorporates five different sets of commands for running SAS (see Figure 3.31). A separate set is needed for each model fitted, depending on how many independent variables should be included. An additional set of commands is used to carry out stepwise regression analysis. However, several streams of commands may refer to the same commands and search keys defined in the previous sections. Of course, the final composition of the command stream submitted to an external package in a particular situation also depends on the handling of incomplete parts of the commands.

As explained in the guidelines for writing rule routines, the argument CODEREC specifies which set of commands Express should use when a package is executed. The possible values 1,2,3,... of CODEREC correspond to the streams of commands defined by the successive

records in the current section, counted in the same way from the top of the section. For example, if the section begins at record number 201 and CODEREC returns the value 4, the command stream beginning at record 204 should be executed. The description of a particular command stream may extend over several records. Thus, if one stream occupies the first two records of this section, the next stream will begin at the third record, and hence there is no command stream numbered 2. If the next stream of commands is to be executed in this case, the CODEREC argument must return the value 3. In other words, the CODEREC argument refers to the physical record number, as counted from the top of the present section, rather than the natural ordering of command streams.

Field 1 (46 positions, left adjusted): This field contains a string consisting of the characters 0 and 9 only. Each character refers to a particular command. The code 9 indicates that the command must be included in the stream of commands generated. Field no. 4 specifies the record number of the first potential command for this stream. If, for example, this record number is 400, Express will determine whether this command should actually be included in the stream by reading the first character in field no. 1. If this is equal to 9, then the command is written to the file used as input to the external package. If the character is 0, no action is taken. To continue, the system considers the command located in record number 401. The second character in field no. 1 is read in order to determine whether this command should be included or not. This procedure is repeated until a blank character is found in field no. 1. Only 46 positions are available in field no. 1. If a particular stream of commands requires more characters, a plus sign can be inserted as the last character in field no. 1, to indicate that more characters follow in field no. 1 in the next record. In the continuation record, only field no. 1 should contain any codes at all (and possibly field no. 6; see below).

Field 2 (2 positions, right adjusted): When Express generates a data file for use by an external package, the name of the file is stored in the data base. The number associated with the record containing this name must be supplied by the set of rules. As explained in Section 3.5.1, this can be done by direct specification of the record number in the argument RECORD in the routines ENFIL and VARFIL. Alternatively, an indirect reference may be made by assigning either of the integers 1 or 2 to the argument RECORD. If RECORD equals 1, Express will read the record number, to be used for storage of the file name, from field no. 2 in the current section of the code file. This facility makes it easy to change the record used to store the file name, without the recompilation of rules required by a change in the explicit record specification.

Field 3 (2 positions, right adjusted): This field can be used, in exactly the same way as field no. 2, to specify a record number in the data base where information is stored about a data file generated by Express. Field no. 3 is used if the argument RECORD equals 2.

Field 4 (4 positions, right adjusted): This field gives the record number of the first command in the command stream.

Field 5 (3 positions, right adjusted): For each stream of commands, it must be specified which external package should be executed. Field no. 5 is used for this purpose, with an integer code referring to a number in the list of different packages defined in the system file (see Figure 3.9). If this integer equals 1, the first package on the list is executed; if it equals

2, the second package is executed, etc. In addition, the integer 0 can be used to indicate that the results should be extracted from the most recent output already produced. In this case, no external packages are executed.

Field 6 (27 positions, left adjusted): In the same way that field no. 1 indicates which commands should actually be included in the command stream, field no. 6 shows which search keys should be used. This field should also contain a string consisting of characters 0 or 9. In each case, the character 9 indicates that the corresponding search key should be used, whereas the character 0 shows that the key should be ignored. Again, more search keys may be needed than can be accomodated in the space available. If a plus sign is inserted at the end of the field, the string can be continued in field no. 6 in the next record.

Field 7 (4 positions, right adjusted): This field serves the same purpose for search keys as field no. 4 does for commands. Thus, field no. 7 indicates the record number of the first search key associated with the current command stream. The combined information from fields 6 and 7 determines precisely which search keys should be used.

Field 8 (4 positions, right adjusted): When a search key is defined, a record number must be specified (in field no. 5, in the search key section), to indicate where the result extracted should be stored. To make it possible to use the same search keys in different streams of commands, it is possible to add a particular number to this address. Field no. 8 is used to specify such a jump code. When the control passes to the program extracting and storing results, it immediately reads the number in this field. This number is then multiplied by DATANO-1, where DATANO is the argument representing the variable number, passed to the extraction program by the set of rules program. (It should be noted that this argument DATANO is not obtained from record 10 in the data base.) The resulting number will be added to the address given in the definition of the search key.

Field 9 (4 positions, right adjusted): The jump code referred to in field no. 8 must not necessarily be active in all situations. Field no. 9 can specify a maximum value for the DATANO argument, for use in connection with this jump code. Thus if the argument DATANO passed to the program extracting and storing results exceeds the value in field no. 9, the record number given in the definition of the search key will not be modified by the jump code.

Field 10 (4 positions, right adjusted): This field is also used to modify the address indicating where the result extracted should be stored. The number specified in this field will be added to the original address (from field no. 5, in the search key section), regardless of the value of the DATANO argument. Fields 8 and 10 can also be used at the same time.

Examples

Figure 3.31 shows the definitions in the code file for the set of rules carrying out regression analysis. The stream of commands defined in record no. 501, which corresponds to regression analysis with a single independent variable, starts reading commands from record number 521. Comparing Figure 3.26 with the string in field no. 1 in Figure 3.31, we note that the MODEL command for stepwise regression is the only one excluded from the command file in this

situation. The also applies to the next three command streams, for regression analysis with two, three and four independent variables, respectively. However, the last command stream in Figure 3.31, for stepwise regression, will only include the second MODEL statement in Figure 3.26.

	1	2	3	4	5	6	7	8	9	10
501	99999999099	16		521	1	99999	921	10	15	-10
502	99999999099	16		521	1	9999999	928	15	15	-15
503	99999999099	16		521	1	999999999	936	20	15	-20
504	99999999099	16		521	1	99999999999	950	0	15	0
505	99999990999	16		521	1	9999999999999	965	0	0	0

Figure 3.31 The records defining the different streams of commands in the set of rules for regression.

Field no. 2 in Figure 3.31 indicates that the name of the data file generated should be stored in record no. 16 in the data base. Field no. 5, specifying which external package should be used, contains the integer 1 for all command streams. This code represents SAS, which is the first package specified in the list defining external packages (see Figure 3.9). With regard to search keys, a more complex coding is needed. If we still consider the stream of commands specied by record no. 501, the search keys are located in the records 921-925, as shown in Figure 3.32.

	1	2	3	4	5	6	7	8	9	10	11	12
921			R-square	0	26							
922			INTERCEP	1	24							
923			INTERCEP	7	21							
924			INTERCEP	10	22							
925			Dependent	0	25					22	1	80

Figure 3.32 Search keys used in the first stream of commands in the set of rules for regression.

A particular problem arises in the command stream for regression analysis with a single independent variable, since is it is not known in advance which one among the four possible variables will be the selected as the independent variable of interest. For this reason it must be possible to store the values extracted from the output at different locations depending on which variable is specified. This problem is handled by using different values of the DATANO argument, depending on the specification made by the end user. If the first independent variable is selected, DATANO will be set equal to 2 (since this is the second variable in use). If the second independent variable is used, DATANO will be equal to 3, and so on.

The first search key is defined in record no. 921. If the argument DATANO equals 2, for the first independent variable, the result is stored in record no. 26 (see Figure 3.32). For the second, third and fourth independent variables, the results must be stored in records 36, 46 and 56, respectively. This is achieved with a jump code equal to 10, which is specified in field no. 8 in Figure 3.31. Field no. 9 contains the number 15, which is simply a sufficiently large number to ensure that the jump code will be used in all situations. The last field contains -10, which is a number that will be added to the address given in the definition of

the search key. This is done because the relevant values of the DATANO argument are 2,3,4,5 rather than the normal set 1,2,3,4. A similar procedure is used for the other search keys in this situation.

Menu for basic problem type

A particular set of rules may be started in an attempt to solve different problems. This section should indicate which problems can be selected by the end user from the basic problem menu for the current set of rules. Each problem of this kind must correspond to a particular rule which can be started to begin the chaining. There is a one-to-one correspondence between admissible problems and records in the present section.

Field 1 (4 positions, right adjusted): Should only be used in the first record in this section. This field should indicate the number of problems to be presented to the end user in the menu.
Field 2 (4 positions, right adjusted): Field no. 3 should assign a text to the particular problem at hand. Field no. 2 should indicate the length of this text.
Field 3 (72 positions, left adjusted): Contains the text associated with the problem which is displayed in the menu.
Field 4 (4 positions, right adjusted): Indicates the number of the rule that will be activated first by Express to solve the problem selected.
Field 5 (4 positions, right adjusted): Each problem may also be assigned a more comprehensive explanation, which is presented to the end user when asked for. If such explanation exists field no. 5 should contain the address of this, as a record number referring to the next section of the code file.

Example

Figure 3.33 shows the specifications for the alternative problems associated with the set of rules performing regression analysis. The menu (displayed in Figure 2.7) includes seven different alternatives, as indicated by field no. 1 in the first record. For executing a regression analysis with a single independent variable, Express must activate rule no. 1 (as specified in field no. 4). A more detailed explanation of this alternative is found in record no. 1040.

	1	2	3	4	5
1021	7	40	1.Regression with one X-variable.	1	1040
1022	0	40	2.Regression with two X-variables.	2	1042
1023	0	40	3.Regression with three X-variables.	3	1044
1024	0	40	4.Regression with four X-variables.	4	1047
1025	0	55	5.Linear regression with all X-variables, one by one.	5	1050
1026	0	65	6.Find the best fit by leaving out not significant factors.	6	1055
1027		45	7.Find the best fit by stepwise regression.	8	1059

Figure 3.33 The section for defining the basic problem menu in the code file for the set of rules performing regression analysis.

