



Durham E-Theses

A knowledge-based system for the estimation of geotechnical properties.

Giolas, Antonis

How to cite:

Giolas, Antonis (1994) *A knowledge-based system for the estimation of geotechnical properties.*, Durham theses, Durham University. Available at Durham E-Theses Online: <http://etheses.dur.ac.uk/964/>

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

**A KNOWLEDGE-BASED SYSTEM FOR THE ESTIMATION
OF GEOTECHNICAL PROPERTIES**

A thesis submitted to the

**School of Engineering
University of Durham**

for the degree of

Doctor of Philosophy

by

Antonis Giolas

December 1994

**A KNOWLEDGE-BASED SYSTEM FOR THE ESTIMATION
OF GEOTECHNICAL PROPERTIES**

A thesis submitted to the

School of Engineering

University of Durham

The copyright of this thesis rests with the author.

No quotation from it should be published without
his prior written consent and information derived

from it should be acknowledged.

for the degree of

Doctor of Philosophy

by

Antonis Giolas

December 1994

To my parents

DECLARATION

I hereby declare that the work reported in this thesis has not been previously submitted for any degree. All material in this thesis is original except where indicated by reference to other work.

STATEMENT OF COPYRIGHT

The copyright of this thesis rests with the author. No quotation from it should be published without his prior written consent and information derived from it should be acknowledged.

ABSTRACT

Ground properties are mainly evaluated from direct measurements obtained from either laboratory or field tests. As an alternative, or in conjunction with test measurements, ground properties can also be estimated from correlations and published summaries of "typical" values. The advantages of their use are that both are simple, easy to use and they provide a cheap, if crude, means for the rapid estimation of ground properties.

A knowledge-based system has been developed to provide a tool for storing and using correlations and "typical" values for the estimation of ground properties. The system was implemented in the ProKappa software, running under X windows on a Sun Spark 2 workstation. The system developed is intended to provide geotechnical engineers with a decision-support tool and to demonstrate the applicability of knowledge-based system technology to the *ground property evaluation problem*.

The development of this system involved the identification and collection of the domain knowledge (knowledge associated with the ground, ground properties, "typical" values and correlations, which was elicited from the relevant technical literature). Generic forms for the representation of correlations and "typical" values in the system were developed (which provide a consistent form for the representation).

The system also incorporates an inference engine, which includes the process that makes use of this knowledge to produce estimations for ground properties; a user interface to facilitate the use of the system; and finally knowledge acquisition modules for updating of existing knowledge, to ensure that the system will maintain its functionality in the future.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr David Toll, for his continuous support and guidance throughout the course of my work. His constant supply of ideas in our conversations was an significant source of inspiration.

This work was partially funded by the Greek NATO Fellowships programs and their contribution is acknowledged.

Also I would like to thank all the members of staff of the School of Engineering and in particular Professor P. B. Attewell for both his social and academic support. Many thanks also to Bernard McEleavey, Steve Richardson and Brian Scurr for their help during the laboratory demonstration, and to Trevor Nancarrow for his contribution in overcoming many software and hardware problems.

Many thanks to my cousin, football and basketball partner Nikitas Vaptismas, the sweet and laid-back Marina Moula, the caring and ever supporting Dora Robi, the round mount of rebound and king of the sting Yianni Giola, Panagiotis "the boss" Dounis and to captain Ilias Papadimitropoulos. Furthermore to Andy Oliver for his tireless help during the development of the program, but moreover for his friendship and his notorious psychoanalytical abilities.

Thanks also to all the lads from the football and basketball teams for their companionship and especially to Raz "main man" Ashraf (respect), "cool" Fanis Antoniou, Evangelos "backshot" Mavrikis and Csilla "sweet wrist" Szabo. In addition my thanks go to Christian Habeck, Sandra (Sandrula) Rothwell, Vicky Malandraki, Panos Kokkonis, Rory Barr, Anna Korea, Adnan Bashir, Alex Iskandar and Haluk Bayracdar, whose friendship made my residence in Durham more

interesting. Special thanks to Andrew "get one together" Shrubbsall for preparing the knot of the tie.

Finally, I dedicate this thesis to my parents, whose care, support, but most important love, helped me to realise all my plans so far. I want you both to know that you are very special to me and I love you.

CONTENTS

CHAPTER 1.....	1
Introduction.....	1
1.1 General.....	1
1.2 Overview of the thesis.....	3
CHAPTER 2.....	6
The application of Knowledge-Based Systems to the estimation of ground properties.....	6
2.1 Introduction.....	6
2.2 KBS components.....	8
2.3 Development stages of KBS.....	9
2.4 KBS in Geotechnical Engineering.....	15
2.5 Discussion.....	20
2.6 Conclusions.....	22
CHAPTER 3.....	23
Development tools.....	23
3.1 Introduction.....	23
3.2 The ProKappa software.....	23
3.2.1 Objects, slots, facets and applications.....	24
3.2.2 Programming languages.....	27
3.2.3 User interface development tools.....	31

3.2.4 Monitors and Active Relations.....	34
3.3.5 Summary.....	38
CHAPTER 4.....	39
Representing the ground and its properties.....	39
4.1 Introduction.....	39
4.2 A model for representing the ground.	41
4.2.1 Rocks.....	42
4.2.2 Soils.....	46
4.3 A model for representing ground parameters.	49
4.4 Implementation in the system.	54
4.5 The representation of "typical" values.	56
4.6 User interface facilities.....	62
4.7 A knowledge acquisition module for typical values.....	69
4.8 Summary.....	78
CHAPTER 5.....	81
Representing correlations in a structured form.	81
5.1 Introduction.....	81
5.2 Representing correlations in a structured form.....	85
5.2.1. Variables.....	85
5.2.2. The estimation procedure.....	88
5.2.3. Applicability.....	88
5.2.4. Reliability.....	90
5.2.5. Comments.....	92
5.3 Implementation in the system.....	92
5.3.1 The Correlation application.....	93

Introduction	93
Variable slots	95
Parameter slots.....	97
Data_Check! slot.....	100
The Estimation Procedure	104
Applicability Slots	105
Reliability	106
5.3.2 The Correction module	106
5.3.3 The CorrUI module.....	107
5.4 Summary.....	119
CHAPTER 6.....	123
A knowledge acquisition module for the implementation of correlations.	123
6.1 Introduction.....	123
6.2 Establishment of the basic parameter.....	124
6.3 Variables and parameters.....	129
6.4 Implementation of estimation procedures.	140
6.5 Applicability definition.	145
6.6 Reliability definition.	151
6.7 Comments.	152
6.8 Updating correlations.	152
6.9 Overview of the Update module.	154
CHAPTER 7.....	156
Discussion - Future development	156
7.1 Discussion.....	156
7.1 Future development.....	162

CHAPTER 8.....	165
Conclusions	165
References	167
Appendix A	A-1
Correlations and Corrections.....	A-1
Part 1: Correlations.	A-1
Part 2: Corrections	A-38
References.....	A-43
Appendix B	B-1
Typical values for ground properties.....	B-1
Appendix C	C-1
The ProTalk Code.....	C-1

LIST OF TABLES

Table 4.1 The representation scheme for ground parameters.....	51
Table 4.2 The facets of the <i>PHI_{peak}</i> slot of the poorly graded sand, very loose well graded sand and angular dense well graded sand objects.....	61
Table 5.1 Basic variables slots and their facets.....	97
Table 5.2 Intermediate parameters slots and their facets.....	99
Table 5.3 An example of a one to one representation for a three variable correlation	102

LIST OF FIGURES

Figure 4.1 Representation of sedimentary rocks.....	44
Figure 4.2 Representation of igneous and metamorphic rocks.....	45
Figure 4.3 Representation scheme for soils	48
Figure 4.4 The slots and facets of the Su_CIU C and Plas objects.....	55
Figure 4.5 The representation of "typical" values of peak effective angle of friction, <i>PHI_{peak}</i> for pure clay	58
Figure 4.6 The object hierarchy for the representation of typical values of angle of friction for clean sand (depending on <i>grading</i> , <i>relative density</i> and <i>angularity</i>).....	61
Figure 4.7 Figure 4.7 The "menu" dialog box..	63
Figure 4.8 The dialog box for browsing the ground types object base.	64
Figure 4.9 A dialog box displaying typical values of dry density for silty sands.....	67
Figure 4.10 The "Parameters search" dialog box.	68
Figure 4.11 The dialog box for selecting parameters to update.....	70
Figure 4.12 The dialog box for specifying the parameters to implement.	71
Figure 4.13 The dialog box for selecting required parameters.	71
Figure 4.14 The dialog box for assigning a required parameter.	72
Figure 4.15 The facets of the <i>UserData</i> slot.....	74
Figure 4.16 The "Required parameters preview" dialog box.	75
Figure 4.17 The dialog box for implementing typical values of the coefficient of volume compressibility, <i>mv</i> , for a high plasticity pure clay	76
Figure 5.1 The slots of the Su7LI correlation object.....	94
Figure 5.2 The facets of the <i>LI</i> slot of the Su7LI correlation object.....	95

Figure 5.3 The facets of the Parameter slot of the Su7LI correlation object	98
Figure 5.4 The representation of applicability	106
Figure 5.5 The "search" dialog box.....	108
Figure 5.6 The dialog box for specifying the correlation's/correction's parameters	109
Figure 5.7 The search settings preview dialog box.....	111
Figure 5.8 The Preview search results dialog box	112
Figure 5.9 The display correlations/corrections dialog box	112
Figure 5.10 A correlation dialog box	113
Figure 5.11 An example of the "Applicability" dialog box.....	117
Figure 5.12 An example of the "Reliability" dialog box.....	118
Figure 5.13 An example of the "Comments" dialog box.....	119
Figure 6.1 The Step1 dialog box.....	124
Figure 6.2 Parameter information display dialog boxes.....	126
Figure 6.3 The dialog box for defining a new qualitative parameter.....	127
Figure 6.4 The dialog box for defining a new quantitative parameter.....	127
Figure 6.5 The Step2 dialog box.....	129
Figure 6.6 The dialog box for specifying applicability range of a quantitative variable.....	131
Figure 6.7 The dialog box for specifying the permissible values of a qualitative variable	132
Figure 6.8 The dialog box for implementing quantitative intermediate variables	134
Figure 6.9 The dialog box for implementing quantitative intermediate parameters	136
Figure 6.10 An example for the Show Preview dialog box.....	137
Figure 6.11 Estimation procedure definition for intermediate variables.....	140
Figure 6.12 The dialog box displaying mathematical functions in ProTalk	142

Figure 6.13 The "estwin" dialog box.....	144
Figure 6.14 The dialog box displaying the permissible values of compressibility	145
Figure 6.15 The dialog box for defining applicable ground types.....	146
Figure 6.16 The dialog box for selecting parameters for applicability restrictions.....	147
Figure 6.17 The dialog box for defining applicability restrictions for a quantitative parameter.....	148
Figure 6.18 The dialog box for defining applicability restrictions for a qualitative parameter	148
Figure 6.19 The "Preview Applicability definition" dialog box.....	150
Figure 6.20 The "Reliability definition" dialog box.....	151
Figure 6.21 The dialog box for implementing comments	152

CHAPTER 1

Introduction.

1.1 General.

Geotechnical engineering aims to describe the behaviour and performance of the ground as a construction material. The assessment of the engineering behaviour of the ground requires the evaluation of its properties. This in turn can be achieved by means of the following methodologies [Kulhawy, 1992]:

- direct measurements, obtained from either laboratory or field tests;
- back analysis of either reduced or full-scale loading tests (for performance properties; e.g. strength or deformation);
- from the measurement of other ground properties, through established empirical correlations based on statistical interpretations of observations in similar past cases;
- by using published summaries for the estimation of "typical", "average", or "representative" values;
- by using well-founded theoretical models (e.g. using Cam clay to estimate the properties of a real clay);
- and by using assumptions of material behaviour within a particular model, which then dictates the values of other properties (e.g. assuming undrained saturated isotropic linear behaviour to infer that the Poisson's ratio, $\nu = 0.5$ and the elastic modulus, $E = 3G$).



The most common way for evaluating ground properties is through geotechnical testing (either in-situ or in the laboratory). Geotechnical testing is probably the most reliable source of information for ground properties, but it is also a very costly and time consuming process. Furthermore the testing conditions may not simulate the ground conditions in-situ and errors may occur as a result of ground disturbance or sample cutting, storage and preparation etc. The same applies to the other methodologies. Therefore, a combined approach to the property evaluation problem would provide more certainty in the prediction and also the ability to cross-check the results of different methodologies.

Extensive geotechnical testing of specific ground types (having specific geological origins) has led to the accumulation of empirical knowledge about their properties. The application of statistical techniques to this knowledge has led to the establishment of "typical" ranges of values of a property for a specific ground type and also empirical relations between ground properties have been defined within a specific ground type. The former are described as "typical" values of ground properties and the latter as correlations between ground properties. Both are used as means for estimating ground properties. Despite the uncertainties associated with their use they provide a cheap and quick means for assessing ground properties and cross-checking geotechnical test results.

The work described here focuses on the use of correlations and published summaries of "typical" values. The aim of this work is to provide a framework for storing and using correlations and "typical" values for the estimation of ground properties. It is thought that this framework, which should be used in conjunction with the other methodologies, will provide a geotechnical engineer with a decision-support tool in the property evaluation problem.

Furthermore the project presented in this thesis aims to demonstrate the applicability of knowledge-based system technology to the area of ground properties estimation from correlations and "typical" values. Knowledge-based system technology provides a medium that can accommodate the representation and use of the knowledge required for the estimation of ground properties.

A knowledge-based system has been developed to provide a tool for storing and using empirical knowledge for the estimation of ground properties. The development of this system involved: the identification, collection and representation of the domain knowledge (relevant to the ground, ground properties, "typical" values and correlations); the implementation of the process that makes use of this knowledge (development of the inference engine); the design and implementation of a user interface to facilitate the use of the system; and finally the design and implementation of knowledge acquisition modules for updating the existing knowledge, to ensure that the system will maintain its usefulness in the future.

In the following section a brief description of the remaining chapters in this thesis is presented.

1.2 Overview of the thesis.

An introduction to knowledge-based system technology is presented in Chapter 2, followed by a review of existing applications in the area of ground properties estimation. The chapter concludes with a general discussion on the development of these applications.

The selection of the development tools (hardware and software) is discussed in Chapter 3. In the last part of the chapter, a detailed description of the ProKappa software (the software used) is presented. It is intended that the reader will become familiar with ProKappa concepts and terminology which are subsequently used in the remaining chapters.

The design and implementation of three of the system's knowledge bases is presented in Chapter 4. These are the Ground, Ground parameters and "Typical" values of ground properties knowledge bases. The chapter concludes with a detailed presentation of the user interface facilities relevant to the use of these knowledge bases and the knowledge acquisition module for "typical" values (including example consultations with the system).

A discussion on the usefulness and limitations from the use of correlations for the estimation of ground properties is presented in the first section of Chapter 5. This is followed by the description of a formal way of representing correlations. Finally, the last section of the chapter is concerned with the description of the part of the system for storing correlations and the user interface facilities for using correlations to infer estimations of ground properties.

In Chapter 6 a detailed description of the knowledge acquisition procedures for implementing new correlations in the system, as well as for updating those already implemented is presented.

Chapter 7 consists of a general discussion of the work presented in this thesis. The main features of the system developed are briefly reviewed followed by recommendations for future improvements. The conclusions resulting from the development of the knowledge-based system for the estimation of ground properties

are presented in Chapter 8. A list of references included within the main text, follows Chapter 8.

Appendix A contains the correlations stored in the system, including a separate reference list. "Typical" values of ground properties are included in Appendix B. Finally, the ProTalk code of the system for the estimation of ground properties is presented in Appendix C.

CHAPTER 2

The application of Knowledge-Based Systems to the estimation of ground properties.

2.1 Introduction.

Geotechnical engineering is a discipline of civil engineering, dealing with the properties of the earth's crust as they relate to construction. The inherent variability of the ground, both spatially and in time, along with the large number of factors affecting ground conditions, and the complexity arising from their combination, have made geotechnical engineering rely heavily on practical experience and empirical knowledge.

Also it is common, in problems related to the ground, that the engineer has to cope with incomplete and/or uncertain data and methodologies. These problems are usually termed ill-defined or ill-structured problems, and sound engineering judgement based on past experience is again a crucial factor.

The knowledge of an area of expertise is generally of two types: the facts of the domains, which is commonly shared declarative knowledge, and the heuristic knowledge, which is knowledge that constitutes the rules of expertise, the rules of good practice, the judgement rules of the field, and the rules of plausible reasoning [Feigenbaum, 1983]. Because of its nature, formalisation of heuristic knowledge using algorithmic programming techniques, has never been entirely successful.

Furthermore, procedural programs can not deal with incomplete data, or ill-defined problems.

Therefore, a need for computer programs that will be able to incorporate heuristic knowledge arises, which can handle ill-defined problems with incomplete and/or uncertain data.

Knowledge-based systems (KBS) are computer programs that contain domain specific knowledge and inference procedures for the solution of ill-structured problems. If these systems operate at an expert's level (simulating the expert's reasoning), they are termed expert systems. Mullarkey [1987], notices that the term knowledge-based systems is a more accurate descriptor of most current systems, since a complete representation of the expert's reasoning scheme is a very complex procedure.

Various other descriptions exist in the literature [Adeli, 1988], but most of them do not necessarily distinguish knowledge-based systems from many conventional programs. Maher and Allen [1987] present some of the distinguishing characteristics. The most important of these are:

- Separation of knowledge and control in KBS, while in conventional programs knowledge and control are integrated.
- Heuristic (inferential) procedures are used in KBS versus algorithmic in conventional programs.
- KBS systems are oriented towards symbolic processing, in contrast to conventional programs which are oriented towards numerical processing.

In the following sections of this chapter a brief account of KBS fundamentals will be presented, followed by a review of some existing KBS applications for the estimation of ground parameters.

2.2 KBS components.

One of the basic features of a KBS is the separation of the domain knowledge from the control of the execution. This leads to the identification of two main components of a KBS, the knowledge base and the inference mechanism.

The knowledge base contains the domain specific knowledge, usually in the form of facts and heuristics. Implementation of the knowledge should be performed in a way that will provide transparency of the knowledge base, so that it can be easily accessed and modified. The latter is a crucial requirement for KBS, since knowledge needs to be continuously updated and modified.

The inference mechanism is the part of a KBS that controls the reasoning process (execution) of the system. It uses the knowledge in the knowledge base, to infer conclusions about the solution of the selected problem. All the conclusions inferred by the system, together with the input for a specific problem, form another part of a KBS, usually termed the context.

The context, also known as the working memory of the system, initially contains the data defining the problem to be solved by the system, but as the reasoning process continues it changes dynamically to incorporate all the intermediate results as well as the solution. At any point during the execution, the amount of information stored in the context, reflects the state of the problem currently being solved by the system.

A typical variation to the basic architecture described above is the blackboard model, which is usually preferred when the problem to be solved requires multiple sources and/or levels of knowledge. In this case the knowledge base is separated into a number of different knowledge sources, that communicate through the blackboard, which also serves as the context of the system.

Additional components of a KBS are the user interface, the knowledge acquisition module and the explanation facility. The user interface is the module that allows users to communicate with the system. Three major requirements for the development of a powerful and easy-to-use user interface are simplicity of form, clarity of expression, and aesthetics. The user interface should also be combined with an explanation facility, that will help users to understand the reasoning process, enabling them to check the conclusions or to learn from the system.

Finally the importance of the development of a knowledge acquisition model is reflected in its dual function; as a means of implementing the knowledge in the system during the development stage, and to ensure updateability of the knowledge after the system's completion.

2.3 Development stages of KBS.

In this section the basic development stages of a KBS are presented as a linear ordered process. This is only done for reasons of simplicity of presentation since it is widely accepted that one does not build a KBS in a simple linear manner, rather the development of such a system is a cyclical and incremental process.

The first part in the development procedure of a KBS is the selection of a project and the identification of the specific tasks to be dealt with by the system. The tasks

to be modelled in the system must be fairly narrow, clearly defined, and domain intensive [Dym, 1987].

After the definition of the project's aims, all the domain specific knowledge needed for tackling the imposed tasks, should be collected. The process of obtaining this knowledge by using text books, Design Standards, Codes of Practice, other available literature, existing knowledge-based systems, as well as by interaction with human experts is known as the knowledge acquisition stage [Miles and Moore, 1994]. Knowledge elicitation is the subsection of knowledge acquisition, which covers the acquisition of knowledge by interaction with human experts. Formal methodologies for knowledge elicitation though many, are usually complex and their use is rather limited. Some of these methods are focused or structured interviews and are based on cognitive psychology techniques (e.g. Cooke and McDonald, 1986). Cordingley [1989], considers three major factors affecting the choice of knowledge acquisition techniques: the nature of the source of the knowledge; what form the knowledge takes; and what is allowed to drive the selection of relevant knowledge and the elicitation process itself. In most geotechnical KBS, knowledge is usually acquired from the technical literature, from structured or unstructured interviews with domain experts, or from questionnaires.

The next development stage is the implementation of the KBS, which consists on software and hardware selection (or in some cases software development), knowledge representation and inference model adoption.

KBS's building tools can be categorised according to hardware sophistication and software system complexity [Mullarkey, 1987]. In terms of hardware, almost all types of computers, from microcomputers up to special purpose workstations, can be useful for building KBS. The selection of hardware always depends on the specific software to be used, but one should bear in mind that climbing the scale of

sophistication increases the capabilities of the system, but may also obstruct its widespread commercial use.

The available software for building KBS can be subdivided into three main categories, namely general purpose programming languages, general purpose representation languages, and expert system shells [Mullarkey, 1987].

The first category includes the conventional programming languages, such as C, FORTRAN, Pascal etc. Their major advantage is portability and compatibility, but since they are mainly oriented towards numerical processing, they do not provide the best environment for KBS development.

General purpose representation languages, such as LISP and PROLOG, are more oriented towards symbolic processing since they were created for building KBS. These are declarative languages in which information is presented in a descriptive form and they provide greater flexibility in implementing KBS.

The third category includes specially developed packages for the rapid prototyping of KBS. These packages usually provide knowledge representation forms and inference mechanisms. Depending on their origin, they could be divided into domain-derived and domain-independent.

In the first case the shell consists of the inference engine and probably the knowledge representation scheme and other components of an existing KBS, such as EMYCIN, [Van Melle, 1979] and EXPERT [Weiss and Kulikowski, 1979]. These systems lack in flexibility and are mostly used for the same class of problem as the original system.

Domain-independent frameworks, on the other hand, are more flexible since they were not developed for a specific application domain. These frameworks may contain multiple forms of knowledge representation and several inference models, and sometimes additional modules, such as explanation facilities. Expert system development environments are the most sophisticated tool for developing KBS. They include features such as: integrated editors, system browsers, debugging tools, user interface and explanation system development facilities. Such systems are usually fully developed system building workbenches. They provide maximum flexibility in KBS development, but they have high software costs, they can only be used on special purpose hardware, and they require trained users.

The next development stage, is the implementation of the knowledge and the control strategy (adoption of inference models) to be used in the system. Knowledge implementation is usually preceded by the development of a knowledge acquisition module. The most common knowledge representation schemes are, logic-based, rule-based and network-based representation (a more detail review of alternative methods for knowledge representation can be found in Gevarter, 1987).

In a logic-based representation scheme, knowledge is represented as assertions in logic. In a rule-based representation scheme, knowledge is represented in the form of rules, usually consisting of an IF part (premise or condition of the rule), a THEN part (conclusion or action) and an optional ELSE part (alternative). These rules are usually termed production rules. They have the advantages of simplicity, homogeneity (of the represented knowledge), and are easily inspected and extended. Their main disadvantage is that general relations between pieces of knowledge are difficult to express.

In a network-based representation scheme, knowledge is represented as nodes, and relations between pieces of information are represented as links between nodes.

Frames, are a generalisation of networks. A frame represents a more general concept about a piece of knowledge which contains attributes (represented as slots), and their values (slot values). Frames can sometimes be structured in a hierarchical mode, enabling the lower level frames to inherit attributes and values from their ancestors. Objects are very similar to frames (sometimes these two terms are used as synonymous in the literature), in that they also contain slots, but are different since they are always structured in a hierarchical mode. Furthermore in objects, slot values can also be pointers to other objects or attached procedures for computing the values of other slots.

The control strategy of a KBS depends on the nature of the problem the system deals with, and the knowledge representation scheme used. The most commonly used inference models, especially in conjunction with a rule-based representation scheme, are forward chaining and backward chaining, but object-based representation schemes can also be coupled with the above techniques.

A forward chaining, or data driven, inference engine works from an initial state of facts to a goal state (conclusions). A backward chaining (deductive reasoning), or goal driven, inference engine assumes a goal state and then reasons back to known facts or data, to check the validity of the assumption. A combination of the above techniques, called mixed chaining, is also allowed in some systems. No formal methodology exists for the selection of an inference model, but forward chaining best simulates a generative (or formative) reasoning process which is usually met in design problems (formation approach), while backward chaining reasoning best simulates diagnostic problems (derivation approach).

The models described above can be combined with other control strategies which dictate the order of activation of the applicable rules (in a rule-based KBS). The most common ones are breadth first search, where all the applicable rules are

executed in turn before failure or success is tested for each one, and depth first search, where the first of the applicable rules is exhaustively explored before examining the next one.

Finally object-oriented programming is an approach in which both information about an object and the appropriate procedures are grouped together in the same data structure (the object), and procedures can be invoked by messages sent to the object from a central controller, or another object. The major advantages of object-oriented programming are that it allows for a truly modular programming environment, where redundancies in coding are kept to a minimum and provides a means of managing large programming projects by breaking up large problems into smaller, independently functioning, highly visible parts [Tello, 1989].

A very important requirement for a KBS is the ability to reason with uncertainty. Uncertainty may be present in the factual and heuristic knowledge implemented in the system and in the input data provided by the user. The most common ways to model uncertainty in data and inference are probability theory, Dempster-Shafer theory of evidence and fuzzy reasoning. A critical review of the above methods can be found in a paper by Groothuizen [1986]. It should be further noted, that uncertainty assessment and modelling in KBS is still a very active and controversial area of research. The main reason for this controversy is that no agreed model of how the human mind processes uncertainty in the real world exists, and inevitably there can be no single method with which this uncertainty can be represented [Miles and Moore, 1994]. Consequently, they recommend that the decision to include or not uncertainty in a KBS (and if yes, which model of uncertainty should be used) should only be made by the system's developer.

In the concluding development stage the user interface, usually coupled with some form of explanation facility, is implemented, followed by the evaluation of the

system's performance. The most common types of user interfaces are question and answer, menus, icons, form filling, command languages and natural language. Each of the above types are associated with advantages and disadvantages which are discussed by Sutcliffe, [1988].

2.4 KBS in Geotechnical Engineering.

KBS technology has been applied in the area of geotechnical engineering and several systems have been developed that deal with a wide variety of problems encountered in the area. A detailed review of many of these systems is given by Moula *et al* [1994]. This section is focused only on KBS that deal with estimation of ground properties.

CASS (Computerised Adviser on Soil Strength) [Gillette, 1991], is an expert system to provide advice on the selection of soil shear strength parameters for use in slope stability analysis.

CASS can help in the interpretation of the results of triaxial confined and unconfined compression, direct shear, and field vane tests. According to the information regarding soil type, test procedures, and the results (provided by the user), the system attempts to infer shear strength parameters for use in slope stability analysis. In the case of field vane tests a correction is applied to the results (e.g. Bjerrum, 1972), while in triaxial tests performed without back-pressure saturation, an empirical method is used to account for the effects of capillarity in the test specimen, on the values of c' and ϕ' .

If no test results are available the system can estimate possible ranges for strength parameters, from soil descriptions and from correlations with classification parameters, such as grading, plasticity etc. The system also cross-checks the various inputs and highlights inconsistencies (e.g. when a soil is described as slightly plastic, yet a plasticity index of 40 is provided by the user), allowing the user to correct them at the stage of their identification without having to restart the consultation. Finally it provides the user with recommendations involving general information and warnings relevant to the soil and test types (if any) in question.

Knowledge was implemented in the system using the rule-based expert system shell Personal Consultant Plus (PC+) and runs on an AT-class personal computer having extended memory. The shell consists of a backward chaining inference engine which drives the rules to infer the desired conclusions, a development interface which translates rules written in a "near English" form into LISP, and a user interface which can display graphics as a part of queries and conclusions. The knowledge was acquired from the technical literature and from interviews with a domain expert (an experienced USBR employee having nearly 50 years of service) and was translated into rules for PC+.

Davey-Wilson [1991], presents a KBS for the estimation of peak effective angle of friction of non-cohesive soils, supplemented by a simulation of the execution of the shear box test.

The system is able to estimate the peak effective angle of friction of a non-cohesive soil, to a perceived maximum accuracy of $\pm 1^\circ$, using as input descriptive soil parameters such as: particle size distribution, grain size, in-situ density and homogeneity. The level of accuracy of the parameter in question depends on the quality of the input soil description (the more information available the higher the

accuracy of the value of the parameter). The knowledge was acquired from the literature [Terzaghi and Peck, 1967] and was implemented in the knowledge base in a simple rule-based structure of if-then-else statements. The same system can also simulate the execution of the laboratory shear box test with step by step interaction with the user. The author suggests that the educational part of the program could be further developed by adding sound effects or digitised photographs, or even by linking it with a video.

The system was implemented using the object oriented software HyperCard, and runs on an Apple Macintosh computer. The HyperCard software is a series of cards, each of which is a separate object, that can be filled in with pictures or text. A stack of cards (application) can contain up to 32000 cards and can be easily combined with other stacks. HyperCard enables an easy construction of a highly graphical environment, and also includes an object oriented language, called HyperTalk.

The system incorporates a graphical interface which allows the selection of the parameters required by the system (such as angularity of the grains), to be made from comparisons with pictures of soil grains and their size, type and distribution.

CONE [Mullarkey, 1986; Mullarkey and Fenves, 1986] is a development prototype KBS, for the interpretation of raw Cone Penetration Test data (measuring the resistance generated by pushing a cone into the ground).

The function of the system incorporates a validity scan of the raw CPT data (the cone resistance q_c and the local side friction f_s), classification of the soil types, including profiling of the layers, and inference of the effective angle of friction of sands and undrained shear strength of clays. Soil type classification is based on the use of two soil classification systems - Dutch [Begemann *et al*, 1982] and Douglas

& Olsen [1981] classification systems - plus a third one, which is a fuzzy set representation of the raw data from the Douglas & Olsen classification system. The shear strength parameters of sands and clays are estimated using empirically and rationally based methods.

Both linguistic data (soil classification) and numerical data (ϕ' , S_u), along with their incorporated uncertainties (vagueness and statistical variability respectively), are represented as fuzzy sets with respect to the linguistic variable. The soil type for example, is represented as a three element fuzzy set (sand, silt, clay) along with the corresponding numerical values indicating the membership of each element in it (membership process). The appropriateness of a soil classification system (with respect to site location), the accuracy of the system in respect to certain soil types (Belief), and the relative importance of the inferred information (Weight) are also expressed as linguistic variables (fuzzy sets represented over a five-valued universe). The Belief and Weight are used as fuzzy set modifiers incorporating the uncertainty in a certain piece of information (soil type, or shear strength parameters). Finally, during the translation process, the modified fuzzy set is translated to a verbal (soil type) or numerical (ϕ' , S_u) descriptor accompanied with its belief value.

Cone has been implemented using OPS5 rules, grouped according to their specific subtasks (as rulesets), and LISP functions. The system, is classified as a development prototype, and has been validated using published cases and proved to be fairly reliable (80% accuracy). A typical run of CONE may take up to 1.5 hours on a lightly loaded Dec-20, depending on the length of the CPT log.

The Rock Mass Classification system, RMC [Juang and Lee, 1989], is an expert system for the assessment of the engineering behaviour of rock masses.

The knowledge in the system was predominately based on Bieniawski's [1976] geomechanics classification scheme. According to this knowledge, there are six major parameters affecting the classification of a rock mass: strength of the intact rock material, rock quality designation (RQD), spacing and condition of joints, groundwater condition, and joint orientation. Knowledge was implemented as facts and rules, and was stored in external databases. The system may be grouped into five sections: the declaration section, the rules section, the input section, the parallel processing section, and the consolidation section.

After the declaration section, the system, using the facts in the database and 11 initial meta-rules, generates 182 production rules for subsequent use. In the next stage, the problem specific data is input to the system through an external program, called GETDATA, which is actually a user interface. During the parallel processing stage, rules are processed enabling some preliminary conclusions (for the values of the factors affecting classification) to be reached. In the last stage, an external program called FUZZY, performs a fuzzy manipulation of the results, incorporating the different supports for the preliminary conclusions, arriving thus at the final conclusions.

The system has been developed in the expert system shell FLOPS, (Fuzzy LOGic Production System) and runs under the MS-DOS environment on microcomputers. FLOPS features approximate reasoning, using fuzzy logic, deductive and inductive reasoning, and supports both forward and backward chaining inference mechanisms. It employs a relational structure for data stored on a blackboard, and has the ability to call other programs and exchange data through the blackboard (thus overcoming the limitations of using a small computer).

The system was tested for a limited number of case studies and its results compared favourably with domain experts' opinions. Some future developments include

further testing and calibration by experts, and the incorporation of additional geological factors.

2.5 Discussion.

The systems presented above deal with specific areas of estimation of ground properties. Their major characteristic is that they address a very restricted area of application; Cass deals with estimation of shear strength parameters for slope stability, Davey-Wilson's KBS with the estimation of peak friction angle of non-cohesive soils, Cone with the interpretation of CPT results and the RMC system, which has relatively the most extended area of application, with the engineering classification of a rock mass.

Another common feature of the systems is that they employ empirical or semi-empirical procedures to estimate ground properties. Cass employs semi-empirical and empirical procedures to correct the results of field and laboratory tests and, as well as Davey-Wilson's system, uses soil descriptions to produce estimates of shear strength parameters. Cone uses both empirical and rationally-based methods for the interpretation of CPT results. Finally the RMC system generates its results relying completely on an empirical procedure (Bieniawski's engineering rock mass classification).

The use of empirical procedures is accommodated with uncertainty that should be incorporated either into the knowledge directly, or in the inference procedure. This is a crucial requirement, since the inclusion of uncertainty of the estimated parameters will provide an evaluation of the risk involved with their subsequent use in analysis or design. Of course there are many aspects of uncertainty in the estimation of ground parameters and the identification and evaluation of each one

(along with their combined effects on the estimated parameter) is no trivial task. An attempt to identify the different aspects of uncertainty, associated with ground properties estimation, can be found in Kulhawy, [1992] and will be discussed further in Chapter 5.

In the systems presented above different methods for incorporating uncertainty have been used. In Cass the results of field and laboratory tests are corrected in order to provide a conservative design value (to be used in slope stability analysis). The correction procedures actually produce a range of values for the desired parameters, but the user is only presented with a value that falls in the lower band of this range (conservative estimation), which might be "safe", in terms of design, but may in fact prove to be uneconomical. Furthermore the system allows the user to input only one value to describe a series of tests (probably with different results), thus eliminating consideration of the spatial variability, which otherwise could be useful for identifying "weak" spots in the ground.

In Davey-Wilson's system, uncertainty is introduced in the inference mechanism as an amount proportional to the quantity of information provided by the user. It is supposed by the system that four descriptive qualitative parameters are adequate to produce an estimate of peak effective angle of friction of non-cohesive soils with an accuracy of $\pm 1^\circ$. This of course is not always the case, because there are many more factors affecting the angle of friction of a soil which are not included in the inference, and because the friction angle of special soil types (such as sands with a small percentage of fines) cannot be estimated correctly from a procedure based on data obtained from typical soils (e.g. clean sands).

In contrast to the above the Cone and RMC systems employ a more formal way of representing uncertainty in the inference. In Cone all the estimated information (soil type, shear strength parameters) is represented as fuzzy sets, and the variables

Weight and Belief (the reliability and applicability of the estimation) are used as modifiers, to incorporate uncertainty into the inferred parameter. Each estimated value for a parameter is accommodated with its belief, which serves as a measure of the quality of the information.

In the RMC system, fuzzy logic is used to incorporate the different supports for the preliminary conclusions. Therefore all the inferred information is combined with respect to its relative importance and uncertainty, thus providing a weighted combination of the factors affecting engineering rock mass classification.

2.6 Conclusions.

The systems developed in the area of estimation of ground properties are few, having a restricted area of application, are predominately based on empirical procedures, and employ different methods of incorporating uncertainty in their inference mechanisms.

There is a need for development of new applications, that will address a wider area of application (instead of relying on a limited number of estimation procedures), in order to provide a more general framework for estimating ground properties. This framework should be based on a consistent and complete representation scheme for ground properties and procedures for their estimation. The latter should be represented in a form that will allow for the inclusion of the quality to the inferred information, and also for updating, as well as implementing new estimation procedures, ensuring that the system will maintain its usefulness in the future.

CHAPTER 3

Development tools.

3.1 Introduction.

The system for the estimation of ground properties was implemented using the ProKappa software, running under X windows on a Sun Spark 2 workstation. The main reasons were that this software and hardware were available at the time and that it was thought that they could provide a favourable environment for the development of the desired system application. The advantages and drawbacks of implementing a KBS using an expert systems development environment on a workstation are discussed in §2.3.

In the remaining sections of this chapter an introduction to the ProKappa software will be presented. This is done in order that the reader will become familiar with the terminology used and to illustrate some of the ProKappa features that have been used in the development of the system.

3.2 The ProKappa software.

ProKappa is a C-based software development system that integrates object-oriented programming, rule-based reasoning and SQL database access in an easy to use graphical environment. The ProKappa system incorporates a number of features for

the rapid prototyping and development of systems applications. These features are presented in the following sections. The ProKappa software is discussed in greater detail in the ProKappa User's Guide [Intellicorp, 1991].

3.2.1 Objects, slots, facets and applications.

Objects are structures that provide a formal way of organising related pieces of data. They can represent templates for sets of real world entities or individual entities. Objects are used to hold descriptive data about the entity, thing, item, concept, category or template being represented. They can also contain special functions called methods, which define behaviour for the entity being represented.

The ProKappa system has two kinds of objects: classes and instances. Classes are templates for sets of entities with common characteristics. Instances represent individual objects in the application domain. Classes and instances are organised hierarchically so that information specified in a class is inherited by its instances. Object hierarchies may have any number of levels. The terms subclass and superclass are used to describe the relationships between objects at different levels in the hierarchy. For a class object all the classes in lower levels of the hierarchy are its subclasses and all in higher levels, its superclasses. In addition to the terms class, subclass, superclass and instance, the following terminology is used to describe possible relationships among objects:

- | | |
|--------|--|
| parent | The object directly above a specified object in the hierarchy, also referred as a direct superclass. An object may have more than one parents. |
| child | The object directly below a specified object in the hierarchy, also referred as a direct subclass or a direct instance. |

- ancestor A superclass of an object.
descendant A subclass or instance of an object.

Both classes and instances contain slots, which are used to represent characteristics or attributes of objects. Slots represent three types of information:

- Attributes, or descriptive information about an object.
- Actions, called methods, that the object can perform.
- Relationships to other objects in the system.

According to their values, slots can be subdivided into the following categories: single-value slots, multi-value slots and method slots. The two former are used to store values as symbols (a sequence of alphanumeric characters including underscores and exclamation marks), strings (a sequence of characters and spaces surrounded by double quotes), numbers, lists, arrays or pointers to other objects. Method slots store values which are pointers to procedures that define behaviour for the object.

The ProKappa object system supports inheritance. There are two types of inheritance: slot inheritance, which is the inheritance of the existence of a slot to its subclasses and instances; slot value inheritance, which is the inheritance of slot values from the slot of the parent object to the slot of the subclass or instance. Slot or slot value inheritance may be blocked at any level in the object hierarchy, preventing the slot or slot value(s) from being inherited further down. The ProKappa system also provides complex types of slot value inheritance. For example an object with multiple parents can inherit values from one or more of its parents (the latter only applies to multi-value slots). Finally it should be noted that slots can only be created at class level, since they represent structures common to all instances of a class.

The information stored in slots can be further refined by the use of facets. Facets, which can be regarded as slots into slots, are used to include additional information about slots or slot values. Facets have the same types as slots (single-value or multi-value) and can be inherited. The differences between slots and facets are that facet values cannot be pointers to methods and that facets can be created both at class and instance level.

The ProKappa object system supports arbitrarily complex hierarchies of objects. A collection of one or more object hierarchies forms an object base. An object base together with the functions, rules and user interface form the integral parts of a data structure, called a ProKappa application. When an application is loaded or saved all of its associated objects and files (containing functions and rules) are loaded or saved respectively.