Particular text for this set of rules

This is the last section of the code file. It is used to store different kinds of text associated with the current set of rules. It includes more comprehensive explanations of slots and explanations needed in the selection of variables or basic problems. Furthermore, explanations of the various rules, referred to when the stack is shown on the screen, must be stored in this section. In each case, the address, indicating where the text is stored, must be inserted into the appropriate location as described above.

Field 1 (4 positions, right adjusted): Several sentences which should be displayed on the screen at the same time must be regarded as a combined text. Field no. 1 of the first record for any text must contain an integer, indicating how many records the text occupies.
Field 2 (4 positions, right adjusted): Field no. 2 should indicate the length of the text for a particular record. Optionally, Express can itself determine the length of any record as the number of characters from the first position in field no. 3 to the last non-blank position. If field no. 2 is left blank, the correct length will be automatically inserted by Express when the text is considered for the first time. If field no. 2 already contains a length specification, this number is left unchanged.
Field 3 (72 positions, left adjusted): This field should contain the actual text. The text representing a comprehensive explanation to a slot, contained in a single record, must not exceed 65 positions.

Example

Figure 3.34 shows part of the text defined in the code file for the set of rules carrying out regression analysis. In view of the address specifications shown in Figure 3.7, record no. 1031 should provide an explanation to the first variable selected. It is essential that the explanations to the different variables follow each other with no empty records in between. As indicated in Figure 3.7, the explanations to the different rules, each occupying one record, start at record no. 1065. The record immediately preceding the first explanation of any rule, in this case record no. 1064, should always include a text which will be displayed when the stack is empty.

	1	2	3
1031	1	60	This is the dependent variable in the regression analysis.
1032	1	63	This is the first of the independent variables. Denoted by X1.
1033	1	64	This is the second of the independent variables. Denoted by X2.
.			
.			
1064	1	15	Stack is empty
1065	1	28	Rule for linear regression.
1066	1	52	Rule for multiple regression with two X-variables.
1067	1	54	Rule for multiple regression with three X-variables.
1068	1	53	Rule for multiple regression with four X-variables.

Figure 3.34 Some examples of text contained in the last section of the code file in the set of rules for regression.

3.6.3 Editing the data base

When information has been inserted into the code file, the new set of rules is almost ready to be executed. However, a few particular numbers must also be inserted into the data base. This section describes the steps needed to prepare the data base. It also provides a survey of the general usage of the different fields in the data base. It is only intended as an explanation in connection with the debugging of the set of rules. It is not necessary for the knowledge engineer to understand all details of the data base, although the basic structure is essential to the writing of a new set of rules.

The following information must be inserted before a set of rules is ready to be executed:

Record 1: This record is used to store information about the number of variables a set of rules can handle.

Field 1 (4 positions, right adjusted): This field must indicate the maximum number of variables that can be selected for the set of rules. As a simple example, this number is equal to 2 for the rules comparing location parameters. In the set of rules for regression analysis the maximum number is 5 (4 independent variables and 1 dependent variable).

Field 4 (4 positions, right adjusted): This field contains the minimum number of variables that must be selected to execute the set of rules. Thus, both in the rules for regression analysis and in the rules comparing location parameters, this number is equal to 2.

Only fields 1 and 4 should be filled in by the knowledge engineer in record no. 1. Express also uses field no. 5 to record how many variables are actually selected by the end user. In the rules comparing location parameters, this number is always equal to 2, but in the rules for regression analysis it ranges between 2 and 5.

Record 3: When the end user selects new variables for analysis or leaves Express, without specifying that information collected should be retained, almost all records in the data base must be erased. This is carried out by inserting the a zero into field no. 2 in the corresponding records, indicating that the slot value is unknown. A particular routine will do this for all relevant records, beginning

at the record number given in field no. 1 in record 3. This is because the initial part of the data base contains essential information which should not be erased. It is possible in principle to specify any number in this field, but the convention has been adopted that the number should be equal to 11 plus the maximum number of variables allowed for this set of rules. Thus this specification is equal to 13 for the rules comparing location parameters and 16 for the rules for regression analysis.

All information supplied by the user is now hopefully in its correct place. Debugging is the next step. During this phase it is helpful to have access to a list of the possible codes included in the fields of the different kinds of records in the data base. This description does not cover the first 10 records which are used for other particular purposes. The records from number 4 to 9 is used by Express to store slot numbers to be set to unknown it the end user during the analysis desides to exit from the chaining of rules (see the subroutine UTNULL in section 3.5.1). As described earlier, record number 10 will keep record of the active variable number. This record contains the record number 10 in the first 4 positions. Then in position 5 a one digit integer tells if the variable number if set or not (0 means not yet found), while the actual variable number is given in positions 6 and 7.

Records	**Description**
Records for storing information about variables	Consider a certain sequence of records starting at record no. 11. The total number of records in this sequence is equal to the maximum number of variables allowed in the set of rules. As described previously, these records are used to keep track of the variables selected by the end user in a particular session. Thus in the rules for regression analysis, records 11-15 are used for this purpose, with record no. 11 containing information about the first variable selected (the dependent variable), record no. 12 containing information about the second variable (the first independent variable), and so on. These records do not have the same subdivision into fields as the remaining parts of the data base. In the first four positions, representing field no. 1, the number assigned to the variable in the data storage is inserted. Position 5, representing field no. 2, contains an integer which is either 0, indicating that no variable has yet been selected, or 1, if the selection has been made. The name of the variable will be automatically inserted into positions 6-10, while the number of observations and the number of missing values will appear in positions 14-17 and 20-13, respectively.

Records for storing name of data files

A certain number of records are set aside for information about data files used during execution of external packages. For example, Figure 3.25 includes four records used for this purpose in the rules comparing location parameters. The first pair of files corresponding to these records contain the values of the two variables considered separately. In the remaining two files, both variables are present. Two distinct files are needed in this case because of the different organization of the input data required by the various procedures in SAS.

Records for storing results

The remaining part of the data base forms a uniform sequence of records, used to store the slot values included in the analysis. The subdivision of records in the data base into separate fields has largely been adapted to this part. The different fields are:

Field 1 (4 positions, right adjusted): This fields simply contains the record number.

Field 2 (1 position): This is a very important field, consisting of a single position only. An integer is inserted to specify the present state of the slot. The different codes are:

Code	Description
0	This code is present when the slot value has not yet been found.
1	This integer indicates that the value assigned to the slot is a result extracted from the output of an external package.
2	The result has been assigned to the slot in the set of rules program (using the TILGDB routine).
3	All results determined by the end user are assigned this code.
5	Slots defined as blocks (plots or tables) in the code file are given the code 5 when they have been determined.
8	If a search key specifies that a string should be found matching a variable name and the search fails, the code 8 is inserted for the slot (see Section 3.6.2).
9	This code is inserted if Express has failed in its efforts to determine the slot value.

Field 3 (1 position): When both the end user and Express have assigned a value to a slot, it will be stated during the presentation whether Express agrees with the value given by the user. (For numerical values, differences of at most 10 per cent are allowed). Field no. 3 keeps track of this relation. Four values are used for this purpose. The code are set to either 1 or 2 if the system and the user agrees. In contrast if there are disagreement the codes 3 and 4 are used. The

codes 1 and 3 are used for slotvalues that have been found by an external package, while the codes 2 and 4 are used for values found by the rules themselves.

Field 4 (4 positions, right adjusted): This field is used to store the address (record number) in the log file of the last reference to the present slot.

Field 5 (4 positions, right adjusted): When the LESGDB and TILGDB routines are used, we may specify a particular "target" for a slot. This means that Express can keep track of which results depend on other results. Such connections are recorded in the relationship file. Express automatically inserts a corresponding address in field no. 5 in the data base. This refers to the location in the relationship file with information on logical relations to other slots.

Field 6 (20 positions, left adjusted): This field covers 20 positions and is used to store the value assigned to the slot. Field no. 6 is normally used for values determined by the system itself and not for values set by the end user.

Field 7 (20 positions, left adjusted): If the end user determines the value of a slot, the result will be stored in this field.

The description above will be slightly different if the slot in question is defined as a block. The first five fields will contain the same information, but in field no. 6, the first eight positions are used to record one or two addresses in the file used to store blocks (plots and tables). Express automatically distinguishes between small and large blocks, depending on the number of lines involved. Small blocks comprise less than 10 lines. The two addresses indicate where Express can find a small and a large block, respectively.

3.6.4 Debugging

A set of rules normally needs some debugging before it executes as intended. This mainly involves checking the specifications in the code file, in particular for consistency with the calls to routines made in separate rules. Only general guidelines can be given here although it has been useful in these situations to have a figure available describing the relationship between the different rules (as, for example, Figure 3.5). This figure should be compared with the actual rules as they have been coded.

A common error involves an incorrect address returned to the main program when a rule should start another rule. Such problems may often be corrected in the code file, although sometimes the error is located in the set of rules program. Then the program must be recompiled and linked when the error has been corrected.

To locate errors in general, the system variable DEBUG (Section 2.6.1) should be turned on.

Useful additional information will then be listed on the screen during the analysis. This includes information about subroutines executed from the rules, specification of data files generated for use in external packages, and information about commands and search keys used. The following routines will print the argument values involved when they are invoked: HENTPAR, SENDPAR, PAASTB, LESSTB, LESGDB and TILGDB. Figure 3.35 shows how this information is typically presented.

```
Values at exit from the LESGDB routine.
```

Routine	RECORD	VALUE	RULENO	TARGET	FOUND
LESGDB	51	999.00000	21	0	False

Figure 3.35 Part of the log, when the system variable DEBUG is turned on.

Routine	Record	File name	Name	#obs	#mis
ENFIL	15	C:\EXPRESS\SYSFIL\315.PXE	alco	975	0
			tobac	975	0

Figure 3.36 Example of information presented when a data file is generated.