Applications may be subdivided into modules and sub-applications. Modules are much like applications themselves; they may contain an object base, functions and all the other components that make up applications. However, unlike applications, modules must always be part of another application. Sub-applications are applications which are required or used by other applications. For example, an application may either require or use its sub-applications, depending on whether the application's objects are children of those in the sub-application or not. The main advantage in using sub-applications and modules is that a large and complex task (that must be dealt with by the application) can be broken down to several less complex tasks, each of which is dealt with by a module or a sub-application. The resulting reduction in the complexity of the task makes the overall application easier to implement, develop and update.

It should be noted that in the rest of the thesis applications, modules and objects are expressed with bold characters and slots and facets with italics.

3.2.2 Programming languages.

The ProKappa environment provides two languages that can be used for adding actions to, or performing tasks in applications. These are the C language (as extended by ProKappa) and the ProTalk language.

The ProKappa environment supports an ANSI standard compatible version of the C programming language plus several libraries of C functions for use specifically within a ProKappa application.

The ProTalk language is a special language for writing functions and rules within the ProKappa environment. Just like any other programming language, it has its own syntax. It is particularly suited for writing functions and rules that access and update information in a ProKappa object base. The ProTalk language has the following features:

- A set of predefined functions for interacting with ProKappa applications, objects, slots and facets and performing tasks such as:
 - Creating applications, objects, slots and facets.
 - Performing utility functions such as sorting lists.
 - Obtaining input from users.
 - Printing.
- Syntax for referring to information stored in an object base that can be used for:
 - Accessing or modifying values in slots and facets.
 - Creating and modifying relationships between objects.
 - Retrieving information about objects, slots and facets.
 - Sending messages to objects.

- Programming constructs such as:
 - Assignment of values to variables.
 - Basic arithmetic operators.
 - Comparison operators.
 - Conditional statements.
 - Iteration constructs.
- Ability to call C functions and incorporate C code.
- Built-in backtracking mechanism (non-deterministic language).

The ProTalk language provides a type of expression called a knowledge expression, for referring to information in an object base (objects, slots, facets and applications).

The major types of knowledge expressions are:

- Slot values: object.slot
- Facet values: object.slot..facet
- Instances of a class: instanceof class
- Subclasses of a class: subclassof class
- Ancestors of an instance: classof instance
- Ancestors of a class: superclassof class.

The last four knowledge expressions can be modified by the use of direct, to restrict the expression to the "direct" instances, classes, subclasses or superclasses, where direct means one level below or above in the object hierarchy.

The ProTalk language also provides value changing operators and comparison operators, that can be used in conjunction with knowledge expressions to change the information in an object base, or compare the values of two knowledge expressions.

Knowledge expressions can also be used in conjunction with search modifiers to retrieve information from an object base. Retrieval can happen either deterministically or non-deterministically. Statements that perform deterministic searches must always have exactly one solution, and they never fail. If a deterministic search is performed for a statement with more than one solutions, it will create an error. Statements that perform non-deterministic searches can have zero, one or multiple solutions, and they can fail. The ProTalk search modifiers are:

Deterministic approach:

- no modifier: For use with single-value slots and facets only. Generates a single value, or Null if there is no value.
- all: Generates a list of all the values, or the empty list `()` if there are no values.

Non-deterministic approach:

- find1: Generates one solution. Fails if there is no value.
- find: Generates one solution each time the statement is executed. Fails if there is no solution. Can be re-entered if the system backtracks to it.
- find N: Generates one solution each time the statement is executed. Fails if there is no solution. Can be re-entered if the system backtracks to it, at most N times.

The ProTalk language can be used for writing functions and rules. A ProTalk function is made up of one or more ProTalk statements. Each simple statement ends in a semicolon (;). A compound statement is a sequence of zero or more statements wrapped in a pair of curly brackets ({ }). Each statement consists of some combination of ProTalk operators (value changing, comparison or search operators listed above), expressions (e.g. knowledge expressions), programming constructs,

function calls and variables. A function is defined by placing the keyword "function" in front of the function name, which is followed by a pair of parenthesis enclosing the function's arguments, separated by commas. A pair of curly brackets encloses the body of the code. Additional features of ProTalk functions are:

- Local variables do not need to be declared.
- Functions may or may not return values.
- All arguments are ProTypes (symbols, strings, numbers, lists, arrays, dates, objects, slots or facets).

When a function is pointed to by an object's method slot it is called a method. Methods are functions stored in an object which specify the actions that this object can perform. The difference between methods and functions are that the former are defined by the keyword "method" and that they contain by default two arguments. The first argument (?self) is bound to the object that contains the method and the second (?slot) to the method slot that contains the pointer to the method.

The ProKappa system allows the writing of rules to reason about objects, their relationships and their attributes. Rules can only be written in the ProTalk language. They are a combination of statements grouped together in rulesets. The two main approaches in using rules in ProKappa are the forward chaining or the backward chaining approach. When appropriate a mixed chaining (forward and backward interchanging) approach can also be used.

3.2.3 User interface development tools.

The ProKappa system allows the building of customised graphic end-user interfaces to applications. It contains two system supplied applications for their development: the **Active Images** and the **Dialog Box** applications.

The **Active Images** system application is a tool for building business and instrumentation images to represent slot values graphically. The images in the **Active Images** application provide a variety of graphic displays for both viewing (output images) and modifying (input images) slot values in objects. These images are specialised objects that can be attached to one or multiple slots (the slots may even belong to different objects) and display their values in a variety of forms. Images are displayed inside image panels, which are customised X windows. This tool has not been utilised for the development of the user interface of the system for the estimation of ground properties. Therefore it will not be discussed in any further detail.

The **Dialog Box** system application is used for creating windows, called dialog boxes, which are used for obtaining arguments or specifying options required by a process about to be executed and for displaying information (e.g. on the progress of a certain processing action).

Each dialog box is represented in the ProKappa environment as an instance of either the **DialogBoxPositional** or the **DialogBoxAuto** class object, depending on whether the dialog box is created programmatically (a set of system supplied functions can be utilised in the C or ProTalk languages for creating and manipulating dialog boxes) or by using the Interface Workbench (a ProKappa browser which allows interactive creation and manipulation of dialog boxes). These two objects belong to the **Dialog Box** application.

A dialog box obtains its functionality from its components, which are called the dialog box controls. Each control is also represented as an instance of an appropriate control class in the **Dialog Box** application. These classes represent the types of dialog box controls supported by the ProKappa environment. Dialog box controls can be subdivided into the following three categories:

Display Controls

Text Display: It displays a value or a set of values to the user. If more than one value is specified, each value is displayed on a separate line.

Pixmap Display: Used to display bitmap images.

Input Controls

Entry Box: It allows the user to type a value into the dialog box. The user can start entering text after selecting the entry box with the mouse.

Radio Buttons: These are used when the user must specify "one-of-many" possibilities. One button is selected at all times and only one button can be selected. When the user selects a new button the old is automatically deselected. Finally the developer is allowed to specify which button is selected by default.

Check Buttons: They allow the user to select several choices out of many. Check buttons are selected and deselected by toggling; the first click with the mouse selects them, the second deselects them, the third reselects them and so on.

List Box: A list box holds a list of items which the user can select. The display capacity, which is by default set to five lines, can be set to the desired number by the developer. By default, a list box allows single selection; if the user selects an item, any

other selected item is deselected. The developer is allowed to specify whether single or multiple selections are allowed. In the latter case selection and deselection of an item is performed by toggling.

Option Menu: An option menu displays the currently selected value out of a number of possible values. Clicking on an option menu pops up a list of all possible values, from which a new value may be selected.

Action Controls

Push Button: A ProKappa push button simulates a physical push button. It displays a label, and it is "pushed" by clicking on and releasing the mouse. Whatever activity is associated with the push button is performed at the time it is pushed.

Push Button Row: The push button row allows specification of a row of push buttons with one object. The system creates as many push buttons as specified and arranges them in a horizontal row. All dialog boxes have by default a push button row control which is called command row control and contains two buttons, labelled "OK" and "Cancel". Additional command row buttons can be created and the labels of the default ones can be changed. The buttons are used to either initiate or cancel the behaviour of the dialog box.

As mentioned earlier each control is represented as an object. All control objects contain slots which hold information associated with their title, values (labels, or selection items), foreground and background colours, fonts (of both their title and values), horizontal and vertical positions (in the dialog box), and the dialog box to which they belong. Furthermore, different types of control contain additional slots

which are associated with the control's specific characteristics and functionality (e.g. a maximum number of lines slot for a list box). The user interface developer has to set or manipulate the values of these slots to obtain the desired outcome. Each dialog box and dialog box control object also contains a *UserData* slot which can be used for storing user-supplied data. Finally each non-display control has an associated React! method slot which can be used to define what happens when the user interacts with the control, e.g. invoke an action when a push button is depressed.

3.2.4 Monitors and Active Relations.

Monitors, also called active values or demons, execute behaviour whenever the value of the slot to which they are attached is either accessed or modified. They are represented as ProKappa objects with one or more specialised method slots. Monitor objects are created as subclasses or instances of either the **SmallMonitor** or the **LargeMonitor** classes in the system supplied application, **SystemApplication**. They are attached to slots using facets on the slot (which they create). Their behaviour is specified as the value of the method slot in the monitor object (a function that must be supplied by the application developer). The developer is allowed to specify that their behaviour occurs as follows:

- When slot values are accessed (WhenNeeded monitors). This monitor type is typically used: to provide reports or alarms; to convert a value from one format to another; to calculate a value only when the value is needed by the application.
- When slot values are changed, before the new value(s) are placed into the slot (BeforeChanged monitors). Some typical uses of BeforeChanged monitors are: checking that a value falls into a specified range; coercing a value type (e.g. if the value is an object); converting a value from one format to another.

- When slot values are changed, after the new value(s) go into the slot (AfterChanged monitors). AfterChanged monitors can be used to: make changes to dependant information in the application (e.g. changing the values of other slots when the value of the monitored slot reaches a critical threshold) and to provide reports or alarms.
- When a monitor is attached to a slot (WhenAttached monitor)
- When a monitor is detached from a slot (WhenDetached monitor)

The **Active Relations** application provides tools to make slots responsible for calculating their own values and for certain other properties of their values. It allows slots to function like cells of a spreadsheet program, gathering up data from other slots and using that data to compute the value(s) for this slot. Also like a spreadsheet, constraints can be imposed to slots, in the type of value that they may contain. The system consists of four parts: value type enforcement, value type coercion, slot value inverses and slot formulas.

Value types are declarative mechanisms for specifying that a slot's value must be of a particular ProType (e.g. a ProKappa list, symbol, object etc.), or that its value must come from a specified list of values. Additional conditions in the value may also be specified, such as its range (for a numerical value), or, if it is an object, whether it is a class or an instance, and the class of which the object is a descendant may also be specified. The **Active Relations** application can take one of four different types of action if a value that does not conform to the value type is entered into the slot. The four enforcement types are:

- Allow enforcement means that the value acts merely as a comment to the slot; the system does no validation of added values.
- Discard enforcement means that invalid values are discarded by the system, but no message or warning is given.

- Convert enforcement means that the system attempts to convert bad values; if it cannot convert them, they are discarded silently. It must be noted that these conversions are very simple; more complex conversions can be dealt with by the value check system.
- Alarm enforcement means that the system prints a message or puts up an alarm dialog box when an invalid value is entered. The alarm enforcement type may also be set to call a function (specified by the developer of the application) which handles bad values, or raises an exception.

The value check system allows the developer to specify arbitrary ProTalk code which the system uses to validate new values for the slot. The code is executed whenever a new value is entered into the slot. In addition to validating a value, the code can modify the new value before the slot is updated, perhaps converting from one representation to another, for example. The value check system is most useful when the preset value types provided by the system are not adequate to fully represent the value allowed in the slot. It is also useful when the conversions offered by the value type system are not flexible enough.

An inverse links two objects bidirectionally, representing an "is-associated" relationship between the two objects: "object A is associated with object B". To create an inverse between two objects, one may simply give each object a slot whose value is the other object. Setting up inverses between particular objects in this way is easy. However, in many cases, inverses exist between potentially large numbers of objects, with the inverse defined not on the objects themselves but on the classes from which the objects descend.

The slot formula tool is used for attaching a ProTalk code fragment to a slot whose purpose is to calculate the value of that or any other slot. This calculation may use other slot values, do queries over the object base, or simply make a mathematical

calculation. The slot formula system maintains links between the slot containing a formula and any slots mentioned in the code of the formula, allowing a change in a mentioned slot's value to cause the formula slot's value to update ("set-mode" formula). A "get-mode" formula is run in response to an operation that "gets" the value of the slot. The slot formula uses the ProTalk language pattern matching capability to loop implicitly over all values satisfying it, relying on ProTalk's backtracking capabilities to generate multiple values and to search through the object base. The most common use for slot formulas is to step through a simulation or a "real-time" monitoring system: one slot is updated and slot formulas update all the others. This works even if the next value of a slot depends on its current value: if a formula contains an implicit dependency on its own value, the system intercepts this potentially infinite loop and cuts it off after one iteration (if required more iterations can be specified).

Slot formulas are represented as objects, which belong to modules automatically generated by the system. These modules are attached to the application that the slot formula is created. For example if a slot formula is attached to a slot in the **correlation** application, the system will automatically generate an **AR_correlation** module (Active Relations in the **correlation** application) and all the slot formula objects will be stored there.

Slot formulas are attached as facets to the slot in which they are created. If any other slots are referenced in the slot formula function, then facets are also attached to these. These facets are used as links between the slot formula object and the slots that are required for the evaluation of the slot formula slot.

For example a slot formula can be attached to an *Area* slot of an object. The slot formula incorporates a function for calculating the area of a rectangle, based on the values of the *width* and *length* slots, also attached to the same object. A slot formula

object will be created which contains a function (the slot formula function) for calculating the area of a rectangle. A number of facets is also attached to the *Area*, *width* and *length* slots, linking them with the slot formula object. Therefore each time the slot formula needs to be executed, the required values are obtained from the *width* and *length* slots and the result of the calculation is placed in the *Area* slot.

3.3 Summary.

The system was implemented using the ProKappa software, running under X windows on a Sun Spark 2 workstation.

ProKappa is a C-based software development system that incorporates object-oriented programming in an easy to use graphical environment. Hierarchically structured objects, which contain slots and facets are used for representing data, stored inside applications. The ProKappa system also incorporates ProTalk, a special language for writing functions and assigning behaviour to objects, slots and facets. It also provides the **Dialog Box** system application, a special tool for building customised end-user graphics interfaces to applications.

Finally, the ProKappa system incorporates tools, which can be used by real-time monitoring applications. These are Monitors and Active Relations. The former are used for monitoring values of slots (and execute behaviour whenever the value of the monitored slot is either accessed or modified), and the later provide tools to make slots responsible for the calculation of their own values. The slot formula tool (part of the Active Relations system) was utilised in the development of the system for the representation of the estimation procedures of correlations (§5.3.1).

CHAPTER 4

Representing the ground and its properties.

4.1 Introduction.

The first stage in the development of a KBS for the estimation of ground properties was the collection of correlations and published summaries that provide "typical" values of properties. Both of these were obtained from searches in the relevant technical literature.

The knowledge acquisition stage lasted for more than a year. During this period a large volume of technical papers and textbooks was examined for correlations and summaries of typical values. The main sources of correlations and "typical" values were a report and a textbook: the "Manual on Estimating Soil Properties for Foundation Design" [Kulhawy and Mayne, 1990], and "Correlations of Soil Properties" [Carter and Bentley, 1991], respectively.

Also, databases containing measurements for ground properties were examined for possible relations between the incorporated ground parameters. In one case, regression analysis of the variables resulted in the establishment of a new correlation (estimation of constant volume effective angle of friction for sands and gravels, from mineralogy and angularity of grains; data from Stroud, 1988). The knowledge collected is presented in Appendices A (correlations) and B (typical values of ground properties).

When an adequate amount of correlations and published summaries was available, it became apparent that a representation of the ground and its parameters was essential. The need to represent the ground can be demonstrated by the fact that each correlation or set of typical values is relevant to a limited range of the ground spectrum. Furthermore, correlations merely describe the interrelationships between ground parameters. Therefore a representation scheme for ground parameters is also required by the system. Hence, the second part of the knowledge acquisition stage concerned the collection of the knowledge that is required for the establishment of representation schemes for the ground and its properties.

The ground is a highly variable and complex material and its behaviour is influenced by a wide range of factors. The problems arising from the variability of the ground can be reduced by subdividing it into more specific types, or classes, of ground. This subdivision is based on certain characteristics (usually a few, easily measured ground parameters), which are common for the members of the same class of the ground. The implementation of this concept in ground related problems has led to the establishment of ground classification systems.

In the case of soils, the most common classification systems are the Unified Classification [A.S.T.M. Standards, 1983] and the British Soil Classification [B.S. 5930, 1981] systems. Identification and classification in both systems is based on grain size (particle size distribution), liquid limit, plasticity index and organic content. The reason for the widespread use of classification systems lies in the assumption that members of the same class will have a similar pattern of behaviour. This can be demonstrated from:

- a. the existence of correlations between parameters, applicable to specific classes
- b. the existence of "typical" ranges of values of certain parameters within a class.

It should be stressed that knowledge of a few ground parameters and the identification and classification of a ground type are not adequate to fully describe its behaviour. Furthermore, correlations and "typical" values are merely approximations, and the assessment of ground behaviour should always rely on the actual measurements of ground parameters (based on geotechnical testing). A more detailed analysis of the usefulness and drawbacks from the use of correlations and "typical" values as a means for estimating values for ground parameters, is presented in Chapter 5.

In the following sections of this chapter a ground representation scheme will be presented, followed by the implementation of a Ground Types Knowledge Base, which in conjunction with a Ground Parameters Knowledge Base, provide a means for storing knowledge for the estimation of ground parameters. This is followed by the representation of "typical" values in the system. The chapter concludes with the presentation of the user interface and knowledge acquisition facilities provided by the system.

4.2 A model for representing the ground.

The ground is subdivided into two fundamental categories: soil and rock. From an engineering point of view the distinguishing characteristics between the two are structure, strength and degree of lithification. Soil is any naturally occurring loose or soft material resulting from the mechanical and chemical disintegration of rock or the decay of vegetation, whilst rock is any hard, indurated or consolidated massive geological material [West, 1991].

This and various other definitions in the literature, do not provide a clearcut distinction between soil and rock and allow for the possibility of overlap; e.g.

strongly cemented sands and very stiff clays are close to the definition for rock, while a weak completely decomposed rock is close to the definition for soils. Nevertheless the distinction between soil and rock is a fundamental one in geotechnical engineering and despite a few transitional ground types it is otherwise adequately defined.

Further refinement of these two fundamental ground classes for the purposes of a complete ground representation scheme was predominately based on the British Soil classification system and the identification scheme of rocks for engineering purposes [B.S. 5930, 1981]. The choice of the two systems was made so that the complete ground representation scheme will be in accordance with the code of practice for site investigation in the UK. In the two systems, both soils and rocks are classified according to grain size (particle size distribution for soils), but soils are also classified from plasticity and organic content, whereas rocks are classified based on their formation process and mineralogy.

The basic principle during the development of this ground representation scheme was that ground classes should be represented in a hierarchical form, initiating with the most general classes and developing towards the more specific. Furthermore, each new class level was specified with either the introduction of a new classification parameter or with the specification of a more restricted range of evaluation (compared to the range of the parent class).

4.2.1 Rocks.

Rock is subdivided into sedimentary, igneous and metamorphic, based on the introduction of mode of formation at that level of classification. The rocks classified under the heading sedimentary were originally soils which have

subsequently been lithified by the geological process of consolidation and cementation. Igneous rocks are formed from solidation of molten rock material, generated within the earth's crust, which may have been injected into the rocks of the crust as intrusive masses, or may have found its way to the surface as lava. Finally, metamorphic rocks, are formed from already existing sedimentary, igneous, or other metamorphic rocks by recrystallization in the solid state, under conditions predominately controlled by heat and/or stress.

Sedimentary rocks are subdivided into siliceous, calcareous, carbonaceous and saline, based on their mineralogy. Siliceous sedimentary rocks are further subdivided into clastic and pyroclastic depending on the origin of their fragments (the latter originate from volcanic rocks). Both clastic and pyroclastic sediments are further refined, according to the size of their grains, as rudaceous (consisting of gravel, cobble and boulder size grains), arenaceous (sand size grains), and argillaceous (silt and clay size grains). Clastic sediments also comprise amorphous or cryptocrystalline sediments, such as chert or flint. Calcareous sedimentary rocks are subdivided according to grain size into the same first three divisions. Finally saline rocks comprise rock types such as gypsum, anhydrite, halite etc. (formed by deposition of salts due to evaporation of salt rich waters in an enclosed basin).

Igneous rocks are subdivided according to their mineralogy into acid (much quartz), intermediate (some quartz), basic (little or no quartz) and ultra basic. Each of these four groups is further subdivided, based on crystal size considerations, into coarse (>2mm), medium (0.6-2mm), fine (<0.6mm) and amorphous (no crystals).

Finally, metamorphic rocks are subdivided, based on their structure, into foliated and massive. Of these two only the former class is further subdivided, according to grain size, into coarse, medium, fine and amorphous.

All the specific rock types fall into one or sometimes more of the lower level rock classes defined above. For example, conglomerate and breccia are classified within the rudaceous class of the clastic sedimentary rocks; volcanic glass is classified within both the basic and intermediate amorphous classes of igneous rocks, thus identifying that its mineralogy can be described either as intermediate or basic. The representation scheme for rocks is presented in Figures 4.1 (sedimentary) and 4.2 (igneous and metamorphic).

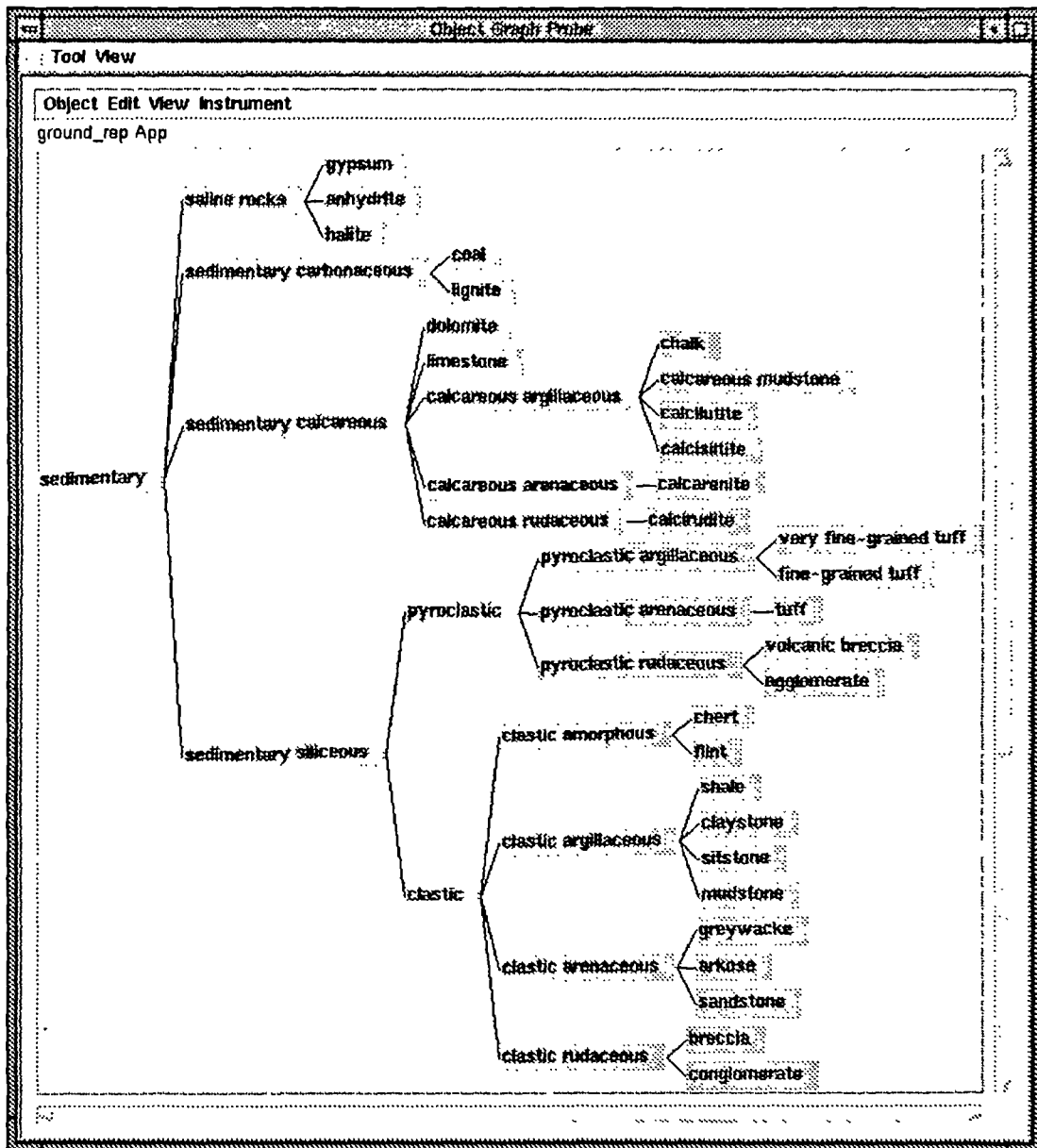


Figure 4.1 Representation of sedimentary rocks.

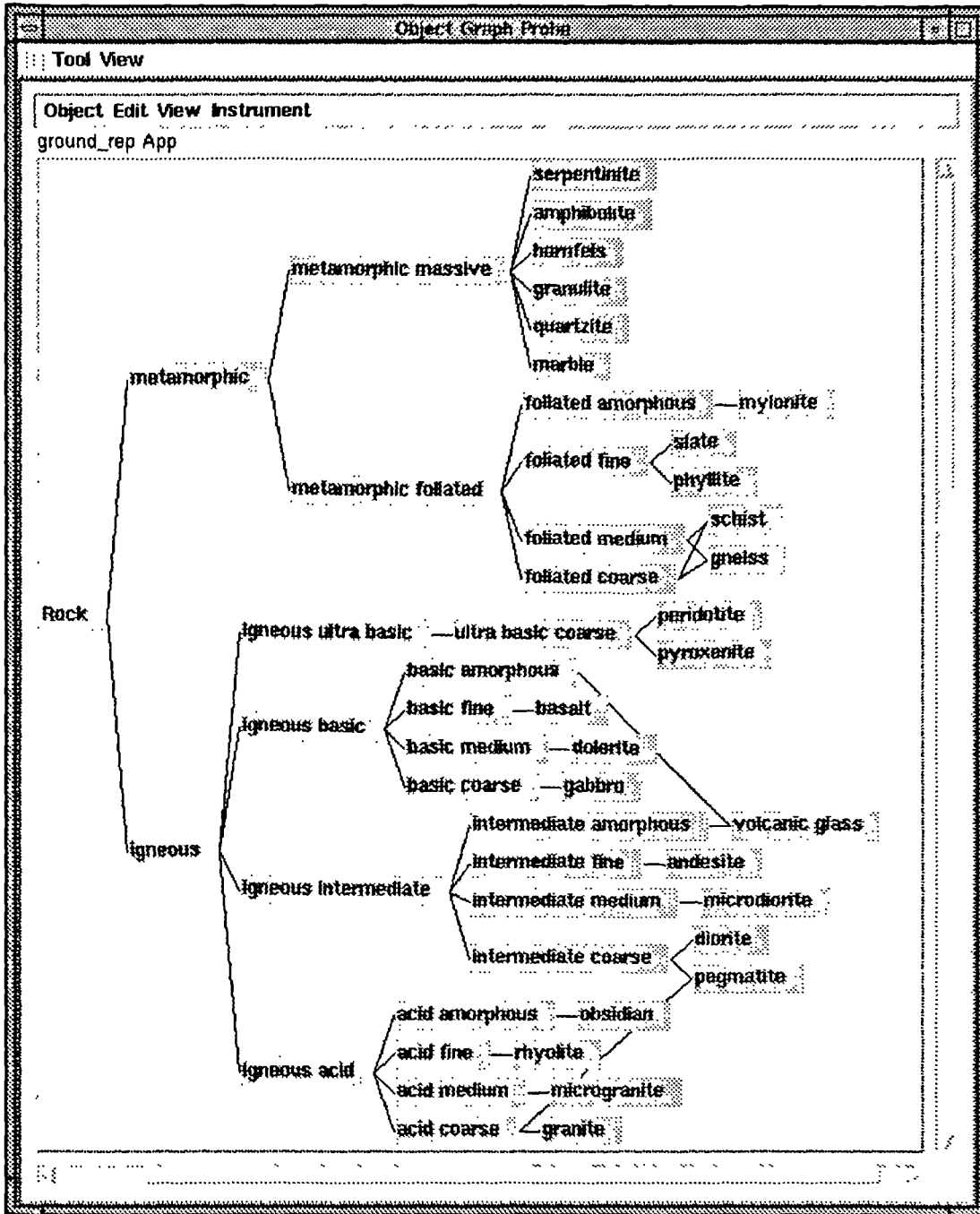


Figure 4.2 Representation of igneous and metamorphic rocks.

4.2.2 Soils.

Soil can be primarily subdivided, in terms of its origin, into naturally occurring and artificial soil. The reason for this is that artificial soils such as waste (material with usually extremely variable composition), or backfill (mainly compacted soils) are frequently encountered in geotechnical problems (e.g. stability analysis of waste tips, embankment construction etc.).

Naturally occurring soils are further refined, based on their organic content percentage, to inorganic and organic. This distinction is necessary, since soils that are classified as organic show markedly different engineering behaviour from inorganic soils, usually expressed as much higher compressibility and in the case of fine organic soils, as very high liquid limits. Both organic and inorganic soils are further refined, based on grain size considerations.

Inorganic soils are subdivided into inorganic very coarse, inorganic coarse and inorganic fine. These different classes are defined according to the grain size of their major constituent. Accordingly, very coarse inorganic soils are defined as soils for which the grain size of their major constituent is more than 60 mm. The corresponding range for coarse soils is 0.06-60 mm and for fine grained soils 0-0.06 mm. These classes can be further refined based on the specification of smaller grain size ranges within each class. Hence, very coarse soils are subdivided into boulders (grain size of major constituent greater than 200 mm) and cobbles (grain size ranging between 60 and 200 mm). Coarse soils are subdivided into gravel (2-60 mm) and sand (0.06-2 mm). Finally, fine soils are subdivided into silt (0.002-0.06 mm) and clay (less than 0.002 mm).

These classes can be further subdivided based on the inclusion of secondary constituents (indicated by a -y ending, e.g. silty, sandy etc.) in the new classes

definitions. Gravel can be further subdivided into clean gravel (main constituent gravel, no secondary constituent), sandy gravel (sand being the secondary constituent, or equally the grain size of the secondary constituent ranging between 0.06 to 2 mm), silty gravel, clayey gravel etc. The same refinement can be applied to the sand, silt and clay classes. The prefix clean is used to indicate the lack of any secondary constituent for gravel, sand and silt. The prefix pure is correspondingly used for clays.

Further refinement of the clean gravel class, based on the adoption of smaller grain size ranges, leads to the establishment of three new subclasses namely: coarse gravel (grain size ranging between 20 and 60 mm), medium gravel (6 to 20 mm) and fine gravel (2 to 6 mm). For clean sand the corresponding subclasses are: coarse sand (0.6 to 2 mm), medium sand (0.2 to 0.6) and fine sand (0.06 to 0.2 mm). Finally clean silt can be subdivided into coarse silt (0.02 to 0.06 mm), medium silt (0.006 to 0.02 mm) and fine silt (0.002 to 0.006 mm).

Organic soils are subdivided into coarse and fine. The definitions for the two subclasses coincide with the definitions for the corresponding subclasses of the inorganic class. The former can be further refined to organic sand and the latter to organic silt and organic clay. The organic gravel and organic very coarse classes were not included in this representation scheme mainly due to the lack of relevant information. Finally the organic class comprises fibrous soils (e.g. peat) that cannot be classified based on grain size considerations. These soils are classified separately under the organic soil class.

The complete representation scheme for soils is presented in Figure 4.3.

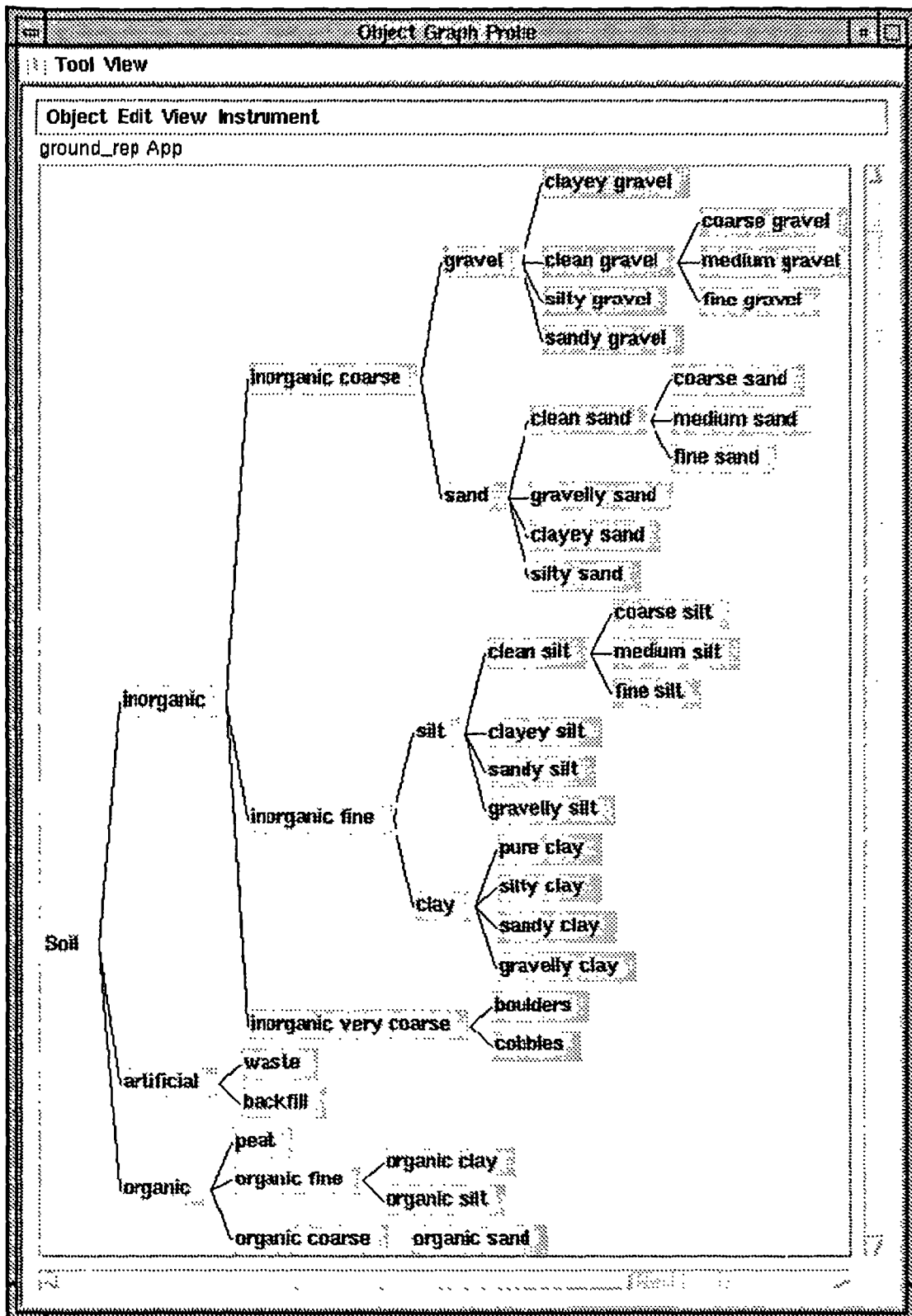


Figure 4.3 Representation scheme for soils

It should be noted that the soil representation scheme can be further expanded with the inclusion of major (indicated by the "prefix" very, e.g. very sandy gravel) and/or minor (indicated by the prefix "slightly", e.g. slightly silty sand) constituents in the definitions of soil classes. Equally these modifiers can also be added to more specific soil classes, such as coarse sand (e.g. clayey coarse sand). Furthermore, a detailed soil description can contain combinations of even more than one major, secondary and minor. However, for the purposes of a soil representation scheme that is used for the estimation of soil parameters, such a level of refinement cannot be justified, since most of the available knowledge is not oriented towards such complex soil types.

4.3 A model for representing ground parameters.

The assessment of the engineering behaviour of the ground can be made through the evaluation of ground parameters. This in turn requires the establishment of a ground parameters representation scheme. The main difficulty associated with this task is the number of parameters used to describe various aspects of engineering behaviour in conjunction with the lack of any formal representation schemes.

The representation scheme for ground parameters utilised here, was based on the principle that parameters can be classified into groups, or parameter categories. For the purposes of a representation scheme that forms part of the basis of a system for the estimation of ground parameters, this was thought to be a beneficial requirement. The main reason for this is that a search for a specific parameter can be significantly limited if its category is first identified, and secondly because it ensures homogeneity in the representation with relation to the ground representation scheme.

The terminology used for the established parameter categories is in accordance with the terminology used in the literature, thus ensuring that their names are meaningful and it is easy to understand which parameters are classified under each of them. Each parameter category can be linked, either directly or indirectly, to one or more different aspects of the engineering behaviour of the ground. An example of a direct parameter category is strength parameters, which are direct expressions of the strength of the ground. An example of an indirect parameter category is stress parameters (e.g. vertical and horizontal effective stress). These have an indirect (but similar) effect on various aspects of ground behaviour, for example they have an effect on the evaluation of strength parameters.

Another problem that was encountered during the implementation of this scheme, was associated with the representation of parameters whose evaluation depends on the actual testing conditions and procedure. For example the undrained shear strength of a specific soil can take different values if this is measured by means of a triaxial compression, a triaxial extension, or a direct shear test. The reason for this is that different responses of a soil are to be expected when subjected to different stress and strain conditions. Therefore, in order to avoid misinterpretations, each type of undrained shear strength was represented as a different parameter (e.g. undrained shear strength in triaxial compression, triaxial extension, direct simple shear etc.). The same principle was applied to other performance parameters, such as effective angle of friction, Young's modulus, shear modulus etc.

Finally, the parameters contained in this representation scheme are distinguished in terms of their evaluation, as either quantitative or qualitative. The former are the ones, which are evaluated by numbers, while the latter are evaluated through verbal descriptors. The inclusion of qualitative parameters in the representation scheme, was dictated by the fact that frequently in geotechnical engineering qualitative descriptions are used for various aspects of the engineering behaviour of the ground.

The parameter categories and their corresponding parameters currently implemented in the system are presented in table 4.1.

Parameter category: Strength parameters	
Parameter	Description
Su	Peak undrained shear strength
Su_CAUC	Peak undrained shear strength from anisotropic consolidation triaxial compression tests
Su_CIUC	Peak undrained shear strength from isotropic consolidation triaxial compression tests
Su_DSS	Peak undrained shear strength from simple shear tests
SU_FV	Peak undrained shear strength from field vane tests
Su_rem	Remoulded undrained shear strength
PHI_peak	Peak effective angle of friction
PHI_TC	Peak effective angle of friction in triaxial compression
PHI_rem	Remoulded effective angle of friction
PHI_cv	Constant volume effective angle of friction
PHI_res	Residual effective angle of friction
c_peak	Peak effective cohesion intercept
c_res	Residual effective cohesion intercept
St	Sensitivity number (ratio of peak over remoulded shear strength)
Sens	Sensitivity (qualitative)
consist	Consistency (qualitative, e.g. "soft", "firm" "stiff" etc.)
Parameter category: Stress history parameters	
Parameter	Description
age	Age of a deposit (in years)
Age	Qualitative evaluation of the age of a deposit (e.g. "young")

sigma_p	Preconsolidation pressure
OCR	Overconsolidation ratio
OC_state	Overconsolidation state (qualitative, e.g. "NC")
Parameter category: Deformation parameters	
Parameter	Description
nu_d	Drained Poisson's ratio
E	Elastic modulus (Young's modulus)
E_d	Drained elastic modulus
E_ds	Drained secant elastic modulus
E_dt	Drained tangential elastic modulus
M	Constraint modulus
M_dt	Drained tangential constraint modulus
M_ds	Drained secant constraint modulus
G	Shear modulus
Parameter category: Compressibility parameters	
Parameter	Description
mv	coefficient of volume compressibility
Cc	compression index
Cr	recompression index
comp	compressibility (qualitative)
Parameter category: Flow parameters	
Parameter	Description
Cv	coefficient of consolidation
k	coefficient of permeability
Parameter category: Particle size distribution parameters	
Parameter	Description
d50	grain size corresponding to 50% passing
d10	grain size corresponding to 10% passing

C _{cur}	coefficient of curvature
C _u	coefficient of uniformity
grading	grading (qualitative e.g. "well graded")
Parameter category: Density parameters	
Parameter	Description
e	void ratio
n	porosity
gamma _{dry}	dry unit weight
gamma _{sat}	saturated unit weight
gamma _{bulk}	bulk unit weight
d _{dry}	dry density
d _{sat}	saturated density
d _{bulk}	bulk density
MDD	maximum dry density (compacted)
D _r	relative density
Relative _{density}	relative density (qualitative e.g. "very loose")
Parameter category: Stress parameters	
Parameter	Description
k _o	coefficient of earth pressure at rest
sigma _{vo}	effective overburden pressure
sigma _{ho}	effective horizontal pressure
sigma _m	mean effective stress = $(\sigma_{vo} + 2\sigma_{ho})/3$
Parameter category: Field test parameters	
Parameter	Description
N _{SPT}	Number of blows from SPT
q _c	Cone resistance from CPT
q _T	Corrected cone resistance from CPTU
K _D	Dilatometer horizontal stress index

mu_2D	Field vane shear strength correction factor, after Bjerrum
mu_3D	Field vane shear strength correction factor, (including 3D end effects)
mu_Aas	Field vane shear strength correction factor, after Aas
Parameter category: miscellaneous	
Parameter	Description
wtp	water table location (qualitative e.g. "above water table")

Table 4.1 The representation scheme for ground parameters

4.4 Implementation in the system.

The representation schemes for rock and soil were implemented in the system as an object base, which is a part of the **ground_representation** application. This object base is made up of a top level object, **Ground**, which has two subclasses, **Soil** and **Rock**. Below these two objects are the specific classes of soil and rock, hierarchically structured in the order shown in Figures 4.1, 4.2 and 4.3.

The representation scheme for ground parameters was also implemented in the system as an object base, part of the **GPar** (Ground Parameters) application. It consists of the top level object **Parameters**. The subclasses of the **Parameters** object, are the parameter categories, and the subclasses of each one of the parameter categories are the corresponding parameters (in accordance with Table 4.1).

The object names for parameters are shorthand descriptions of the actual parameter names. For example the name of the object representing the undrained shear strength from triaxial isotropically consolidated compression tests is **Su_CIUC**. Each object contains a slot, *name*, which contains a string with a more detailed

description of the parameter. Each parameter object also contains a *format* slot which, depending on the evaluation of the parameter, can be bound to either "quantitative", or "qualitative". In the first case, the *format* slot contains a *units* facet, with the units of the parameter (if any). In the second case, a multi-value facet named *per_val* (permissible values) is attached to it, containing one or more lists with the permissible values, which are used to qualify the parameter. For example, the qualitative parameter plasticity can be evaluated from either the list (low, intermediate, high, very high, extremely high), or alternatively from the (lower, upper) list. It should be noted that qualitative parameters can only be evaluated from the qualifiers defined in the permissible values facet. Of course, the user has the ability to modify these lists of permissible values. The slots and facets of a quantitative and a qualitative ground parameters are presented in Figure 4.4.

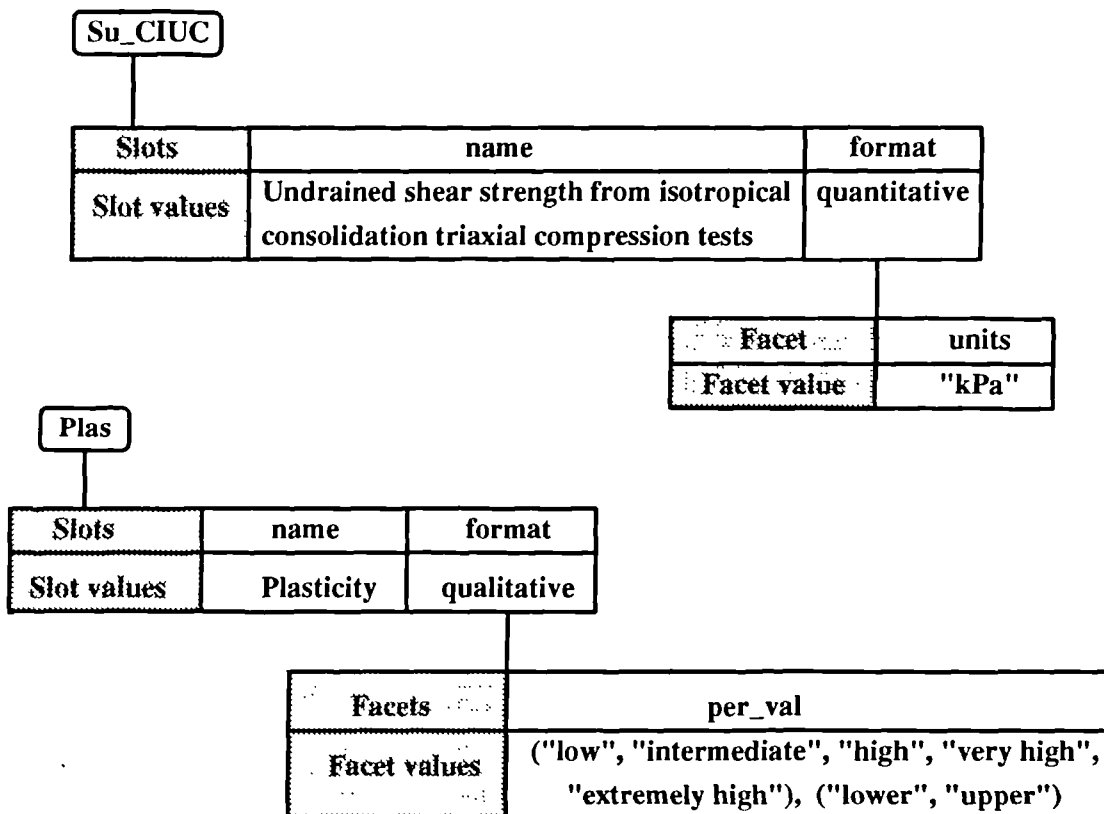


Figure 4.4 The slots and facets of the **Su_CIUC** and **Plas** objects.

4.5 The representation of "typical" values.

As mentioned in §4.1 there exist typical ranges of values for certain parameters within a ground class. Bearing in mind that a ground class is defined by specifying ranges of values for the parameter or parameters the classification system is based upon, it becomes apparent that typical values describe relations between these and the parameter they evaluate. Furthermore, the extent of the range of typical values of a parameter is usually in direct proportion to the degree of refinement of the corresponding ground class. This can be explained by considering that the more specific is a ground class, the smaller the range of values of the classifying properties becomes. This subsequently leads to a decrease in the range of typical values.

It should be stressed that, taking for example the case of soils, whose classification is predominately based on organic content and grain size, there are very few relations that merely rely on these two ground parameters (and these are crude). The establishment of more reliable relations requires the evaluation of yet more ground parameters (especially parameters related to soil or rock descriptions); for example grading and relative density for coarse soils or plasticity for fine soils. These properties can be easily evaluated either from visual inspection, or from simple classification tests. A common characteristic of these parameters is that they are mostly of a qualitative format. Therefore they are evaluated from one of the specified lists of qualifiers (the permissible values, see §4.4).

During the collection of published summaries of typical values it became apparent that the estimated parameters were mostly of a quantitative format (e.g. angle of friction, dry density etc.). This observation along with the above remarks form together the requirements that a representation of typical values of ground parameters should meet.

Two additional requirements affecting the selection of the representation type are the ability to alter the estimated values of the parameter and to add new parameters that affect this parameter's evaluation. The former requirement is dictated by the nature of this type of knowledge, which is empirical. Therefore, as new knowledge is accumulated the ranges of values for a parameter may need to be reconsidered. The latter is necessary in cases where the influence of a new parameter affecting the evaluation of the parameter that is being estimated, is quantified. For example the effective angle of friction for a sand depends on its grading and relative density. It is also affected by the angularity of its grains but currently this effect cannot be quantified. If in the future this knowledge becomes available the system needs to be able to update the representation for the angle of friction of sand without having to reimplement all the knowledge (rather to add to that already existing).

The typical values of a ground parameter for a specific ground type are represented as a slot in the object representing that ground type. The slot is named after the shorthand description of the parameter. Since the estimated parameters are mostly of quantitative format, the slot's value is a list containing their minimum, average and maximum values. If any of this information is not available it is substituted with the letter "u" (indicating that the corresponding value is unknown).

If a parameter can be estimated more precisely based on the evaluation of other parameters (for a specific ground type), then the system creates a number of objects, each of which corresponds to one of the permissible values of these other parameters. The typical values are stored in the slots of these created objects. In the case of a single parameter with n permissible values, the system will create n objects as subclasses of the ground type object. Each object will correspond to one of the n values of the quantifying parameter and the typical values for it will be stored in the estimated parameter's slot, which is inherited from the ancestor ground type class.

The representation of "typical" values of peak effective angle of friction (PHI_{peak}) for a **pure clay**, is presented in Figure 4.5. These depend on plasticity ($Plas$) which is evaluated from the ("lower plasticity", "upper plasticity") set of permissible values. Therefore two objects are created: **lower plasticity pure clay** and **upper plasticity pure clay**.

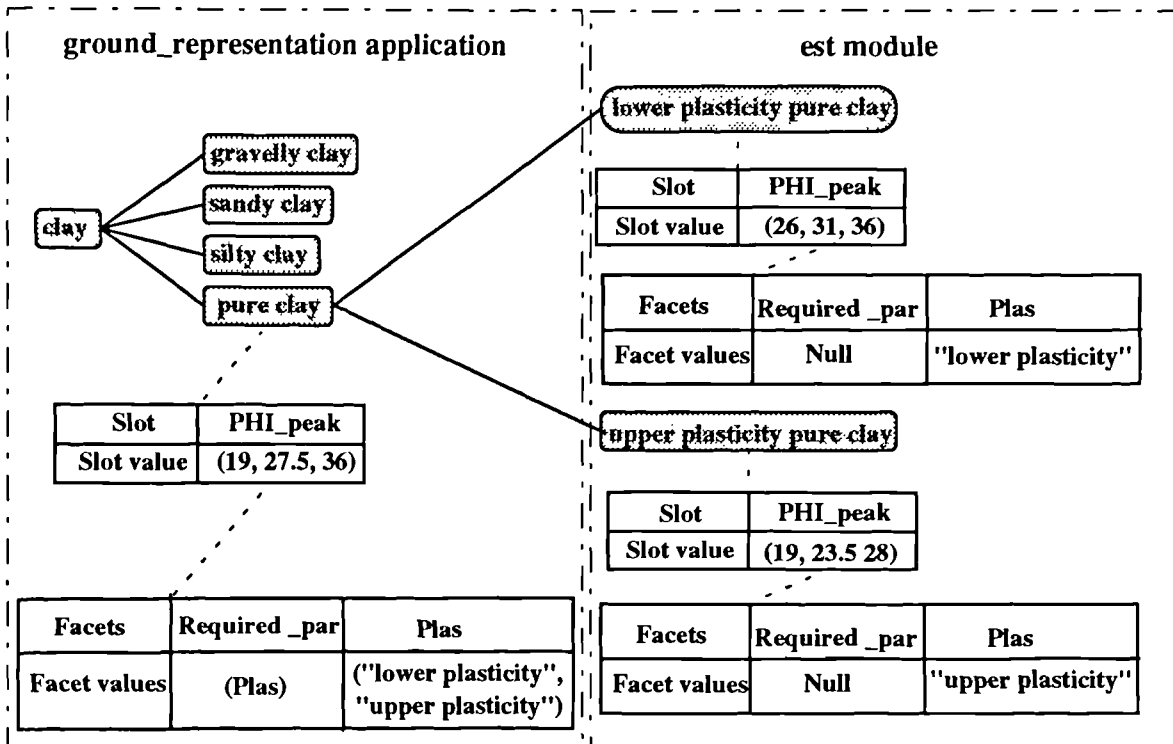


Figure 4.5 The representation of "typical" values of peak effective angle of friction, PHI_{peak} for **pure clay**.

These created objects are stored in the object base of the **est** (estimate) module which belongs to the **ground_representation** application, so that they can be distinguished from the actual ground type objects (stored in the object base of the **ground_representation** application). It should be noted here that the ProKappa software allows objects that belong to an application or module to have parents or children that belong to a different application or module (in this case the ground type object which belongs to the **ground_representation** application, has subclass objects which belong to the **est** module).

When these objects are created, two or more facets are created, attached to the slot which represents the parameter to be estimated (in the original ground type object): the *Required_par* facet, which contains the names of the parameters affecting the evaluation of the estimated parameter; and the parameters' facets, each of which represents a required parameter and contains a list of permissible values for this parameter. The value of the *Required_par* facet in the descendants of the ground type object is set to "Null". The values of the parameters' facets are set to the permissible value of the parameters that the object represents, or to "Null" otherwise.

In the example shown in Figure 4.5 the *PHI_peak* slot on **pure clay** shows the minimum, average and maximum typical values for that object (19, 27.5, 36). However, the *Required_par* facet on the *PHI_peak* slot in **pure clay** is set to (*Plas*) indicating that plasticity affects the evaluation of *PHI_peak*. The *Plas* facet then contains the list ("lower plasticity, "upper plasticity") indicating permissible values for plasticity. On the subclasses **lower plasticity pure clay** and **upper plasticity pure clay** the *PHI_peak* values are specific to the quantifying parameter *Plas* with values of (26, 31, 36) for the "lower plasticity" range. The facets on that slot indicate *Required_par* set to "Null" and the *Plas* facet set to "lower plasticity".

In the case of two or more required parameters the object hierarchy created by the system has as many levels of descendants as the number of the required parameters and the direct subclasses of the ground type object are as many as the sum of the permissible values of all the required parameters. This can be demonstrated by the following example: as mentioned earlier the effective angle of friction of sands depends on grading, with permissible values: ("well graded", "poorly graded"), relative density, with values: ("very loose", "loose", "medium dense", "dense", "very dense") and angularity of grains, with values: ("rounded", "angular"). The objects

that are created by the system (the complete object hierarchy is shown in Figure 4.6) can be divided in three groups:

- The first group (area II in Figure 4.6) contains two objects that correspond to the permissible values of the parameter grading (**well graded sand** and **poorly graded sand**). The *grading* facet of each object is set to the permissible value of grading they represent. The other two parameter facets are set to "Null" (see Table 4.2). The values of effective angle of friction, they contain correspond to well graded and poorly graded sand respectively.
- The second group (area III in Figure 4.6) contains 15 objects relevant to the evaluation of the relative density parameter. Five of these are created as direct subclasses of the **sand** object and correspond to the evaluation of relative density of a sand (Note that the symbol ... is used in the figure to indicate the presence of objects which are not all shown). The other ten are created as five subclasses of the **well graded sand** object and five of the **poorly graded sand** object. For each of these ten objects two required parameters are evaluated, namely relative density and grading. Therefore the values of their corresponding facets are set to the appropriate qualifiers of the two parameters (see Table 4.2).
- Finally the third group (area IV in Figure 4.6) which covers the evaluation of the angularity of grains parameter, contains 36 objects. Two of these are direct subclasses of the **sand** object, 10 are direct subclasses of the objects corresponding to the evaluation of relative density (the five former objects of the previous group), 20 are direct subclasses of the objects corresponding to the evaluation of both grading and relative density (the 10 latter objects of the previous group) and finally four are subclasses of the **well graded sand** and **poorly graded sand** objects respectively.

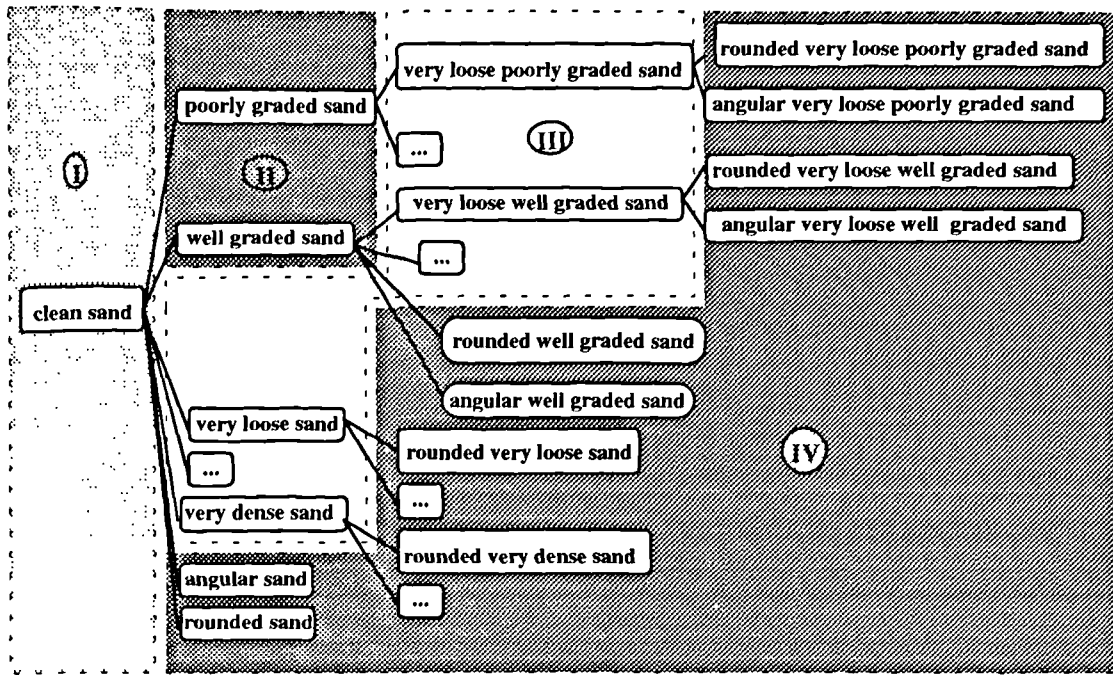


Figure 4.6 The object hierarchy for the representation of typical values of angle of friction for **clean sand** (depending on *grading*, *relative density* and *angularity*).

Facets	Objects		
	poorly graded sand (II)	very loose well graded sand (III)	angular dense well graded sand (IV)
<i>Required_par</i>	Null	Null	Null
<i>grading</i>	"poorly graded"	"well graded"	"well graded"
<i>relative density</i>	Null	"very loose"	"dense"
<i>angularity</i>	Null	Null	"angular"

Table 4.2 The facets of the *PHI_peak* slot of the **poorly graded sand**, **very loose well graded sand** and **angular dense well graded sand** objects.

The representation of typical values is thought to be both efficient and adequate. The representation is adequate because its structure provides the ability to estimate the parameter in question (in this case the effective angle of friction) even when information for all the three required parameters is not available: e.g. when only

information of grading and angularity of grains is available; it is also efficient because the desired information can be retrieved directly from the slot, whose facets are set to the user specified values: e.g. grading is set to "well graded", angularity of grains is set to "angular" and relative density is set to "unknown" (or is not set at all).

Furthermore, it satisfies the imposed requirements (ease of modification and expandability) since the values of each slot can be easily accessed and modified; and also new parameters can be added to the "Required parameters" slot and all the corresponding objects will be created at the end of the already existing object hierarchy.

4.6 User interface facilities.

A user interface has been developed that allows the user of the system to browse and update the ground and ground parameters object bases. It is also used for accessing and displaying typical values of ground parameters for a selected ground type. The user interface has been developed based on the **DialogBox** system provided by the ProKappa environment (§3.2.3).

The user interface session initiates with the appearance of the "menu" dialog box (Figure 4.7). This dialog box incorporates a list box which contains six options. Each of these options initiates a different section of the user interface. Selection of the first or second option (by clicking on to the desired option and pressing the "OK" button in the command row of the dialog box) will result in the appearance of one of the dialog boxes which have been created for browsing and updating the ground and ground parameters object bases.

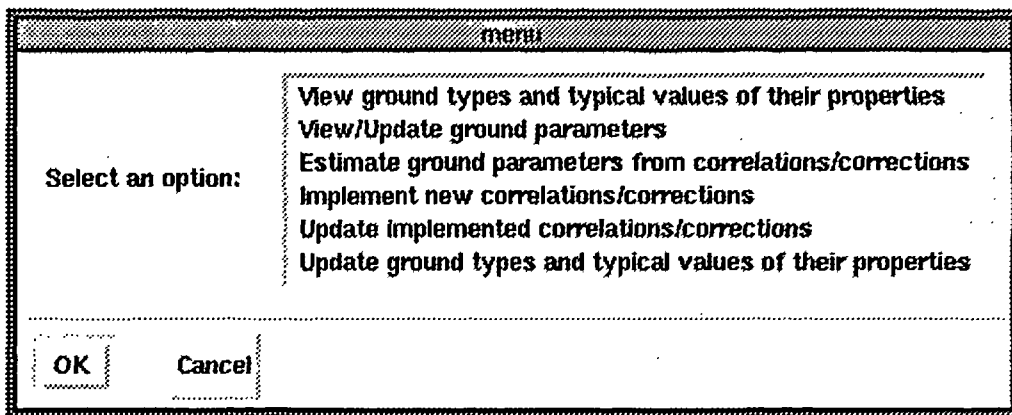


Figure 4.7 The "menu" dialog box.

Each of these dialog boxes contains a list box for displaying the objects of the object hierarchies and three push button controls (labelled "Forward", "Back" and "Reset" respectively) for moving forward and backward in the hierarchies. The list box displaying ground types initially contains two items: "Soil" and "Rock"; the parameters display list box initially contains all the parameters categories.

The user may select a ground type or a parameter category (by clicking on it with the mouse) and then press the "Forward" button. This will invoke the execution of a function that will search the appropriate object hierarchy for the direct subclasses of the selected object. For example if "Soil" has been selected, the list box will display the objects "organic", "inorganic" and "artificial".

Correspondingly, if a parameter category has been selected the list box will display the parameters classified under it. Pressing the "Back" button will cause the list box to display the items of the previous level (e.g. "Soil" and "Rock", or the parameter categories). The dialog box for browsing the ground types object base is shown in Figure 4.8.

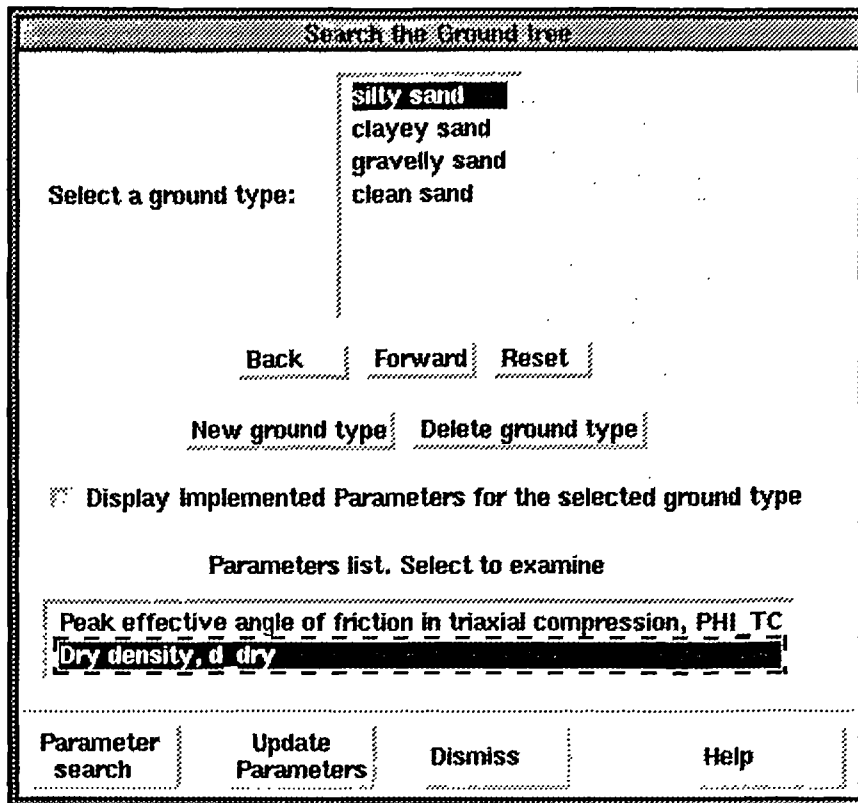


Figure 4.8 The dialog box for browsing the ground types object base.

In the case of the ground hierarchy, since it incorporates more than two levels of subclasses, the system keeps track of the selection history. When the user is moving forward in the ground hierarchy, the system stores all the previous levels in the *UserData* slot (§3.2.3) of the list box object. Each level is represented as a list, which contains the selection items of that level. The lists are ordered according to their selection history. Subsequently the system can display these levels in a reverse order if required by the user. Furthermore, each time a previous level is displayed, the item that had been selected is displayed first, demonstrating the exact route the user has followed.

The dialog box shown in Figure 4.8 can also be used for creating new ground types and removing existing ones. The "New ground type" push button is used for the former task and the "Delete ground type" for the latter.

When the "New ground type" button is pressed a dialog box appears on screen asking the user to specify a name for the new ground type and its parent ground type(s). When the user supplies the dialog box with the required input he/she may press the "Create" push button (in the command row of the dialog box). This in turn will invoke a function which checks if a name is specified, if another object with the specified name already exists and if at least one parent ground type object has also been specified. If no inconsistencies exist, a new object will be created, whose parents are the user specified ground type objects. The newly created object is stored in the object base of the **expand** module which belongs to the **ground_representation** application. As with objects used for storing "typical" values (which belong to the **est** module), user defined ground types are stored in the **expand** module so that they can be distinguished from system supplied ground types (stored in the **ground_representation** application).

The reason for providing the ability to create "user defined" ground types is to allow the user of the system to store ground types which are geographically and/or geologically defined; e.g. London clay. Typical values data can be added to these objects, thus allowing the creation of a user defined (project specific) knowledge base.

The "Delete ground type" button can be used for removing user defined ground types from the ground hierarchy (stored in the **expand** module). The system supplied ones cannot be deleted. This is done in order to preserve the integrity of the system's knowledge base. The removal of a ground type is a three staged procedure: initially a ground type must be selected from the ground types display list box; then the "Remove ground type" button must be pressed; and finally the system pops up a dialog box asking the user to confirm the requested action. It should be noted that if the selected ground type object is a parent of other ground type objects, a dialog box will pop up on screen informing the user that the selected

objects is a parent object and ask him/her if he/she still requires the object's removal, since this will cause the removal of all the lower level objects. A similar dialog box appears on screen when the object to be removed is used for storing parameters' typical values data. The dialog box warns the user that removal of the object will result in the loss of this data and asks him/her to either proceed with or cancel the requested action.

Finally the ground types display dialog box (Figure 4.8) incorporates a check button control ("Display implemented parameters for the selected ground type"), which is used to display the parameters for which typical values exist, for a selected ground type. If this button is clicked on, a second list box appears within the dialog box, which is used to display parameters (this is shown in Figure 4.8). If then a ground type is selected in the ground types display list box, the former will display the parameters, for which "typical" values have been implemented in the system (if any exist). The user then may select from this list a parameter to estimate (by clicking on it with the mouse). This in turn will cause a new dialog box to appear on screen. The dialog box for estimating typical values of dry density for a silty sand is shown in Figure 4.9.

This dialog box contains a list box for each required parameter. Each of these list boxes displays the parameter's permissible values. The appropriate data is obtained from the *Required_par* and the parameters facets (in this case the *Relative_density* facet) attached to the estimated parameter's slot (*d_dry*) of the ground type object. The user then may evaluate the required parameter (select one of the displayed permissible values) and click on the "Estimate" button to obtain the minimum mean and maximum values for the dry density of silty sand. The relevant information is retrieved from the *d_dry* (dry density) slot of the subclass of the **silty sand** object, whose *Relative_density* facet is set to "medium dense" (the name of this object is **medium dense silty sand**). If no value for the relative density is specified and the

"Estimate" button is pressed, the min., mean and max. values displayed will be retrieved from the d_{dry} slot of the **silty sand** object (the same will happen if the value "unknown" is selected).

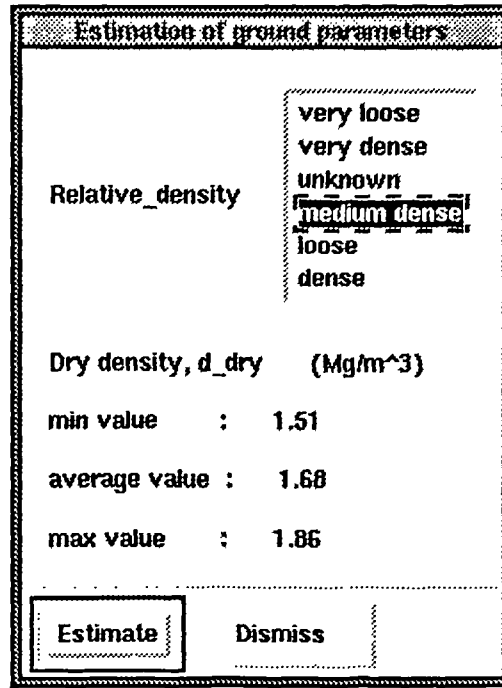


Figure 4.9 A dialog box displaying typical values of dry density for silty sands.

The user interface of the system also provides the ability to search for ground types for which a ground parameter value meets some user defined requirements. For example a search can be performed for ground types for which the angle of friction is greater than a user supplied value. This task is handled by the dialog box shown in Figure 4.10. This dialog box can be called from the dialog box displayed in Figure 4.8, by clicking on the "Parameter search" button in its command row.

The dialog box shown in Figure 4.10 contains a list box that displays parameters, for which typical values have been implemented in the system. The user may select any of these parameters and then specify a value (type a number in the entry box) and a search criterion (click on the appropriate radio button). The search criteria provided cover the following cases: searching for ground types for which the user

supplied value of the selected parameter is either greater (than their minimum value), lower (than their maximum value), or "in range" (greater than the minimum and lower than the maximum). After the user specifies the value and search criterion, he/she may click on to the "Search for:" push button controls to initiate the search. The "Basic ground types" button invokes a function which searches for system supplied objects (ground types in the **ground_representation** object base) that contain a slot named after the selected parameter and the slot's values meet the specified requirements. Correspondingly, the "Specific ground types" performs a similar search for user defined objects (in the object base of the **expand** module).

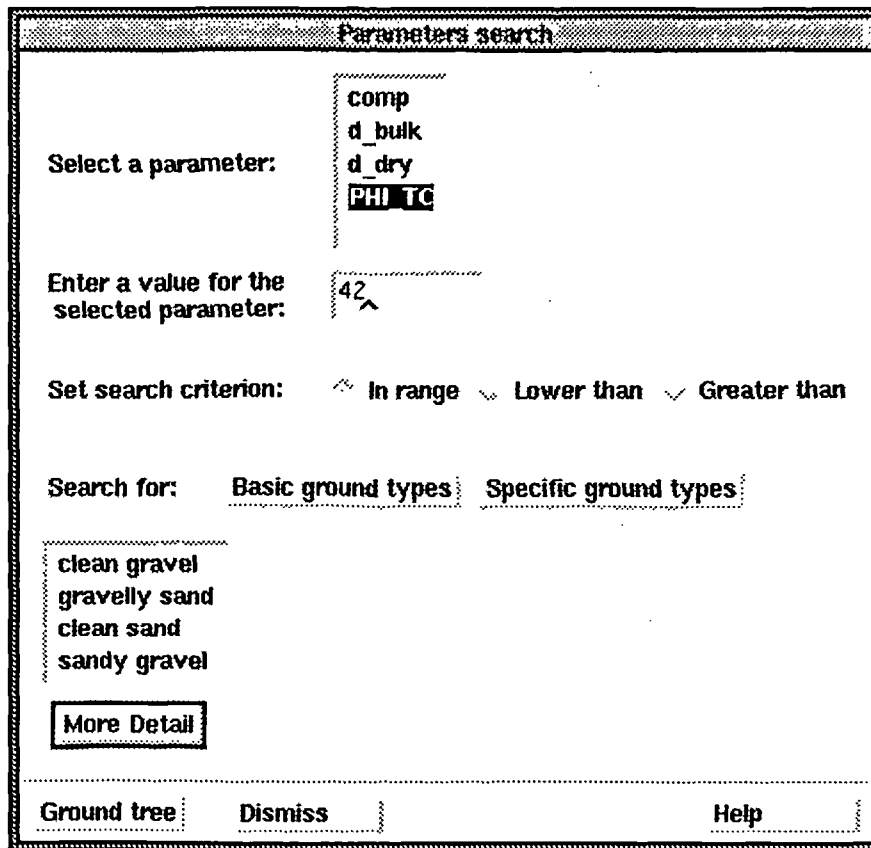


Figure 4.10 The "Parameters search" dialog box.

The results of the search are displayed in the list box below the "Search" push button. A button labelled "More Detail" also appears below the list box. If one of the displayed ground types is selected and the "More Detail" button is pressed, the list box will now display more detailed descriptions of the selected ground type that

meet the user defined search requirements; e.g. if **clean sand** is selected and the "More Detail" button is pressed, the list box may display descriptions such as: **well graded clean sand**, or **very dense well graded clean sand** etc. (Note that the "More detail" objects belong to the **est** module). Furthermore, the label of the "More Detail" button will change to "Back". If the user clicks on the "Back" button the initial list of ground types will be displayed in the list box and the button label will again change to "More Detail".

4.7 A knowledge acquisition module for typical values.

A most important requirement for a KBS is the development of a knowledge acquisition module. Knowledge acquisition modules are useful both during the development stage to enter the necessary knowledge into the system and after this as a means of updating the existing knowledge, so that the system will maintain its functionality in the future. In section 4.6 the ways in which the ground and ground parameters object bases could be updated were presented. In this section the module for updating the typical values of ground parameters will be presented. This module was developed, based on the **DialogBox** system, provided by the ProKappa software (§3.2.3). It is used for adding typical values of ground properties to selected ground types as well as to update the already existing ones.

The implementation of new sets of typical values of parameters, or the updating of already implemented ones is invoked from the dialog box for browsing the ground types object base (Figure 4.8). The user must first select a ground type from the ground types display list box and then click on the "Update Parameters" push button, on the command row of this dialog box. If the button is pushed without previously selecting a ground type a "warning" dialog box will appear on screen prompting the user to select a ground type.

Clicking on the "Update Parameters" button will invoke a function that searches for parameters, for which sets of "typical" values (for the selected ground type) have been implemented. If the search is successful, then the parameters that can be estimated will be retrieved and displayed inside the list box, contained in the dialog box shown in Figure 4.11.

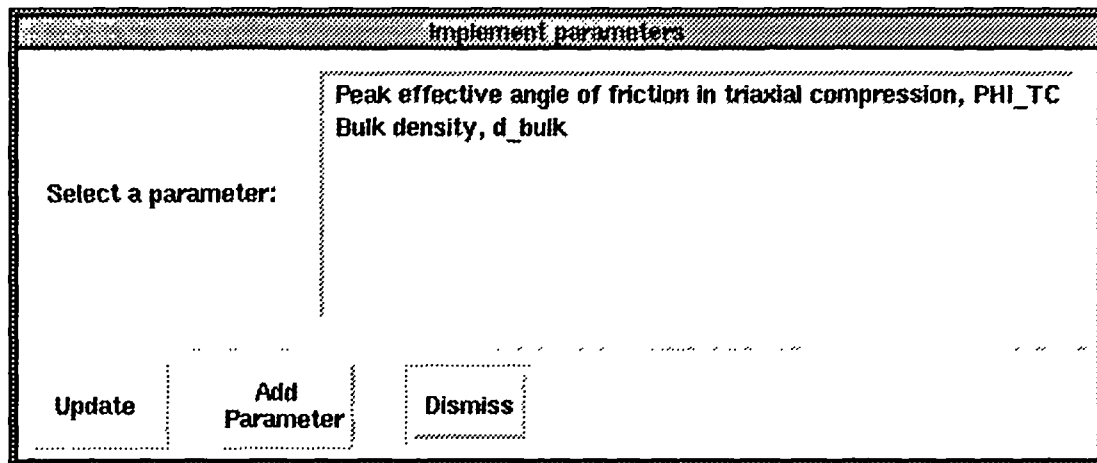


Figure 4.11 The dialog box for selecting parameters to update.

If the desired parameter is displayed in this list, the user may select it and press the "Update" button in the command row. Otherwise, he/she should click on the "Add Parameter" button. This last action will result in the appearance of the dialog box shown in Figure 4.12. This dialog box also appears if the initial search for implemented parameters fails (i.e. no "typical" values for any parameter have been implemented for that ground type).

The dialog box displayed in Figure 4.12 is very similar to the dialog box used for browsing the ground parameters object base. This dialog box is used for specifying the parameter for which "typical" values will be provided. The user must locate the parameter (using the parameters display list box and the "Back", "Forward" and "Reset" button) and then select it. If the parameter does not exist in the ground parameters object base, then it must be created. This can be done by means of either

the "New quantitative parameter", or the "New qualitative parameter" push button in the dialog box.

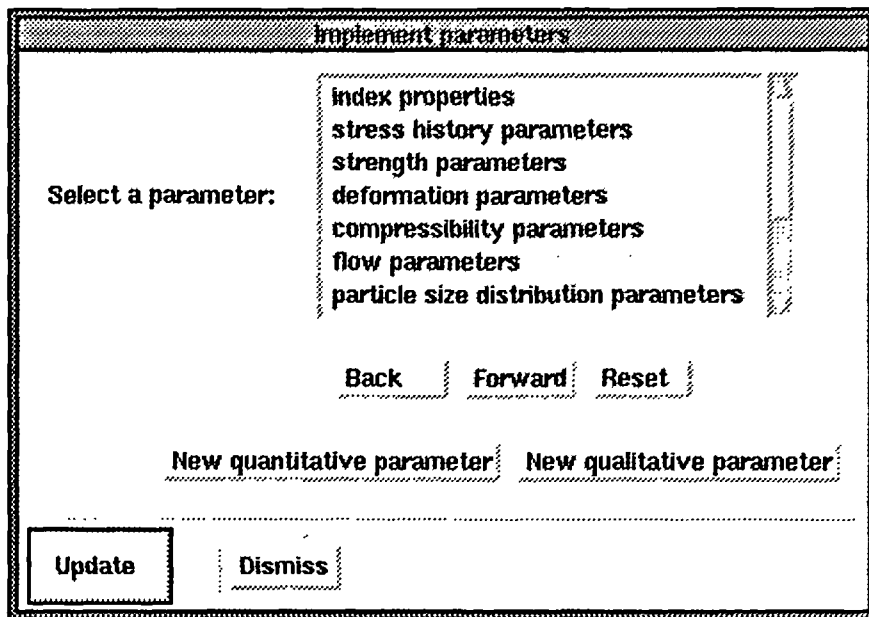


Figure 4.12 The dialog box for specifying the parameter to implement.

After the parameter has been selected the user must click on the "Update" button to continue with the implementation procedure. This action will result in the appearance of a dialog box on screen which is used for specifying the required parameters for the estimation of the selected parameter. The "Required parameters" dialog box is displayed in Figure 4.13.

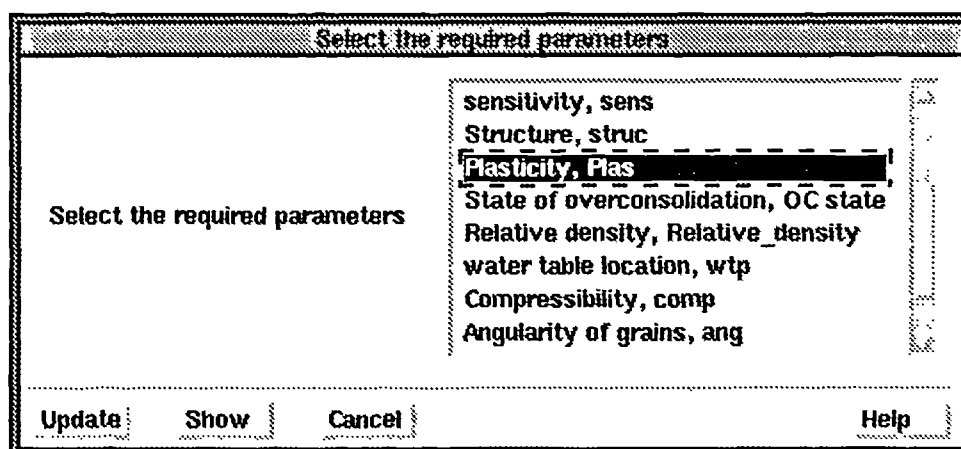


Figure 4.13 The dialog box for selecting required parameters.

Before the "Required parameters" dialog box appears on screen, a function is executed which searches the ground parameters object base and retrieves all the qualitative parameters. This is done in accordance with the observation that required parameters are mostly of a qualitative format (§4.5). The identified parameters are displayed in the dialog box's list box.

When a parameter is selected (from the list box) a function is invoked which retrieves the permissible value set(s) for the parameter. This is followed by the appearance of a dialog box displayed in Figure 4.14 (in this case the selected parameter is plasticity).