When a data file is generated for use by an external package, additional information is displayed as shown in Figure 3.36. The program indicates which one of the two file generating routines is used, and the file name is shown with the number of records. The names of variables included in the file are displayed, with the number of observations and the number of missing values.

The last process that can be examined with the debugging tool is the generation of commands for external packages and the extraction of results from the output. Figure 3.37 shows what kind of information is displayed for command generation. Each command is shown in the original form stored in the code file. If a command includes incomplete parts, it is explained how it is completed. For example, the name of the file containing the data must be inserted into the second command in Figure 3.37. Below the actual command the debugging tool indicates that the additional information needed is obtained from record no. 4041 in the code file (ADD=4041). This information then directs the system to the actual string to be inserted into the command, at record no. 38 in the data base (REC=38). Since the flag FL1 differs from zero, the system knows that the information searched for is stored at this location. Now the actual string inserted is shown following the INFO label. The two flags FL1 and FL2 correspond to the two codes described in the section "Additional information for commands" which may be read from a file (the data base) using prescribed formats (see Figure 3.26) to ensure that the proper information is found and should be inserted into the incomplete command. If the second flag FL2 equals -99, this means that it is not in use, as is usually the

case. The quantity SHR will differ from zero only if the incomplete field in question is deleted from the command.

Figure 3.38 shows the information presented by the debugging tool when values are extracted from the output produced by an external package. In addition to the search key and the actual value extracted, two records number are displayed. First, the number of the record in the code file containing the search key is presented. At the end of the line, the record number in the database where the value is stored is given.

```
Number of the stream of commands:     1

Commands as they are stored in the code file (rec. no. inserted first).

3442     1     0 /PROB TITLE = 'Deciding number of levels for a variable'.
3443  4041     0 /INPUT FILE = '%0'.
                 ------------------------------------------------------------
                 No.   1    ADD=4041 REC=38   FL1=6    SHR=0    FL2=-99
                 (System)  INFO=  C:\EXPRESS\SYSFIL\638.PXE
                 ------------------------------------------------------------
3444     1     0    VARIABLES = 1.
3445     1     0    FORMAT = FREE.
3446  4042     0 /VARIABLE NAMES = %0.
                 ------------------------------------------------------------
                 No.   1    ADD=4042 REC=11   FL1=1    SHR=0    FL2=-99
                 (System)  INFO=  case.
                 ------------------------------------------------------------
3447     1     0 /PRINT COUNT.
3448     1     0    LINESIZE = 80.
3449     1     0 /END
```

Figure 3.37 Information presented when commands for an external package are generated.

```
In the output file produced by the execution of the external package,
EXPRESS has used these search keys, and has extracted these values:

    (REC)  SEARCH KEYS        VALUES                 (REC)


  (4442)  MAX                 1.0000000              ( 103)
  (4443)  MIN                 0.0000000              ( 102)
  (4444)  DISTINCT            2                      ( 101)
```

Figure 3.38 The debug information presented during extraction of values from the package output.

Errors of a different kind arise if essential information in the system file is erased or inserted incorrectly. Express will then not execute properly, and frequently a Fortran error message will indicate that a particular file cannot be opened. Errors involving missing or inappropriate file names in the system file can be very hard to correct. It is often impossible to use the Express editor since the missing files are needed for starting the program. The editor requires the presence of several files before any one can be edited. In some cases this problem can be circumvented by running a particular auxiliary program, which is able to edit the system file without entering the standard Express editor. The program is started by typing C:\EXPRESS\EDSYS.EXE on the command line. However, even this simplified editor must have access to a few additional files. If this program also fails, it is advisable to go through the complete installation of Express once more.

Chapter 4

UNIX VERSION FOR THE X WINDOW SYSTEM

4.1 INTRODUCTION 124
4.1.1 Purpose of the UNIX version 124
4.1.2 Implementation 124

4.2 DESCRIPTION 126
4.2.1 The main screen 126
4.2.2 Rules 127

4.3 THE EDITOR 128
4.3.1 Introduction 128
4.3.2 The system file 129
4.3.3 The code file and the data base 130

4.4 SIMULATIONS 131

4.1 INTRODUCTION

4.1.1 Purpose of the UNIX version

The version of Express running under UNIX is intended to be a supplement to the main PC version. There were several reasons why it was decided to create such a simplified UNIX version. First, we wanted to investigate in general how portable Express was. Second, it was of interest to explore the possibility of running Express under a more standardized window manager system. In the PC version, all program code in connection with windows and pull down menus is incorporated into Express itself, in special assembler routines.

Finally, a major motivation behind the UNIX implementation was the wish to speed up the program, making it possible to include a simulation module. UNIX computers (work stations) were available which were much faster than the PCs. Especially when external software was executed, the excessive amount of time required on a PC made it difficult to carry out extensive simulations. The simulation module built into the UNIX version makes it easy to go through a simulation cycle many times, using Express at each cycle to analyse a different data set according to any standard set of rules. The data set can be created each time by a suitable external random number generator. The simulation module will keep track of how many times the different alternative conclusions are reached by Express, and also to some extent how many times different paths are being followed through the rules in order to reach these conslusions.

It must be emphasized that the UNIX implementation constitutes a very limited version of Express. Many features of the PC version have not been incorporated. But our experience with this basic UNIX version indicates that it should be relatively simple to construct a complete UNIX version.

4.1.2 Implementation

The software used in the UNIX implementation includes SUN Fortran, SUN C and XView. Since most of the subroutines written for the PC version were coded in standard Fortran, a major part of the program code could be transferred quite easily. However, the restructuring of the main program posed a major problem. In the PC version the main program is in full control at any time. It initiates various tasks that must be carried out and then waits until they are completed before proceeding. A window manager system such as XView reacts on interrupts (and is notification based), which makes it possible for the user or other programs to influence the program flow at almost any time. For this reason, major changes were required in the structure of the main program of Express (written in C as required by XView). For the user, however, it may be more significant that new knowledge bases (sets of rules)

may be implemented in the UNIX version in almost the same way as in the PC version.

A new version of the editor was also constructed for the UNIX adaptation. Almost the entire editor has now been written in C and XView.

4.2 DESCRIPTION

4.2.1 The main screen

Figure 4.1 shows the main screen in the UNIX version. It differs from the PC version as some items in the main menu have been deleted. The option "Simulations" has been added. Furthermore, the screen has been divided vertically into three parts rather than two. The upper and lower parts are used for the same purpose as in the PC version, although some of the displays may differ as windows have now been defined using XView. The middle part of the screen has been inserted mainly to show additional information when the simulation module is active. Also, some information generated when an external package is executed will be displayed in this window. A feature not available in the PC version is the separate editing facilities in each part of the screen. Thus, the user may at any point insert his own comments before storing the text from a part of the screen in a file.

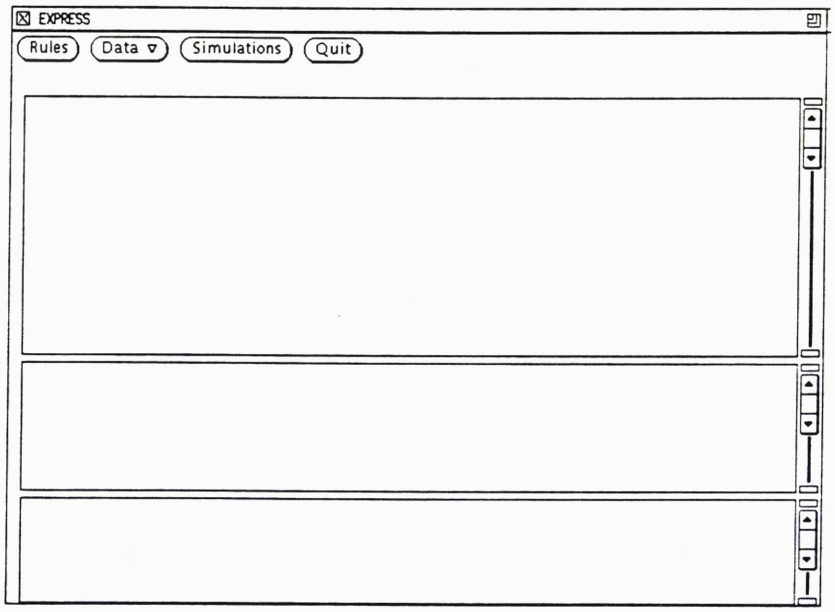


Figure 4.1 The main screen in the UNIX version of Express.

In the main menu, rules are selected in the same way as in the PC version. However, no explanations are available of the different items in the basic problem menu. For entering data into the data storage, the procedure is the same in the two versions, although listing of data is not possible in the UNIX version. Selecting data to be used in an analysis is also carried out in the same manner. Selections made under the "Simulations" option are described in Section 4.4. When the program exits, all information in the data base is preserved.

4.2.2 Rules

Rules are coded in almost the same way in the two versions of Express. Some changes have been introduced in the main program of each set of rules, as the UNIX version of Express does not escape to an outer shell when an external package is about to be executed. As indicated in the description of the PC version in Section 3.5.2, certain arguments to the main program in Express must then be stored and later retrieved. This is not necessary in the UNIX version, since any package execution is started directly from the currently active rule. Thus the statement usually labelled 1000 (see Section 3.5.2) must be omitted. Furthermore, the particular subroutines used for transferring arguments have not been implemented in the UNIX version and the subroutine UTNULL has been removed. All other library subroutines should be used in accordance with the explanation in Section 3.5.2.