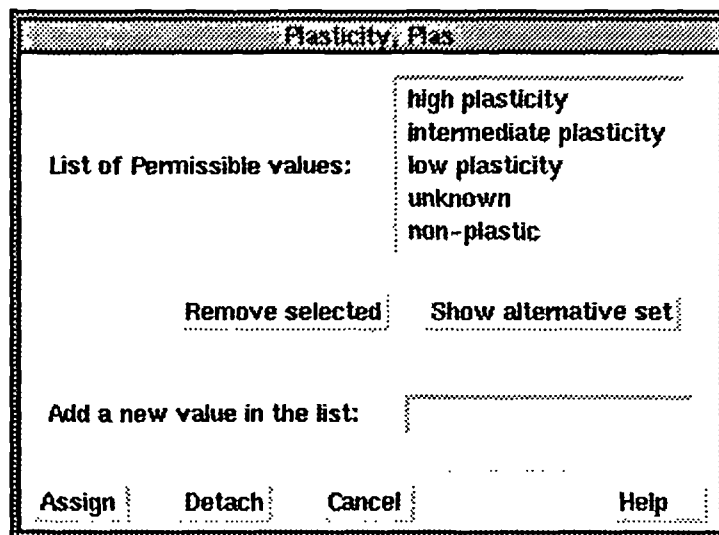


Figure 4.14 The dialog box for assigning a required parameter.

This dialog box contains a list box displaying a set of permissible values for the parameter. If more than one sets exist, then all the remaining sets (except the one being displayed in the list box) will be stored inside a list in the *UserData* slot of the list box object. A push button with the title "Show alternative" (set of permissible values) will appear below the list box. This button incorporates a function for displaying (inside the list box) all the sets of permissible values for the variable. Each time this button is pressed the currently displayed set of permissible values is

placed at the end of the *UserData* slot list. The first element of that list (another set of permissible values) will be taken off the list and its values will be displayed inside the list box. In this way the user can alternatively view all the sets.

The dialog box also contains an entry box incorporating a method for adding new values to the permissible values list box. The new value is typed inside the entry box and by pushing the "Enter" button (on the keyboard), it appears inside the list box. Finally the dialog box also incorporates a "Remove selected" button for removing selected items from the list box. These last two functions allow the user to redefine the permissible values of the parameter. In this case, i.e. if a new set of permissible values is defined, it will be added to those already existing in the *per_val* (permissible values) facet of the *format* slot of the parameter object (in the **GPar** object base).

Finally the dialog box incorporates a command row with an option for assigning the parameter under examination as a required parameter ("Assign"), an option for detaching the parameter ("Detach") from the required parameters' list (if the parameter has already been defined as a required parameter) and an option for dismissing the dialog box without performing any action ("Cancel"). If the "Assign" button is pressed, a temporary facet is created in the *UserData* slot of the required parameters display list box object (Figure 4.13). This facet is named after the shorthand description of the selected parameter and contains a list with the user specified permissible values. Furthermore the name of this facet is added to the list of values of a facet, named *Required_par*, which is also attached to the same slot. The facets of the *UserData* slot are presented in Figure 4.15. Finally the dialog box shown in Figure 4.15 is taken off screen and control of the execution is returned to the dialog box for selecting required parameters (Figure 4.13).

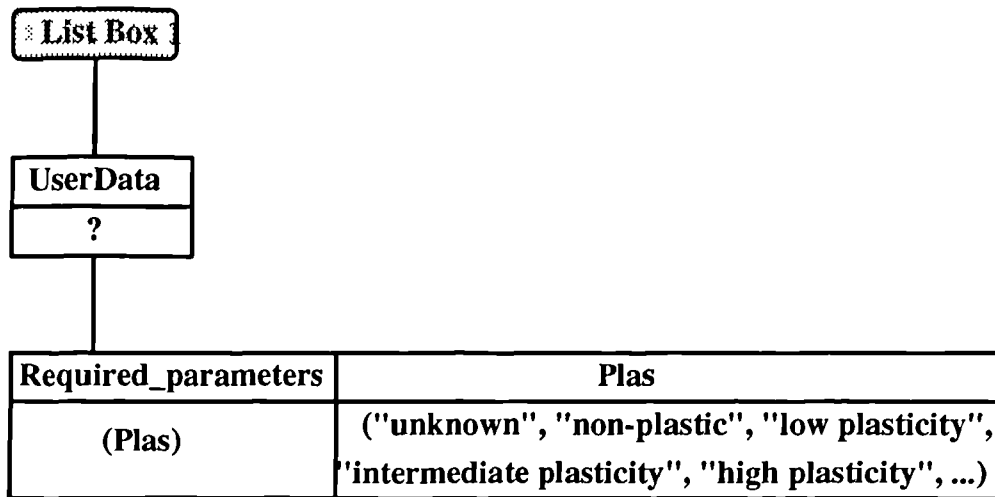


Figure 4.15 The facets of the *UserData* slot.

The procedure described above may be repeated until all the required parameters are specified. The user may then wish to view the specified required parameters. This can be done by pressing the "Show" button in the command row of the Figure 4.13 dialog box. A "Required parameters preview" dialog box will appear on screen containing a list box which displays the required parameters (Figure 4.16). From there the required parameters can be selected for examination. The selection of a parameter will result in the reappearance of the dialog box of Figure 4.14. This time the contained list box will display the specified set of permissible values. The user may then alter the displayed set, remove the parameter from the required parameters list (click on the "Detach" button), or dismiss the dialog box ("Cancel").

It must be noted that in the case of updating the typical values of a parameter that already exist in the system, the "Required parameters preview" window will also display the existing required parameters, even if these are not specified during the current required parameter selection stage.

Finally when the required parameters settings are complete, the user may click on the "Update" button to proceed with the implementation of the typical values. The "Required parameters preview" dialog box will reappear on screen, this time

prompting the user to confirm the requested action. If the user wishes to continue, he/she must click on the "Continue" button at the command row of the dialog box. To cancel the requested action the user must click on the "Reset" button. The "Required parameters preview" dialog box is displayed in Figure 4.16.

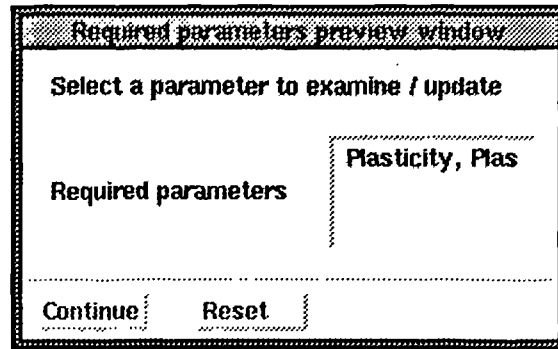


Figure 4.16 The "Required parameters preview" dialog box.

The control of the execution is then passed to a function which retrieves the appropriate information from the *Required_par* and the parameters facets (stored in the *UserData* slot of the required parameters display list box object). This information is used for the establishment of the object hierarchy which will be used for storing the typical values of the selected parameter.

Initially a dialog box appears on screen asking the user to specify the min., mean and max. values of the parameter for the selected ground type (e.g. the coefficient of volume compressibility, m_v for **pure clay**). The user may then supply the requested information (or parts of it; e.g. only the min. value) and press the "OK" button to update the **ground_representation** application. The requested information can be supplied by typing into the appropriate entry boxes (Figure 4.17).

The procedure will continue with the reappearance of the Figure 4.17 dialog box; this time the user will be requested to provide information for the typical values of the parameter for the object representing a more specific ground type (in this case a

pure clay of intermediate plasticity). This process will be repeated until the typical values of the estimated parameter for all the combinations of the permissible values of the required parameter(s) have been evaluated.

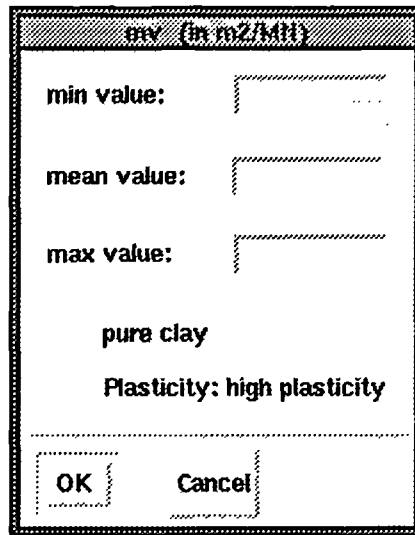


Figure 4.17 The dialog box for implementing typical values of the coefficient of volume compressibility, mv , for a **high plasticity pure clay**.

The course of action of the function which is responsible for the creation of the necessary object hierarchy, is described in detail below. This function is called with one argument (*?parents_list*), which initially is set to a list that contains the basic ground type object, e.g. (**pure clay**). The function then accesses the first required parameter (from the *Required_par* facet of the ground type object). Subsequently it creates all the objects which correspond to the established set of permissible values for the first required parameter (each one in turn). Each time an object is created the dialog box of Figure 4.17 is put on screen, prompting the user to specify the "typical" values for the newly created object. After the "OK" or "Cancel" button is pressed the control of the execution is returned back to the function, which continues with the creation of a new object. Furthermore the function sets the *Required_par* facet value to a list which contains the rest of the required parameters (for all the created objects, including the basic ground type object).

After all the objects, which correspond to the permissible values of the first required parameter have been created, the control of execution is passed to another function. This function checks the *Required_par* facet of the ground type object and its new subclasses, to determine whether or not any contain values. If they do not, it will stop the updating procedure. Otherwise, the initial function will be invoked, this time with the *?parents_list* argument set to a list containing both the basic ground type object and its subclasses. Each of these objects will be used as a parent for the objects that correspond to the permissible values of the second required parameter. The procedure will continue until the *Required_par* facet in all the objects no longer contain any more required parameters.

It should be noted that the initial function will be called as many times as the number of required parameters. Each time, it will create a set of objects that corresponds to the permissible values of a required parameter. An illustration of these sets of objects can be found in Figure 4.6 (each set corresponds to one of the II, III and IV areas).

The procedure for updating the typical values of parameters that have already been implemented is the same as that described above. It should be mentioned that if a set of typical values of a parameter for a ground type (which may be further described by the evaluation of a number of required parameters) already exists, then these values will be retrieved from the corresponding objects and will be displayed inside the appropriate entry boxes as default values (in the dialog box of Figure 4.12). The user is allowed to either confirm or alter the default set of values and update the corresponding object with this information.

4.8 Summary.

A representation scheme for the ground is presented in this chapter. The various ground types are hierarchically structured, based on a limited number of classification parameters. The ground hierarchy initiates with the more general classes (e.g. soil, rock) and develops towards the more specific (e.g. sand, claystone). Each new class level is specified by either the introduction of a new classification parameter, or by the specification of a more restricted range of evaluation for this parameter (compared to the range of the parent class).

Furthermore, a representation scheme for ground parameters is also presented in this chapter. The various ground parameters are grouped into parameter categories, which correspond to different aspects of engineering behaviour. The ground parameters are distinguished in terms of their evaluation as either quantitative or qualitative. The former are evaluated by numbers, the latter through verbal descriptors.

Both schemes were implemented in the system as object bases, which belong to the **ground_representation** and **GPar** applications respectively. Each ground class, parameter or parameter category is represented as an object. The parameter objects contain a *format* slot, which is either bound to "quantitative" or "qualitative". In the first case two facets are attached to the slot, containing the units and the number of decimal points that should be used for the expression of the parameters values. In the second case a *per_val* (permissible values) facet is attached to the slot, containing one or more sets of qualifiers for the parameter.

An analysis of "typical" values of ground properties revealed that each set of typical values corresponds to a specific ground class and the estimated parameter is always of quantitative format. It should also be noted that more restricted ranges of typical

values can be established for more specific ground classes. For example the range of variation for dry density of a well-graded, medium dense clean sand is smaller than the corresponding range for clean sand (because the evaluation of dry density of clean sand depends on grading and relative density). When additional parameters are used for the specification of more restricted ranges of "typical" values, then these parameters (called the required parameters) are invariably of qualitative format.

Typical values of ground properties are represented in the system as the value of a slot, named after the estimated parameter (e.g. dry density, *d_{dry}*), which is attached to a ground class object (e.g. **silty sand**). The value of this slot is a list, that contains the minimum, mean and maximum values of the property for the ground class.

When a number of required parameters is introduced, the system creates an object hierarchy for the representation of "typical" values. This hierarchy initiates with the ground class (the basic ground type, e.g. **sand**). Below that a number of subclasses is created, each corresponding to a "more detailed" ground type, defined from the evaluation of one or more of the required parameters (e.g. **well-graded medium dense clean sand**). The "typical" values for each of these "more detailed" ground types are represented as the value of the estimated property slot, that is inherited from the basic ground type object. The "more detailed" ground types are created in the object base in a module (called **est**) in order to distinguish them from the basic ground types (which belong to the **ground_representation** application).

A user interface has also been created for browsing the ground types and ground parameters knowledge bases. It is also used for displaying the "typical" values of properties for selected ground types.

Finally, knowledge acquisition modules have also been implemented in the system for creating new ground types, ground parameters and implementing new sets of "typical" values as well as for updating the already implemented ones.

CHAPTER 5

Representing correlations in a structured form.

5.1 Introduction.

The process of correlation is that of identifying the mutual relationship between objects under examination. When referring to the ground, these objects are the ground properties, and a correlation between any number of them is a measure of their statistical interdependency.

The most common way for evaluating ground properties is through geotechnical testing (either in-situ or in the laboratory). But even when test results are available, there might be a need to interpolate or extrapolate the available results in order to predict the values of the desired parameters in positions, times, or under conditions that have not been considered during testing, for various reasons (such as cost-effectiveness; lack of time or equipment for additional testing etc.). Furthermore it is well known that no laboratory or field test accurately simulates the boundary conditions, modes of loading and stress paths followed in a field problem, and calibration of the results against observations of the performance of the ground is required to account for the discrepancies between measured and actual values of the parameters in question.

Therefore, the extensive use of correlations in the area of geotechnical engineering primarily arises from the need to give predictions for the ground behaviour when

limited or no relevant test results are available. Also empirical correlations are used for calibrating test results against observations of field performance, thus rationalising the selection of parameters for design; to provide comparisons between the results of different tests for the same parameter; and as a means of linking field test parameters with other ground parameters, which exhibit similar general trends but because of the boundary conditions being dissimilar, a theoretically-based linkage is very difficult to achieve. The last case can be extended to incorporate all those cases where a similar general trend between two parameters is identified, but because of complexity of ground conditions, or because of inability to identify and quantify all the influencing factors, an empirical correlation is used to describe their relation (e.g. when relating undrained shear strength (S_u) with SPT N value measurements).

Another major advantage from the use of correlations in the design process is cost-effectiveness. During preliminary design, where very accurate predictions of the design parameters is not a basic requirement, ground parameters can be estimated from correlations together with a limited number of tests. During the final design stage, correlations, suitably validated from test results, can be used for site variability assessment, reducing the amount of testing required for the site.

Finally correlations can be used for cross-checking the consistency of test results, thus identifying variations from normal behaviour. In summary, correlations are simple, easy to use and they provide a cheap, if crude, means for the rapid estimation of ground properties.

Of course the use of correlations can have some important drawbacks, mainly due to their empirical nature, or due to lack of appreciation of the restrictions associated with their use.

When trying to correlate two or more parameters, it is very important that an understanding exists as to why these parameters should be correlated, in other words to establish the physical meaning of the correlation. Correlating is certainly not only plotting a best fit curve from a series of data, because the lack of physical explanation in a phenomenological relationship can lead to erroneous results, when applied to practice.

Secondly the data that will be used for the correlation should be of "good quality". It should be checked for inconsistencies and for "suspicious" values, to ensure that it does not add to the uncertainty of the correlation. In the case where test results are used, coherence of the data can be achieved if standardisation of test types and test conditions is established. Otherwise the increase in the uncertainty of the correlation will reflect the differences between test types and conditions used. Unfortunately there are many correlations in the literature made from "low quality" data, that originate from a mixture of test types and procedures (e.g. correlations of plasticity index with angle of friction for clays, obtained from a mixture of isotropic and anisotropic triaxial compression tests).

It should also be emphasised that the more empirical the nature of the correlation, the higher will be the uncertainty and the more likely that significant deviations from the actual value of the estimated parameter will be observed in the field. A common example demonstrating the above is the use of SPT N value (N_{SPT}) measurements for the prediction of almost every performance property of both sands and clays. For decades engineers have tried to establish relations for a broad range of properties (from friction angle and relative density of sand to undrained shear strength and elastic modulus of clay, and a few others) by simply using the dynamic driving resistance of a particular sampler size. Most of these relations naturally show a large degree of scatter, which is to be expected when parameters that are not directly related are linked through an empirical relation.

Another important point is the age of the correlation. It is very common that young or immature correlations change rapidly when used in practice and as the understanding of the correlated parameters improves [Kulhawy, 1992]. For example the correlation between relative density, D_r and N_{SPT} from Terzaghi and Peck [1948], was subsequently altered by many researchers, to incorporate additional components that improved its precision. In its current form it incorporates nine parameters [Skempton, 1986], most of them in the form of corrections that account for different testing procedures, and the state of the tested soil. Therefore one should be very cautious when using a correlation which has not been extensively validated against field cases.

The most common source of error originates from the extrapolated use of a correlation; i.e. when a correlation is used for ground types and conditions that are different from those that were accounted for during its development. The associated risk in such cases should be quantified, or at least be the subject of sound engineering judgement. This of course implies a knowledge of the theoretical background of the correlation (if any), and the realisation of the need to cross-check the correlation's results against information from more reliable sources. Again the extensive use of a correlation in field cases will provide a better feel for its reliability and its limitations.

In conclusion, correlations can be a very useful means for estimating ground parameters, if the negative implications from their use are minimised. The basic requirements towards this direction should be:

- When creating correlations, avoid non rationally-based relations between parameters and use good quality data obtained through standardised procedures.

- When using correlations, avoid those that have not undergone extensive validation, and appreciate the theoretical background and the limitations of the relation.

5.2 Representing correlations in a structured form.

A correlation contains many pieces of information. The summation of these describes the relation between a number of parameters. If these different pieces of information are isolated from each other, the parts that form the correlation can be identified as distinct entities, and subsequently categorised into the following groups:

- The correlation's variables
- The estimation procedure
- The correlation's applicability
- The correlation's reliability
- Comments on the correlation

It is believed that these five parts adequately represent the information associated with a correlation. They will be presented in more detail in the following sections.

5.2.1. Variables.

The term variables is used here to describe both the independent and dependant variables of the correlation.

The independent variables, henceforth called **variables** for reasons of simplicity, represent all the input information, required for the execution of the correlation. Variables are usually, but not exclusively, ground properties, either alone or in a combination of more than one. Therefore they can be further subdivided into basic and intermediate variables. The term intermediate is here used to underline that these variables are the product of a combination of basic variables, and therefore need to be evaluated indirectly.

Also variables can be divided into those which have numerical values and those which are described through verbal descriptors. The former will be termed quantitative, and the latter qualitative variables. It should be noted here that intermediate variables can only be quantitative, while basic variables can be either of the two types.

A quantitative basic or intermediate variable can be fully described by its name, units (if any), and the maximum and minimum values, defining the limits of application of the correlation with respect to the variable. The range of applicability, can be any subgroup of the range from which the variable can be evaluated; e.g. there are correlations [Lerouiel *et al* 1983] for which plasticity index can only take values between 5 to 70, even though it can generally take values between 0 and 140 or more. This is imposing a restriction on the correlation preventing extrapolation outside its limits of application. Therefore this information should never be neglected.

It should also be emphasised, that by distinguishing between basic and intermediate variables, restrictions at both levels can be explicitly incorporated into the body of the correlation.

A qualitative basic variable can be fully described by its name and a list of words, henceforth termed the permissible values, which are used as its qualifiers. These values can either be translated into numbers, and therefore directly incorporated in the execution procedure, or used as pointers to invoke alternative estimation procedures. In the first case, translation will be performed through a mapping function, the definition of which should be incorporated into either the description of the variable, or the estimation procedure.

The dependant variables, henceforth called **parameters**, represent the totality of the results produced by the correlation's execution. In most cases there is only one parameter that is to be estimated, but there are also correlations which produce additional results. The term intermediate parameters will be used to describe all the additional results, generated through a multi-staged procedure, towards the estimation of the basic parameter.

A correlation proposed by Aas *et al* [1986] demonstrates the need to identify and describe intermediate parameters as parts of a correlation. This correlation is used for estimating the average undrained shear strength of clays in triaxial compression, extension, and direct shear, from field vane shear strength, plasticity index and effective overburden pressure. The estimation is performed as a two-staged procedure. In the first stage the variables are used to produce an estimation of the state of overconsolidation of the clay. Depending on whether the clay is classified as normally consolidated or overconsolidated, different relations are used in the second stage, in order to produce an estimate of S_{uav} . The intermediate parameter, state of overconsolidation, is an essential requirement for the execution of the second stage, and is also yet another piece of information obtained through the correlation. This can possibly be used in further analyses, cross-checking of data, or even as a variable in another correlation.

It should also be noted, that intermediate parameters and the parameter of the correlation are also subdivided into quantitative and qualitative.

5.2.2. The estimation procedure.

This is the part of the correlation describing the interdependency of variables and parameters in a mathematical and/or logical form. In its simplest form, it consists of a single formula linking the basic variables with the parameter of the correlation. In its most complex form, it becomes a three-staged procedure. During the first stage, the intermediate variables are calculated from the corresponding basic variables, through their predefined functions. In the second stage all the intermediate parameters are estimated. Finally the basic parameter of the correlation is estimated in the concluding stage of the estimation procedure.

The estimation procedure can also be simple or complex in terms of its incorporated mathematical form. Usually it features simple equations, but often iterative procedures are used for the estimation of the basic variable; e.g. the relative density of sands can be estimated from cone penetration test results, through an iterative procedure, [Jamiolkowski *et al*, 1985]. Irrespective of its simplicity or complexity, the estimation procedure is the most suitable and straightforward part of a correlation for algorithmic representation.

5.2.3. Applicability.

The applicability is the part of a correlation containing the restrictions that define the limits of its application. There are many different kinds of restrictions associated with a correlation, the ground type(s) for which the correlation applies,

being the most typical of these. The applicable ground types can be very broad categories (e.g. all coarse soils, clays), or in some cases very specific (e.g. relatively uniform, inorganic, clean, fine to coarse sands).

Furthermore, the state of the ground, as described from a number of ground properties (density, plasticity, mineralogy, ageing, overconsolidation state etc.), can impose additional restrictions to the application of a correlation. For example, a correlation for the estimation of relative density from the cone tip resistance from CPT tests [Jamiolkowski et al, 1985], is relevant to relatively uniform (grading), clean, normally consolidated (overconsolidation state), predominately quartz (mineralogy) sands where k_0 (the earth pressure at rest), is about 0.45. Therefore, the definition of its applicability requires an evaluation of the above ground properties (stated inside brackets).

There even exist correlations for which an initial assessment of the parameter in question is required in order to confirm its applicability. For example there are correlations for the estimation of the undrained shear strength applicable only to soft clays. To identify a clay as soft, one needs to actually perform an evaluation of its undrained shear strength (in this case to assume $S_u < 40$ kPa). Fortunately, most of the above parameters can be easily assessed through visual inspection and simple classification and strength tests.

It becomes apparent that in order to apply a correlation with confidence, for a specific ground type, one has to compare its type and properties with the applicable types and properties of the correlation. This in turn requires information about the ground type and at least a crude evaluation of the properties relevant to the restrictions.

For a formal evaluation of the applicability of a correlation, each comparison could be assigned a weight, that would provide a measure of its relative importance, and a score, describing the goodness of the comparison. Detailed comparisons with weights and scores, would require a quantitative evaluation of how the applicability of the correlation alters, when departing from the predefined standards (the restrictions of applicability). Unfortunately, very rarely is such information incorporated into correlations, and even then, it is inadequate for quantitative evaluation. For example, it is known that if correlations for the estimation of compressibility of inorganic clays are used to estimate the compressibility of organic clays, they will tend to produce much lower values, but a quantitative assessment of how much lower is not feasible.

Therefore, even though comparisons for checking the applicability of a correlation should always be performed, formal methodologies for assessing applicability that incorporate quantitative aspects, can not be applied, for the above reasons. Alternatively, the applicability of a correlation with respect to specific ground types and properties restrictions could be expressed in terms of a linguistic variable, which can accept the values low, medium and high applicability. In this way it is ensured that qualitative information with respect to the applicability will be represented in the system. For example, with consideration of the previous example, the applicability of the correlation for inorganic clays can be described as high, while for organic clays it would be low.

5.2.4. Reliability

The reliability of a correlation is a measure of the validity of the estimated results, with respect to the uncertainties associated with the estimation process, and the risk involved with their subsequent use for analysis, or design.

Uncertainties are always introduced in any evaluation process, and therefore also in correlations. Some represent the inherent material and property variability, some represent measurement errors, and some represent modelling inadequacies, or inaccuracies (transformation error) [Kulhawy, 1992]. In correlations especially, further uncertainties are introduced when a mixture of results from different soil types and test types and procedures is used in the dataset for the establishment of the correlation. Even though the individual effects of each type of uncertainty are difficult to assess and quantify, their overall effect results in a scatter of the data points around the correlation's function. This scatter can be represented by the coefficient of fit, r^2 and the standard deviation, σ (statistical interpretation). The reliability of a correlation can be further described by the number of data points, n , since larger datasets provide a more reliable estimation of the above statistical parameters.

The extent of evaluation and the age of a correlation are also crucial factors for assigning values to its reliability. The use of an "immature" correlation, that has not been extensively evaluated in field problems, is always associated with higher risks, and therefore lower reliability. Of course these risks can not be easily quantified, but equally they should not be neglected when assessing reliability.

Therefore the evaluation of reliability can involve both quantitative and qualitative aspects. The former will be represented through statistical parameters (r^2 , σ , n), wherever such information is available. The latter is expressed in terms of a linguistic variable (reliability score) with verbal descriptors, such as low, medium and high. The qualitative evaluation of reliability may be based on information obtained from the literature (either from the authors of the correlations, or from others, who have subsequently used and modified the original correlation).

Finally, it should be noted that reliability will be decreased as a result of the extrapolated use of the correlation outside of the predefined ranges as described by restrictions on the correlation's variables. The same applies when the correlation is used for ground types and conditions different from those described by the correlation's applicability.

5.2.5. Comments.

Comments are all the additional information relevant to a correlation that cannot be included in any of the above categories. This information will be presented in a textual form. It can be in the form of explanations, that will help the user understand the background of the correlation; additional warnings with respect to the applicability and reliability of the correlation; and recommendations relevant to the subsequent use of the correlation's results for analysis and design. The reason for including comments in the correlation's body, is to increase the confidence of the user and to warn him/her when necessary.

5.3 Implementation in the system.

Correlations have been implemented in a hierarchically structured object base, which forms a part of the **Correlation** application. This application is supplemented with three application modules (ProKappa applications and modules are presented in §3.2.1). The **Correlation** application and the **Correction** module (corrections are discussed in §5.3.2) are used for storing correlations in the system. The **CorrUI** module is the correlation's user interface and the **Update** module is a knowledge acquisition module for implementing new correlations in the system (or updating those already existing). In the following sections of this chapter a detailed

review of the **Correlation** application and the **Correction** and **CorrUI** modules will be presented, with emphasis on the functionality of the system as a whole. The knowledge acquisition module is presented in Chapter 6.

5.3.1 The Correlation application

Introduction.

Each correlation is represented as an object in the object base of the **Correlation** application. This object base consists of a top level object (**Correlations**) and all its subclasses, which are the correlations. Each correlation object contains all the necessary information, featuring variables and parameters, estimation procedure, applicability, reliability and comments of the correlation, in the form of slots and facets (slots and facets are presented in §3.2.1) attached to the object. All of the slots and their facets will be presented in detail in this section. All correlation objects contain the following slots:

- Variable slots. Each variable (basic or intermediate) will have its own slot for input of values.
- A *Parameter* slot, which contains a detailed description of the basic parameter.
- Parameters slots. Each parameter (basic or intermediate) will have its own slot for output of values.
- A *Data_Check!* slot. A method slot for performing a check on the values of variables.
- Applicability slots, for storing the applicability restrictions.
- A *Reliability* slot, for storing the reliability information.
- An *error* slot, for storing errors and warnings.
- A *Comments* slot, containing the comments of the correlation in a textual form.

- A *Parameters_needed* slot, containing a string with a description of the parameter and the basic variables of the correlation followed by the correlation reference.
- A *winame* slot, containing the name of the dialog box object through which the correlation will be represented at the interface stage (§5.3.3) and
- A *wintitle* slot, containing the title of the dialog box, which also has an *author* facet (the correlation's author(s) and date) attached to it.

All the above slots (except the Variable slots and Parameter slots) and their corresponding facets are defined in the **Correlations** object and inherited by every correlation (since every correlation object is a subclass of **Correlations**). The *DataCheck* method (the value of the *Data_Check!* slot) is also defined in the **Correlations** object. An example showing the slots of a correlation object (**Su7LI**) is presented in Figure 5.1.

Slot Edit View Instrument	
	Su7_LI
Comments(mv)	?
Data_Check!	"?Correlations.Data_Check!"
error(mv)	?
High_Applicability(mv)	(clay@ground_rep, sens)
LI	(0, 1)
Low_Applicability(mv)	?
Medium_Applicability(mv)	?
Parameter	"Undrained shear strength, Su"
Parameters_needed(mv)	"Su from Liquidity index, LI, Skempton and Northey, 1952 (7)."
Reliability	"low"
Su(mv)	(81.0, 97.0 ...), (6.0, 9.0 ...) ...
winame	Su7
wintitle	"Su = f(LI), Skempton and Northey, 1952"

Figure 5.1 The slots of the **Su7LI** correlation object

Variable slots.

Each variable of the correlation is represented as a single value slot. The slot name is a shorthand description of the variable (e.g. LI, for liquidity index) and its value is either a list of numbers (one or more values for a quantitative variable), or a string (for a qualitative variable). A number of facets is also attached to the variable slot, defining the type (basic or intermediate variable) and the format (quantitative or qualitative) of the variable, the type and name of interface control (required by the interface module, as will be shown in §5.3.3), a facet, named *variable*, containing a string with a more detailed description of the variable (e.g. the variable facet of the slot LI, is bound to "liquidity index, LI"), and finally facets containing the restrictions of applicability relevant to the variable. An example of the facets for the *LI* slot of the *Su7LI* correlation object is given in Figure 5.2.

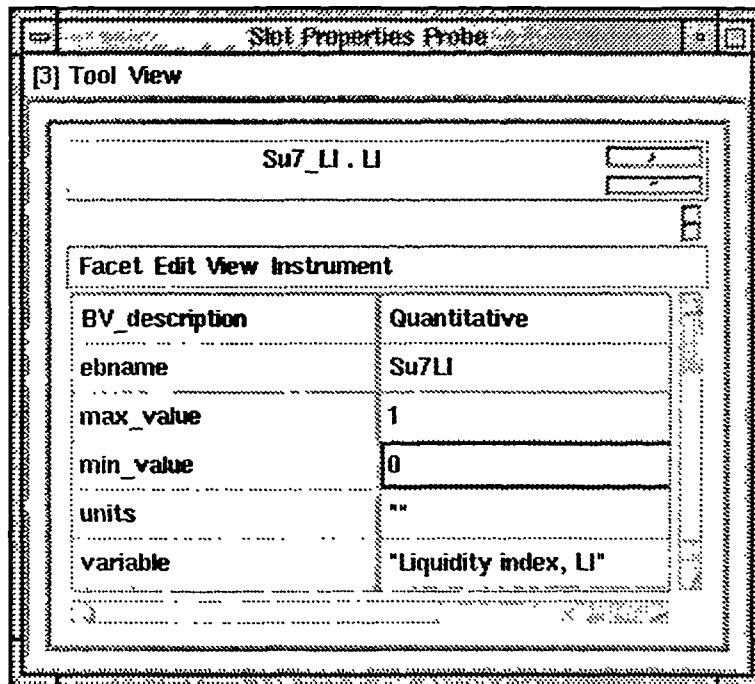


Figure 5.2 The facets of the *LI* slot of the *Su7LI* correlation object

In the case of a basic variable, a single-value facet named *BV_description* (Basic Variable description), is created, which depending on the format of the variable, can

accept the values qualitative or quantitative. In the case of an intermediate variable, a facet named *IV_description* (Intermediate Variable description), is created, that can only accept the value quantitative (as mentioned in §5.2.1).

If the *BV_description* or *IV_description* facet of the variable is bound to quantitative, then four more facets are created in the slot, namely the *max_value* facet, containing the maximum value of the variable for which the correlation applies, the *min_value* facet, containing the minimum value, the *units* facet containing the units of the variable, and the *ebname* facet (entry box name), containing the name of the entry box object, through which the variable will be represented at the interface stage (this is discussed in §5.3.3). In the case of an intermediate variable the facet is called *tdname* (text display name) containing the name of the Text Display object, through which the variable will be represented at the interface stage. The maximum and minimum values define the range of applicability of the correlation with respect to the variable, and the inclusion of units in the definition of the variable minimises the danger of errors, as a result of using wrong units. An example of a quantitative basic variable slot and its corresponding facets is shown in Table 5.1. It should be noted that an intermediate variable slot also contains the same facets, the only distinguishing characteristic being that the *BV_description* facet is substituted with the *IV_description* facet.

If the *format* facet is bound to qualitative, a facet named *per_val* (permissible values) is created, containing a list with the permissible values of the variable; e.g. in a correlation for the estimation of relative density, the quantitative variable compressibility is introduced, which can take the values low, medium, and high, [Jamiolkowski *et al*, 1985]. A second facet, *lbnname* (list box name), is also created, which contains the name of the list box object, through which the variable will be represented at the interface stage (see §5.3.3). An example of a qualitative basic variable slot and its corresponding facets is presented in Table 5.1.

Quantitative basic variable		Qualitative basic variable	
Slot name	Slot value	Slot name	Slot value
<i>LI</i>	(0.8, 0.9)	<i>Plas</i>	"low"
Facet names	Facet values	Facet names	Facet values
<i>BV_description</i>	Quantitative	<i>BV_description</i>	Qualitative
<i>ebname</i>	Su1LI	<i>lbname</i>	Su1Plas
<i>variable</i>	"liquidity index, LI"	<i>variable</i>	"Plasticity, Plas"
<i>max_value</i>	1.2	<i>per_val</i>	("low", "intermediate",...)
<i>min_value</i>	0.0		
<i>units</i>	?		

Table 5.1 Basic variables slots and their facets.

Parameter slots.

The basic parameter of the correlation is represented by two slots. The name of the first, which is a multi-value slot, is a shorthand description of the parameter and its values can be ordered lists of triples or series of strings, depending on the format of the parameter.

The second (single-value slot), is named *Parameter* and its value is a string with a more detailed description of the basic parameter. A *parameter* facet, which contains a pointer to the multi-value parameter slot, along with facets containing control objects names (see §5.3.3), and a *format* facet, describing the parameter's format, are attached to the *Parameter* slot. If the *format* facet is bound to quantitative, then a *units* and a *num_of_dec* (number of decimal points) facets are also created. The *Parameter* slot also incorporates a *SlotFormula* facet, which contains a pointer to the slot formula object used for the representation of the estimation procedure and an *estp* facet containing the user defined text for the creation of the estimation procedure function (both are discussed in the Estimation Procedure section).

Finally, the *Parameter* slot also includes facets containing names for Text Display objects for the representation of the correlation's results and a *cr* facet which contains the name of the command row of the dialog box which is used for representing the correlation at the interface stage (see §5.3.3). If the basic parameter is of quantitative format, it will contain the *tdname*, *tdmin*, *tdmax*, *tdav* and *tdmam* facets (representing the name, minimum, maximum, average and overall min., mean and max. values of the parameter respectively). Otherwise, the *Parameter* slot only contains a *tdname* facet. An example of the facets of the of the *Parameter* slot on the **Su7LI** correlation object is given in Figure 5.3.

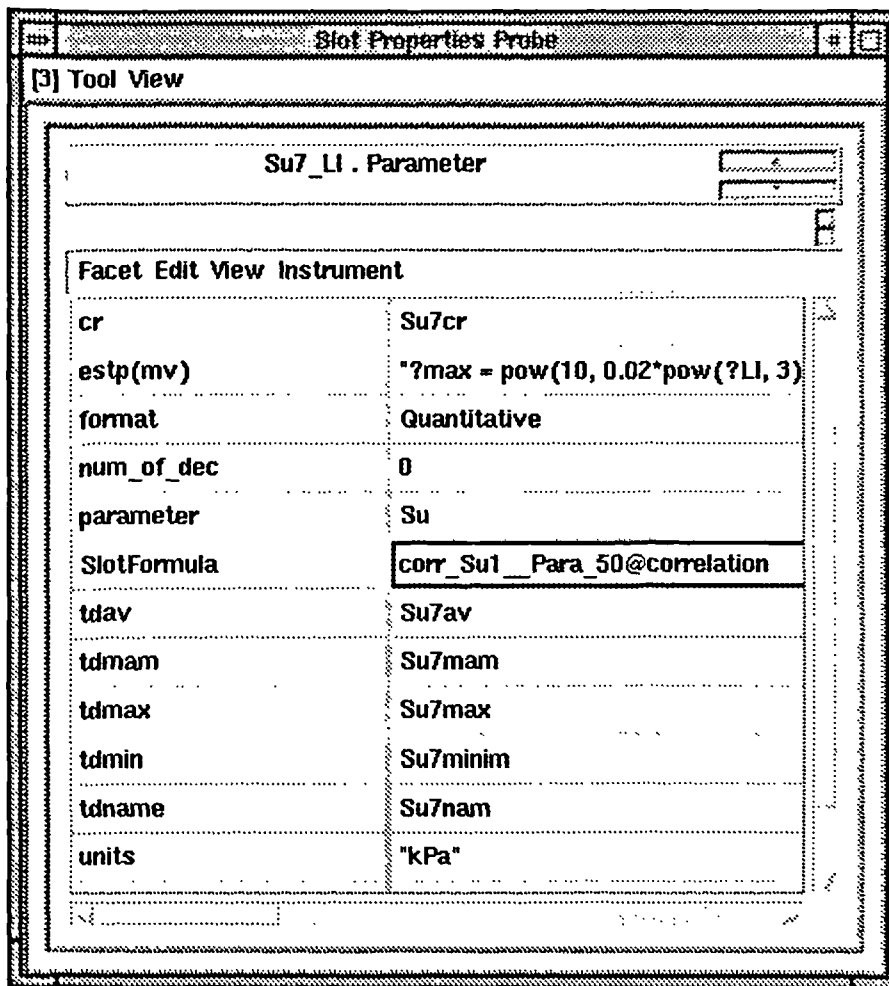


Figure 5.3 The facets of the Parameter slot of the Su7LI correlation object

Intermediate parameters are represented as multi-value slots with names that are shorthand descriptions of the parameter. The slots and facets for a quantitative and a qualitative intermediate parameters are presented in Table 5.2.

Quantitative intermediate parameter		Qualitative intermediate parameter	
Slot name	Slot value	Slot name	Slot value
<i>mu</i>	(0.8, 0.82, 0.85), (0.83, 0.85, 0.87), ...	<i>OC_state</i>	"normally consolidated", "lightly overconsolidated"
Facet names	Facet values	Facet names	Facet values
<i>IP_description</i>	Quantitative	<i>IP_description</i>	Qualitative
<i>parameter</i>	"Field vane correction factor, after Bjerrum"	<i>parameter</i>	"Overconsolidation state"
<i>tdname</i>	Su11munam	<i>tdname</i>	Su10OC_statetd
<i>tdmin</i>	Su11mumin		
<i>tdmax</i>	Su11mumax		
<i>tdav</i>	Su11muav		
<i>tdmam</i>	Su11mumam		
<i>units</i>	?		
<i>num_of_dec</i>	2		

Table 5.2 Intermediate parameters slots and their facets.

A facet, *IP_description* is attached to the slot, which is either bound to quantitative or qualitative, depending on the format of the parameter. For quantitative parameters, the slot values are lists of ordered triples, each of which contains the estimated minimum, average, and maximum values of the parameter, for a set of single values of the variables. For qualitative parameters, their slot values are a series of strings, each obtained from a set of single values of the variables. A *parameter* facet, containing a more detailed description of the parameter, and facets containing names for text displays (*tdname*, *tdmin*, *tdmax*, *tdav*, *tdmam*) for the representation of the correlation's results (see §5.3.3), are also attached to the parameter slot. Finally, if the parameter's format is quantitative, a facet, named

units, containing the parameter's units, and a facet *num_of_dec*, containing the number of decimal points that will be used in the expression of the results, are also created. The number of decimal points is in accordance with the expected accuracy of the correlation's results and the relative magnitude of the parameter (e.g. relative density, usually ranging between 0.0 and 100.0, is presented with one decimal point, and dry unit weight, usually between 1.00 and 2.50, with two).

Data_Check! slot.

Each correlation object inherits a *Data_Check!* method slot to provide the relevant value checking functions. The *Data_Check!* slot contains a method (implemented in ProTalk and stored in a file, which is a part of the **Correlation** application), which checks the variables' values. The method can be invoked if a message is sent to that slot. The course of action of the method follows a number of stages which are associated with functions, called from the DataCheck method, to perform different value checks. An outline of these stages is presented here followed by an analytical description of the checking procedures.

- search for quantitative basic variables
- format checking of quantitative basic variables
- check basic variables' values are in range
- list length checking for all basic variables
- search for intermediate variables
- calculate values for intermediate variables
- check intermediate variables calculated values are in range

Initially, the DataCheck method searches the correlation object for quantitative basic variables. These are identified by searching for slots of the correlation object which contain a facet *BV_description*, the value of which is bound to quantitative.

Following the identification of a *variable* slot, a format check function is invoked. The values of the slot are assessed and checked for being of the correct list format (numbers inside brackets, separated by commas). If this is not the case, or if the slot does not contain any values, a message, informing of the observed inconsistency, will be sent to an error slot, which is also attached to the correlation object. The control will be passed back to the method and execution will continue with the check of another variable.

If no error exists in the format, the method will call another function that checks each of the values of the initial variable, by comparing them with the minimum and maximum values, as these are stated in the *min_value* and *max_value* facets (if these facets contain any values). Every inconsistency will be reported to a *warnings* facet, which is a multi-value facet attached to the *errors* slot. All warnings are explicit to each of the values outside the variable's applicability range: e.g. if the list (-1, 1, 2.2) is assigned to the variable liquidity index, with range of applicability 0 to 2, then two strings will be added to the warnings facet: "the value of LI equal to -1 is not between 0 and 2" and "the value of LI equal to 2.2 is not between 0 and 2".

After checking the first variable, another basic quantitative variable will be identified and the checking procedure will be repeated for it, until all the basic quantitative variables of the correlation have been checked. Backtracking of the method's execution is based on a non-deterministic search, using the ProTalk search modifier *find* (described in §3.2.2).

The last action of the method, incorporates a call to a third function which checks the number of values in the list for each of the basic variables. If all the basic variable lists do not contain the same number of values, an error will be generated because then a one to one representation of the variables' list values is impossible. This can be illustrated if the variables' list values are arranged in a matrix format,

with number of columns equal to the number of variables, and number of rows equal to the number of values of each variable. In this way, each row provides one result set for the correlation's parameters, and the number of rows is equal to the number of the estimated result sets of the correlation's parameters. An example for a correlation with three basic variables, each of which is bound to a list with n elements is shown in table 5.3.

variable 1	variable 2	variable 3	parameters
value11	value21	value31	\Rightarrow result set 1
value12	value22	value32	\Rightarrow result set 2
value13	value23	value33	\Rightarrow result set 3
\vdots	\vdots	\vdots	\vdots
value1n	value2n	value3n	\Rightarrow result set n

Table 5.3 An example of a one to one representation for a three variable correlation.

After the `DataCheck` method has finished checking the quantitative basic variables, and if no errors or warnings have been reported to the corresponding slot and facet, it will try to establish the existence of intermediate variables. If the search for intermediate variables proves successful, a message will be passed on to the function contained into a slot formula object.

Intermediate variables are associated with a slot formula, which is used to store their estimation procedures (slot formulas are presented in §3.2.4). This slot formula is pointed to by a *SlotFormula* facet in the *Parameters_needed* slot. The value of this facet is a pointer to a slot formula object, which is stored in the **AR_Correlation** application module (Active Relations in the **Correlation** application).

The slot formula object contains a user supplied function which is used for the calculation of the values of the intermediate variable. This function is implemented in ProTalk and uses as input the values of the basic variables to calculate values for

the intermediate variable. An example of part of the ProTalk code could be: "?IV_1 = ?Su_FV/?sigma_vo;", which would calculate the values of intermediate variable 1 (?IV_1) as the ratio of the undrained shear strength from the field vane test (?Su_FV) and the overburden pressure (?sigma_vo). It should be further noted that no uncertainty is associated with the calculation of intermediate variables (intermediate variables are calculated rather than estimated).

The calculation function also has to manipulate multi-value lists to produce all the values of intermediate variables, so that each set of values of the basic variables will produce one value. The slot formula object also incorporates a slot which contain references to all the slots, whose values are required by the slot formula for the calculation of the intermediate variable's values (the relevant basic variables slots). The detail of the implementation of slot formulas is presented in §6.4.

After the calculation of the values of all the intermediate variables, control is passed back to the DataCheck method. The method now tries to identify slots that contain *IV_description* facets, and calls the function that checks if each one of their values falls within the range defined by the minimum and maximum values of applicability of the variable (the format check is now omitted since errors are impossible at that stage). Any observed inconsistency is again reported to the warnings facet.

If at the end of the checking procedure the error slot contains any values, then all the corresponding errors should be corrected before proceeding with the estimation procedure. However, the existence of warnings does not prevent the execution of the correlation. In contrast to errors, the decision is totally dependant upon the user of the system, highlighting the need to distinguish between errors and warnings. The former occur when the values of the variables are not of the correct format, or when the sets of the variables' values cannot be defined, thus making the execution impossible. On the other hand warnings are merely extrapolations outside the limits

of the correlation's application, which should not impede its execution. Of course the estimated results may prove erroneous, but the user would be informed about the associated risk. Furthermore, he/she will also be able to assess the correlation's performance outside its applicability limits, and in some cases even reconsider these limits. Finally, it should be remembered that these limits are not always clearly defined and sometimes were determined as a result of inadequately extended testing procedures. For example, a correlation for the estimation of the sensitivity of clays [Skempton and Northey, 1952] is applicable only to clays with a liquidity index, LI, ranging between 0 and 1.2. There is a possibility that this correlation can also provide meaningful results for larger or lower values of LI, but probably because no test results are available, the applicability limits of LI are restrained to the above range.

The Estimation Procedure.

The next step, after the checking of the basic and intermediate variables, is the execution of the estimation procedure of the correlation. The estimation procedure is a user supplied ProTalk function, contained in the slot formula object, pointed to by the *SlotFormula* facet in the *Parameter* slot. The function's input consists of all the basic and intermediate variables' values. The body of the function is an algorithmic representation of the correlation's estimation procedure (stored in the *estp* facet in the *Parameter* slot). This procedure is contained in a loop, which is executed as many times as there are values in the list for the variable, producing thus an equal number of values (or triples of values) for the correlation's parameters. When information for the variation of the values of quantitative format parameters (either in terms of standard deviation, or minimum and maximum limits of variation etc.) is available, this is included in the function. The output of the estimation procedure consists of ordered triples (in the order: minimum, average, and maximum) of values for quantitative parameters, or simple strings for the qualitative

ones. All the output is directly placed in the corresponding slots in the correlation object. The implementation of slot formula functions for the estimation of the correlation's parameters is presented in §6.4.

Applicability slots.

The applicability of the correlation is represented by three multi-value slots. These are the *High_Applicability*, *Medium_Applicability* and *Low_Applicability* slots. Each of these slots may contain one or more lists. Each list should contain at least one element, which is an applicable ground type, and if more elements exist, the ground type will always be the first element of the list. This element is a pointer to an object in the **ground_representation** object base.

Further restrictions on applicability are represented as multi-value facets attached to the relevant applicability slot. Each facet is named after a shorthand description of the parameter (a pointer to the parameter object in the GPar object base; see §4.4) imposing the restriction and this name is also included in the slot. The descriptors of that restriction are either a range of numbers, or some of the permissible values of the parameter (depending on whether this parameters is of quantitative or qualitative format). These descriptors are placed in a list, which in turn is placed in the parameter's facet. The first element of this list is the applicable ground type, followed by the restriction descriptors. In this way each of the restrictions for a specific parameter is linked to the corresponding ground type.

The parameter facet is created as a multi value facet so that it can also accept other lists of restrictions, which correspond to other, also applicable, ground types. For example a correlation which is highly applicable to well-graded sands and poorly-graded gravels, will be represented by two lists in the *High_Applicability* slot (Figure 5.4).

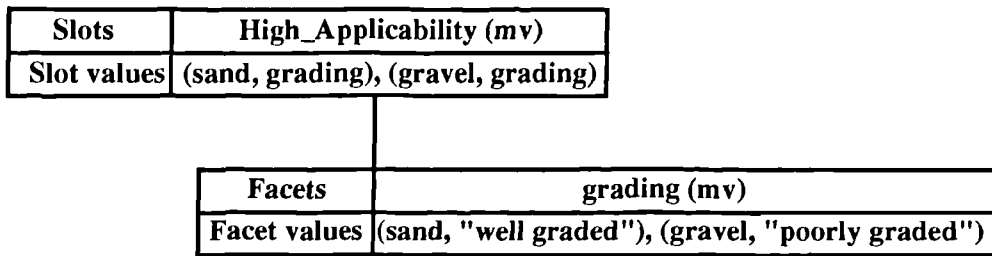


Figure 5.4 The representation of applicability.

The first list contains the element **sand** and the element *grading*. The second contains the elements **gravel** and *grading*. Finally a facet, named *grading* is attached to the *High_Applicability* slot. This facet contains a list with the element **sand** followed by "well graded", and a list with the element **gravel** followed by "poorly graded".

Reliability.

The reliability of the correlation is represented as a single-value slot, which is bound to either "low", "medium", or "high", as described in §5.2.4. The facets *sd* (standard deviation, σ), *r2* (coefficient of fit, r^2), and *n* (the number of data points) are attached to the *Reliability* slot and contain the appropriate information, whenever this is available.

5.3.2 The Correction module.

A special case of correlations are those which describe the relation between two different modes of the same parameter. For example, relations exist between the field vane undrained shear strength and average field undrained shear strength of an embankment or a cutting (e.g. Bjerrum, 1972, Azzouz *et al*, 1983, and Aas *et al*, 1986). Similar relations also exist between undrained shear strengths or peak effective friction angles, obtained through different testing procedures (triaxial

compression, triaxial extension, direct shear, plane strain shear etc.). These relations are usually termed corrections, because the estimation of one type of undrained shear strength from the other, is made through multiplication of the latter with a correction factor (which can be a constant number, or a function of other soil properties). Corrections are implemented in the system, in exactly the same way as correlations. All corrections are subclasses of the **Correlations** object, but they are separated from them, since they belong to the **Correction** module. In the rest of this chapter as well as in the following chapter the term correlations will be used to cover both correlations and corrections, unless it is specifically mentioned otherwise.

5.3.3 The CorrUI module.

The **Correlation** application and the **Correction** module together form the knowledge base of the correlations system. The **CorrUI** module is a user interface, which provides a means of assessing and using correlations to estimate ground properties and present them to the user. Its implementation is based on the **DialogBox** application, provided by the ProKappa software (§3.2.3).

The interface session starts with the appearance of the "menu" dialog box (Figure 4.7). If the user selects the "Estimate ground properties from correlations / corrections" option (and click on to the "OK" button), a dialog box will appear on screen, which is used for searching the Correlation and Correction object bases. This dialog box, named "search", is presented in Figure 5.5. The "search" dialog box contains a list box, with various options for performing restricted searches in the correlation object base. These restrictions are associated with:

- the specification of the correlation's basic parameter,

- the specification of the correlation's variables
- the specification of the correlation's applicable ground types
- the specification of the correlation's reliability
- the specification of the correlation's reference (author - year)

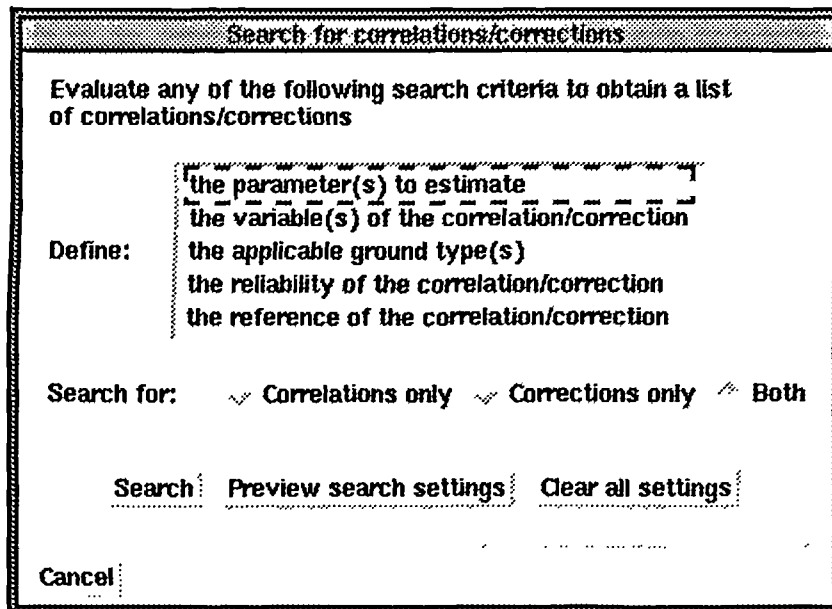


Figure 5.5 The "search" dialog box.

If the "Define the parameter(s) to estimate" option is selected, this invokes the execution of a function, which searches all the correlation and correction objects for the values of their Parameter slots. A dialog box, containing a list of the basic parameters of all the correlations and corrections, will then appear on screen, shown in Figure 5.6.

The user can select as many of these parameters as is required and then press the "Update" button, to include this restriction in the search criteria. The function of the "Update" button is to store the selected parameters in a *par* (parameters) facet attached to the *UserData* slot of the "Search" push buttons object. From there they can be assessed and used as input by a search function.

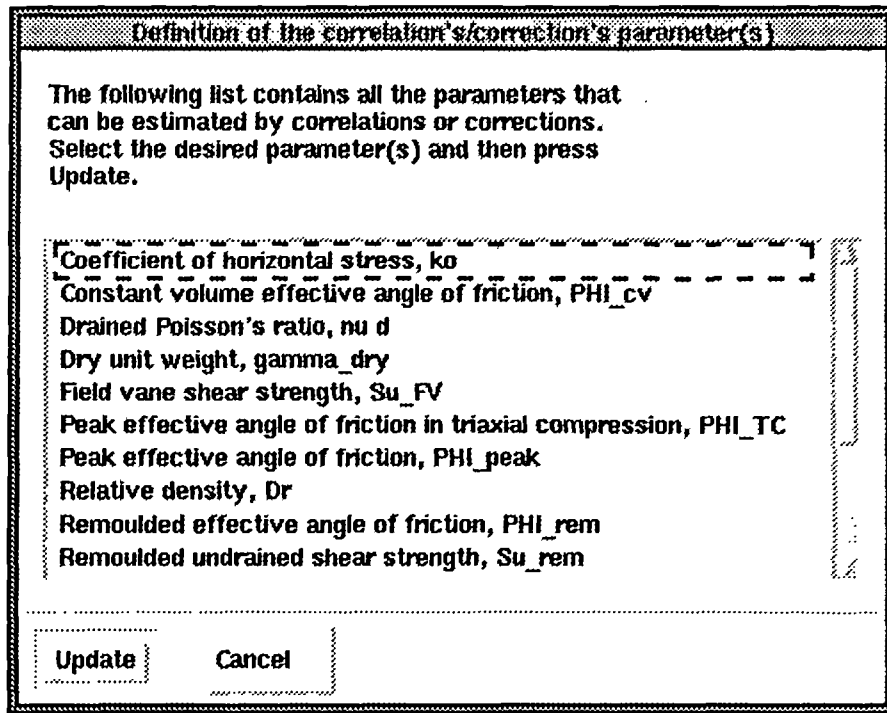


Figure 5.6 The dialog box for specifying the correlation's/correction's parameters.

If after this, the user clicks on the "Search" push button (in the dialog box in Figure 5.5), a search function will be invoked, which will identify all the correlations and corrections that have one of the specified parameters as the basic parameter.

If the "Define the variable(s) of the correlation/correction" option is selected, a similar function is used to produce the list of basic variables. This function searches all the correlation objects for slots that contain a *BV_description* facet. A dialog box, containing a list of the basic variables of all the correlations and corrections, will appear on screen. If some of the listed variables are selected and included into the search criteria (stored in a *var* facet in the *UserData* slot of the "Search" button object), then a search in the correlation and correction object bases will produce a list of correlations, which will contain at least one of the specified variables.

In the case of setting restrictions in terms of the applicable ground types, the corresponding search function will not only search for the correlations and corrections that contain the specified ground types in their applicability slots, but

also for the ones that contain subclasses of each of those. For example if a search is performed for correlations and corrections applicable to gravel, then the search function will also try to establish correlations and corrections that are applicable to sandy gravel, clean gravel etc. The reason for this is that for example clean gravel is a specific case of gravel and therefore each correlation applicable to gravel, is by definition applicable to clean gravel as well.

It should also be mentioned that applicability parameter restrictions are not included in the applicability search criteria (only ground types), because it was thought that their inclusion would affect the functionality of the search module, making it cumbersome and in effect slower. This can be explained by considering that in order to define parameter applicability restrictions, one has first to define the parameters that impose the restrictions and then their restriction descriptors. This is already a three-stage process (selection of the ground type, selection of the parameter and specification of the parameters values or descriptors) which would have to be repeated when defining a second set of applicability restrictions.

Another option for a restricted search for correlations/corrections, is by defining the reliability score(s). The user can specify one or more of: "high", "medium" and "low" reliability score values. Finally, search restrictions can be made by specifying the author-date reference of the correlation. Selection of the last option produces a list of authors-dates references from which the user can select the desired ones.

It should also be noted that the "search" dialog box contains a radio buttons control which can be used to limit the search to either the **correlation** or **correction** object base. By default its value is set on "Search for both" (correlations and corrections).

After defining all the search criteria, the user may press the "Preview search settings" button to check the restrictions settings. This action results in the appearance of the dialog box shown in Figure 5.7.

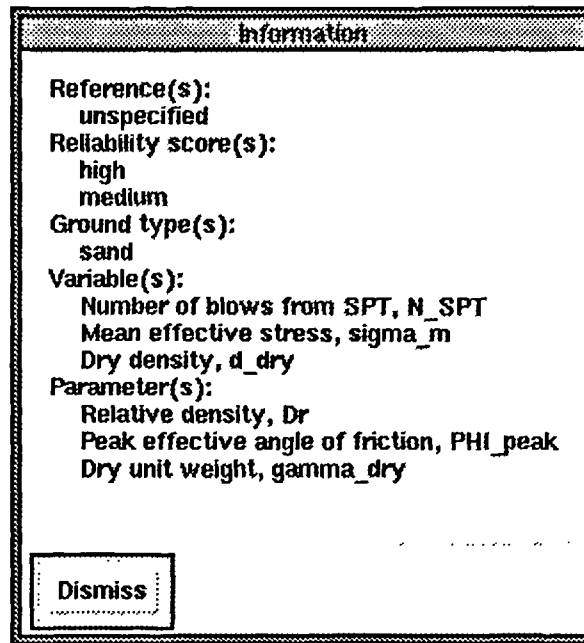


Figure 5.7 The search settings preview dialog box

After dismissing the dialog box which displays the search criteria, the user may change the settings, or press the "Search" button to activate the search function. This function first assesses all the defined restrictions, which have been stored in the appropriate facets of the *UserData* slot of the "Search" button object. Then the appropriate parts of the search function are executed one after the other. The results of each individual search are stored as lists of correlations and/or corrections (depending on the value of the limit search radio buttons) in the *UserData* slot. After this, the result lists are combined to produce an overall list of correlations/corrections, that are common to all of the result lists.

Finally a dialog box appears on screen (see Figure 5.8) that contains the number of correlations and/or corrections identified for each search and also the number for the overall search results. The user may update the search settings and repeat the

search, or press the "Correlate" button (see Figure 5.8) to obtain a list of the identified correlations.

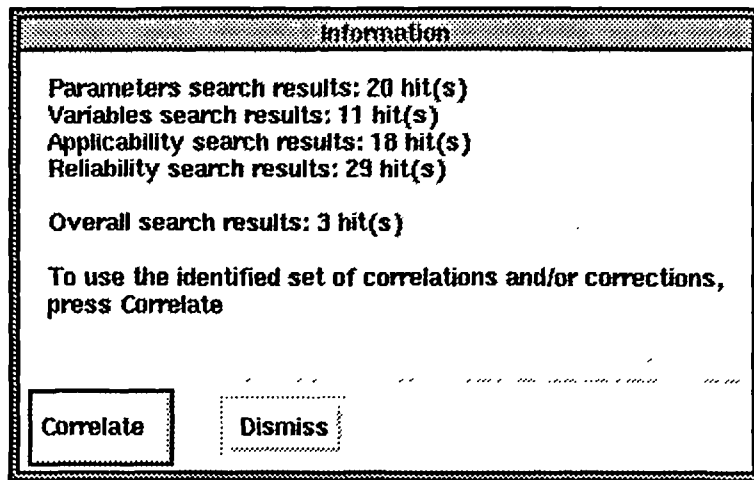


Figure 5.8 The Preview search results dialog box.

If the "Correlate" button is pressed a dialog box (Figure 5.9) appears on screen, which contains a list of the identified correlations. The user may choose from this list any number of correlations to use. Therefore the user is allowed to simultaneously consult more than one correlations for the estimation of a parameter and compare the results (if more than one correlations for that parameter exists). After selecting the desired ones, the user should press the Correlate button. This button activates a function that creates the dialog boxes and dialog box controls, which are used to represent each of the selected correlations.

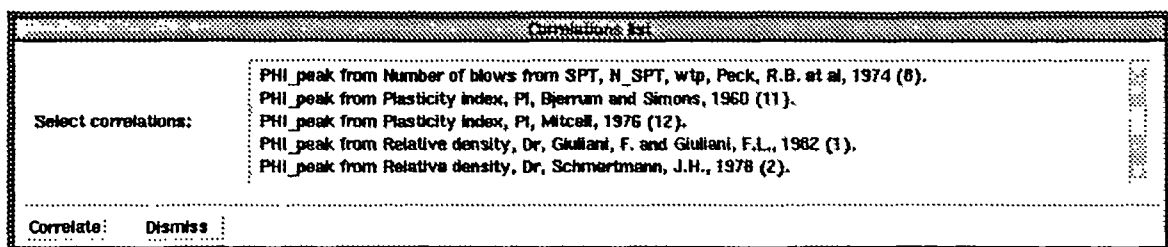


Figure 5.9 The display correlations/corrections dialog box.

Each correlation, is represented as a dialog box window, containing the basic and intermediate variables and parameters and a command row control, for executing the correlation, and displaying its applicability, reliability and comments. The name of the dialog box is obtained from the *winame* slot of the correlation object. The dialog box title is built up from the shorthand descriptions for the basic parameter and variables of the correlation, which are followed by the correlation's reference (obtained from the *wintitle* slot). The information displayed in the title is thought to be adequate for identifying and distinguishing between correlations. An example for a correlation window is presented in Figure 5.10.

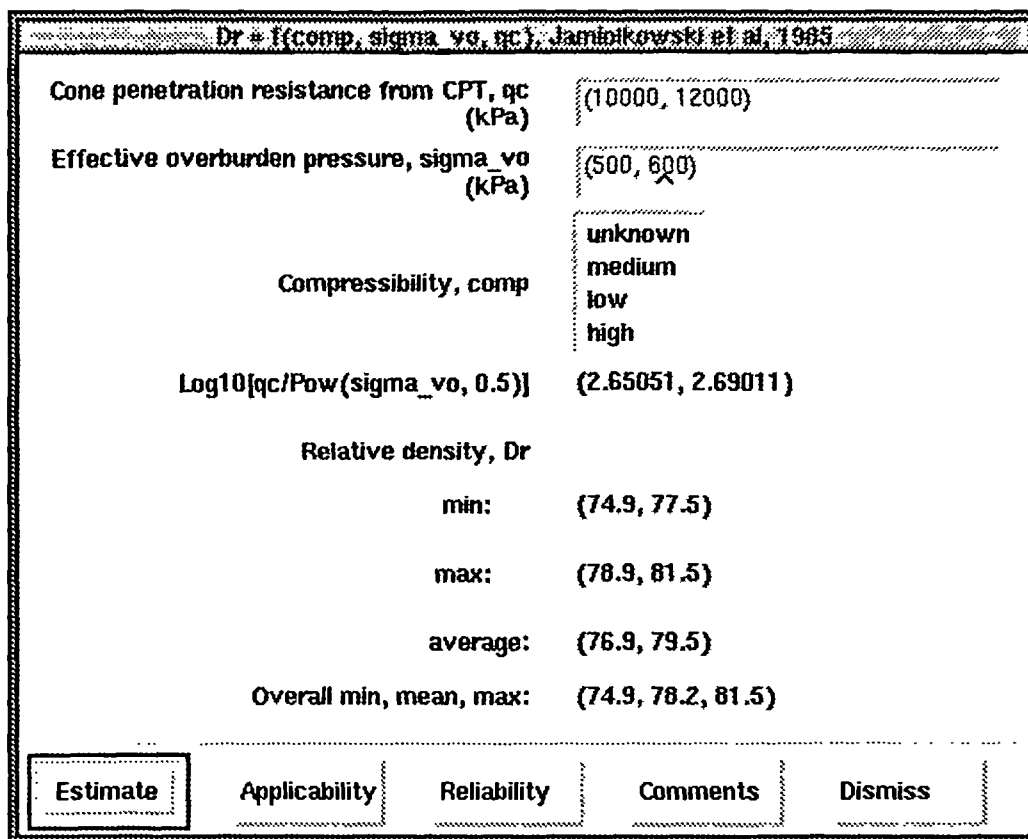


Figure 5.10 A correlation dialog box.

Inside the correlation window and following a top to bottom order, the following features can be identified:

- the quantitative basic variables, which are represented as entry boxes. Figure 5.10 shows two of these (*qc* and *sigma_vo*). The name for each of these entry box objects is obtained from the *ebname* (entry box name) facet of the variable's slot. The title of the entry box is the name of the variable, followed by a list of values. If the slot of the variable in the correlation object contains any values (data may have been imported either manually, or from a data import module), these will be passed on to the entry box as default values. Of course the user can change any of the values before execution of the correlation takes place.
- the qualitative basic variables, which are represented as list boxes. Figure 5.10 shows a list box for compressibility. The name for each of these list box objects is obtained from the *lbyname* (list box name) facet of the variable's slot. These list boxes contain a number of options for evaluating the variable, the totality of which are the permissible values contained in the corresponding facet of the slot representing the variable in the correlation object. The use of a list box for the representation of qualitative variables is justified by the way that such variables are defined. That is they can only accept values contained in the permissible values list. Furthermore, the use of list boxes (instead of entry boxes) simplifies the checking procedure, since incorrect entries are impossible. Finally it should be noted that only one value can be selected each time (single selection list boxes are used).
- the quantitative intermediate variables, which are represented as single line text displays, composed of the title, which is the name of the variable ($\text{Log}_{10}[\text{qc}/\text{Pow}(\text{sigma_vo}, 0.5)]$ in Figure 5.11), and after the execution of the correlation, the calculated values of the variable. The name for each of these text display objects is obtained from the *tdname* (text display name) facet of the variable's slot. It should be remembered that no uncertainty is present in the calculation of intermediate variables from the basic variables, since the former

are simply mathematical combinations of the latter (intermediate variables are calculated, rather than estimated).

- the quantitative intermediate parameters, which are represented as five text display objects. The first obtains its name from the value of the *tdname* facet, attached to the intermediate parameter's slot. It displays the full name of the intermediate parameter. The following three obtain their names from the *tdmin*, *tdmax* and *tdav* facets of the intermediate parameter's slot. These are used to display the minimum, maximum, and average estimated values of the parameter (each set of three corresponds to a single set of basic variables). Finally the last one (obtaining its name from the *tdmam* facet), contains values representing the overall minimum, average (defined as the arithmetic mean of all the average values) and maximum for the parameter.
- the qualitative intermediate parameters, which are represented as single line text displays (obtaining their name from the *tdname* facet of the parameter's slot), composed of the title, which is the name of the parameter and the estimated parameter values, which are expressed in a textual form.
- the basic parameter of the correlation, which depending on its format is represented in the same way as intermediate parameters (either quantitative or qualitative). Figure 5.10 shows values for the relative density parameter.
- the command row, containing buttons for executing the correlation ("Execute"), displaying the applicability, reliability and comments of the correlation ("Applicability", "Reliability", and "Comments", respectively) and a button for dismissing the correlation window ("Dismiss"). Actions are invoked by clicking on the appropriate button.

The "Execute" button invokes a function which identifies the correlation object and sends all the values of the quantitative and qualitative basic variables, contained inside the corresponding entry and list boxes of the correlation window, to the appropriate slots. It then invokes the DataCheck function which checks the imported data for inconsistencies. If any inconsistencies are identified, the DataCheck function fails and execution is passed back to the initial function, which now tries to identify the existence of errors or warnings in the corresponding slot and facet respectively of the correlation object. If the error slot contains any values, then all these are passed on as text to an "Error" window, which subsequently appears on screen. Execution of the correlation stops and the user is prompted to correct the observed inconsistencies, before retrying to re-execute the correlation.

If no errors exist, then all the warnings are passed on as text to a "warning" dialog box window, which subsequently appears on screen. Execution of the correlation is paused but not stopped. The user now has the option to continue the execution of the correlation, without correcting the inconsistent values (by clicking on the "Continue" button in the "warning" window), or to halt the execution (by clicking on "Reset").

If the user selects to continue (or if no warnings or errors exist), then a message will be sent to the function of the slot formula that calculates intermediate variables (if the correlation contains any intermediate variables). The calculation of the intermediate variables will be followed by the execution of the DataCheck function, which now will check the generated intermediate variables for possible inconsistencies. If no warnings are generated (errors are impossible at that stage, as noted in §5.3.1), execution will continue with the estimation of the correlation's parameter(s). This is done by invoking the function of the slot formula attached to the *Parameter* slot (see §5.3.1). The estimated values for the intermediate parameters and the basic parameter will be stored in the appropriate slots in the

correlation object, and from there will be imported into the correlation window as values of the corresponding text displays.

It should be further noted that the system allows for easy modifications of the data contained in the entry boxes and the selected values in the list boxes. The user can alter any number of the input values and then execute the correlation to get an alternative set of results. Sensitivity checking of each variable can therefore be performed easily and quickly.

The "Applicability", button is used to access and display the applicability of the correlation. By clicking on this button a function is invoked which searches the *High_Applicability*, *Medium_Applicability* and *Low_Applicability* slots of the correlation object. The information contained in these slots is retrieved and the appropriate text is automatically generated by the function. The generated text is displayed inside an "Applicability" dialog box. An example of the "Applicability" dialog box for a correlation, highly applicable to uniformly graded, normally consolidated, quartz clean sands, with horizontal stress ratio at rest (k_0) between 0.4 and 0.5 [Jamiolkowski *et al*, 1985] is presented in Figure 5.11.

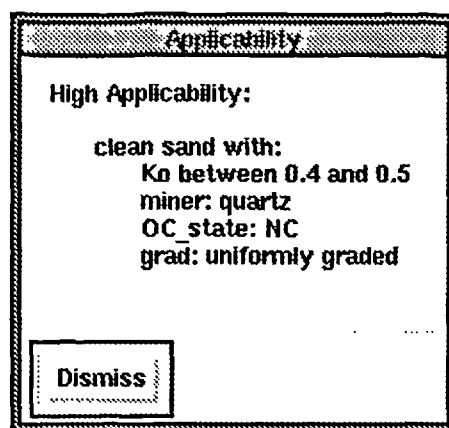


Figure 5.11 An example of the "Applicability" dialog box.

The "Reliability" button invokes a function which retrieves the relevant information from the *Reliability* slot (and its facets) of the correlation object and displays it inside a "Reliability" dialog box. The "Reliability" dialog box for the same correlation is presented in Figure 5.12.

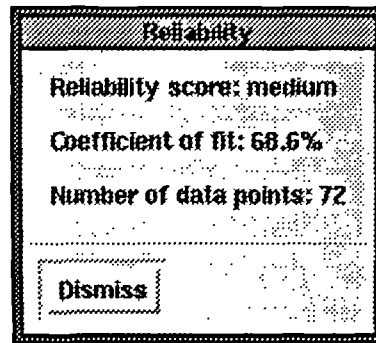


Figure 5.12 An example of the "Reliability" dialog box.

Finally, the "Comments" button activates a function, which searches all the basic and intermediate quantitative variables for applicability ranges. This is done by searching for slots that contain a *BV_description* or an *IV_description* facet whose value is set to "quantitative". Subsequently, the *max_value* and *min_value* facets of each one of the identified variables are accessed. The function, depending on whether the maximum, or minimum, or both values are defined, will generate the appropriate expression. For example if the variable "liquidity index, LI" has an applicability range of 0 to 1.2, the generated expression will consist of the variable's full name ("liquidity index, LI") followed by "should be between" "0" (the minimum value) "and" "1" (the maximum value).

This generated text is appended to the text contained in the *Comments* slot and is placed inside a "Comments" dialog box. An example of the "Comments" dialog box is presented in Figure 5.13.

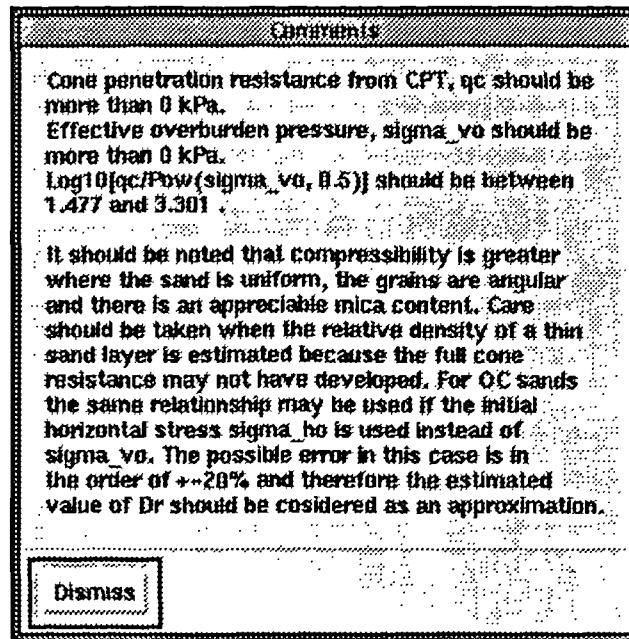


Figure 5.13 An example of the "Comments" dialog box.

5.4 Summary.

The evaluation of ground properties is one of the most important problems in geotechnical engineering. A common methodology for addressing this problem is by using correlations. Correlations are simple, easy to use and they provide a cheap, if crude, means for the rapid estimation of ground properties. The main disadvantages of correlations are the use of non rationally-based relations, the use of relations that have not undergone extensive validation and extrapolations outside their limits of application.

A substantial amount of correlations has been collected and subsequently analysed. The purpose of this analysis was to identify the knowledge types, that describe all the different pieces of information contained in a correlation. The result of this analysis was the identification of the following parts:

- The correlation's variables

- The estimation procedure
- The correlation's applicability
- The correlation's reliability
- Comments on the correlation

The correlation's variables are both the dependant and independent variables. The former are called **parameters** and the latter **variables**. Variables can be divided into basic and intermediate, the former being simple ground or test parameters, while the latter are mathematical combinations of two or more of the former. Equally, parameters are *divided into the basic parameter and the intermediate parameters*. The former is the parameter that is estimated by the correlation and the latter are expressions of all the intermediate results produced during the estimation of the basic parameter. Finally, the basic variables along with the basic and intermediate parameters can also be divided into quantitative and qualitative (the former are evaluated numerically, the latter through textual descriptors).

The estimation procedure is the part of the correlation that describes the interdependency of variables and parameters in a mathematical and/or logical form.

The applicability of a correlation refers to the ground types (e.g. clean sand), and also any parameter restrictions (e.g. grading: well-graded), for which the correlation is applicable. A ground type along with its associated parameter restrictions define an applicability set. The applicability of a correlation is expressed as a combination of these sets with their applicability scores (one of "high", "medium" and "low"). The applicability score is a qualitative expression of the applicability of the set, with which it is associated.

Furthermore, each variable is associated with an applicability range (quantitative variables) or a set of qualifiers (the permissible values of a qualitative variable).

The specification of these ranges or sets of values also defines the limits of the correlation's applicability.

The reliability of a correlation is expressed quantitatively: as a range of variation around the estimated value, from the standard deviation, the coefficient of fit and the number of data points used for the creation of the correlation; and qualitatively: from the reliability score (with values "high", "medium" and "low").

The comments of a correlation are used to represent any additional information (in a textual form) relevant to a correlation that cannot be included in any of the above categories.

A generic form for the representation of correlations has been developed. Each correlation is represented as an object (in the object base of the **correlation** application), which contains slots and facets, which in turn are used for representing the various knowledge types contained in the correlation (e.g. slots for representing variables, parameters, applicability etc.). The only exception is in the representation of the correlation's estimation procedure (which is represented as a function).

Finally, a user interface module has been developed for the use of correlations for the estimation of ground properties. The development of the system is based on the **DialogBox** system application provided by ProKappa.

The user interface provides the user with the ability to perform restricted searches in the correlation application. A number of options is provided by the system which can be used for specifying criteria in a correlation's search (e.g. search for correlations that contain a specific variable or parameter or have specific reliability score etc.). A list of correlation is subsequently presented to the user, who has the option to simultaneously use any number of them.

Each correlation is represented as a dialog box window, containing the variables and their values, and the parameters. This dialog box contains a button, which is used for the execution of the correlation. It also features three options for displaying the applicability, reliability and comments of the correlation.

CHAPTER 6

A knowledge acquisition module for the implementation of correlations.

6.1 Introduction.

The **Correlation** application (presented in §5.3.1) is supplemented by the **Update** module, which is the system's knowledge acquisition module. The **Update** module is used for importing new correlations into the system, as well as for updating the already implemented correlations. Similar to the **CorrUI** module (§5.3.2), implementation of the **Update** module was based on the **DialogBox** application, so that it can be considered as a part of the system's user interface.

The implementation of a new correlation in the system is a six stage procedure. During the first stage the correlation object is created and the basic parameter is implemented. The second stage covers the implementation of the variables and parameters of the correlation. This is followed by the implementation of the correlation's estimation procedure. Finally, the last three stages cover the definitions of applicability, reliability and comments, respectively.

In the following sections of this chapter a detailed description of the updating procedures will be presented.

6.2 Establishment of the basic parameter.

The implementation of a new correlation in the system is initiated by selecting the "Implement new correlations/corrections" option in the Figure 5.7 dialog box. This action results in the appearance of the dialog box (Step1), which is used for defining the correlation's basic parameter. The Step1 dialog box is shown in Figure 6.1.

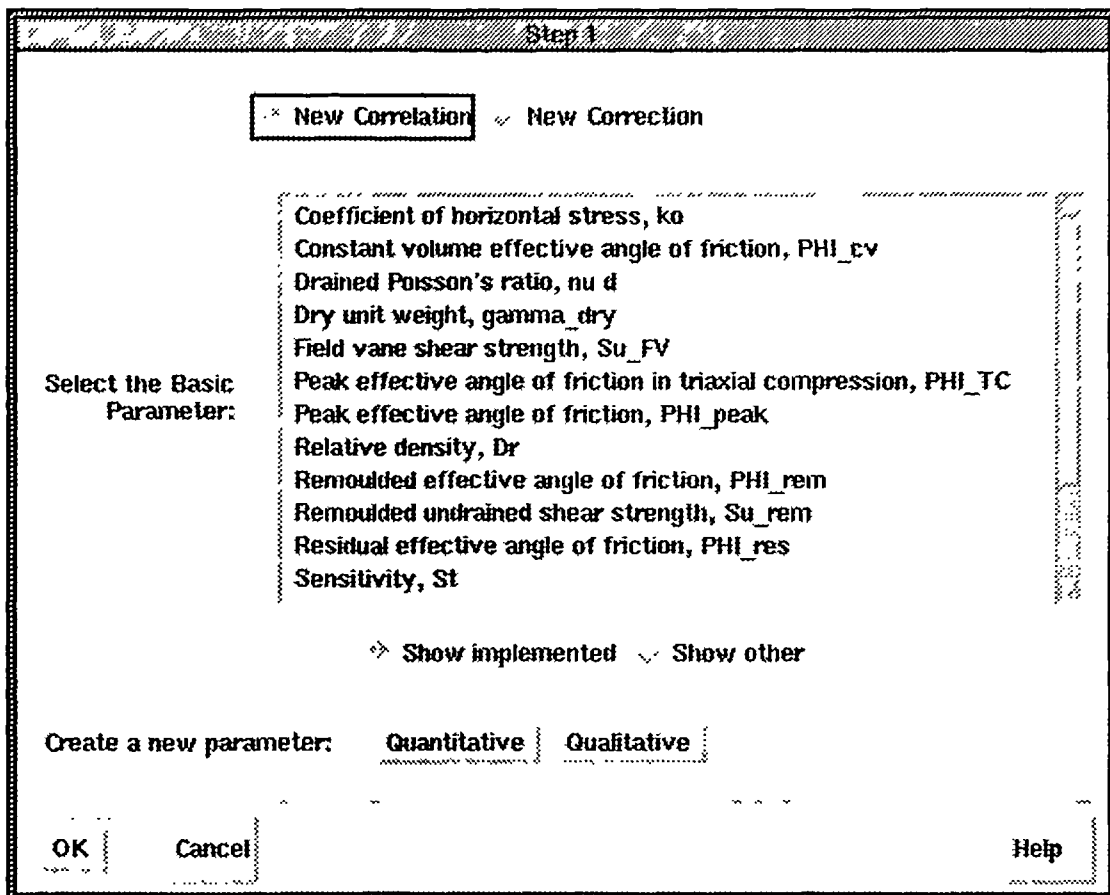


Figure 6.1. The Step1 dialog box.