Since there is no need to escape to an outer shell, the commands needed to execute external software are slightly different. If we consider the example in Figure 3.19, the same rule implemented in the UNIX version will appear as shown in Figure 4.2.

```
      SUBROUTINE REGTRE(RULENO,DATANO,RECORD,ADDRESS)
      INTEGER*2 RULENO,DATANO,RECORD,ADDRESS,VARNO(3)
      REAL VALUE1
      LOGICAL FINISH,UNTRUE,FOUND,TRUE
C
      TRUEEX = .TRUE.
      UNTRUE = .FALSE.
C
C  Finds p-value for t-pooled test.
C
      CALL PAASTB(RULENO,DATANO)
      RECORD=37
      DATANO=3
      CALL LESGDB(RECORD,VALUE1,RULENO,51,FOUND)
      IF ( .NOT. FOUND ) THEN
         VARNO(1) = 2
         VARNO(2) = 1
         VARNO(3) = 2
         CALL ENFIL(VARNO,1,RULENO,TRUEEX)
         CALL SKR10(DATANO)
         CALL JOBB(RULENO,DATANO)
         CALL LESGDB(RECORD,VALUE1,RULENO,51,FOUND)
      ENDIF
      RECORD=51
      IF (VALUE1 .LT. .10) THEN
         CALL TILGDB(RECORD,0.0,UNTRUE,0)
      ELSE
         CALL TILGDB(RECORD,1.0,UNTRUE,0)
      ENDIF
      CALL LESSTB(RULENO,DATANO,FINISH)
      ADDRESS = 1
999   CONTINUE
      END
```

Figure 4.2 Rule for executing a t-test.

Since the execution of an external package no longer must be treated as an exit to a special rule, the different executions can be numbered 1, 2, 3, etc. Thus, we must not subtract a number from the RULENO argument (used in the routines ENFIL and JOBB) to execute the correct stream of control language. The main difference is that a temporary exit must be made from the rule in the PC version before a package can be started, whereas the routine JOBB executes the package directly in the UNIX version. In this case the rule will not be restarted when the execution has finished and it is therefore necessary to include an additional call to the routine LESGBD after the call to JOBB.

4.3 THE EDITOR

4.3.1 Introduction

Since only one knowledge base can be implemented in each physical copy of Express in this version, the system file has been simplified. In addition, some other files have been removed, such as the file containing definition of windows. However, the code files are almost identical in the two versions.

We shall outline how information is stored in the different files. Figures 4.3 and 4.4 show the basic main screen of the editor and the screen when the code file is edited, respectively. To edit a particular file, click the mouse on the relevant item, and the contents of the file will be displayed. The editing buttons at the right will then become available and the editing can begin.

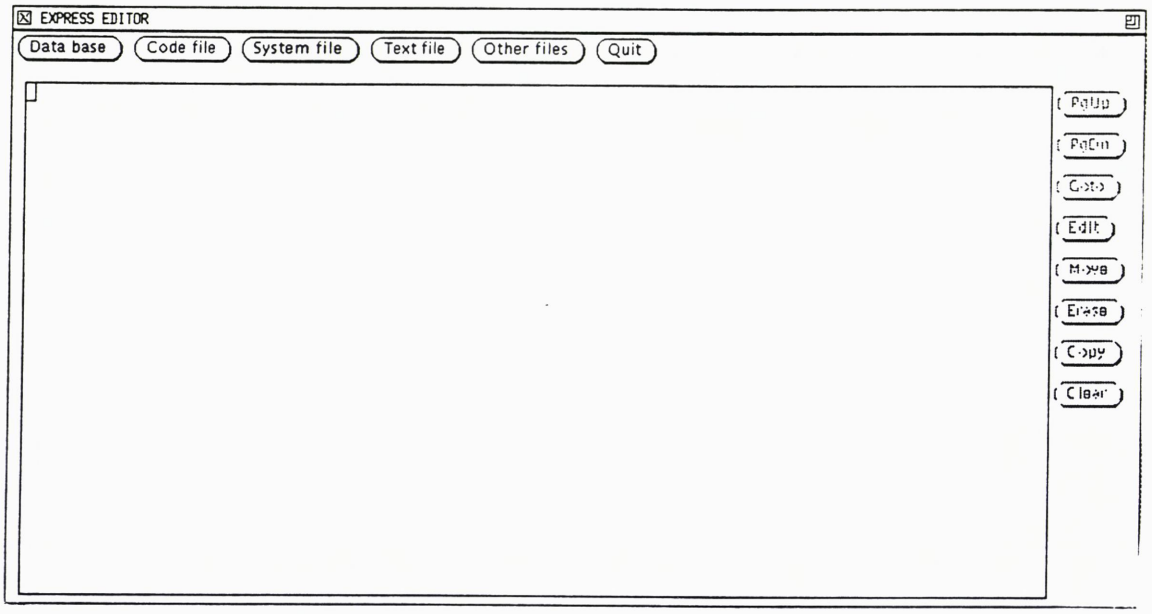


Figure 4.3 The basic screen in the editor.

When a particular record is edited in the code file, all fields belonging to this record are shown separately (Figure 4.4). This is a slightly improved facility as compared with the PC version. However, in the UNIX version, editing with automatic subdivision of records is only available for the code file.

Figure 4.4 Editing the code file using the new version of the editor.

4.3.2 The system file

Records 11-99 define the external packages used. Records 11-49 contain the actual commands executing packages. These commands do not constitute "BAT files" as in the PC version but represent system commands such as

/bmdp/bmdp 7d express/styr.pxe express/utskrift.pxe

Records 51-99 include codes for missing values used in the different packages.

Records 100-209 are used by the system for various purposes, including management of data storage and simulation. If it is intended that simulation should be available for the current knowledge base, some additional codes must be given in the records starting with number 210. Record 210 must contain the command that invokes the external random number

generator. This might be a command of the type

/home/u/aarseth/nagnorm

where the program "nagnorm" generates normally distributed observations and writes them to a file. How Express should read these values is indicated in record 212 in the system file and in the following records.

212	3				
213	1	10	varen	20	/home/u/aarseth/nagnorm.dat
214	11	20	varto	20	/home/u/aarseth/nagnorm.dat
215	21	30	vartre	20	/home/u/aarseth/nagnorm.dat

Figure 4.5 Codes necessary for retrieving the generated data into Express.

Figure 4.5 gives an example of how three variables located in the file nagnorm.dat may be retrieved by Express. The position of each variable must be specified first. In this example each variable occupies 10 positions. Thus for the first variable, the limits 1 and 10 are indicated. For the second variable, the limits inserted are 11 to 20 and for the third variable produced by the generator, the limits are 21 to 30. Each variable must be assigned a particular name. Six positions are used on each record for this purpose. Before the name of the file containing the data is given, the number of observations generated in each simulation must be stated. In the same way as other numerical items, this number is written in a field occupying 4 positions, right justified.

4.3.3 The code file and the data base

The information stored in the code file and the data base is almost identical to that in the PC version. The only record which must be filled in by the knowledge engineer in the data base is record number 3. As in the PC version, this should specify the number of the first record to be erased when the information in a particular data base is deleted.

In the code file, records 2-9 are used to divide the file into different sections. These are the same sections as used in the PC version. In record no. 10, two integers must be inserted (each occupying four positions, with right justification). The first number indicates the lowest number of variables that the set of rules can handle, while the second number gives the maximum number. From record no. 11 on, one record is assigned to each of the possible variables which can be included. The number of records used must be equal to the maximum number given in record 10. In these records a short explanation of each variable should be provided. This text will be presented to the user when variables are selected from the data storage. In the PC version, these explanations could be located almost anywhere in the code file, as the system file included an address indicating where the text started (Section 3.4). The remaining information to be stored in the code file is identical to that in the PC version.

4.4 SIMULATIONS

The new version makes it possible to simulate the strategy implemented in Express. Section 4.3.2 explains which codes must be included in the system file in this case. Some simple specifications must also be made in two additional files. When a simulation is started, the system will keep track of two different counts. It will record the number of times each particular value of a slot has been found, and the number of times this value is used as a basis for other inferences. Any call to the routines LESGDB or TILGDB will increase the second count, whereas the first count will be increased whenever a value has been found by an external package.

The information to be inserted in the main additional file (a direct access file with record length 100) determines which quantities should be considered in the simulation. When a knowledge base has been built, several slots might be included which are not of interest in this connection. If the corresponding record in the additional file is left empty, this particular slot will not be simulated. Thus, the slot value will be determined when needed but no records will be kept of the values found in different cycles.

Figure 4.6 shows how information should be inserted into this additional file. The first field can be coded either with 0, 1, 2 or 3, where a zero value prevents the slot from being simulated, as in record no. 101 in the example. The slots defined in records 100, 102 and 103 will be simulated. The slot values defined in records 100 and 102 will be divided into two groups, using 0.5 and 0.05 as cutpoints, respectively. The definition in record no. 103 includes two cutpoints and thus the values generated will be distributed into three groups. Moreover, as the first code in this record is equal to 2, all values generated will be written to a file. In this way it is possible to study the complete distribution of values for the slot being simulated. If this option is selected, the second additional file must indicate which files should be used to store the values in question. In the corresponding records of this file (also a direct access file with record length 100), the appropriate file name must be given after the first four positions have been left blank.

100	1	2	0.5								
101	0										
102	1	2	0.05								
103	2	3	0.05	0.1							
104	3		1.0								

Figure 4.6 Codes to be inserted in the additional auxiliary file for simulations.

In certain simulations it may be useful to fix a particular slot value. This can be done by inserting the code 3 into field no. 1 in the first additional file, followed in field no. 3 by the number to be used throughout the complete set of simulations. An example is given in record 104 in Figure 4.6.

When these codes have been inserted, the simulation process can be started as shown in Figure 4.7. The simulation module is activated by giving the number of simulations desired and clicking the ON button. If an item is selected from the basic problem menu, the simulation begins. The middle part of the main screen indicates the current number of repetitions in the simulation loop, whereas the upper part displays the results when the simulations have been completed.

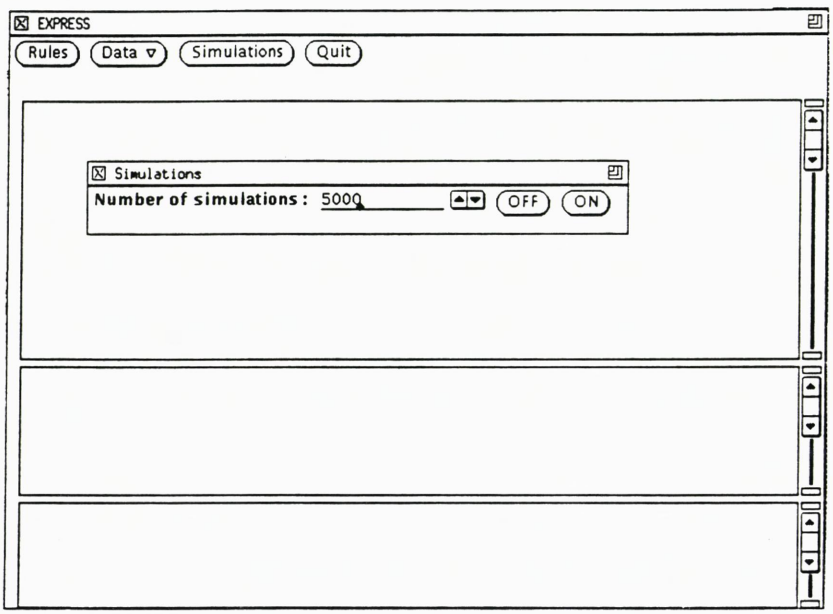


Figure 4.7 The menu for activating simulations.

Appendix A

SAMPLE SET OF RULES

A.1 INTRODUCTION 134

A.2 LISTING OF RULES 136

A.3 LISTING OF THE CODE FILE 141
A.3.1 Definition of slots 141
A.3.2 Codes for execution of packages 142
A.3.3 Commands for packages 142
A.3.4 Additional information for commands 142
A.3.5 Search keys 142
A.3.6 Menu for basic problems 143
A.3.7 Particular text for this set of rules 143

A.4 LISTING OF THE SAMPLE SESSION LOG FILE 144

A.1 INTRODUCTION

This appendix provides a complete example of a basic set of rules carrying out a simple one-way analysis of variance. This is the same set of rules as considered in the sample session in Section 2.7.1. The hypothesis of interest corresponds to identical location parameters for different samples. The ordinary F-test is applied if the samples are normally distributed and appear to have equal variances. Both the decision on normality and the decision concerning equality of variances are based on the data set. The normality assumption is checked using the Shapiro-Wilk test, while equality of variances is tested by the Levene test. If the normality assumption appears to be satisfied but not the assumption concerning equal variances, the Brown-Forsythe test, adapted to this situation, will be selected. Finally, if the assumption about normality fails, the non-parametric Kruskal-Wallis test will be used to address the basic question about location parameters.

To serve as an illustration, the strategy has been kept rather simple. A more extensive strategy might have involved, for example, multiple comparison tests of the various samples. Nonetheless, the strategy considered, based on several tests which are not independent, raises fundamental questions about the overall performance. Such issues are not dealt with here.

Figure A.1 shows a strategy map for the procedure outlined above. To decide whether all variables involved appear to be normally distributed, rule no. 4 is called several times. For this reason, various rules have been included for deciding on normality. Rule no. 2 merely starts the process, while rule no. 3 acts as a loop. Each time the loop is executed, this rule calls rule no. 4 to decide on normality for a particular variable. Rules marked with "P" can start an execution of the external package BMDP.

This simple strategy illustrates the flexibility of the rules. The loop facility may be especially important in other problems. The remaining part of this appendix provides a listing of the rules and the contents of the code file.

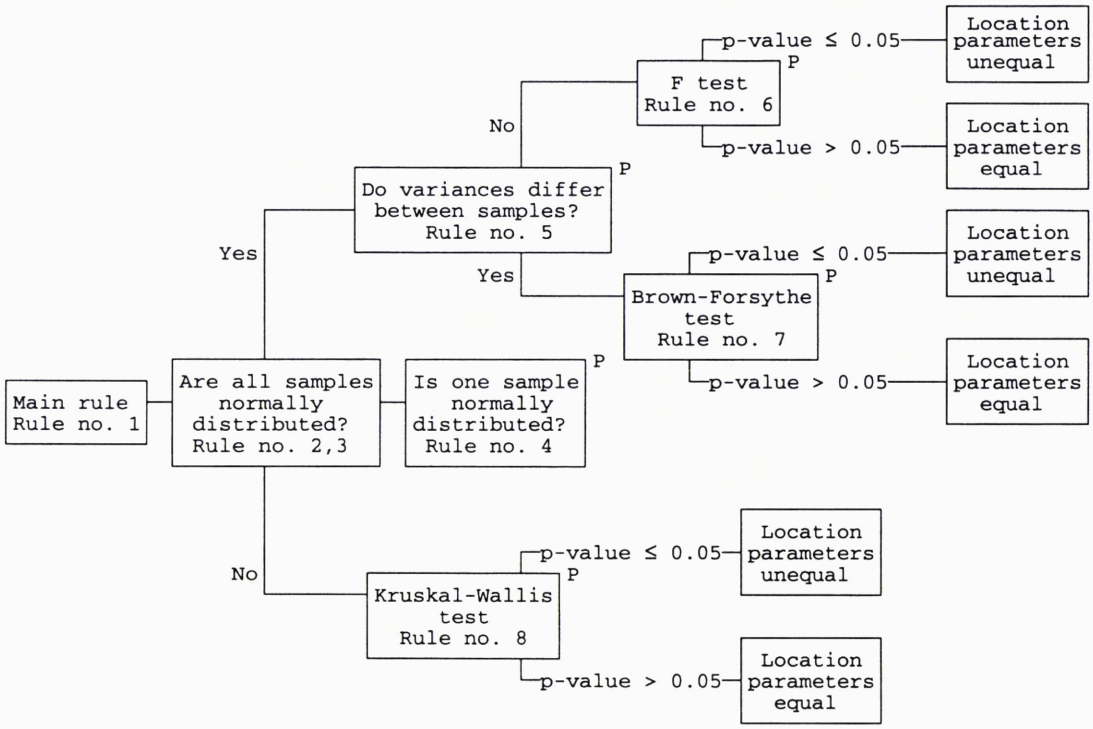


Figure A.1 Simple strategy map.