This dialog box contains a radio buttons control which prompts the user to specify if the new object will be a correlation or correction. The default value of the radio buttons is set to "New Correlation", so unless the user clicks on the "New Correction" button, a new correlation object will be created.

Below the radio buttons there is a list box initially containing all the ground parameters that have been used as basic parameters in the correlations and corrections which have already been implemented. The radio buttons control below the list box, which is initially set to "Show implemented", can be used for viewing an alternative set of parameters. By clicking on the "Show other" button, the list will now display all the parameters contained in the Parameters object base (§4.4), except those listed previously.

It should be noted that by clicking on the radio buttons (which control the selection items of the parameter display list box), a function is activated which executes a non-deterministic search to produce the appropriate list of parameters. The search for the "implemented" parameters (invoked by clicking on "Show implemented") is performed by retrieving all the values of the *Parameter* slot contained in every correlation and correction (duplication of information is not allowed; so even if a parameter can be found in two or more correlations, it will produce only one selection item). The remaining parameters (search invoked by clicking on "Show other") are obtained by excluding the "implemented" parameters from all the parameters contained in the **GPar** (parameters) object base.

If the basic parameter to be used in the new correlation is contained in any of the two parameters lists, it can be specified by selection (clicking with the mouse). If a parameter is selected, a dialog box containing information about the selected parameter will appear on screen. In the case of a qualitative parameter this information is the name and the shorthand description of the parameter, while for a quantitative parameter it will also include the units and the number of decimal points (§5.3.1). The information dialog boxes for a quantitative and qualitative parameter are displayed in Figure 6.2

As can be seen (in Figure 6.2), the information dialog boxes contain a command row control with "OK" and "Cancel" buttons. If the user clicks on the "Cancel" button, the control will return back to the Step1 dialog box without any other action. On the other hand, the "OK" button will activate a function that will create a new object in either the **Correlation** application, or in the **Correction** module (depending on the value of the relevant radio buttons control). The name of the new object will be **correl_temp**, which is a temporary name. It will be shown in §6.3 that the object's permanent name is established after the implementation of the correlation's variables.

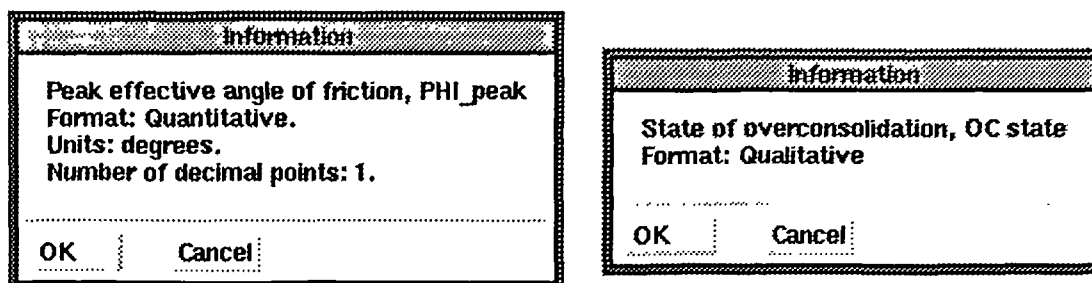


Figure 6.2. Parameter information display dialog boxes.

The newly created object automatically inherits all the slots and facets, contained in the top level **Correlation** object (§5.3.1). The displayed information about the basic parameter (contained in the information dialog box) will be passed on to the *Parameter* slot and its corresponding facets (described in §5.3.1). Furthermore, a multi-value slot, named after the shorthand description of the basic parameter, will be created in the **correl_temp** object. This slot will be used for storing the basic parameter's estimated values (§5.3.1). Finally, both the parameter information and the Step1 dialog boxes are taken off screen and the implementation procedure continues with the establishment of the correlation's variables and parameters (described in §6.3).

If the parameter is not already present in the GPar object base, then it can be specified by the "Create a new parameter:" push buttons control (Figure 6.1). In the case of a new qualitative parameter, a dialog box will appear on screen (Figure 6.3)

The dialog box is titled "New Qualitative Parameter Definition". It contains the following fields and controls:

- Full Name:** A text input field containing "Compressibility".
- Sorthand Description:** A text input field containing "comp".
- Select the parameter category:** A list box with the following items: "index properties", "stress history parameters", "strength parameters", "deformation parameters", and "compressibility parameters". The "compressibility parameters" item is highlighted.
- Buttons:** "OK" and "Cancel" buttons are located at the bottom left.

Figure 6.3. The dialog box for defining a new qualitative parameter.

The dialog box is titled "New Quantitative Parameter Definition". It contains the following fields and controls:

- Full Name:** A text input field containing "Relative density".
- Sorthand Description:** A text input field containing "Dr".
- Units:** A text input field containing "%".
- Select the number of decimal points:** A numeric input field containing "1".
- Select the parameter category:** A list box with the following items: "compressibility parameters", "flow parameters", "particle size distribution parameters", "density parameters", and "miscellaneous". The "density parameters" item is highlighted.
- Buttons:** "OK" and "Cancel" buttons are located at the bottom left.

Figure 6.4. The dialog box for defining a new quantitative parameter.

This dialog box prompts the user to specify the full name and a shorthand description of the parameter, as well as the parameter category (see §4.3).

In the case of a quantitative parameter (Figure 6.4) the user may also specify the units and number of decimal points of the parameter.

By clicking on the "OK" button, a check will be performed (to see if the name, shorthand description and parameter category have been specified, or if the parameter has already been defined). If the check produces any errors, these will be reported inside an ERROR dialog box, and the user will be asked to correct them before continuing any further.

If no errors exist, a new object will be created in the **GPar** object base, as a subclass of the specified parameter category object. The information provided by the user will be passed on to the appropriate slots and facets of the newly created parameter object. The name of the object will be the specified shorthand description of the parameter and the full name will be placed in its *name* slot (parameter objects and their slots are presented in §4.4). In the case of a quantitative parameter the *format* slot is bound to "Quantitative" and the units and the number of decimal points will be placed into the corresponding facets. Furthermore, a new correlation object (*correl_temp*) will be created and the specified information for its basic parameter will be passed on to the appropriate slot and facets.

Finally both the parameter definition and the Step1 dialog boxes are taken off screen and the implementation procedure continues with the establishment of the correlation's variables and parameters.

6.3 Variables and parameters.

After the definition of the new correlation object and its basic parameter, a new dialog box window appears on screen (Figure 6.5). This window (Step2) contains two list boxes, each supplemented with radio button controls and push button controls, used for the implementation of the quantitative and qualitative basic variables of the correlation. It also contains two push button controls for the implementation of intermediate variables and parameters and an entry box for defining the correlation's reference.

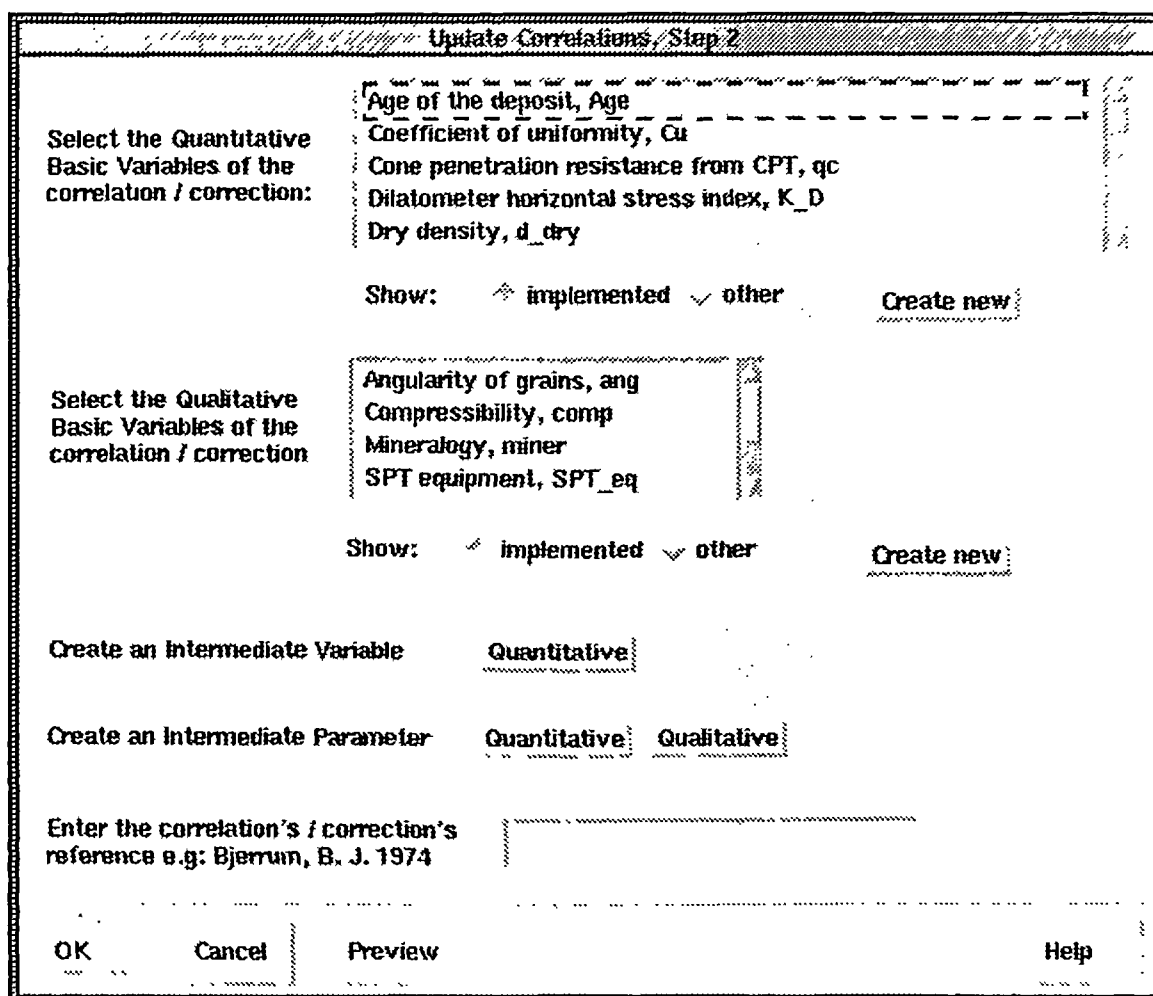


Figure 6.5. The Step2 dialog box.

Each of the list boxes initially contains all the parameters, quantitative and qualitative respectively, that have been used as basic variables in correlations and corrections which have already been implemented. The radio button controls ("Show implemented", "Show other") below each of the list boxes are used to activate functions, which control the parameters displayed in the list boxes. The radio button values are initially set to "Show implemented". Clicking on the "Show other" button, will cause all the remaining quantitative or qualitative parameters in the **GPar** object base, that have not been used as basic variables, to be displayed in the corresponding list box. The activated functions search for slots in all the correlation objects that contain a *BV_description* facet (indicating that the slot is used for representing a basic variable), whose values are bound to either "Quantitative" or "Qualitative".

If the parameter to be used as a basic variable can be found among the selection items of the list box, then it may be selected. The selection of the parameter invokes a function, which depending on whether the parameter has or has not been used as a basic variable in existing correlations, performs a different search. In the first case the function will search for a slot (through which the variable has been represented) in any of the correlations objects. When the first slot meeting the search criterion is identified, the search will stop and the information contained in the slot (and its facets) will be retrieved. In the second case the function will search for an object in the ground parameters object base, whose *name* slot contains the selected parameter name. Consequently, the necessary information will be retrieved from the slots and facets of the parameter object. The retrieved information will be displayed in a dialog box.

If the variable is of quantitative format, then the dialog box will contain a text display, displaying the name, shorthand description and the units (if any) of the variable, and two entry boxes for defining the minimum and maximum values of the

variable (if any) for which the new correlation or correction applies. These entry boxes may contain, as default values, the values of the *min_value* and *max_value* facets of the identified slot (if the parameter has already been used as a basic variable in an already implemented correlation). The default values are displayed to provide a typical range of values for the selected variable and they are not intended to be kept unaltered. They may be altered in order to meet the requirements for the new correlation or correction. An example of a dialog box for defining quantitative variables is shown in Figure 6.6.

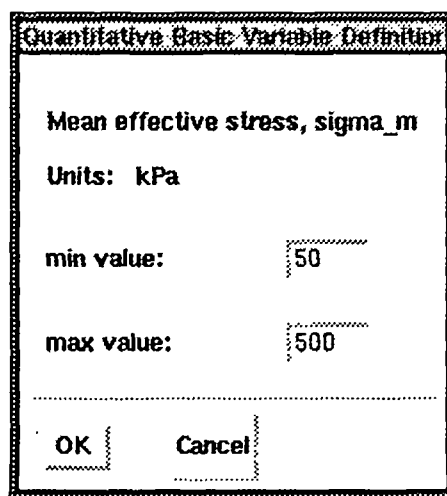


Figure 6.6. The dialog box for specifying applicability range of a quantitative variable.

If the variable is of qualitative format, then the dialog box shown in Figure 6.7 will appear on screen. This dialog box contains a text display with the name and a shorthand description of the parameter and a list box containing a set of permissible values for the parameter. The displayed set of values is retrieved from the *per_val* (permissible values) facet, attached to either the *variable* slot in a correlation object (if the parameter has already been used as a variable), or otherwise from the *format* slot of the parameter object (in the **GPar** object base).

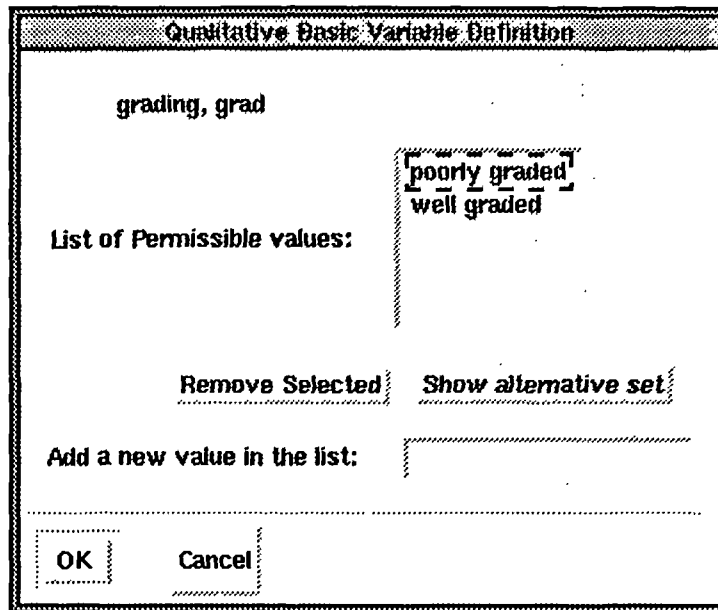


Figure 6.7 The dialog box for specifying the permissible values of a qualitative variable.

If the permissible values facet of the *format* slot of the object by which the parameter is represented in the **GPar** object base, contains more than one sets of permissible values, then all the remaining sets (except the one being displayed in the list box) will be stored inside a list in the *UserData* slot of the list box object. A push button with the title "Show alternative" (set of permissible values) will appear below the list box. This button incorporates a function for displaying (inside the list box) all the sets of permissible values for the variable. Each time this button is pressed the currently displayed set of permissible values is placed at the end of the *UserData* slot list. The first element of that list (another set of permissible values) will be taken off the list and its values will be displayed inside the list box. In this way the user can alternatively view all the sets.

The dialog box also contains an entry box incorporating a method for adding new values to the permissible values list box. The new value is written inside the entry box and by pushing the "Enter" button (on the keyboard), it appears inside the list box. Finally the dialog box also incorporates a "Remove Selected" push button for

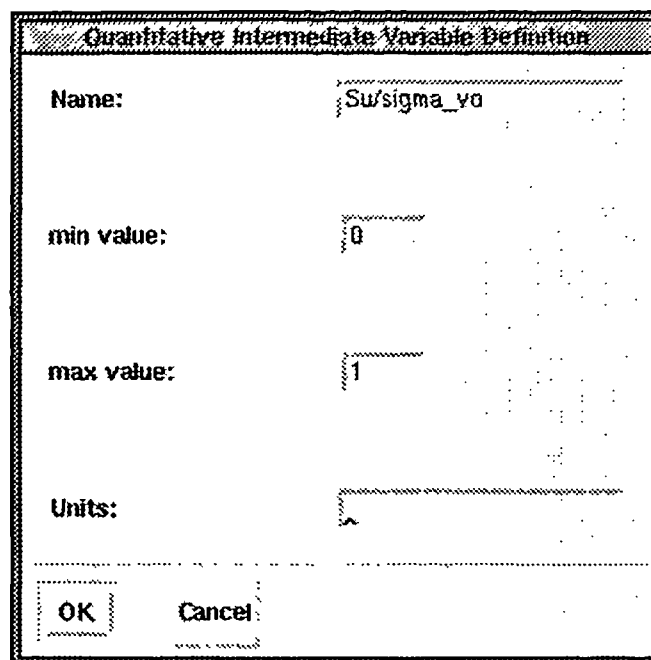
removing items from the list box (one item may be removed each time by selecting it and then pressing the "Remove Selected" button). These last two functions allow the user to redefine the permissible values of the variable in accordance with the correlation restrictions. In this case, i.e. if a new set of permissible values is defined, it will be added to those already existing in the *per_val* (permissible values) facet of the *format* slot of the parameter object (in the **GPar** object base).

The user then has the option either to update the correlation object by pushing the "OK" button, or simply dismiss the dialog box ("Cancel") and return the control of the execution back to the Step2 window (which is insensitive while the variable dialog box is on screen). The correlation object is now updated by the creation of a single value slot (the name of which is the shorthand description of the variable) and its corresponding facets (described in §5.3.1) containing the information provided by the user in the dialog box.

If the required parameter is not already contained in the **GPar** object base then a new parameter must be defined using the "Create new" push buttons (below the list boxes in the Step2 dialog box, Figure 6.5). The definition of a new parameter that will also serve the new correlation as a basic variable, is a two step procedure. Initially a dialog box appears on screen which is used for creating the parameter object in the Parameters object base. The "New Qualitative Variable" dialog box is the same as that presented in Figure 6.3 and the "New Quantitative Variable" dialog box is the same as that presented in Figure 6.4. The user should supply the dialog box with the required information and then press the "OK" button. The user supplied information will be checked for inconsistencies. If no errors exist, a new parameter object will be created in the **GPar** object base. Otherwise, an ERROR dialog box will appear on screen, prompting the user to correct all errors before proceeding any further.

Then a second dialog box will appear on screen, either for specifying the minimum and maximum values of the variable (if the new parameter is of quantitative format, see Figure 6.6), or the permissible values (if the new parameter is of qualitative format, see Figure 6.7). It should be noted that the entry boxes (or list box) in this dialog box will not contain any default values, since the parameter is not predefined. After the applicability range (or the set of permissible values) is specified, the user must press the "OK" button to update the correlation object with the new variable.

Intermediate quantitative variables can also be added to the correlation object, using the "Create an Intermediate Variable" push button control, in the Step2 dialog box (Figure 6.5). If the user clicks on the "Quantitative" button, the "Quantitative Intermediate Variable Definition" dialog box appears on screen (Figure 6.8).



The image shows a dialog box titled "Quantitative Intermediate Variable Definition". It contains the following fields and controls:

- Name:** A text box containing "Su/sigma_y0".
- min value:** A text box containing "0".
- max value:** A text box containing "1".
- Units:** An empty text box.
- Buttons:** "OK" and "Cancel" buttons at the bottom left.

Figure 6.8. The dialog box for implementing quantitative intermediate variables.

This dialog box is used to define the name, units and minimum and maximum limits of applicability for the intermediate variable. The shorthand description of intermediate variables (which is used as the name of the slots that represents the variable) is created by the system and takes the form IV_number. The number is

produced by incrementing the number of intermediate variables in the correlation object by one. For example the first intermediate variable slot is named IV_1; the second IV_2 etc. There are two reasons for the automatic definition of the shorthand description of intermediate variables. The first is that intermediate variables are "dummy" parameters and they are not included in the Parameters object base. Therefore, there is no need to define both a shorthand description and a full name. Furthermore their shorthand description is never presented to the user at any stage. The second reason is to avoid slot names that may contain illegal characters, such as "/". For example a possible shorthand description for the intermediate variable undrained shear strength over effective overburden pressure is: S_u/σ_{vo} , which contains the illegal character "/".

After the requested information about the intermediate variables is placed into the appropriate entry boxes, the user should click on the "OK" button to update the correlation object. The input information is checked for inconsistencies, or missing information. If the check produces any errors, these are reported to the user. The user is prompted to correct all errors before proceeding. If no errors exist the correlation object is updated and a new dialog box appears on screen. This dialog box (Figure 6.11), is used for implementing the function for calculating the values of the intermediate variable. The implementation of estimation procedures for intermediate variables is discussed later in §6.4.

Finally, intermediate parameters can also be added to the correlation object, by using the "Create an Intermediate Parameter" push buttons controls (Step2 dialog box; Figure 6.5). If the "Quantitative" button is pushed, a dialog box appears on screen (Figure 6.9).

This dialog box contains the same controls (a list box, radio buttons and push buttons) that are used for the definition of basic variables. Intermediate parameters

can be implemented in the system by selection. It should be noted that there is no need to define applicability ranges (for quantitative parameters), or sets of permissible values (for qualitative parameters). If a parameter, displayed in the list box, is selected, then an information dialog box will appear on screen (similar to one of the information dialog boxes for basic parameters, shown in Figure 6.2). If the "OK" button, in the information dialog box is pressed, the selected intermediate parameter will be added to the correlation object (the representation of intermediate parameters is covered in §5.3.1). If the parameter is not contained in the Parameters object base, then it must be created by using the "Create new" push button control. The procedure for creating a new intermediate parameter is the same as the procedure for defining new basic parameters (§6.2).

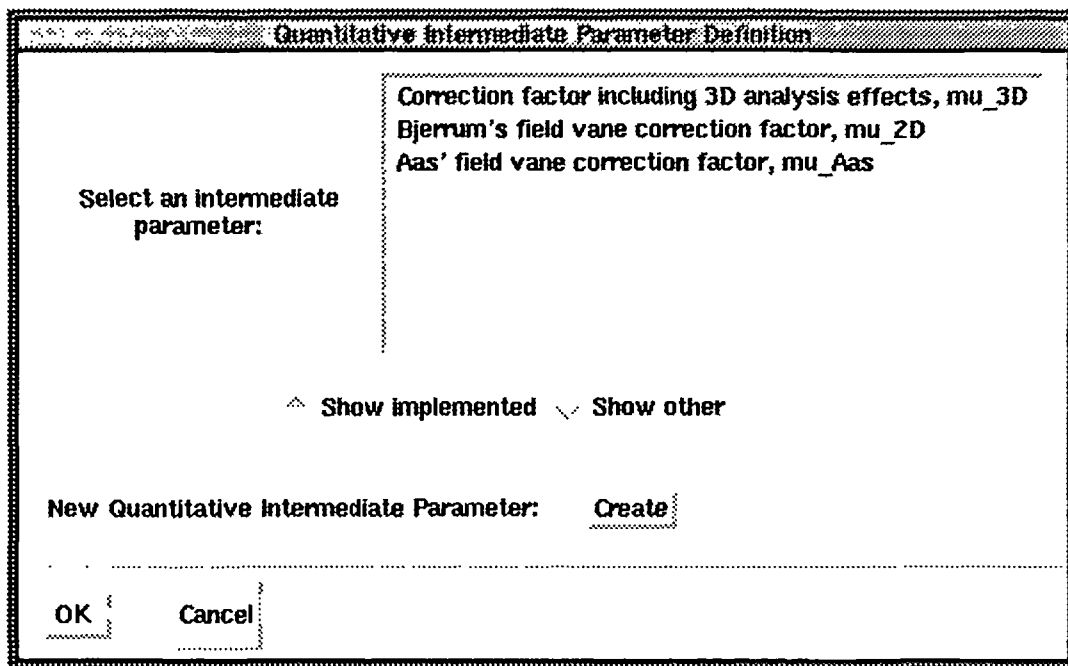


Figure 6.9. The dialog box for implementing quantitative intermediate parameters.

It should be mentioned that the implementation of a parameter, either as a variable or a parameter of the correlation, activates a function which excludes this parameter from all the display lists of variables and parameters. This function prevents the same parameter being implemented twice in the same correlation object.

When the implementation of variables and parameters of the correlation is over, or at any stage during their implementation the user can examine them by using the "Show Preview" option (in the command row of the Step2 dialog box, shown in Figure 6.5). By pushing this button a function is invoked which searches the correlation object to identify all the implemented basic, intermediate variables and parameters in the correlation object. These are displayed in the dialog box shown in Figure 6.10.

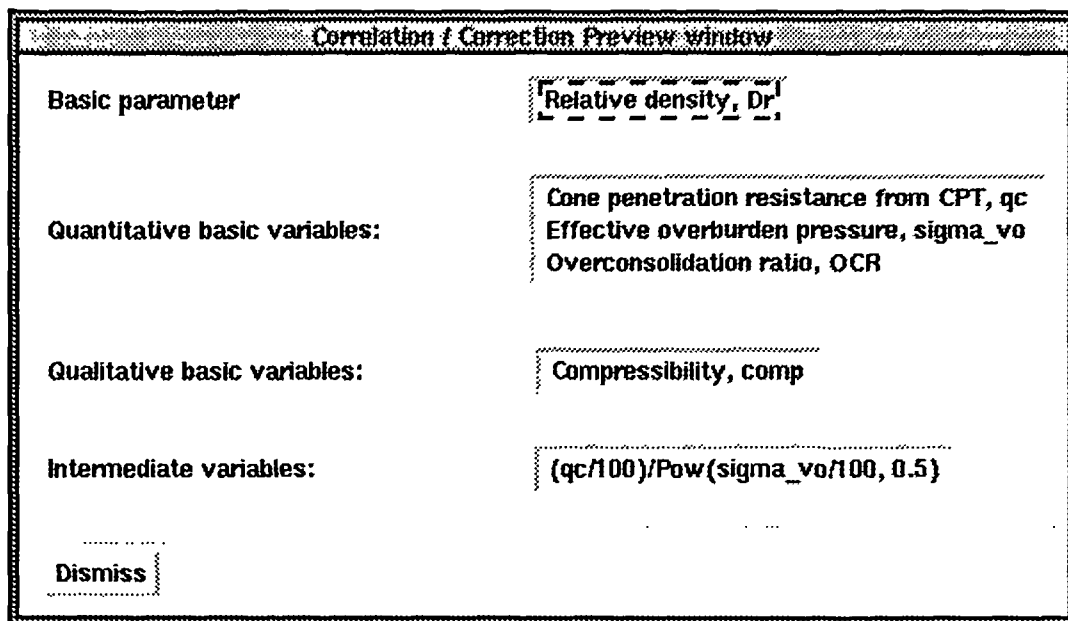


Figure 6.10. An example for the Show Preview dialog box.

The correlation's variables and parameters are placed into list boxes based on whether they are basic or intermediate, quantitative or qualitative. From there they can be examined by selection (clicking with the mouse).

The selection of a variable invokes a function that puts a dialog box on screen. For quantitative variables this dialog box is the same as that shown in Figure 6.6, while for qualitative same as that in Figure 6.7. In the first case the entry boxes display, as default values, the specified min. and max. values for the variable, and in the second case the list box displays the specified set of permissible values. The user

can therefore modify the applicability range, or set of permissible values of the variable. By pressing "OK" the correlation object is updated with the modified information. The command row of these dialog boxes also contains a "Remove variable" option, which can be used to remove the variable from the correlation object (by deleting the slot which represents the variable). The selection of a parameter causes a parameter information display dialog box to appear on screen (similar to one of those shown in Figure 6.2). The command row of this dialog box is also supplemented with a "Remove Parameter" option, for removing the parameter from the correlation object.

After all the correlation's variables and parameters have been implemented, the user should type the correlation's reference inside the reference entry box (in the Step2 dialog box, Figure 6.5). It is recommended, without being restrictive, that this should be done in an *authors-date order (as in the example in the entry box title)*, thus preserving homogeneity in the representation.

Finally the user should press the "OK" button in the command row of the Step2 dialog box. Pushing of the "OK" button will invoke a function that performs a few checks in the correlation object and updates its name and some of its slots and their facets. The first action of this function is to check if at least one quantitative or qualitative basic variable has been implemented. If no basic variable is found, an error dialog box will appear on screen prompting the user to define at least one basic variable before continuing. Secondly, it checks the reference entry box to ensure that a reference has been supplied. If the reference entry box is empty, the ERROR dialog box (mentioned above) will appear on screen, asking the user to supply a reference for the correlation.

If the above checks are successful, then the function counts all the correlations and corrections, which have the same basic parameter as the correlation under

examination. The maximum number is then incremented by one (henceforth the correlation's number) and is appended as a string to the shorthand description of the basic parameter (also expressed as a string). The resulting string is converted into a symbol (by removing the quotes around it) and placed in the *winame* slot of the correlation object. For example if five correlations and/or corrections exist for the estimation of undrained shear strength ("Su"), then the value of the *winame* slot of a sixth correlation will be Su6. This name will be used as the dialog box object name (as mentioned in §5.3.1 and §5.3.3), by which the correlation will be represented in the interface session. This name, which is obviously unique, is used as the initial part of all the dialog box control names (these controls are used to represent variables and parameters, §5.3.3). In the case of variables, the values of their *ebname* and *lbnam*e facets respectively (§5.3.1), are made up from the *winame* slot value and the shorthand description of the variable. For example the value of the *ebname* facet of the liquidity index (LI) variable will be Su6LI. For the overconsolidation ratio (OCR) quantitative intermediate parameter the values of the *tdname*, *tdmax*, *tdmin* and *tdmean* facets will be Su6OCR, Su6OCRmax, Su6OCRmin and Su6OCRmean respectively. In this way all the names for the dialog box and its controls are unique. This is an essential requirement since object names in ProKappa must be unique.

The function also creates a string with the parameter and the basic variables of the correlation (their full names) followed by the correlation's reference and the correlation's number, and places it in the *Parameters_needed* slot (§5.3.1). A similar string (containing shorthand descriptions for the basic variables) is placed in the *wintitle* slot. The reference is stored in the *authors* facet (§5.3.1). The last action of this function is to take off screen the Step2 dialog box. The updating procedure will continue with the implementation of the correlation's estimation procedure (described in §6.4).

If the user clicks on the "Cancel" button in the Step2 dialog box, the implementation procedure will return to the previous stage. The Step2 dialog box will be taken off screen and the Step1 dialog box will appear on screen. Furthermore the correlation object will be deleted as well as all its slots and their corresponding facets.

6.4 Implementation of estimation procedures.

The implementation of intermediate variables is a two stage procedure. The first stage was presented in §6.3. The second stage is the implementation of the estimation procedure for the variable. Estimation procedures for intermediate variables can be implemented in the system by means of the dialog box displayed in Figure 6.11.

Define the formula for the Intermediate Variable

Select a Quantitative Basic Variable: Field vane shear strength, Su_FV
Effective overburden pressure, sigma_vo

Select the Intermediate Variable: Su_FV/sigma_vo

Always finish each statement with a semicolon (;) !!

Enter the formula:

Select a line to examine: ?IV_1 = ?Su_FV/?sigma_vo;

Figure 6.11 Estimation procedure definition for intermediate variables.

This dialog box contains two list boxes. The first displays the names of all the quantitative basic variables of the correlation. Therefore, it is recommended that the basic variables should always be implemented before intermediate variables. The

second contains the intermediate variable, which is currently being implemented. The dialog box also includes an entry box where the estimation procedure code is written and a list box for displaying this code.

The code of the estimation procedure should be written in ProTalk syntax. The dialog box incorporates features for the fast and syntactically correct writing of simple ProTalk statements. These are presented below:

- The two list boxes, which contain the basic variables and the intermediate variable. Whenever the user wants to write a reference to any of the above variables, he/she should select the desired one (click on with the mouse), and this will be presented inside the entry box in a ProTalk variable format. For example, assume that the user wants to write the statement: $IV_1 = Su_{FV}/\sigma_{vo}$; he/she has to first select the intermediate variable from the list box. A ?IV_1 expression appears inside the entry box (which is a reference to the intermediate variable in a ProTalk variable format). Then he/she should enter in the entry box the "=" (set equal to) operator. The second step is to select the field vane undrained shear strength variable from the list box. A ?Su_FV expression appears in the entry box. Then he/she should add the "/" (divide) operator, select the effective overburden pressure variable from the list box and finally type a ";" (semicolon) at the end of the statement. The ProTalk expression inside the entry box is now: "?IV_1 = ?Su_FV/?sigma_vo;"
- A text display (below the list boxes, see Figure 6.11), reminding the user to finish each statement with a semicolon ";"
- A "View Functions" option in the command row of the dialog box (see Figure 6.11). Whenever pressed, a dialog box appears on screen (Figure 6.12), displaying most of the mathematical functions that can be used within ProTalk.

The user may select the desired one (click on with the mouse), press the "Export" button and this will appear inside the entry box.

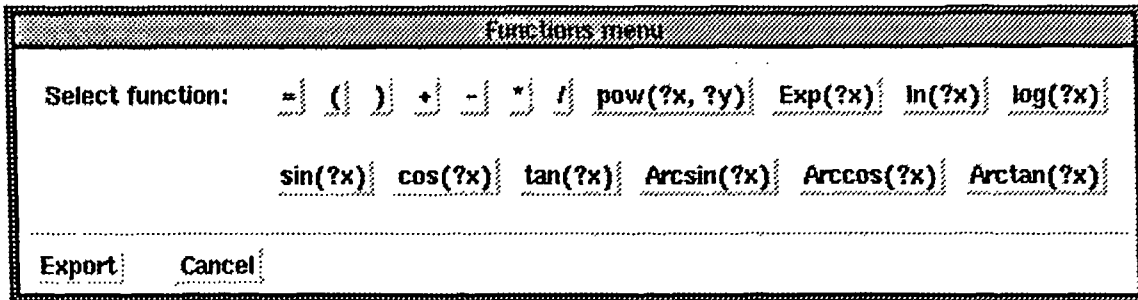


Figure 6.12. The dialog box displaying mathematical functions in ProTalk.

After the end of a statement the user should press the Enter button (on the keyboard) to add it into the display list box. The display list box incorporates a function that allows the user to examine the displayed ProTalk statements. The user can select any statement to examine and this will appear inside the entry box, where it can be edited. After updating the statement the Enter button should be pressed and the updated statement will be redisplayed, in its original position, inside the display list box. When the code is complete the user should press the "Update" button in the command row of the dialog box (Figure 6.11).

The invoked function appends all the items of the display list box in a string. The system will then generate some more statements and append those to that string. In its final form the updated string contains a complete ProTalk function that can be used for the calculation of the intermediate variable's values. The user supplied statements are placed inside a *for-do* loop so that the function can generate a list of values for the intermediate variable (as mentioned in §5.3.1). With reference to the previous example the complete function will be as follows:

```
{   bound inputs;
    ?list = `();
```

```

        /* ?list is defined as the empty list */
for ?i from 0 to ListLength(?self.Su_FV)-1;
        /* ?self is a variable bound to the correlation object */
do {  ?Su_FV = ListNth(?self.Su_FV, ?i);
        ?sigma_vo = ListNth(?self.sigma_vo, ?i);
        ?IV_1 = ?Su_FV/?sigma_vo;
        ?list = AppendLists(?list, `(?IV_1));  }
?self.IV_1 = ?list;
}

```

Finally a slot formula is created, which is attached to the *Parameters_needed* slot (see §5.3.1), and the string is passed to it as its function. The system will then try to compile this function. If the function compiles, an information dialog box appears on screen, informing that the installation of the slot formula function was successful. Otherwise a message informing of the compilation failure is displayed and the user is asked to correct all errors, before trying to recompile.

The estimation procedure of the correlation itself is implemented by means of the dialog box (named *estwin*) displayed in Figure 6.13. This dialog box is used for implementing or updating the estimation procedures of correlations. The structure of this dialog box is the same as that of the dialog box for the implementation of the estimation procedures of intermediate variables (Figure 6.11). The main difference is that the display list box in the *estwin* dialog box contains all the variables and parameters of the correlation (rather than only the quantitative basic variables).

The selection of a qualitative basic variable from the display list box will cause:

- a. the appearance of the variable, in a ProTalk variable format, inside the entry box

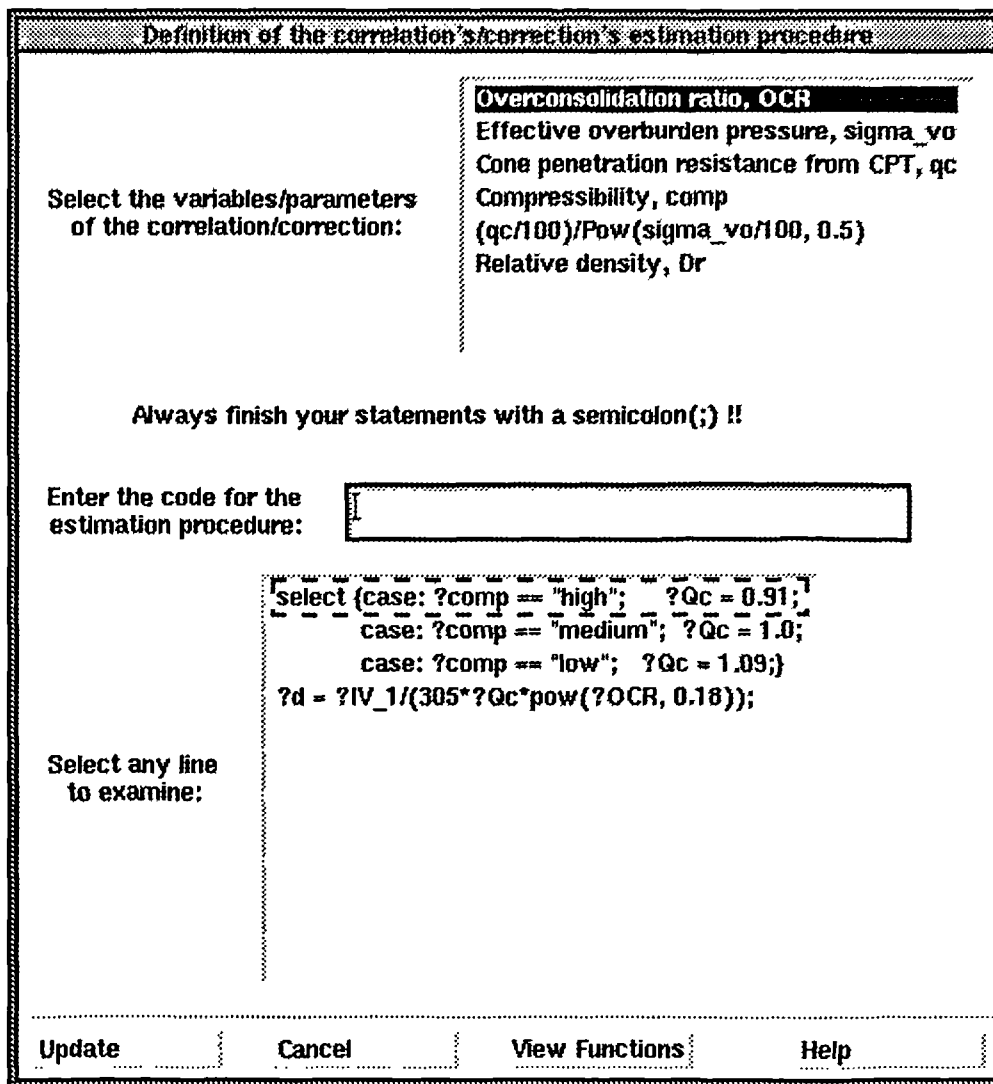


Figure 6.13. The estwin dialog box.