A.2 LISTING OF RULES

```
$STORAGE:2
      PROGRAM ONEWAY
C
C  Set of rules for one-way analysis of variance
C
      INTEGER*2 SNR,DATANR,POSTNR,HOPP,TIL,CODEREC
      LOGICAL SLUTT
C
      CALL INITRE()
      CALL HENTPAR(DATANR,POSTNR,CODEREC,HOPP)
1     CALL LESSTB(SNR,DATANR,SLUTT)
      IF (SLUTT) THEN
         CODEREC = 0
         GOTO 999
      ENDIF
C
10    CONTINUE
      GOTO(101, 102, 103,104, 105, 106, 107, 108,999,999,
     $1000,1000,1000) SNR
C
101   SNR=1
      CALL REGEN(SNR,DATANR,POSTNR,TIL)
      GOTO (1,10)TIL
C
102   SNR=2
      CALL REGTO(SNR,DATANR,POSTNR,TIL)
      GOTO (10,105,108)TIL
C
103   SNR=3
      CALL REGTRE(SNR,DATANR,POSTNR,HOPP,TIL)
      GOTO (1,10)TIL
C
104   SNR=4
      CALL REGFIR(SNR,DATANR,POSTNR,HOPP,TIL)
      GOTO (1,10)TIL
C
105   SNR=5
      CALL REGFEM(SNR,DATANR,POSTNR,TIL)
      GOTO (106,107,10)TIL
C
106   SNR=6
      CALL REGSEK(SNR,DATANR,POSTNR,TIL)
      GOTO (1)TIL
C
107   SNR=7
      CALL REGSJU(SNR,DATANR,POSTNR,TIL)
      GOTO (1)TIL
C
108   SNR=8
      CALL REGATT(SNR,DATANR,POSTNR,TIL)
      GOTO (1,10)TIL
C
1000  CODEREC = SNR - 10
999   CONTINUE
      CALL SENDPAR(DATANR,POSTNR,CODEREC,HOPP)
      CALL STENGRE()
      END
```

```
      SUBROUTINE REGEN(SNR,DATANR,POSTNR,TIL)
C
C  Main rule for the analysis.
C
      INTEGER*2 SNR,DATANR,POSTNR,TIL
      REAL VERDI
      LOGICAL SLUTT,FUNNET
C
      CALL PAASTB(SNR,DATANR)
      POSTNR = 700
      CALL LESGDB(POSTNR,VERDI,SNR,0,FUNNET)
      IF ( .NOT. FUNNET ) THEN
         TIL = 2
         GOTO 999
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END



      SUBROUTINE REGTO(SNR,DATANR,POSTNR,TIL)
C
C  Main rule for deciding if all variables are normally distributed.
C
      INTEGER*2 SNR,DATANR,POSTNR,TIL
      REAL VERDI
      LOGICAL SLUTT,FUNNET
C
      CALL PAASTB(SNR,DATANR)
      POSTNR = 701
      CALL LESGDB(POSTNR,VERDI,SNR,700,FUNNET)
      IF ( .NOT. FUNNET ) THEN
         CALL NYPOST(50,0.0,1,0,1)
         TIL = 1
         GOTO 999
      ELSE
         IF ( INT(VERDI) .EQ. 1 ) THEN
             TIL = 2
         ELSE
             TIL = 3
         ENDIF
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
999   CONTINUE
      END



      SUBROUTINE REGTRE(SNR,DATANR,RECORD,HOPP,TIL)
C
C  This rule generates a loop. Each time the loop is executed, the
C  assumption about normality is checked for a particular variable.
C
      INTEGER*2 SNR,DATANR,RECORD,TIL,CODE,INNE,MAXI,MINI,
     $NUMBER,HOPP
      REAL VALUE,USERVA
      LOGICAL SLUTT,FOUND
C
      CALL PAASTB(SNR,DATANR)
      CALL KODER(50,CODE,VALUE,USERVA)
      CALL NYPOST(50,VALUE+1.0,1,0,1)
      INNE = INT(VALUE)+1
      CALL ANTUT(MAXI,MINI,CODE,NUMBER)
C
10    CONTINUE
      IF ( INNE .LE. NUMBER ) THEN
         RECORD = 115 + (INNE - 1)*HOPP
         CALL LESGDB(RECORD,VALUE,SNR,700,FOUND)
         IF ( .NOT. FOUND ) THEN
            DATANR = INNE
            CALL SKR10(DATANR)
            TIL = 2
```

```
            GOTO 999
         ELSE
            INNE = INNE + 1
            CALL NYPOST(50,REAL(INNE),1,0,1)
            GOTO 10
         ENDIF
      ELSE
         KNORM = 1
         DO 20 I=1,NUMBER
            RECORD = 115 + (INNE - 1)*HOPP
            CALL KODER(RECORD,CODE,VALUE,USERVA)
            IF ( CODE .EQ. 3 ) VALUE = USERVA
            IF ( INT(VALUE) .EQ. 0 ) THEN
               KNORM = 0
            ENDIF
20       CONTINUE
         CALL NYPOST(701,REAL(KNORM),1,2,1)
      ENDIF
C
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END



      SUBROUTINE REGFIR(SNR,DATANR,POSTNR,HOPP,TIL)
      INTEGER*2 SNR,DATANR,POSTNR,TIL,VARNR(3),HOPP,MAALNR,CODE
      REAL VERDI1,VERDI2,NORMAL,VALUE,USERVA
      CHARACTER SKLEVL*1,KULEVL*1
      LOGICAL SLUTT,SANN,USANN,FUNNET
      SANN = .TRUE.
      USANN = .FALSE.
C
C  Rule that tests the normality assumption for a particular
C  variable.
C
      CALL PAASTB(SNR,DATANR)
      CALL HNTNR(1,6,DATANR)
      IF ( DATANR .EQ. 0 ) THEN
         TIL = 1
         GOTO 999
      ENDIF
C
C  Present histogram of the frequency.
C
      POSTNR = 106 + (DATANR - 1)*HOPP
      CALL LESGDB(POSTNR,VERDI1,SNR,0,FUNNET)
      IF ( .NOT. FUNNET ) THEN
         VARNR(1) = 1
         VARNR(2) = DATANR
         CALL VARFIL(VARNR,16+DATANR,SNR,SANN)
         CALL SKR10(DATANR)
         TIL = 2
         GOTO 999
      ENDIF
C
C Use the Shapiro-Wilk test to decide on normality.
C
      POSTNR = 114 + ( DATANR - 1 )*HOPP
      MAALNR = 115 + ( DATANR - 1 )*HOPP
      CALL LESGDB(POSTNR,VERDI,SNR,MAALNR,FUNNET)
      POSTNR =115 +(DATANR-1)*HOPP
      IF ( VERDI .LT. 0.05 ) THEN
         CALL TILGDB(POSTNR,0.0,SANN,701)
      ELSE
         CALL TILGDB(POSTNR,1.0,SANN,701)
      ENDIF
C
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END
```

```
      SUBROUTINE REGFEM(SNR,DATANR,POSTNR,TIL)
      INTEGER*2 SNR,DATANR,POSTNR,TIL,MAXI,MINI,CODE,VARNR(7)
      REAL VERDI1
      LOGICAL SLUTT,FUNNET,SANN
C
C  Rule that tests for equal variances.
C
      SANN = .TRUE.
      CALL PAASTB(SNR,DATANR)
      POSTNR=703
      DATANR=7
      CALL LESGDB(POSTNR,VERDI1,SNR,700,FUNNET)
      IF ( .NOT. FUNNET ) THEN
         CALL ANTUT(MAXI,MINI,CODE,VARNR(1))
         DO 10 I=1,VARNR(1)
            VARNR(I+1) = I
10       CONTINUE
         CALL SKR10(DATANR)
         CALL ENFIL(VARNR,20,2,SANN)
         TIL = 3
         GOTO 999
      ENDIF
      IF ( VERDI1 .GT. 0.05) THEN
         CALL TILGDB(699,1.0,SANN,700)
         TIL = 1
      ELSE
         CALL TILGDB(699,0.0,SANN,700)
         TIL = 2
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
999   CONTINUE
      END


      SUBROUTINE REGSEK(SNR,DATANR,POSTNR,TIL)
      INTEGER*2 SNR,DATANR,POSTNR,TIL
      REAL VERDI1
      LOGICAL SLUTT,SANN,USANN,FUNNET
      SANN = .TRUE.
      USANN = .FALSE.
C
C  This rule decides on the question of equal location parameters,
C  when the both assumptions concerning normality and equality of
C  variances are fulfilled.
C
      CALL PAASTB(SNR,DATANR)
      POSTNR=704
      CALL LESGDB(POSTNR,VERDI1,SNR,700,FUNNET)
      POSTNR=700
      IF (VERDI1 .LE. 0.05) THEN
         CALL TILGDB(POSTNR,1.0,USANN,0)
      ELSE
         CALL TILGDB(POSTNR,0.0,USANN,0)
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END
```

```
      SUBROUTINE REGSJU(SNR,DATANR,POSTNR,TIL)
      INTEGER*2 SNR,DATANR,POSTNR,TIL
      REAL VERDI1,VERDI2,HOYEST
      LOGICAL SLUTT,SANN,USANN,FUNNET
      SANN = .TRUE.
      USANN = .FALSE.
C
C  This rule decides on the question of equal location parameters
C  when the normality assumption is satisfied but not the assumption
C  concerning equality of variances.
C
      CALL PAASTB(SNR,DATANR)
C      POSTNR=705
C      CALL LESGDB(POSTNR,VERDI1,SNR,51,FUNNET)
      POSTNR=706
      CALL LESGDB(POSTNR,VERDI2,SNR,51,FUNNET)
      POSTNR=700
C      HOYEST = MAX(VERDI1,VERDI2)
      IF (VERDI2 .LE. 0.05) THEN
         CALL TILGDB(POSTNR,1.0,USANN,0)
      ELSE
         CALL TILGDB(POSTNR,0.0,USANN,0)
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END



      SUBROUTINE REGATT(SNR,DATANR,POSTNR,TIL)
      INTEGER*2 SNR,DATANR,POSTNR,TIL,MAXI,MINI,CODE,VARNR(7)
      REAL VERDI1
      LOGICAL SLUTT,FUNNET,USANN
C
C  This rule decides on the question of equal location parameters when the
C  normality assumption is not satisfied.
C
      USANN = .FALSE.
      CALL PAASTB(SNR,DATANR)
      POSTNR=707
      DATANR=7
      CALL LESGDB(POSTNR,VERDI1,SNR,700,FUNNET)
      IF ( .NOT. FUNNET ) THEN
         CALL ANTUT(MAXI,MINI,CODE,VARNR(1))
         DO 10 I=1,VARNR(1)
            VARNR(I+1) = I
10       CONTINUE
         CALL SKR10(DATANR)
         CALL ENFIL(VARNR,20,2,USANN)
         TIL = 2
         GOTO 999
      ENDIF
      POSTNR = 700
      IF (VERDI1 .LE. 0.05) THEN
         CALL TILGDB(POSTNR,1.0,USANN,0)
      ELSE
         CALL TILGDB(POSTNR,0.0,USANN,0)
      ENDIF
      CALL LESSTB(SNR,DATANR,SLUTT)
      TIL = 1
999   CONTINUE
      END
```

A.3 LISTING OF THE CODE FILE

A.3.1 Definition of slots

50			Help - number of variable (loop)							
101	1	23	Number of values in x1	1	11		0	-999	9999	All values
102	1	16	Skewness for x1	1	11		0	-999	9999	All values
103	1	19	Skewness/S.E. for x1	1	11		0	-999	9999	All values
104	1	16	Kurtosis for x1	1			0	-999	9999	All values
105	1	20	Kurtosis/S.E. for x1	1			0	-999	9999	All values
106	1	17	Histogram for x1	7	11					
107	1	11	Mean of x1	1	11		0	-999	9999	All values
108	1	17	The median of x1	1			0	-999	9999	All values
109	1	28	The standard deviation of x1	1			0	0	9999	[0,..)
110	1	19	The variance of x1	1			0	0	9999	[0,...)
111	1	39	Lower limit (95% conf.int of mean-x1)	1			0	-999	9999	All values
112	1	39	Upper limit (95% conf.int of mean-x1)	1			0	-999	9999	All values
113	1	39	W statistic for x1 (test of normality)	1			0	0	1	[0,1]
114	1	35	p-value for x1 - Test of normality	1			0	0	1	[0,1]
115	1	30	'Is x1 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}
201	1	23	Number of values in x2	1	11		0	-999	9999	All values
202	1	16	Skewness for x2	1	11		0	-999	9999	All values
203	1	19	Skewness/S.E. for x2	1	11		0	-999	9999	All values
204	1	16	Kurtosis for x2	1			0	-999	9999	All values
205	1	20	Kurtosis/S.E. for x2	1			0	-999	9999	All values
206	1	17	Histogram for x2	7	11					
207	1	11	Mean of x2	1	11		0	-999	9999	All values
208	1	17	The median of x2	1			0	-999	9999	All values
209	1	28	The standard deviation of x2	1			0	0	9999	[0,..)
210	1	19	The variance of x2	1			0	0	9999	[0,...)
211	1	39	Lower limit (95% conf.int of mean-x2)	1			0	-999	9999	All values
212	1	39	Upper limit (95% conf.int of mean-x2)	1			0	-999	9999	All values
213	1	39	W statistic for x2 (test of normality)	1			0	0	1	[0,1]
214	1	35	p-value for x2 - Test of normality	1			0	0	1	[0,1]
215	1	30	'Is x2 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}
301	1	23	Number of values in x3	1	11		0	-999	9999	All values
302	1	16	Skewness for x3	1	11		0	-999	9999	All values
303	1	19	Skewness/S.E. for x3	1	11		0	-999	9999	All values
304	1	16	Kurtosis for x3	1			0	-999	9999	All values
305	1	20	Kurtosis/S.E. for x3	1			0	-999	9999	All values
306	1	17	Histogram for x3	7	11					
307	1	11	Mean of x3	1	11		0	-999	9999	All values
308	1	17	The median of x3	1			0	-999	9999	All values
309	1	28	The standard deviation of x3	1			0	0	9999	[0,..)
310	1	19	The variance of x3	1			0	0	9999	[0,...)
311	1	39	Lower limit (95% conf.int of mean-x3)	1			0	-999	9999	All values
312	1	39	Upper limit (95% conf.int of mean-x3)	1			0	-999	9999	All values
313	1	39	W statistic for x3 (test of normality)	1			0	0	1	[0,1]
314	1	35	p-value for x3 - Test of normality	1			0	0	1	[0,1]
315	1	30	'Is x3 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}
401	1	23	Number of values in x4	1	11		0	-999	9999	All values
402	1	16	Skewness for x4	1	11		0	-999	9999	All values
403	1	19	Skewness/S.E. for x4	1	11		0	-999	9999	All values
404	1	16	Kurtosis for x4	1			0	-999	9999	All values
405	1	20	Kurtosis/S.E. for x4	1			0	-999	9999	All values
406	1	17	Histogram for x4	7	11					
407	1	11	Mean of x4	1	11		0	-999	9999	All values
408	1	17	The median of x4	1			0	-999	9999	All values
409	1	28	The standard deviation of x4	1			0	0	9999	[0,..)
410	1	19	The variance of x4	1			0	0	9999	[0,...)
411	1	39	Lower limit (95% conf.int of mean-x4)	1			0	-999	9999	All values
412	1	39	Upper limit (95% conf.int of mean-x4)	1			0	-999	9999	All values
413	1	39	W statistic for x4 (test of normality)	1			0	0	1	[0,1]
414	1	35	p-value for x4 - Test of normality	1			0	0	1	[0,1]
415	1	30	'Is x4 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}
501	1	23	Number of values in x5	1	11		0	-999	9999	All values
502	1	16	Skewness for x5	1	11		0	-999	9999	All values
503	1	19	Skewness/S.E. for x5	1	11		0	-999	9999	All values
504	1	16	Kurtosis for x5	1			0	-999	9999	All values
505	1	20	Kurtosis/S.E. for x5	1			0	-999	9999	All values
506	1	17	Histogram for x5	7	11					
507	1	11	Mean of x5	1	11		0	-999	9999	All values
508	1	17	The median of x5	1			0	-999	9999	All values
509	1	28	The standard deviation of x5	1			0	0	9999	[0,..)
510	1	19	The variance of x5	1			0	0	9999	[0,...)
511	1	39	Lower limit (95% conf.int of mean-x5)	1			0	-999	9999	All values
512	1	39	Upper limit (95% conf.int of mean-x5)	1			0	-999	9999	All values
513	1	39	W statistic for x5 (test of normality)	1			0	0	1	[0,1]
514	1	35	p-value for x5 - Test of normality	1			0	0	1	[0,1]
515	1	30	'Is x5 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}
601	1	23	Number of values in x6	1	11		0	-999	9999	All values
602	1	16	Skewness for x6	1	11		0	-999	9999	All values
603	1	19	Skewness/S.E. for x6	1	11		0	-999	9999	All values
604	1	16	Kurtosis for x6	1			0	-999	9999	All values
605	1	20	Kurtosis/S.E. for x6	1			0	-999	9999	All values
606	1	17	Histogram for x6	7	11					
607	1	11	Mean of x6	1	11		0	-999	9999	All values

608	1	17	The median of x6	1			0	-999	9999	All values	
609	1	28	The standard deviation of x6	1			0	0	9999	[0,..)	
610	1	19	The variance of x6	1			0	0	9999	[0,...)	
611	1	39	Lower limit (95% conf.int of mean-x6)	1			0	-999	9999	All values	
612	1	39	Upper limit (95% conf.int of mean-x6)	1			0	-999	9999	All values	
613	1	39	W statistic for x6 (test of normality)	1			0	0	1	[0,1]	
614	1	35	p-value for x5 - Test of normality	1			0	0	1	[0,1]	
615	1	30	'Is x5 normally distributed?'	4	4		1	0	1	{1-Yes, 0-No}	
700	1	33	'Do location parameters differ?'	2	2		1	0	1	{1-Yes, 0-No}	
701	1	28	'Are all variables normal?'	4	3		1	0	1	{1-Yes, 0-No}	
702	1	31	'Are the variances identical?'	2	4		1	0	1	{1-Yes, 0-No}	
703	1	32	p-value for equal variances	1	12		0	0	1	[0,1]	
704	1	36	p-value for equal groups (ordinary)	1	12		0	0	1	[0,1]	
705	1	33	p-value for equal groups (Welsh)	1	12		0	0	1	[0,1]	
706	1	33	p-value for equal groups (Brown)	1	12		0	0	1	[0,1]	
707	1	32	p-value for Kruskal-Wallis test	1	13		0	0	1	[0,1]	

A.3.2 Codes for execution of packages

900	99999999			1000	6	99999999999999	1200	100		6	
901	9999999			1012	12	9999	1220				
902	99999999			1022	13	9	1230				

A.3.3 Commands for packages

1000	1180		/INPUT FILE = '%0'.
1001	1		FORMAT = FREE.
1002	1		VARIABLES = 1.
1003	1181		/VARIABLE NAMES = %0.
1004	1		/PRINT WSTAT.
1005	1		NO CONT.
1006	1		LINESIZE = 80.
1007	1		/END
1008	1184	0	/INPUT FILE = '%0'.
1009	1	0	VARIABLES = 2.
1010	1	0	FORMAT = FREE.
1011	1182	0	/VARIABLE NAMES = %0, %1.
1012	1182	0	/HISTOGRAM GROUPING IS %1.
1013	1182	0	VARIABLE IS %0.
1014	1	0	/END
1015	1184	0	/INPUT FILE = '%0'.
1016	1	0	VARIABLES = 2.
1017	1	0	FORMAT = FREE.
1018	1182	0	/VARIABLE NAMES = %0, %1.
1019	1182	0	GROUPING = %1.
1020	1182	0	/TEST VARIABLE = %0.
1021	1	0	KRUSKAL.
1022	1	0	/END

A.3.4 Additional information for commands

1180	10	11	16	(4X,I1,A40)			
1181	10	11	10	(4X,I1,A6)			
1182		11	20	(4X,I1,40X,A6)			
1183		11	20	(4X,I1,46X,A6)			
1184		11	20	(4X,I1,A40)			

A.3.5 Search keys

1200	CASES READ	00	COUNTED. .	0	101							
1201			SKEWNESS	0	102							
1202			SKEWNESS	1	103							
1203			KURTOSIS	0	104							
1204			KURTOSIS	1	105							
1205			KURTOSIS	1	106				3	12	1	80
1206			MEAN	0	107							
1207			ST.DEV.	0	109							
1208			MEDIAN	0	108							
1209			VARIANCE	0	110							
1210			LOWER 95%	3	111							
1211			UPPER 95%	3	112							
1212			W STATISTIC	0	113							
1213			SIGNIFICANCE	1	114							
1214	HISTOGRAM		PROBABILITY	2	704				1		-11	0
1215			WELCH	3	705							
1216			FORSYTHE	3	706							
1217			LEVENE'S	6	703							
1218	KRUSKAL		SIGNIFICANCE	1	707							

A.3.6 Menu for basic problems

```
1300|   3|  61|1. Decide whether location parameters differ between samples|  1|   |
1301|    |  56|2. Determine whether a variable is normally distributed     |  3|   |
1302|    |  51|3. Decide whether variances differ between samples          |  4|   |
```

A.3.7 Particular text for this set of rules

```
1310|   1|  19|The stack is empty
1311|   1|  10|Main rule
1312|   1|  50|Rule activating normality tests for all variables
1313|   1|  49|Rule deciding on normality for a single variable
1314|   1|  46|Rule deciding whether variances are identical
1315|   1|  69|Rule for equality of loc.param. under normality and homoscedasticity
1316|   1|  71|Rule for equality of loc.param. under normality and heteroscedasticity
1317|   1|  61|Rule for equality of location parameters under non-normality
1318|    |    |
1319|    |    |
1320|    |    |
1321|   1|  16|Executing BMDP.
1322|   1|  16|Executing BMDP.
1323|   1|  16|Executing BMDP.
    .    .    .
    .    .    .
1350|   1|  45|Select the first variable, referred to as x1.
1351|   1|  46|Select the second variable, referred to as x2.
1352|   1|  45|Select the third variable, referred to as x3.
1353|   1|  46|Select the fourth variable, referred to as x4.
1354|   1|  45|Select the fifth variable, referred to as x5.
1355|   1|  45|Select the sixth variable, referred to as x6.
```