- b. a dialog box to appear on screen, containing a list box with all the permissible values of the variable. Selection of any of these values results in their appearance inside the entry box. This dialog box is useful as a reminder of the set of permissible values and as a means of minimising manual entry, (reducing the risk of typing errors). An example of the dialog box, displaying the permissible values for the compressibility variable of the same correlation is presented in Figure 6.14.

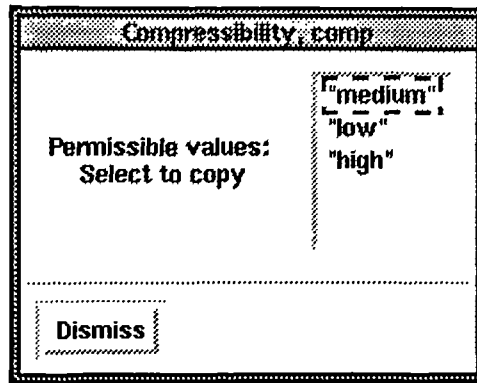


Figure 6.14 The dialog box displaying the permissible values of compressibility.

All other functions and procedures are similar to those described above (for estimation procedures of intermediate variables). If the slot formula function (now attached on the *Parameter* slot) compiles, the implementation of the new correlation will continue with the definition of applicability.

6.5 Applicability definition.

The definition of the applicability for correlations is a four stage procedure. The first is relevant to the definition of an applicable ground type. The second to the selection of the parameters that impose applicability restrictions (for the selected ground type). The third to the definition of the restrictions for each of the parameters. Finally, the assignment of an applicability score for the specified applicability set (the selected ground type and its associated parameter restrictions) is performed in the fourth stage. This procedure must be repeated for each applicability set.

The dialog box used for the selection of the applicable ground types, is shown in Figure 6.15.

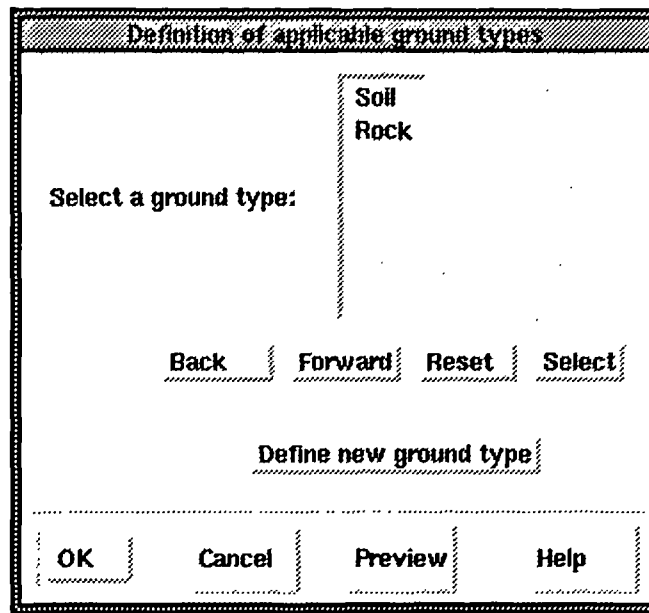


Figure 6.15 The dialog box for defining applicable ground types.

The list box and the first three push buttons ("Forward", "Back", "Reset") are used for searching the ground object base (see §4.4). When the desired ground type is found and displayed in the list box, it must be selected (click on with the mouse) and then the "Select" button should be pressed. If the desired ground type cannot be found, it should be created by using the "Define new ground type" push button (defining new ground types is presented in §4.5). After the new ground type is created it should be displayed in the list box and from there it should be selected.

Once a ground type has been selected a new dialog box will appear on screen, which inquires of the user if the selected ground type will be supplemented with parameter related restrictions. If the answer is negative, the procedure will continue with the assignment of an applicability score to the selected ground type. Otherwise a dialog box for selecting parameters (in order to define parameter associated restrictions) will appear on screen. This dialog box is displayed in Figure 6.16.

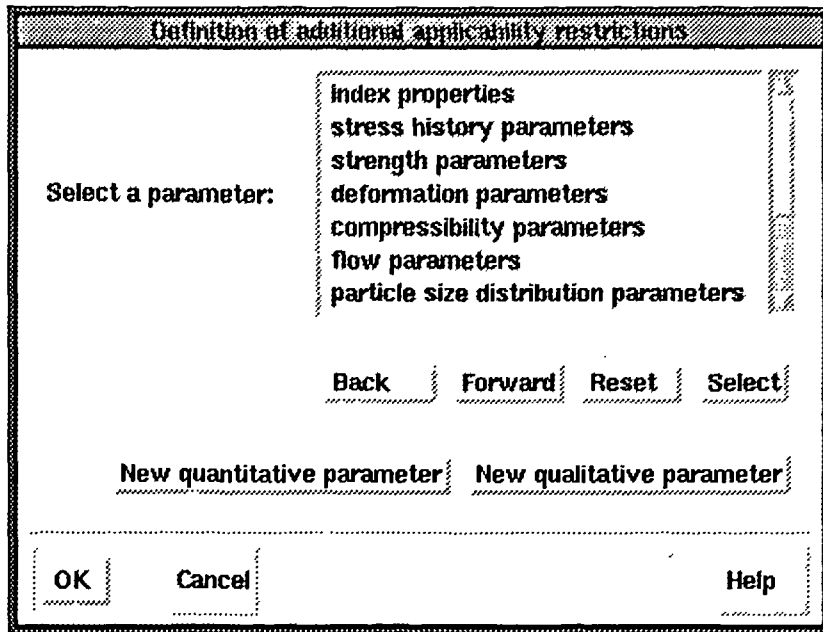


Figure 6.16. The dialog box for selecting parameters for applicability restrictions.

The list box (of the dialog box in Figure 6.16) initially contains all the parameter categories (obtained from the Parameters object base). The user should select the desired parameter category and then press the "Forward" button. The parameters classified under the selected parameter category will be displayed in the list box. If the desired parameter is found among the displayed parameters, it should be selected (click on with the mouse and then press the "Select" button). If the desired parameter is not contained in the Parameters object base, it should be created. New parameters can be created by means of the "New quantitative parameter" and "New qualitative parameter" push buttons. The procedure for creating new quantitative or qualitative parameters is presented in §6.2 (Figures 6.3 and 6.4). If a parameter is selected or created, then the applicability restrictions are defined by means of the dialog box shown in Figure 6.17 (for a quantitative parameter), or the dialog box shown in Figure 6.18 (for a qualitative parameter).

The dialog box in Figure 6.17 displays the name and units of the selected (or created) parameter. The user must specify the min. and max. values defining the applicability range for the parameter by typing into the appropriate entry boxes.

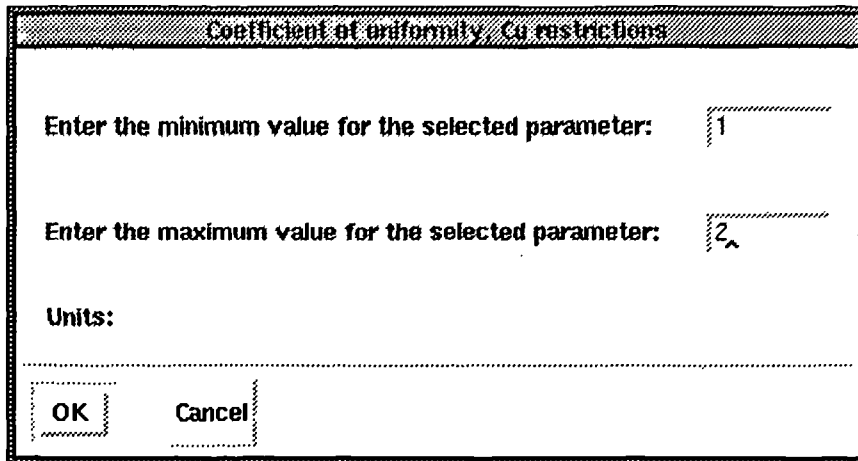


Figure 6.17. The dialog box for defining applicability restrictions for a quantitative parameter.

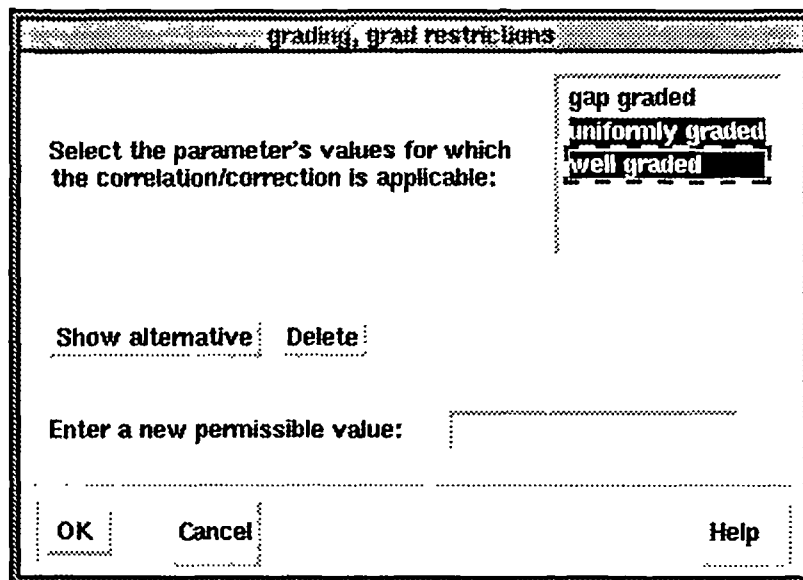


Figure 6.18. The dialog box for defining applicability restrictions for a qualitative parameter.

If a qualitative parameter is selected, the invoked function will retrieve all the sets of permissible values for the parameter (stored in the *per_val* facet attached to the *format* facet of the parameter's object). The set that is retrieved first is displayed in the list box (Figure 6.18). If more than one set exists, then all the remaining sets are stored in the *UserData* slot of the push buttons control object (below the list box). The "Show alternative" button is used to display alternatively the sets of permissible

values (see §6.3). The user can redefine any set using the "Delete" button (removes a selected item from the list box) and the entry box for entering new permissible values. When the desired set of permissible values is displayed, the user may select the appropriate ones and then press the "OK" button.

A temporary facet will be created, named after the shorthand description of the selected parameter. The value of this facet is a list, either containing the min. and max. values defining the applicability range (for a quantitative parameter), or the selected permissible values (for a qualitative parameter). The dialog box for defining the parameter restrictions will be taken off screen and the control of the execution is returned to the dialog box for selecting parameters (Figure 6.16). The user may select another parameter and repeat the procedure.

When all the parameter associated restrictions for the selected ground type have been defined, the user should click on the "OK" button (in the command row of the Figure 6.16 dialog box). A new dialog box will appear on screen which is used for assigning an applicability score to the specified applicability set.

This dialog box contains a list box with the options "high", "medium" and "low". The user will select the applicability score and then he/she should press the "OK" button to update the correlation object. The information contained in the applicability set will be passed to the appropriate applicability slot and facets (the representation of applicability is discussed in §5.3.1). Finally both the dialog boxes for selecting parameters and applicability scores are taken off screen and control of the execution is returned to the dialog box for defining applicable ground types (Figure 6.15).

The user may define a new applicability set following the course of action described above. He/she may also preview the already defined applicability sets by clicking

on the "Preview" button (in the command row of the Figure 6.15 dialog box). The "Applicability Preview" dialog box appears on screen (see Figure 6.19), displaying the settings for the applicability set defined first.

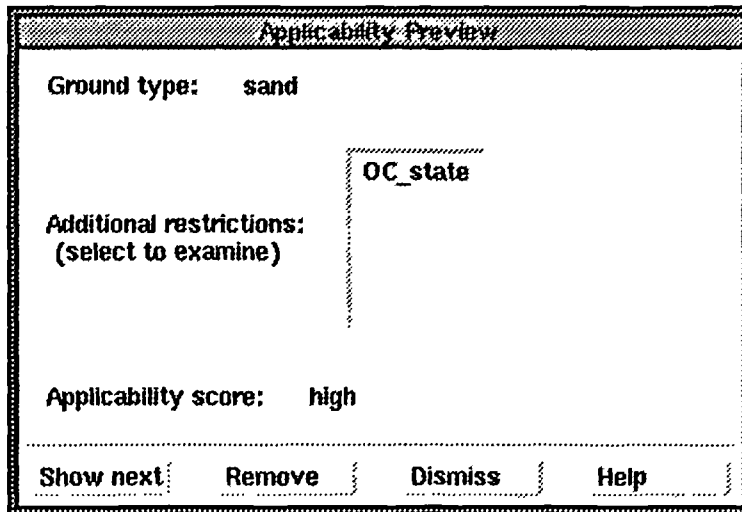


Figure 6.19. The "Preview Applicability definition" dialog box

This dialog box can display one applicability set at the time. In order to view another set the user must click on the "Show next" button (in the command row of the "Applicability Preview" dialog box). In this way the user can exhaustively examine all the applicability sets.

The user may also examine any of the parameters (displayed in the list box). Selection of the parameter will cause one of the dialog boxes, shown in Figures 6.17 and 6.18 respectively, to appear on screen. These dialog boxes will contain the defined parameter restrictions as default values. The user is allowed to redefine the applicability range, or the set of the selected permissible values. The "OK" button should then be pressed for the correlation object to be updated with the altered information. Finally, the "Applicability Preview" dialog box contains a "Remove" push button for removing the currently displayed applicability set from the applicability definition of the correlation object.

When all the applicability sets are defined, the user may click on the "OK" button in the dialog box for selecting applicable ground types. The invoked function checks the applicability definition of the correlation object. If no values exist in any of the applicability slots, an information dialog box appears on screen, warning the user that the applicability of the correlation must be defined before proceeding. Otherwise, the dialog box is taken off screen and the updating procedure continues with the definition of reliability.

6.6 Reliability definition.

The reliability of the correlation is defined by means of the dialog box displayed in Figure 6.20. This dialog box contains a list box for defining the reliability score of the correlation. The user must choose one of the displayed reliability scores. Furthermore, any additional information in terms of coefficient of fit (a percentage), standard deviation (a positive number) and number of data points (used for the correlation), may also be specified by typing into the corresponding entry boxes.

The dialog box is titled "Definition of Reliability". It contains the following elements:

- Select reliability score:** A list box with three options: "low", "medium" (selected), and "high".
- Coefficient of fit (%):** A text entry box containing the value "77.6".
- Standard deviation:** A text entry box containing the value "2.8".
- Number of points:** A text entry box containing the value "544".
- Buttons:** "OK" and "Cancel" buttons at the bottom left.

Figure 6.20. The "Reliability definition" dialog box.

6.7 Comments.

The last stage of the procedure for implementing new correlations is the implementation of comments. This is done with the dialog box shown in Figure 6.21.

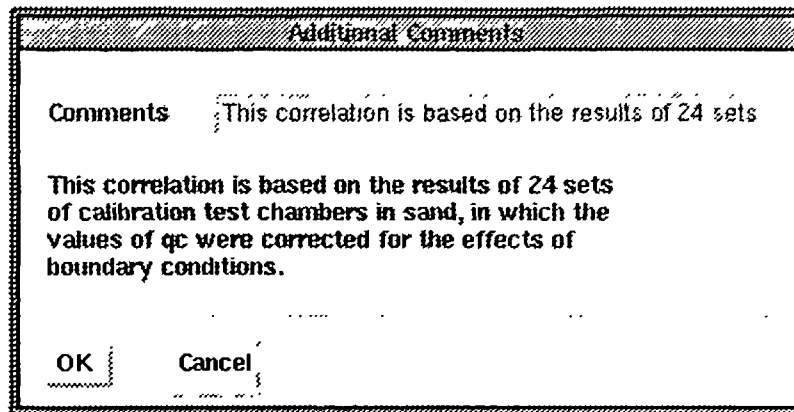


Figure 6.21. The dialog box for implementing comments.

The "Comments" dialog box contains an entry box where the user types the correlation's comments. This entry box incorporates a text manipulating function, which breaks down the text to lines of about 40 characters. The text is then displayed in a text display (below the entry box). Finally, if the "OK" button is pressed, the displayed text will be placed in the "Comments" slot of the correlation object.

6.8 Updating correlations.

The Update module is also used for updating correlations that have already been implemented in the system. The updating procedure initiates by selecting the "Update implemented correlations/corrections" option in the Figure 5.7 dialog box. This action results in the appearance of the "search" dialog box (Figure 5.2), used

for identifying the correlation to update.. Performing searches in the correlation and correction object bases is covered in §5.3.3.

The actual updating of the correlation commences with the activation of a function that renames the correlation object from its original name to `correl_temp`. The Step2 dialog box (Figure 6.5) then appears on screen. The author-date reference is also retrieved from the *author* facet of the *winame* slot of the correlation object. This is displayed in the appropriate entry box (of the Step2 dialog box) as a default value.

The user may click on the "Preview" button to view the parameters and variables of the correlation. As mentioned in §6.3 any of these parameters can be examined by selection. Therefore, applicability ranges, or sets of permissible values of basic variables may be redefined. Furthermore, intermediate parameters and variables may be removed and new ones may be implemented.

If the user clicks on the "OK" button, the correlation object will be renamed (and all the other actions described in §6.3 will also take place) and the "estwin" dialog box (Figure 6.13) will appear on screen. The correlation's estimation procedure will be retrieved from the *string* slot of the slot formula object (§6.4) and will be displayed inside the list box. From there any statement can be examined, updated or deleted and new statements may also be included in the code.

In the next stage the user may examine, alter, or reimplement the applicability sets for the correlation object. The updating procedures concludes with the reliability and comments stages. The corresponding dialog boxes will display the already implemented information, contained in the appropriate slots and facets of the correlation object. This information may be examined and altered if required by the user.

6.9 Overview of the Update module.

The implementation of new correlations in the system as well as the updating of already implemented correlations is a staged procedure. Each of these stages corresponds to one of the five integral parts of a correlation's definition, as these were identified in §5.2.

The only diversion from this methodology is noted in the implementation of the variables and parameters of a new correlation, which takes place in two stages (specifically the basic parameter definition is the first stage and the implementation of the variables and intermediate parameters is a second separate stage). The reason for the introduction of two stages in the implementation of a correlation's variables and parameters is to ensure that the basic parameter cannot be altered during an updating of this correlation. The need for this can in turn be explained by considering that the updating of a correlation may be relevant only to one or more of the following cases: the inclusion of new variables that affect the estimation of the correlation's parameter; alterations to the estimation procedure as a result of either the inclusion of new knowledge or reconsideration of the variables' effect; modifications to the applicability, reliability and comments definitions as a result of new empirical knowledge. On the other hand modification of the basic parameter is meaningless, because this will result in a fundamentally different correlation, which should rather be implemented as a new correlation. Therefore, the implementation of the basic parameter in a separate stage and the omission of this stage during the correlation's updating procedure was thought to be the best way to preserve the correlation's identity.

Furthermore the necessity for the first stage in the implementation of new correlations may also be demonstrated by the fact that it incorporates the selection of whether the new object will be a correlation or a correction, which is also

irrelevant to the updating procedures. Consequently, the development of an initial stage to handle these two tasks (basic parameter implementation, choice between correlation and correction) during the implementation of new correlations and its omission during the updating procedures, allows the rest of the interface to be used for both purposes without the need for modifications.

The implementation of new correlations and the updating of already implemented ones are linear reversible procedures. Each stage of these procedures is based on a central or execution control dialog box, which either contains the appropriate controls for performing all the necessary actions, or it can call other dialog boxes to perform these actions. The central dialog boxes for each stage are: the Step1 dialog box (Figure 6.1; only applicable to the implementation of new correlations); the Step2 dialog box (Figure 6.5); the estwin dialog box (Figure 6.13); the dialog box for defining applicable ground types (Figure 6.15); The Reliability definition dialog box (Figure 6.20); and the Comments dialog box (Figure 6.21).

Each of these dialog boxes contains a "Cancel" button in their command row control, which can be used for reversing the flow of the procedure. In this way, the user is allowed to move backwards one stage, each time the button is pressed, in order to correct possible inconsistencies or entry errors.

Finally it should be noted that similar controls in different dialog boxes perform similar actions. For example an "OK" or "Update" button will always result in the execution of the requested action and will pass the control of the execution to the next stage or sub-stage of the procedure. It is believed that the consistency in the implementation of the interface will help future users to become quickly familiar with the system.

CHAPTER 7

Discussion - Future development.

7.1 Discussion.

The evaluation of ground properties is one of the most important, yet difficult problems in geotechnical engineering. The variety in the material and geological conditions, in conjunction with the extended range of geotechnical problems, has led to the development of a considerable number of methodologies for ground properties evaluation (§1.1).

The work described here aims to provide a framework for storing and using correlations and "typical" values for the estimation of ground properties. It is thought that this framework, which should be used in conjunction with the other methodologies, will provide a geotechnical engineer with a decision-support tool in the property evaluation problem.

The first stage in the development of such a tool was the identification of the requirements that should be met by it. These are outlined below:

- The ability of storing empirical knowledge (correlations, "typical" values) and to use this knowledge to infer estimations for ground properties.
- The ability to incorporate symbolic processing (e.g. so that the developed system will be able to deal with qualitative parameters, applicability of correlations etc.)

- The ability to update the existing knowledge as well as to add new knowledge, so that it will maintain its usefulness in the future.
- To incorporate some expression for the quality of the inferred information (uncertainty assessment).

Knowledge-based system technology can be applied to the property evaluation problem, as it provides a medium that can accommodate the representation and use of empirical knowledge. Furthermore, it can also incorporate symbolic processing, the ability to update the system's knowledge and ways to represent uncertainty (§2.1, §2.2).

The second stage in the development of a KBS for the estimation of ground properties was the collection of correlations and "typical" values of properties.

It should be noted that during the knowledge elicitation stage, the assessment of the quality of the inferred information was not always an easy task. The main reason for this was the lack of relevant information and sometimes the existence of conflicting information. Problems of this nature were handled by the adoption of a conservative approach for uncertainty assessment.

In the case of correlations, if inadequate information was available, low values of reliability were adopted and when conflicting information existed the pessimistic approach was preferred (high values of uncertainty). The adoption of this strategy in the reliability evaluation is intended to make the user of the system aware of the variation associated with the estimation of ground properties from empirical procedures. Furthermore, it was thought that it is better to provide larger ranges of variation for the value of a property, thus increasing the possibility that its actual value will fall within that range, rather than doing the opposite. The user of the system must then consider any additional information about the correlation provided

by the system and use his/her engineering judgement for selecting a fixed value for the property in question. This is in accordance with the intention that the developed system should function as a decision-support tool rather than taking decisions for the user.

Another problem that was encountered during the acquisition of correlations was the lack of a mathematical expression for the incorporated relation. Unfortunately, a significant number of correlations is expressed through graphs that contain a single line function describing the relation between the variables and the estimated parameter. Furthermore, the omission of the data points that were used for the establishment of the relations has also an effect on the assessment of the associated uncertainty and the reliability of the correlation.

When an adequate amount of correlations and "typical" values of ground properties had been collected, this was subsequently analysed in order to produce generic forms of representation. The result of this analysis was the identification of the components that are needed to adequately represent all the knowledge associated with a correlation and the "typical" values of a property respectively (§5.2, §4.5). It also revealed the need for acquiring additional knowledge which is essential for their representation. This was knowledge relevant to the classification of the ground and its properties (§3.2, §4.1, §4.2, §4.3).

The representation forms for "typical" values of ground properties (described in §4.5) and for correlations (described in §5.2) were developed in accordance with the requirements that were identified in the initial stages of the system's development.

Specifically, they both are structured representation forms, which can be used for implementing a large number of correlations and "typical" values in a consistent way, rather than individual pieces of code having to be written to handle each

correlation or set of "typical" values. Furthermore, they incorporate all the different knowledge types which are necessary for an adequate and complete representation of the domain knowledge.

Moreover, they also incorporate expressions for the quality of the inferred information. In the "typical" values representation the associated uncertainty is expressed as a range of variation around the "typical" value. In the correlation representation, uncertainty is expressed both quantitatively: as a range of variation around the estimated value, from the standard deviation, the coefficient of fit and the number of data points used for the creation of the correlation; and qualitatively: from the reliability score (§5.2.4).

The knowledge acquisition and the development of representation forms for that knowledge was followed by the system's implementation. The system was implemented in the ProKappa software (described in §3.4), running under X windows on a Sun Spark 2 workstation.

The ProKappa software provided a favourable environment for the development of the system for the estimation of ground properties. The user-friendly DUI (Developer's User Interface) which incorporates an object browser (with graphical displays of object hierarchies), code debuggers (C and ProTalk workbenches) and an interface development workbench, and the excellent model engine are two of the major advantages of the software.

The extensive use of the software during the system's development has also revealed some of its shortcomings. The most important of these was the instability of the software after long periods of use. In particular, the appearance of internal errors that were usually followed by the automatic shutdown of the software, which were not associated with any form of improper use of the software.

Furthermore the manipulation of large object bases through the object browser significantly slowed down the performance of the system. Finally, the use of the **Active Relations** application (described in §3.4.4) during development sessions caused break-downs in the software due to memory allocation problems. The most probable reason for that is that **Active Relations** is not fully integrated into the ProKappa software, rather it is called by it as an external compatible program.

In either case, these problems were only encountered during the development of the system and not during its use. Hence, the overall performance of the ProKappa software can be described as satisfactory, at least with respect to the developed system.

The implementation of the knowledge in the system was based on the object-oriented approach, which is a feature supported by the ProKappa software. The use of objects for the representation of the domain knowledge proved to be particularly advantageous for the imposed task. This can be demonstrated by various aspects of the developed system. These are presented in more detail in the following paragraphs.

The system incorporates five knowledge bases (specifically: the basic and specific ground types, ground parameters, "typical" values and correlations knowledge bases), which are contained in three separate ProKappa applications and a ProKappa application module (**ground_representation**, **expand**, **GPar**, **est** and **correlation**, respectively). Each of these applications (or modules) contains a number of objects, which incorporate the domain specific knowledge and whenever required, the appropriate behaviour as well.

The main advantage provided by this representation form is that a relatively large task (in this case the estimation of ground parameters) can be broken down into a

number of smaller subtasks, which can be handled with greater ease by the system developer, since these are inherently less complex than the original task.

At the object level, the use of slots and facets allows for the incorporation of the relevant knowledge (properties and behaviour through method slots) in the object's definition. Each slot or facet is used to represent a specific chunk of the knowledge associated with the object (explicit representation of knowledge). Therefore, each individual chunk of knowledge can be accessed, retrieved and modified simply through the assessment of the appropriate slot or facet. The benefits from this are: the transparency of the knowledge base; and that the part of the inference mechanism which is responsible for the manipulation of the system's knowledge is consequently reduced down to the relatively simple task of slot and facet assessment.

A clear separation between the knowledge bases and the inference mechanism has been achieved. New correlations or "typical" values can be added to the system without any need to modify the inference mechanism.

The coupling of the object oriented approach with generic representation forms provides the ability of adding new knowledge in the system. This can be demonstrated by the existence of the knowledge acquisition facilities incorporated in the system (§4.7, chapter 6).

For example, each correlation is represented as an object, containing a number of slots and facets, the totality of which is used to represent all the aspects of the incorporated knowledge. Each of these slots and facets is associated with a slot or facet type, which in turn corresponds to a specific type of knowledge: e.g. slots and facets that represent: variables, parameters, applicability etc.). Therefore, in order to add a new correlation to the system, the user must specify the necessary slot and

facet types and then supply the system with the relevant information for each one of them.

Furthermore, the inference mechanism is based on the identification of a specific slot or facet type and the execution of the action that is associated with it. Hence, both the knowledge and inference in the system are implemented in a modular manner.

The advantage of this implementation can be demonstrated in the case of adding new knowledge types to the existing ones. This would be the only case where the inference mechanism would require changes to achieve this.

Each new knowledge type would need to be represented by one or more new slot and/or facet types. The necessary updating of the inference mechanism is merely relevant to the addition of the appropriate action, which must follow the occurrence of the new knowledge type. Therefore, the modularity in the representation and inference allows the inclusion of new knowledge types without the need for reimplementation of the system.

7.1 Future development.

The system has been developed as a stand-alone module. It is intended that in the future it will form a part of another knowledge-based system, currently being developed at the University of Durham for interpreting geotechnical information from a site investigation [Toll and Oliver, 1993].

The system, known as SIGMA, is being developed in a modular manner, operating around a central database of site investigation information and making use of

general knowledge about geotechnical engineering organised in individual knowledge bases. The site investigation database contains the geotechnical data (results from laboratory and field tests, engineering descriptions of the ground) which requires interpretation. This data can be used as the necessary input for the correlation and "typical" values knowledge bases to produce estimates for the required parameters.

This process can be easily automated, with the implementation of a function that will access all the data available for a particular layer (soil types, measurements of ground properties etc.). This data will be used as the criterion for searching the correlations object base (the functions for searching the correlation object base have already been implemented and are presented in §5.3.3) for the appropriate correlations, which will subsequently be presented to the user of the system. A similar process for an automated parameter search can be applied to the "typical" values knowledge base (this time using the ground type and any additional measurements of ground properties as the necessary input).

An important aspect for the enhancement of the presentation of the correlations results would be the inclusion of graphical routines in the system. For example, a two-dimensional representation of the estimated values of a property versus depth, will provide a better feel for the variation of the parameter's values, rather than a presentation of a list of numbers (especially when continuous measurements along the soil profile for a correlation variable are performed; e.g. in correlations that incorporate field test parameters).

Finally, the knowledge included in the system can be further extended with the inclusion of definitive relations between qualitative and quantitative parameters. For example, the translation of qualitative descriptors for relative density or

undrained shear strength to ranges of values for the corresponding quantitative parameters (D_r and S_u).

Explicit ranges of values exist for each descriptive term. These relations are merely relevant to the properties they involve and do not depend on ground type. Therefore, they can be represented in the appropriate parameter objects (in the parameters object base) as slots and facets.

The slot that will be used for representing the relation may be attached to the qualitative parameter and its value may be set to be a pointer to the object of the corresponding quantitative parameter. A number of facets may also be attached to it, each of which will be named after one of the qualitative parameter's permissible values and its value will be set to a range of values for the corresponding quantitative parameter. This representation can be supplemented by a piece of code that will search and retrieve the corresponding range of values, given a qualitative descriptor. Furthermore, if a number for the quantitative parameter is given, the system will search for the facet containing the range that this value falls into and retrieve the corresponding permissible value of the qualitative parameter (which is the name of that facet). It should be noted that no uncertainty assessment is relevant in this case, since the represented relations are definitions.

CHAPTER 8

Conclusions

The evaluation of ground properties is one of the most important problems in geotechnical engineering. Even though geotechnical testing is probably the most reliable source for obtaining values for ground properties, other methodologies such as the use of correlations and published summaries of "typical" values are also used to address the same problem.

The work described here aims to provide a framework for storing and using correlations and "typical" values for the estimation of ground properties, that will provide geotechnical engineers with a decision-support tool to assist with the property evaluation problem.

Knowledge-based system technology has been employed for tackling the imposed task as it provides a medium that can accommodate the representation and use of empirical knowledge; it also incorporates symbolic processing, the ability to update the system's knowledge and ways of representing uncertainty in the inference. The developed system makes use of all these features and demonstrates the applicability of knowledge-based systems in the area of ground property evaluation.

The system was implemented in the ProKappa software, running under X windows on a Sun Spark 2 workstation. It incorporates four knowledge bases which store information for the ground, its properties, correlations and "typical" values.

Implementation of the knowledge in the system was based on the object-oriented approach which is a ProKappa supported feature. The main advantage gained from this representation was the transparency of the knowledge bases.

The system also incorporates an inference mechanism and user interface facilities, which allow the user of the system to interrogate the knowledge bases and to use this knowledge for the estimation of ground properties.

Generic forms of representation were developed for correlations and "typical" values of ground properties, allowing the implementation of large numbers of the two knowledge types to be made in a consistent way. The utilisation of structured representation forms allows new correlations and sets of "typical" values to be implemented in the system even after the completion of the development stage.

Furthermore, each individual chunk of knowledge contained in a correlation (or a set of "typical" values) can be accessed, retrieved and modified (if necessary), allowing for an easy updating of the already implemented knowledge. To ensure ease of modification and addition of new knowledge, the system has been supplemented with four knowledge acquisition facilities (each of these corresponds to a different knowledge base). In this way it is ensured that the system will maintain its functionality in the future.

References

- Aas, G., Lacasse, S., Lunne, T. and Hoeg, K., (1986), "Use of in-situ tests for foundation design on clay", ASCE Spec. Conf. In-situ '86, Use of in-situ tests in Geotechnical Engineering, Blacksburg Virginia, USA, pp. 1-30.
- Adeli, H. (ed.), (1988), "Expert Systems in Construction and Structural Engineering", Chapman and Hall, London.
- American Society for Testing and Materials (A.S.T.M.) (1983), "Annual Book of ASTM Standards", *Section 4: Construction, Volume 04.08, Philadelphia.*
- Azzouz, A. S., Baligh, M. M., and Ladd, C. C., (1983), "Corrected field vane strength for embankment design", *Journal of Geotechnical Engineering*, Vol. 109, pp. 730-734.
- Begemann, H. K. S. Ph., Jousa, K., te Kamp, W. G. B., Krajicek, P. V. F. S., Heijnen, W. J. and van Weele, A. F., (1982), "Cone Penetration testing", *Civiele and Bouwkundige Techniek*, pp. 3-59.
- Bieniawski, Z. T., (1976), "Rock mass classification in rock engineering", *Proc. Symp. Expl. Rock Eng., Johannesburg*, pp. 97-106.
- Bjerrum, L., (1972), "Embankments on soft ground", state-of-the-art report, *Proc. American Society of Civil Engineers, Spec. Conf. on Performance of Earth and Earth-supported Structures, Lafayette, Ind., USA, Vol. 2*, pp. 1-54.

British Standard (B.S.) 5930, (1981), "Code of Practice for Site Investigations",
British Standards Institution, London.

Carter, M. and Bentley, S. P., (1991), "Correlations of soil properties", Pentech
Press, London.

Cooke, N. M. and McDonald, J. E., (1986), "A Formal Methodology for Acquiring
and Representing Expert Knowledge", Proceedings of the IEEE, October
1986, pp. 1422-1430.

Cordingley, E. S., (1989), "Knowledge elicitation techniques for knowledge-based
systems", from Knowledge Elicitation: principles, techniques and application
(ed. Diaper, D.), pp. 89-173.

Davey-Wilson, I. E. G., (1991), "Geotechnical Laboratory Test Simulation using AI
Techniques, in Artificial Intelligence and Civil Engineering" (ed. Topping B.
H. V.), CIVIL-COMP Press: Edinburgh, pp. 119-124.

Douglas, B. J. and Olsen, R. S., (1981), "Soil Classification Using the Electric Cone
Penetrometer.", in Cone Penetration Test Experience, (eds. Norris, G. M. and
Holtz, R. D.), ASCE, pp. 209-227.

Dym, C. L., (1987), "Implementation Issues in the Building of Expert Systems",
Expert Systems for Civil Engineers, ASCE, New York, pp. 35-45.

Feigenbaum, E. A., (1983), "Knowledge Engineering: The Applied Side", in
"Intelligent Systems; The Unprecedented Opportunity", (ed. Hayes J. E.), Ellis
Horwood Limited, Chichester, UK, pp. 37-55.

- Gevarter, W. B., (1987), "The Nature and Evaluation of Commercial Expert System Building Tools", Computer, pp. 24-41.
- Gillette, D. R., (1991), "An Expert System for Estimating Soil Strength Parameters", Proc. Geotechnical Engineering Congress, in Geotechnical Special Publication, Vol. 1, No. 27, (eds. McLean, F. G., Campbell, D. W. A. and Harris, D. W.), ASCE: Boulder, Colorado, pp. 276-287.
- Groothuizen, R. J. P., (1986), "Inexact Reasoning in Expert Systems - An Integrating Overview", National Aerospace Laboratory NLR, Report No. NLR TR 86009 U.
- IntelliCorp, (1991), "ProKappa User's Guide", Inc. Version 2.0, Publication No: PK2.0-UG-2.
- Jamiolkowski, M., Ladd, C. C., Germaine, J. and Lancellota, R., (1985), "New developments in field and laboratory testing of soils", 11th ICSMFE.
- Juang, C. H. and Lee, D. H., (1989), "Development of an Expert System for Rock Mass Classification", Civil Engineering Systems, 6, pp. 147-156.
- Kulhawy, F. H. and Mayne, P. W., (1990), "Manual on Estimating Soil Properties for Foundation Design", Rpt. EL-6800, Electric Power Res. Inst., Palo Alto.
- Kulhawy, F. H., (1992), "On the evaluation of static soil properties", Slopes and Embankments, pp. 95-115.

- Leroueil, S., Tavenas, F. & Le Bihan, J. P., (1983), "Propriétés caractéristiques des argiles de l' est du Canada" Canadian Geotechnical Journal, Vol. 20 (4), pp. 681-705.
- Maher, M. L. and Allen, R., (1987), "Expert Systems Components", in "Expert Systems for Civil Engineers", (ed. Maher M. L.), ASCE: New York, pp. 3-14.
- Miles, J. C. and Moore, C. J., (1994), "Practical Knowledge-Based Systems in Conceptual Design.", Springer-Verlag, London.
- Moula, M., Toll, D. G., Vaptismas, N., (1994), "Knowledge-Based Systems in Geotechnical Engineering", Geotechnique, (in press).
- Mullarkey, P. W., (1987), "Languages and Tools for Building Expert Systems", in "Expert Systems for Civil Engineers", (ed. Maher M. L.), American Society of Civil Engineers, New York, U.S.A., pp. 15-34.
- Mullarkey, P. W. and Fenves, S. J., (1986), "Fuzzy logic in a geotechnical knowledge based system: CONE", Civil Engineering Systems, 3, 2, pp. 58-81.
- Mullarkey, P. W., (1986), "A Geotechnical KBS Using Fuzzy Logic", in "Applications of A.I. in Engineering Problems", (eds. Sriram D. and Adey R.), Springer-Verlag, Vol. 2, pp. 847-859.
- Skempton, A. W. and Northey, R. D., (1952), "The sensitivity of clays", Geotechnique, No. 3, pp. 30-53

- Skempton, A. W., (1986), "Standard Penetration Test Procedures and the Effects in Sands of Overburden Pressure, Relative density, Particle Size, Ageing, and Overconsolidation", *Geotechnique*, Vol. 36, No. 3, Sept. 1986, pp. 425-447.
- Stroud, M. A., (1988), "The Standard Penetration Test - Its application and interpretation", *Proceedings of the Institution of Civil Engineers Geotechnology Conference*, Birmingham, 6-8 July 1988, pp. 29-51.
- Sutcliffe, A., (1988), "Human-Computer Interface Design", Macmillan Education Limited.
- Tello, E. R., (1989), "Object-Oriented Programming for Artificial Intelligence: a guide to tools and system design", (ed. Addison-Wesley), pp. 15-35.
- Terzaghi, K. and Peck, R. B., (1948), "Soil Mechanics in Engineering Practice", John Wiley and Sons, New York, 584 p.
- Terzaghi, K. and Peck, R. B., (1967), "Soil Mechanics in Engineering Practice", 2nd Ed., John Wiley and Sons, New York, 729 p.
- Toll, D. G. and Oliver, A. J., (1993), "SIGMA: a Knowledge-Based System for the Interpretation of Geotechnical Information", *Proc. SERC Conf. on Informing Technologies for Construction, Civil Engineering and Transport*, (eds Powell J. and Day R.), Brunel University, London, pp. 245-254.
- Van Melle, W., (1979), "A Domain Independent Production-Rule System for Consultation Programs", *Proc. 6th IJCAI*, August 1979, pp. 923-925.

Weiss, S. M. and Kulikowski, C. A., (1979), "EXPERT: A System for Developing Consultation Models", Proc. 6th IJCAI, August 1979, pp. 942-947.

West, G., (1991), "The Field Description of Engineering Soils and Rocks", Geological Society of London Professional Handbook.

Appendix A

Correlations and Corrections

Part 1: Correlations.

1. Correlations for the estimation of peak effective angle of friction, ϕ' .

1. $\phi' = f(D_r)$

Reference: Giuliani and Giuliani, 1982.

Estimation procedure: $\phi' = \text{Arctan} [0.575 + 0.361(D_r/100)]^{0.866}$

Required parameters: Relative density, D_r , with range of values: 0 - 100. ϕ' ranges from 30° to 43°.

Applicability: all coarse soils, especially sands.

Reliability: low (more consistent testing is required).

2. $\phi'_{TC} = f(D_r, \rho_d)$

Reference: U.S. Navy, 1982.