A.4 LISTING OF THE SAMPLE SESSION LOG FILE

```
ACTIVATING RULE
NO.:   1 (Main rule ).
RULES REMAINING ON THE STACK:     1

Considering slot:  'Do location parameters differ?'. The corresponding slot
value has not yet been determined.
The system will attempt to find its value.


ACTIVATING RULE
NO.:   2 (Rule activating normality tests for all variables ).
RULES REMAINING ON THE STACK:     1   2


ACTIVATING RULE
NO.:   3 (Rule deciding on normality for a single variable ).
RULES REMAINING ON THE STACK:     1   2   3

Considering slot:  Histogram for x1. The corresponding slot value has not
yet been determined.
The system will attempt to find its value.
Please note: Some packages will during execution destroy
      the screen produced by EXPRESS, but this screen will
      be restored when the execution is complete.
The following package is executed: BMDP 2D.

        THE FOLLOWING COMMANDS WERE APPLIED:

/INPUT FILE = 'C:\EXPRESS\SYSFIL\617.PXE'.
          FORMAT = FREE.
          VARIABLES = 1.
/VARIABLE NAMES = ONEW1.
/PRINT WSTAT.
          NO CONT.
          LINESIZE = 80.
/END


In the output file produced by the execution of the external package,
EXPRESS has used the following search keys:

          SEARCH KEYS          VALUES EXTRACTED

          COUNTED. .            20
          SKEWNESS             -0.58
          SKEWNESS             -1.051
          KURTOSIS              0.53
          KURTOSIS              0.487
          KURTOSIS             A plot or table has been extracted.
          MEAN                  0.3456730
          ST.DEV.               0.7434019
          MEDIAN                0.4061500
          VARIANCE              0.5526465
          LOWER 95%            -0.0022493
          UPPER 95%             0.6935953
          W STATISTI            0.9355
          SIGNIFICAN            0.2074

ACTIVATING RULE
NO.:   3 (Rule deciding on normality for a single variable ).
RULES REMAINING ON THE STACK:     1   2   3


                    Histogram for x1


                EACH 'H' REPRESENTS      1 COUNT
                EACH '-' REPRESENTS  .250000 UNITS

       H H      L = -1.50000
       H H      U =  1.75000
      HH HH
      HH HH  H
 H H  HH HH  H
L-------------U
```

```
The external package has found:
p-value for x1 - Test of normality     = 0.2074000
This slot is needed to determine: 'Is x1 normally distributed?'

The system has reached the following conclusion
to the question 'Is x1 normally distributed?'
The answer to this question is Yes!
This slot is needed to determine: 'Are all variables normal?'

ACTIVATING RULE
NO.:   2 (Rule activating normality tests for all variables ).
RULES REMAINING ON THE STACK:     1   2

The user has reached the following conclusion
to the question 'Is x2 normally distributed?'
The answer to this question is Yes!
The system has not yet determined this slot value.

The user has reached the following conclusion
to the question 'Is x3 normally distributed?'
The answer to this question is Yes!
The system has not yet determined this slot value.

ACTIVATING RULE
NO.:   1 (Main rule ).
RULES REMAINING ON THE STACK:     1

Considering slot:  'Do location parameters differ?'. The corresponding slot
value has not yet been determined.
The system will attempt to find its value.

The system has reached the following conclusion
to the question 'Are all variables normal?'
The answer to this question is Yes!
This slot is needed to determine: 'Do location parameters differ?'

Considering slot:  'Are the variances identical?'. The corresponding slot
value has not yet been determined.
The system will attempt to find its value.

ACTIVATING RULE
NO.:   4 (Rule deciding whether variances are identical ).
RULES REMAINING ON THE STACK:     1   4

Considering slot:  p-value for equal variances. The corresponding slot
value has not yet been determined.
The system will attempt to find its value.
Please note: Some packages will during execution destroy
      the screen produced by EXPRESS, but this screen will
      be restored when the execution is complete.
The following package is executed: BMDP 7D.

        THE FOLLOWING COMMANDS WERE APPLIED:

/INPUT FILE = 'C:\EXPRESS\SYSFIL\620.PXE'.
       VARIABLES  =  2.
      FORMAT = FREE.
/VARIABLE NAMES = VAR20D, GRP20D.
/HISTOGRAM GROUPING IS GRP20D.
         VARIABLE IS VAR20D.
/END


In the output file produced by the execution of the external package,
EXPRESS has used the following search keys:

          SEARCH KEYS          VALUES EXTRACTED

          PROBABILIT          0.0381
          WELCH               0.0788
          FORSYTHE            0.0390
          LEVENE'S            0.1021

ACTIVATING RULE
NO.:   4 (Rule deciding whether variances are identical ).
RULES REMAINING ON THE STACK:     1   4

The external package has found:
p-value for equal variances     = 0.1021000
This slot is needed to determine: 'Are the variances identical?'
```

```
The system has reached the following conclusion
to the question 'Are the variances identical?'
The answer to this question is Yes!
This slot is needed to determine: 'Do location parameters differ?'

ACTIVATING RULE
NO.:   1 (Main rule ).
RULES REMAINING ON THE STACK:     1

Considering slot:  'Do location parameters differ?'. The corresponding slot
value has not yet been determined.
The system will attempt to find its value.

The system has reached the following conclusion
to the question 'Are all variables normal?'
The answer to this question is Yes!
This slot is needed to determine: 'Do location parameters differ?'

The system has reached the following conclusion
to the question 'Are the variances identical?'
The answer to this question is Yes!
This slot is needed to determine: 'Do location parameters differ?'

ACTIVATING RULE
NO.:   5 (Rule for equality of loc.param. under normality and
homoscedasticity  ).
RULES REMAINING ON THE STACK:     1   5

The external package has found:
p-value for equal means (ordinary F)     = 0.03810000
This slot is needed to determine: 'Do location parameters differ?'

ACTIVATING RULE
NO.:   1 (Main rule ).
RULES REMAINING ON THE STACK:     1

The system has reached the following conclusion
to the question 'Do location parameters differ?'
The answer to this question is Yes!
```

Appendix B

FILES INCLUDED IN EXPRESS

B.1 FILES IN EXPRESS 148
B.1.1 Files related to the sets of rules 148
B.1.2 Main files of the Express system 151

B.1 FILES IN EXPRESS

A short description is given below of all files downloaded to the hard disk during installation of Express. The files are divided into two different groups. The files in the first group are associated with one or several of the sets of rules. These are files a knowledge engineer must be familiar with when he creates a new set of rules or modifies an old one. The second group includes files which can be considered part of the Express system itself. They are merely listed for the sake of completeness and cannot be changed by the knowledge engineer.

B.1.1 Files related to the sets of rules

Path: C:\EXPRESS\SYSFIL

NYSYS.PXE	System file.
ETEKST.PXE	Text file.
EVINDU.PXE	File containing definition of windows.
FREX*.PXE	Files containing the data storage.
HJELP.PXE	File containing general help to Express.
EDHJELP.PXE	File containing help to the editor of Express.
BIBLIO.PXE	File containing the dictionary of Express.
STYR.PXE	File containing the commands to be used in an external package.
STYRTO.PXE	As STYR.PXE but this is a direct file (used when commands are presented to the user).
TABTO.PXE	File containing the stack and slots.
UTSKRIFT.PXE	File containing output.
BMDP*.BAT	Files executing different BMDP programs.
MINITAB.BAT	File executing MINITAB.
UT.BAT	File needed for executing MINITAB (see Figure 3.12).
SAS.BAT	File executing SAS.
STAT.BAT	File executing STATXACT.
DOS.BAT	File executing DOS.

Certain technical files are used internally by the main program of Express only: BOSS.PXE, EXTALL.PXE and ASMBAT.PXE.

New files will be generated in this directory when different sets of rules are executed. These are data files containing variables used during a session, which are read by external packages. The files are in ASCII format, with names composed by numbers and the extension .PXE.

The table below lists the files associated with each set of rules and indicates where the files

are located. In addition to the files in the table, each set of rules includes one or more executable files (*.EXE). Actually, Logistrule is the only set of rules with more than a single executable file. Logistrule has six such files, four located in C:\EXPRESS\LOGREG and the additional two in C:\EXPRESS\LOGREG\TOLKNING.

Finally, four data files with the extension .DAT are located in the directory C:\EXPRESS. These contain data used in sample sessions.

Set of rules	Code file	Data base	Plot and table file	Log file	Relation-ship file	Output file	Help file
One-way ANOVA Path: C:\EXPRESS\ONEWAY	CODE.PXE	GDB.PXE	PLOT.PXE	LOG.PXE	CONECT.PXE	OUTPUT.PXE	ONEWAY.PXE
Regression Path: C:\EXPRESS\REGREG	EKODE.PXE	GDBREG.PXE	PLOTTREG.PXE	LOGREG.PXE	TABREG.PXE	UTSKRREG.PXE	
Two sample comparison Path: C:\EXPRESS\SAMMEN	ECODE.PXE	GDB.PXE	PLOTT.PXE	LOG.PXE	TABELL.PXE	DIRUT.PXE	SAMTO.RFA
Logistrule Path: C:\EXPRESS\LOGREG	CODE.PXE	GDB.PXE	PLOT.PXE	LOG.PXE	TABLE.PXE	OUTPUT.PXE	FORKLAR.PXE
Egenkjor Path: C:\EXPRESS\EGENKJOR	KODE.PXE	GDB.PXE	PLOTT.PXE	LOG.PXE	TABELL.PXE	DIRUT.PXE	
Prove Path: C:\EXPRESS\PROVE	EKODE.PXE	GDB.PXE	PLOTT.PXE	LOG.PXE	TABELL.PXE	DIRUT.PXE	

B.1.2 Main files of the Express system

The files for the main system of Express are:

C:\EXPRESS\EXPRESS.COM
C:\EXPRESS\EXPSTART.EXE
C:\EXPRESS\INSTALL.BAT
C:\EXPRESS\NYSETT.EXE
C:\EXPRESS\MODE.EXE
C:\EXPRESS\EDITOR\ED.EXE
C:\EXPRESS\EDITOR\EDSYS.EXE
C:\EXPRESS\LAGRE\LAGRE.EXE
C:\EXPRESS\MENY\MENY.EXE
C:\EXPRESS\MENY\BIBLIO\BIBLIO.EXE
C:\EXPRESS\MENY\DATA\DATA.EXE
C:\EXPRESS\MENY\SETVAL\SETVAL.EXE
C:\EXPRESS\VELKOM\VELKOM.EXE