Estimation procedure: dependent on D_r as follows:

D_r between 0 and 25: $\phi'_{TC} = 2.5\rho_d + (0.86\rho_d - 0.2)D_r/25 + 23.2$

D_r between 25 and 50: $\phi'_{TC} = 4.36\rho_d + (2.28\rho_d - 0.8)D_r/25 + 23.4$

D_r between 50 and 75: $\phi'_{TC} = 6.64\rho_d + (1.79\rho_d - 0.73)D_r/25 + 22.6$

D_r between 75 and 100: $\phi'_{TC} = 8.43\rho_d + (1.5\rho_d - 0.4)D_r/25 + 21.87$

Required parameters: D_r , relative density with values ranging between 0-100, ρ_d in t/m^3 (Mg/m^3). The range of ρ_d depends on the value of relative density as follows:

D_r between 0 and 25: $0.004*D_r + 1.2 < \rho_d < 0.0044*D_r + 1.89$

D_r between 25 and 50: $0.004*D_r + 1.3 < \rho_d < 0.0048*D_r + 1.89$

D_r between 50 and 75: $0.0036*D_r + 1.4 < \rho_d < 0.004*D_r + 2.12$

D_r between 75 and 100: $0.0036*D_r + 1.5 < \rho_d < 0.0039*D_r + 2.22$

Applicability: all inorganic coarse soils, including inorganic non-plastic silt.

Reliability: low to medium

Comments: The effective angle of friction is also dependent upon soil type. However, its variation due to soil type is negligible and therefore for reasons of simplicity it is not considered here. According to Kulhawy and Mayne [1990], the estimated angle approximates peak angle of friction, in triaxial compression (ϕ'_{TC}).

3. $\phi' = f(D_r, \text{soil type})$

Reference: Schmertmann, 1978.

Estimation procedure: Dependent on soil type as follows:

uniform gravel, or well-graded gravel-sand-silt: $\phi' = 0.08 * D_r + 38$

uniform coarse sand or well-graded medium sand: $\phi' = 0.10 * D_r + 35$

uniform medium sand or well-graded fine sand: $\phi' = 0.12 * D_r + 31$

uniform fine sand: $\phi' = 0.14 * D_r + 28$

Required parameters: relative density D_r with values in the range: 0-100.

Applicability: from gravel to fine sand (silts and silty soils excluded) either uniform or well-graded.

Reliability: low to medium.

Comments: the correlation does not take into account the effect of overburden pressure. It is relevant to predominately, quartz sands and was derived from large-scale shallow footing tests on sand. As an indirect correlation, it involves double uncertainty.

4. $\phi'_{TC} = f(y)$

Reference: Been and Jefferies, 1985.

Estimation procedure: $\phi'_{TC \max} = 40.93y^2 - 37.16y + 34.935$

$\phi'_{TC \min} = 50.14y^2 - 30.97y + 30.8$

$\phi'_{TC \text{mean}} = 45.26y^2 - 34.06y + 32.87$

Required parameters: y' , the state parameter of sand, is defined as the distance of the line $e - \log I_1$ (I_1 : first stress invariant) from the steady state line (SSL) in an $e - \log I_1$ plot, ranging from -0.3, to +0.1. The corresponding ranges of ϕ'_{TC} are: $\phi'_{TC \max}$ from 31.6° to 49.7°, $\phi'_{TC \min}$ from 28.2° to 44.6°, and $\phi'_{TC \text{mean}}$ from 29.9° to 47.1°.

Applicability: sands, silty sands, sand-silt mix.

Reliability: high (for range of ϕ' : 24 - 48) provided the value of y' is accurately measured.

Comments: The angle ϕ' estimated from this correlation corresponds to the drained angle of shearing resistance in triaxial compression (ϕ'_{TC}).

5. $\phi'_{PSC}, \phi'_{TC} = f(D_r, Q, p'_f, \phi'_{cv})$

Reference: Bolton, 1986.

Estimation procedure:

for plain strain compression: $\phi'_{PSC} = \phi'_{cv} + 5\{D_r[Q - \ln(100p'_f/p_a)]/100 - 1\}$

for triaxial compression: $\phi'_{TC} = \phi'_{cv} + 3\{D_r[Q - \ln(100p'_f/p_a)]/100 - 1\}$

if mineralogy = quartz, feldspar $Q=10$

if mineralogy = limestone $Q=8$

if mineralogy = anthracite $Q=7$

if mineralogy = chalk $Q=5.5$

Required parameters: critical state angle, ϕ'_{cv} , in degrees, relative density, D_r , soil mineralogy and compressibility coefficient, Q , mean principal effective stress at failure, p'_f in kPa, and the atmospheric pressure, $p_a \approx 100$ kPa.

Applicability: clean sands.

Reliability: medium.

Comments: For preliminary estimations, if the value of p'_f is not known it can be assumed to be $p'_f = 2\sigma'_{vo}$, which should typically lead to an estimated value of ϕ' within 1° to 2° of the actual value. However, for final design, the value of p'_f corresponding to the specific loading conditions (initial stress state, stress path to failure, test conditions and field problem specifications) should be used.

6. $\phi' = f(N_{SPT}, \text{soil type})$

Reference: Peck, Hanson and Thorburn, 1974.

Estimation procedure:

a. Sands: $\phi' = [(N_{SPT})^{6/9} + 21.6]/0.868$

b. Sandy gravels: $\phi' = -0.002(N_{SPT})^2 + 0.44(N_{SPT}) + 28.4$

Required parameters: N_{SPT} from Standard Penetration Tests, ranging from less than 4, up to more than 50 blows.

Applicability: all coarse soils.

Reliability: low to medium.

Comments: For fine sands, silty sands and silts below the water table. a correction is required for N values greater than 15 according to the following formula: $N' = 15 + 0.5*(N-15)$, (after Terzaghi and Peck, 1967).

7. $\phi'_{TC} = f(N_{SPT}, \sigma'_{vo})$

Reference: Schmertmann, 1975.

Estimation procedure: $\phi'_{TC} = \text{Arctan}[N_{SPT}/(12.2 + 20.3\sigma'_{vo}/p_a)]^{0.34}$

Required parameters: N_{SPT} from Standard Penetration Tests, ranging from less than 4 up to more than 60 blows and effective overburden pressure, σ'_{vo} , ranging between 0 and 300 kPa.

Applicability: all coarse soils.

Reliability: low to medium.

Comments: the estimated values of angle of friction in triaxial compression are conservative. The use of this correlation should be avoided in very shallow depths ($\sigma'_{vo} < 30$ kPa).

8. $\phi' = f(q_c)$

Reference: Meyerhof, 1963.

Estimation procedure: $\phi' = 0.75(q_c/10)^{0.5} + 23.9$

Required parameters: cone penetration resistance q_c in kPa.

Applicability: all coarse soils.

Reliability: low.

9. $\phi' = f(q_c, \sigma'_{vo})$

Reference: Durgunoglu and Mitchell, 1975.

Estimation procedure: $\phi' = b_i - 2(a - a_i)/(a_{i+1} - a_i)$

for $i = 1$ to 7 :

$b_i = \{46, 44, 42, 40, 38, 36, 34\}$

$a_i = \{1.833, 3.25, 5.5, 8.68, 13.05, 19.01, 26.74, 36.5\}$

$a = 1000\sigma'_{vo}/q_c$ and $a_i < a < a_{i+1}$.

Required parameters: q_c from CPT in kPa and range of applicability: 0-60000 kPa, σ'_{vo} , effective overburden pressure in kPa, and range of applicability: 20 - 50 kPa.

Applicability: medium and medium-to-coarse, clean, uniform sand, NC, predominately quartz with some feldspar and perhaps a small amount of mica, particle shape varying from rounded to sub-angular.

Reliability: medium.

Comments: The value of ϕ' from this correlation is usually a lower bound for these types of sand. The correlation does not take into account the effects of soil compressibility. Therefore for the more compressible ones (also with angular grains, higher content of mica, more uniform) ϕ' may be higher by up to 2 degrees. However, the presence of compressible sands can be detected from their friction ratios (if $R_f \geq 0.5$). For cemented sands the value of ϕ' is also underestimated. For OC sands ϕ' (measured as a secant angle) is overestimated by 1 to 2 degrees. To include the effect of a curved strength envelope, corrections should be made at high confining stresses. The reduction of ϕ' depends on D_r as follows:

$0 < D_r < 35$	ϕ' less by 0 to 1 degree.
$35 < D_r < 65$	ϕ' less by 2 to 3 degrees.
$65 < D_r < 85$	ϕ' less by 3 to 5 degrees.
$85 < D_r < 100$	ϕ' less by 5 to 8 degrees.

10. $\phi'_{TC} = f(q_c, \sigma'_{vo})$

Reference: Robertson and Campanella, 1983.

Estimation procedure: $\phi'_{TC} = \text{Arctan} [0.1 + 0.38 \log(q_c/\sigma'_{vo})]$

Required parameters: q_c from CPT in kPa and range of applicability: 0 - 50000 kPa and, σ'_{vo} , effective overburden pressure in kPa, and range of applicability: 50 - 400 kPa.

Applicability: NC, uncemented, quartz sands.

Reliability: medium.

Comments: The remarks made for the previous relation [Durgunoglu and Mitchell, 1975] apply to this relation as well.

11. $\phi'_{TC} = f(q_T, \sigma'_{ho})$

Reference: Houlsby and Wroth, 1989.

Estimation procedure: $\phi'_{TC\ min} = [\ln(q_T/\sigma'_{ho}) - 0.4]/0.16 + 9$

$$\phi'_{TC\ max} = [\ln(q_T/\sigma'_{ho}) + 0.6]/0.16 + 9$$

$$\phi'_{TC\ mean} = \ln(q_T/\sigma'_{ho})/0.16 + 9$$

Required parameters: q_T , from piezocone tests, in kPa, σ'_{ho} , horizontal effective stress, in kPa. $\ln(q_T/\sigma'_{ho})$ ranges from 3 to 7, while ϕ'_{TC} ranges from 30° to 50°

Applicability: sands

Reliability: medium.

12. $\phi' = f(PI)$

Reference: Bjerrum and Simons, 1960.

Estimation procedure: $\phi' = -0.0005PI^2 - 0.1044PI + 34.1$

Required parameters: the plasticity index PI ranges from 5 to 100, while ϕ'_{peak} ranges from 19° to 34°.

Applicability: NC, uncemented insensitive clays.

Reliability: low. The maximum deviation of the value of ϕ'_{peak} is $\pm 4^\circ$.

Comments: $\phi' = \phi'_{peak} = \phi_{cv}$, for NC clays.

13. $\phi' = f(PI)$

Reference: Mitchell, 1976.

Estimation procedure: $\phi' = \text{Arcsin}[0.8 - 0.094 \ln PI]$

Required parameters: the plasticity index PI ranges from 6 to 100, while ϕ'_{peak} ranges from 20° to 39°.

Applicability: NC, uncemented insensitive clays.

Reliability: low. The maximum deviation of the value of ϕ'_{peak} is $\pm 4^\circ$.

2. Correlations for the estimation of critical state angle, ϕ'_{cv}

1. $\phi'_{cv} = f(\text{angularity, } d_{10}, C_u, D_r, \text{mineralogy})$.

Reference: Koerner, 1970.

Estimation procedure: $\phi'_{cv} = 36^\circ + \Delta\phi_1 + \Delta\phi_2 + \Delta\phi_3 + \Delta\phi_4 + \Delta\phi_5$

$\Delta\phi_1 = -6^\circ$ for high sphericity and subrounded shape

$\Delta\phi_1 = +2^\circ$ for low sphericity and angular shape

$\Delta\phi_2 = -11^\circ$ for $d_{10} > 2.0$ mm

$\Delta\phi_2 = -9^\circ$ for $2.0 \text{ mm} > d_{10} > 0.6$ mm

$\Delta\phi_2 = -4^\circ$ for $0.6 \text{ mm} > d_{10} > 0.2$ mm

$\Delta\phi_2 = 0^\circ$ for $0.2 \text{ mm} > d_{10} > 0.06$ mm

$\Delta\phi_3 = -2^\circ$ for $C_u > 2$

$\Delta\phi_3 = -1^\circ$ for $C_u = 2$

$\Delta\phi_3 = -1^\circ$ for $C_u < 2$

$\Delta\phi_4 = -1^\circ$ for $0 < D_r < 50$

$\Delta\phi_4 = 0^\circ$ for $50 < D_r < 75$

$\Delta\phi_4 = +4^\circ$ for $75 < D_r < 100$

$\Delta\phi_5 = 0^\circ$ for quartz

$\Delta\phi_5 = 0^\circ$ for feldspar, calcite, chlorite

$\Delta\phi_5 = 0^\circ$ for muscovite mica

Required parameters: angularity, with values subrounded and angular, particle size of 10% passing, d_{10} , coefficient of uniformity, C_u , relative density, D_r , and mineralogy with values, quartz, feldspar, calcite, chlorite and muscovite mica.

Applicability: single mineral soils from fine sand to gravel.

Reliability: medium

Comments: According to Bolton's theory of dilatancy, relative density should not influence ϕ'_{cv} and therefore $\Delta\phi_4 = 0^\circ$. It should also

be noted that the presence of some silt in a sand deposit will lower the value of ϕ'_{cv} .

2. $\phi'_{cv} = f(\text{mineralogy, angularity})$.

Reference: Data from Stroud, 1988.

Estimation procedure:

$$\phi'_{cv} = 0.96(\text{ang}) + 0.11(\text{ang})^2 - 2.46(\text{min}) + 0.83(\text{min})^2 - 0.06(\text{min})^3 + 30.3 \pm 1.6$$

Note: ang stands for angularity and min for mineralogy.

Required parameters: Mineralogy of the soil and Angularity of the grains.

The values assigned to these parameters can be inferred from the following table:

Mineralogy.	Mineralogy number	Angularity	Angularity number
quartz.	1	well rounded	1
quartz with some feldspar.	2	rounded	2
quartz and feldspar	4	sub-rounded	2.5
feldspar and quartz	6	sub-angular	3
feldspar	8	angular	4
		very angular	5

Applicability: sands and gravels either quartz or feldspar, or combinations of the two.

Reliability: medium to high. $r^2=94\%$, $sd=1^\circ$

Comments: This correlation was produced by the author using data from Stroud [1988]. The scales used for both the mineralogy and the angularity are arbitrary.

3. Correlation for the estimation of remoulded angle of friction, ϕ'_{rem} .

1. $\phi'_{rem} = f(PI)$

Reference: Gibson, 1953.

Estimation procedure: $\phi'_{rem} = 79.9/PI + 0.00066PI^2 - 0.1634PI + 27$

Required parameters: the plasticity index PI ranges from 12 to 120, while

ϕ'_{rem} ranges from 15° to 32°.

Applicability: all fine soils.

Reliability: low.

4. Correlation for the estimation of residual angle of friction, ϕ'_{res} .

1. $\phi'_{res} = f(PI)$

Reference: Gibson, 1953.

Estimation procedure: $\phi'_{res} = 0.00224PI^2 - 0.432PI + 30.96$

Required parameters: the plasticity index PI ranges from 0 to 110, while ϕ'_{res}

ranges from 8° to 31°.

Applicability: all fine soils.

Reliability: low.

5. Correlations for the estimation of undrained shear strength, S_u

1. $S_u = f(LI)$

Reference: Skempton and Northey, 1952.

Estimation procedure: Probable upper limit: $S_{u_{max}} = 10^{(0.02LI^3 + 0.8LI^2 - 1.8LI + 2.05)}$

Probable lower limit: $S_{u_{min}} = 10^{(-0.137LI^3 + 1.37LI^2 - 2.39LI + 1.91)}$

Required parameters: Liquidity index LI, ranging between 0 and 1.0, while S_u ranging between 5 and 120 kPa.

Applicability: clays of low to moderate sensitivity with natural moisture contents below their liquid limit ($LI < 1$).

Reliability: low.

Comments: the correlation presented above is a combination of correlations obtained from Skempton and Northey's original work.

2. $S_{uFV} = f(\sigma'_{vo}, PI)$

Reference: Skempton and Bjerrum, 1957.

Estimation procedure: $S_{uFV} = \sigma'_{vo}(0.11 + 0.0037*PI)$

Required parameters: effective overburden pressure, σ'_{vo} , and plasticity index, PI.

Applicability: NC clays.

Reliability: low.

Comments: Skempton's correlation can be useful for preliminary estimations and for cross-checking laboratory data, since the required parameters are relatively easy to obtain. The S_u obtained corresponds to field vane shear strength, S_{uFV} .

3. $S_{uFV} = f(\sigma'_p, PI)$

Reference: Chandler, 1988.

Estimation procedure: $S_{uFV} = \sigma'_p(0.11 + 0.0037*PI)/(1 \pm 25\%)$

Required parameters: effective preconsolidation pressure, σ'_p and plasticity index, PI.

Applicability: OC clays, not fissured, organic, or sensitive.

Reliability: low (accuracy $\pm 25\%$).

Comments: Chandler suggest that the Skempton and Bjerrum [1957] correlation can be applied to OC clays if the effective overburden pressure is substituted with the preconsolidation pressure.

4. $S_{u_{FV}} = f(\sigma'_p, PI)$

Reference: Leroueil, Tavenas and Le Bihan, 1983.

Estimation procedure: $S_{u_{FV}} = \sigma'_p(0.2 + 0.0024*PI)$

Required parameters: plasticity index PI with range of application: 5 - 60% and preconsolidation pressure, σ'_p , in kPa. $S_{u_{FV}}/\sigma'_p$ ranges from 0.2 to 0.35.

Applicability: soft clays from Eastern Canada with $PI < 60$.

Reliability: low.

Comments: this correlation cannot have global applicability. It is mainly applicable to soft clays with low to medium values of plasticity index.

5. $S_{u_{TC}} = f(\sigma'_i, PI)$

Reference: Wroth and Houlsby, 1985.

Estimation procedure: $S_{u_{TC}} = \sigma'_i(0.129 + 0.00435*PI)$

Required parameters: the effective overburden stress after isotropic consolidation, σ'_i and plasticity index, PI.

Applicability: NC clays.

Reliability: low.

Comments: the undrained shear strength estimated from this correlation, corresponds to conditions of triaxial compression after isotropic consolidation, $S_{u_{TC}}$.

6. $S_{\text{UDSS}} = f(\sigma'_p)$

References: Jamiolkowski, Ladd, Germaine and Lancellota, 1985.
Mesri, 1989.

Estimation procedure: a. $S_{\text{UDSS}} = (0.23 \pm 0.04)\sigma'_p$

b. $S_{\text{UDSS}} = 0.22\sigma'_p$

Required parameters: effective preconsolidation pressure, σ'_p .

Applicability: low OCR clays with low to moderate PI.

Reliability: low.

Comments: undrained shear strengths obtained from the above relationships, corresponding to direct simple shear conditions (S_{UDSS}), can only be considered as approximations.

7. $S_u = f(\sigma'_{\text{vo}}, \text{OCR})$

Reference: Jamiolkowski, Ladd, Germaine and Lancellota, 1985.

Estimation procedure: $S_u = \sigma'_{\text{vo}} * (0.23 \pm 0.04) * \text{OCR}^{0.8}$

Required parameters: effective overburden pressure, σ'_{vo} , in kPa and the overconsolidation ratio, OCR.

Applicability: non-cemented clays with a plasticity index less than 60%.

Reliability: low

Comments: this correlation can be used for preliminary estimations, or to cross-check other estimation methods.

8. $S_{\text{UCTUC}}, S_{\text{UCAUC}} = f(\sigma'_{\text{vo}}, \phi'_{\text{TC}})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: a. $S_{\text{UCTUC}} = 0.012\phi'_{\text{TC}}\sigma'_{\text{vo}}$

$S_{\text{UCTUC min}} = 0.01\phi'_{\text{TC}}\sigma'_{\text{vo}}$ and $S_{\text{UCTUC max}} = 0.015\phi'_{\text{TC}}\sigma'_{\text{vo}}$

b. $S_{\text{UCAUC}} = 0.0117\phi'_{\text{TC}}\sigma'_{\text{vo}}$

$S_{\text{UCAUC min}} = 0.01\phi'_{\text{TC}}\sigma'_{\text{vo}}$ and $S_{\text{UCAUC max}} = 0.014\phi'_{\text{TC}}\sigma'_{\text{vo}}$

Required parameters: effective overburden pressure, σ'_{vo} , in kPa and the angle of friction in triaxial compression, ϕ'_{TC} , ranging between 20° and 50°.

Applicability: intact NC clays.

Reliability: medium. $r^2 = 0.686$, $sd = 0.055 * \sigma'_{vo}$.

Comments: data bases of 81 intact and 1 fissured clays, and 71 intact and 1 fissured clays respectively were used to produce the two relations. The undrained shear strength estimated from the first relation, corresponds to conditions of triaxial compression after isotropic consolidation, S_{uCIUC} , whilst the second to triaxial compression after anisotropic consolidation, S_{uCAUC} , (consolidation to either k_o stresses using special testing procedures, or estimated k_o stresses, or finally, some general anisotropic stress which may or may not correspond to k_o conditions). The reliability of the correlations increases for $\phi'_{TC} > 20^\circ$.

9. $S_{uTC} = f(\sigma'_{vo}, \phi'_{TC})$

References: Wroth and Houlsby, 1985.

Estimation procedure: $S_{uTC} = \sigma'_{vo} 0.5743(3 + \sin\phi'_{TC}) / (3 - \sin\phi'_{TC})$

Required parameters: effective overburden pressure, σ'_{vo} , in kPa and the angle of internal friction from triaxial compression tests, ϕ'_{TC}

Applicability: NC. clays.

Reliability: low to medium.

Comments: the undrained strength estimated from this correlations corresponds to undrained strength obtained from triaxial compression tests, S_{uTC} .

10. $S_{uTC} = f(\sigma'_{vo}, c', \phi', k_o, A_f)$

Reference: Thorne, 1984.

Estimation procedure: $S_{uTC} = (c' \cos \phi'_{TC} - U_o \sin \phi'_{TC}) / [1 - (1 - 2A_f) \sin \phi'_{TC}]$

$$U_o = -\sigma'_{vo} (1 + 2k_o) / 3$$

Required parameters: the effective cohesion, c' , the angle of internal friction, ϕ' , Skempton's A parameter at failure, A_f and the parameter U_o relating to the average effective stress, $\sigma'_m = 1/3 * (\sigma'_{vo} + 2\sigma'_{ho})$ as follows: $U_o = -\sigma'_m$

Applicability: both NC. and OC. clays.

Reliability: low to medium.

Comments: the undrained strength estimated from this correlations corresponds to undrained strength obtained from triaxial compression tests, S_{uTC} .

11. $S_u = f(K_D, \sigma'_{vo})$

References: Marchetti, 1980.

Lacasse, and Lunne, 1988.

Estimation procedure: Marchetti: $S_{uTC} = 0.22(0.5K_D)^{1.25}$

Lacasse and Lunne: $S_u = a(0.5K_D)^{1.25}$

for triaxial compression, (S_{uTC}) $a=0.20$

for direct shear, (S_{uDSS}) $a=0.14$

for field vane strength, (S_{uFV}) $0.17 < a < 0.21$

Required parameters: Effective overburden pressure, σ'_{vo} , in kPa and the dilatometer horizontal stress index, K_D , ranging from 1 to 20.

Applicability: soft clays.

Reliability: medium.

Comments: the Marchetti correlation was found to be off by a factor of 2 in stiff, old, UK clays, while in soft clays, it works reasonably

well. The Lacasse and Lunne correlation depends on the type of test used to establish the shear strength.

6. Correlation for the estimation of remoulded undrained shear strength, $S_{u\text{ rem}}$.

1. $S_{u\text{ rem}} = f(\text{LI})$

Reference: Leroueil, Tavenas and Le Bihan, 1983.

Estimation procedure: $S_{u\text{ rem}} = 1/(\text{LI} - 0.21)^2$

Required parameters: liquidity index, LI with range of application: 0.4 - 3.0, while the remoulded shear strength, $S_{u\text{ rem}}$, ranges from 0.1 to 30 kPa.

Applicability: soft clays.

Reliability: medium to high.

Comments: the values of liquidity indices were obtained using the cone penetrometer. The relation between LI and $S_{u\text{ rem}}$ is very well defined within the range of application. The data set consists of a large number of soft clays from North America, Sweden and Norway, fitting closely to the recommended curve. It can also be noted that with an estimation of the clay's sensitivity the above correlation can be used for producing an estimate of the intact shear strength.

7. Correlation for the estimation of sensitivity, St

1. $St = f(\text{LI})$

Reference: Skempton and Northey, 1952.

Estimation procedure: $St = e^{(0.64\text{LI}^3 - 1.28\text{LI}^2 + 2.88\text{LI} - 0.186)}$

LI ≤ 0.4 :	no correction.
0.4 > LI ≤ 0.8:	Sens = range (Sens - 2, Sens + 2)
0.8 > LI ≤ 1.2:	Sens = range (Sens - 2, Sens + 4.5)

Required parameters: the liquidity index LI.

Applicability: relatively moderate sensitivity clays with natural moisture contents below their liquid limit.

Reliability: medium.

8. Correlations for estimation of relative density, D_r

1 $D_r = f(N_{SPT}, \sigma'_{vo})$

References: a. Gibbs and Holtz., 1957.

b. Bazaraa, 1967.

Estimation procedure: a. $D_r = 1.5(N_{SPT}/F)^{0.222} - 0.6,$

$$\text{and } F = 0.000065(\sigma'_{vo})^2 + 0.168(\sigma'_{vo}) + 14$$

b. $D_r = 0.2236\{N_{SPT}/[a + b(\sigma'_{vo}/10)]\}^{0.5}$

$$\text{and if } \sigma'_{vo} \geq 150 \quad a=1 \quad \text{and} \quad b=0.2$$

$$\text{else } \sigma'_{vo} \leq 150 \quad a=3.25 \quad \text{and} \quad b=0.05.$$

Required Parameters: the number of blows, N from the SPT and σ'_{vo} the effective overburden pressure in kPa.

Applicability: sands.

Reliability: low.

Comments: the relationships presented here were obtained from regression of the variables from Giuliani and Giuliani [1982], based on the earlier work from Bazaraa [1967] and Gibbs and Holtz [1957].

2. $D_r = f(N_{SPT}, \sigma'_{vo}, C_u, OCR)$

Reference: Marcuson and Bieganousky, 1977.

Estimation procedure:

$$D_r = 100 * \{ 12.2 + 0.75 * [222 N_{SPT} + 2311 - 711 OCR - 779 (\sigma'_{vo} / 100) - 50 C_u^2]^{0.5} \}$$

Required Parameters: the number of blows N obtained from the SPT, σ'_{vo} the effective overburden pressure in kPa, the coefficient of uniformity, C_u (scalar), and the overconsolidation ratio OCR

Applicability: unaged sands, with OCR equal to 1 or 3.

Reliability: medium. $r^2 = 0.77$.

Comments: the relationship describing the correlation was obtained from regression analysis of the data from collected from Marcuson and Bieganousky, referring to unaged, NC or lightly OC sands.

3. $D_r = f(N_{SPT}, \sigma'_{vo}, D_{50}, t, OCR, ER, d, SM, l)$

Reference: Skempton, 1986.

Estimation procedure:

$$D_r = (C_{ER} C_B C_S C_R C_N N_{SPT}) / (C_P C_A C_{OCR}).$$

The coefficients of the above relation are given in the following table:

Factor	Equipment variables	Term	Value
Energy ratio	Pilcon, Dando, UK	C_{ER} , (ER)	1.0
	old standard, UK		0.8
	Safety hammer, USA		0.9
	Donut hammer, USA		0.75

Factor	Equipment variables	Term	Value
Borehole diameter	65 mm < d < 115 mm	$C_{B, (d)}$	1.0
	d=150 mm		1.05
	d=200 mm		1.15
Sampling method	Standard sampler	$C_{S, (SM)}$	1.0
	Sampler without liner		1.2
Rod length	l > 10 m	$C_{R, (l)}$	1.0
	6 m < l < 10 m		0.95
	4 m < l < 6 m		0.85
	3 m < l < 4 m		0.75

for fine medium dense NC sand: $C_N = 2/(1 + \sigma'_{vo}/100)$,

for coarse dense NC sand: $C_N = 3/(2 + \sigma'_{vo}/100)$,

and for fine OC sand: $C_N = 1.7/(0.7 + \sigma'_{vo}/100)$,

$C_p = 60 + 25 \log D_{50}$, $C_A = 1.2 + 0.05 \log(t/100)$, and $C_{OCR} = OCR^{0.18}$.

Required parameters: the correction factors C_{ER} , C_B , C_S , C_R are related to field procedures and are defined in the above table, the vertical effective overburden pressure σ'_{vo} , ranging between 50 and 300 kPa, the particle size of the sand at 50% finer, the time t (in years) describing the age of the deposit, and the overconsolidation ratio OCR

Applicability: NC sands and fine OC sands

Reliability: medium.

Comments: the value of N is initially corrected for field procedures to an average energy ratio of 60%, to produce N_{60} , then a correction is applied to take account of effective overburden pressure, producing $(N'_1)_{60}$, and final adjustments for mean particle

size, age of deposit and overconsolidation ratio, producing a value of N directly correlated to the square of D_r . The relations describing the effects of particle size, ageing and OCR, were obtained from [Kulhawy and Mayne, 1990].

4. $D_r = f(q_c, \sigma'_m, K_o)$

Reference: Baldi, Bellotti, Ghionna, Jamiolkowski and Pasqualini, 1986.

Estimation procedure: $D_r = \ln\{q_c/[205(\sigma'_m/1000)^{0.51}]\}/2.93$

Required Parameters: cone resistance q_c in kPa and mean effective stress σ'_m in kPa, where $\sigma'_m = (\sigma'_{vo} + 2\sigma'_{ho})/3$. The range of q_c is 0 to 80000 kPa and the corresponding range of σ'_m is 50 to 500 kPa.

Applicability: sands both NC and OC

Reliability: medium to high

Comments: the determination of D_r requires an estimation of K_o . This can be made from additional measurements in the CPT or from other means.

5. $D_r = f(q_c, \sigma'_{vo})$

Reference: Jamiolkowski, Ladd, Germaine and Lancellota, 1985.

Estimation procedure: $D_r = -98 + 66 \cdot \log_{10}[10 \cdot q_c / (10 \cdot \sigma'_{vo})^{0.5}]$

$$k_q = 1 + 0.2(D_r - 30)/60$$

The procedure is as follows:

Step 1: determination of D_r from the formula.

Step 2: determination of K_q from the calculated value of D_r

Step 3: determination of D_r again with $q'_c = q_c/k_q$, until no further variation of the value of D_r is observed.

Required parameters: the cone resistance q_c and the effective vertical overburden pressure σ'_{vo} . The units for both

parameters are kPa and $q_c/(\sigma'_{vo})^{0.5}$ ranges from 95 to 6325 while D_r ranges from 15 to 100%.

Applicability: relatively uniform, clean, NC, predominately quartz sands where the in-situ horizontal stress ratio, K_o , is about 0.45.

Reliability: high for the type of sands quoted here.

Comments: the correlation was based on data from calibration chambers for five different NC sands. The relationship between relative density, D_r and cone resistance, q_c , of a sand is greatly affected by its compressibility. The value of D_r from the formula is applicable to sands of moderate compressibility. For the more compressible ones the value should be increased by up to 6.5. The opposite is true for the less compressible ones. Compressibility is greater where the sand is uniform, the grains are angular and there is an appreciable mica content. It should also be mentioned that in a thin layer of sand an underestimate of the value of D_r may be obtained, because the full cone resistance may not have developed. For OC sands, the same relationship may be used, if the initial horizontal effective stress σ'_{ho} is used instead of σ'_{vo} . The possible error in this case is in the order of $\pm 20\%$, and therefore the value of D_r obtained should be considered as an approximation.

6. $D_r = f(q_c, \sigma'_{vo}, \text{compressibility, OCR})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $(D_r/100)^2 = (q_c/100) / [305 * Q_c * OCR^{0.18} * (\sigma'_{vo}/100)^{0.5}]$

where Q_c , the compressibility factor and

if compressibility = high $Q_c = 0.91$

compressibility = medium $Q_c = 1.0$

compressibility = low $Q_c=1.09$

Required parameters: the cone resistance q_c , the effective vertical overburden pressure σ'_{vo} , the compressibility factor, Q_c , and the overconsolidation ratio, OCR. The units for q_c and σ'_{vo} are kPa and $Q_c=(q_c/100)/(\sigma'_{vo}/100)^{0.5}$ ranges from 0 to 450. OCR ranges between 1 and 15.

Applicability: sands.

Reliability: medium. $r^2=77.6\%$, and standard deviation, $sd=13$.

Comments: the correlation is based on the results from 24 sets of calibration test chambers in sand, in which the values of q_c were corrected for the effects of boundary conditions.

8. Correlation for estimation of dry unit weight, γ_d

1. $\gamma_d = f(D_r, C_u)$

Reference: Giuliani and Giuliani, 1982.

Estimation procedure: $\gamma_d = 1.33 * (1 + 0.0023D_r) * C_u^{0.1}$

Required parameters: relative density D_r and coefficient of uniformity C_u which can be obtained from the P.S.D. of the soil.

Applicability: sands.

Reliability: low to medium.

9. Correlation for estimation of drained Poisson's ratio, ν_d .

1. $\nu_d = f(\phi'_{TC})$.

Reference: Trautmann, and Kulhawy, 1987.

Estimation procedure: $\nu_d = 0.1 + 0.3(\phi'_{TC} - 25^\circ)/(45^\circ - 25^\circ)$

Required parameters: the angle of friction in triaxial compression, ϕ'_{TC} , ranging between 25° and 45°.

Applicability: all coarse soils.

Reliability: low.

Comments: the correlation reflects that denser soils (higher effective friction angle), have higher values of v_d .

10. Correlations for the prediction of coefficient of earth pressure at rest k_o .

1. $k_o = f(\phi'_{TC})$.

Reference: Jáký, 1944.

Estimation procedure: $k_o = 1 - \phi'_{TC}$

Required parameters: the angle of friction in triaxial compression, ϕ'_{TC} , ranging between 10° and 50°.

Applicability: all NC soil types.

Reliability: medium.

Comments: A regression analysis of 124 different specimens of various soil types by Mayne and Kulhawy [1982], showed a similar relation with $k_o = 0.97(1 - \sin\phi'_{TC})$, and $r^2 = 70\%$, standard deviation, $sd = 0.06$ (90% of the points fall in the range of $k_{o,est} \pm 0.1$).

2. $k_o = f(\sigma'_{vo}, K_D, q_c)$.

Reference: Baldi, Bellotti, Ghionna, Jamiolkowski and Pasqualini, 1986.

Estimation procedure: $k_o = 0.376 + 0.095K_D - 0.0046(q_c/\sigma'_{vo})$

Required parameters: the cone resistance q_c , the effective vertical overburden pressure σ'_{vo} and the dilatometer

horizontal stress index K_D . The units for q_c and σ'_{vo} are kPa. K_D should be between 1 and 20.

Applicability: sands

Reliability: low to medium.

3. $k_o = f(q_c, D_r, \sigma'_{vo})$.

Reference: Kulhawy, Jackson and Mayne, 1989.

Estimation procedure:

$$k_o = (q_c/p_a)^{1.25} / [35e^{(D_r/20)} (p_a/\sigma'_{vo})], \quad p_a \approx 100 \text{ kPa}$$

Required parameters: cone resistance q_c , effective overburden pressure σ'_{vo} , both in kPa, and relative density D_r , ranging between 20 and 100.

Applicability: both NC and OC sands.

Reliability: low to medium.

Comments: the correlation was obtained from calibration chamber tests data.

4. $k_o = f(PI)$ or $f(LL)$

Reference: Larsson, 1977.

Estimation procedure: $k_o = 0.0077 \cdot PI + 0.3 \pm 0.1$

$$k_o = 0.0072 \cdot LL + 0.15 \pm 0.12$$

Required parameters: plasticity index, PI, ranging between 0 and 60, or liquid limit, LL, ranging between 0 and 90.

Applicability: NC inorganic soft Swedish clays.

Reliability: low to medium for the type of clay quoted above.

Comments: the correlation was based on data from soft inorganic Swedish clays (local correlation). Organic clays deviate significantly from the above relation. A similar analysis [Kulhawy and

Mayne, 1990] with more clays from worldwide, demonstrated the lack of correlation between k_o and PI ($r^2=14.7\%$).

5. $k_o = f(LI, \sigma'_{vo})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure:

$$k_o = 0.5[(p_a/\sigma'_{vo})^{0.5} 10^{(0.56 - 0.81LI)}] \quad p_a \approx 100 \text{ kPa.}$$

Required parameters: liquidity index, LI, and effective overburden pressure, σ'_{vo} , in kPa.

Applicability: NC inorganic clays.

Reliability: low.

Comments: this is an indirect correlation, and therefore it involves double uncertainty. It should only be used as a first order estimator of k_o .

6. $k_o = f(\sigma'_{vo}, K_D, S_u)$.

Reference: Lunne, Powell, Hauge, Uglow and Mokkelbost, 1990.

Estimation procedure:

$$\text{for young clays } (S_u/\sigma'_{vo}) \leq 0.8: \quad k_o = aK_D^{0.54}, \quad 0.28 \leq a \leq 0.38$$

$$\text{for old clays } (S_u/\sigma'_{vo}) \geq 0.8: \quad k_o = 0.68K_D^{0.54}$$

Required parameters: the undrained shear strength S_u , the effective vertical overburden pressure σ'_{vo} and the dilatometer horizontal stress index K_D , ranging between 2 and 30.

Applicability: clays.

Reliability: medium.

Comments: the accurate determination of the quotient S_u/σ'_{vo} is not required for the execution of the correlation. It is merely used as an indicator of the age of the clay. The uncertainty relevant to the estimation of k_o in young clays is $\pm 20\%$.

11. Correlations for the prediction of overconsolidation ratio OCR

1. $OCR = f(k_o, \phi'_{TC})$.

Reference: Schmertmann, 1983.

Estimation procedure: $OCR = [k_o / (1 - \sin \phi'_{TC})]^{(1.25 / \sin \phi'_{TC})}$

Required parameters: the horizontal stress index k_o and the angle of internal friction corresponding to triaxial compression ϕ'_{TC} .

Applicability: sands

Reliability: low to medium.

Comments: the angle of friction can be estimated from dilatometer tests and corresponds to that measured in a drained triaxial compression test. k_o can also be estimated from the dilatometer test.

2. $OCR = f(\sigma'_{vo}, LI)$.

Reference: Wood, 1983.

Estimation procedure: $OCR = 10^{[1 - 2.5 \cdot LI - 1.25 \cdot \log(\sigma'_{vo}/pa)]}$

Required parameters: effective vertical overburden pressure, σ'_{vo} , in kPa and liquidity index, LI, ranging between - 0.2 and 1.

Applicability: unstructured, low sensitivity fine soils.

Reliability: low to medium.

Comments: the original relation contains the critical state parameter Λ , which is assumed equal to 0.8. The correlation is applicable only to insensitive soils at the critical state, but it can also be considered as a useful approximation for the types of soils quoted above.

3. $OCR = f(S_{uCIUC}, \sigma'_{vo}, \phi'_{TC})$ or $f(S_{uCAUC}, \sigma'_{vo}, \phi'_{TC})$

Reference: Mayne, 1988.

Estimation procedure: $OCR_{CTUC} = [(S_{uCTUC}/\sigma'_{vo})/0.75*\sin\phi'_{TC}]^{1.43}$, or

$$OCR_{CAUC} = [(S_{uCAUC}/\sigma'_{vo})/0.67*\sin\phi'_{TC}]^{1.28}$$

Required parameters: undrained shear strength from isotropically, or anisotropically consolidated triaxial compression tests, S_{uCTUC} , and S_{uCAUC} , respectively, effective overburden pressure σ'_{vo} , (both in kPa) and effective angle of friction in triaxial compression, ϕ'_{TC} .

Applicability: clays.

Reliability: low to medium.

Comments: the relation was based on data obtained from isotropically and anisotropically consolidated triaxial compression tests.

4. $OCR = f(\sigma'_{vo}, K_D, S_u)$.

Reference: Lunne, Powell, Hauge, Uglow and Mokkelbost, 1990.

Estimation procedure:

$$\text{for young clays} \quad (S_u/\sigma'_{vo}) \leq 0.8: \quad OCR = 0.3K_D^{1.17} \pm 30\%$$

$$\text{for old clays} \quad (S_u/\sigma'_{vo}) \geq 0.8: \quad OCR = 2.7K_D^{1.17} \pm 30\%$$

Required parameters: the undrained shear strength S_u , the effective vertical overburden pressure σ'_{vo} and the dilatometer horizontal stress index K_D , ranging between 2 and 30.

Applicability: clays.

Reliability: medium.

Comments: the accurate determination of the parameter S_u/σ'_{vo} is not required for the execution of the correlation. It is merely used as an indicator of the age of the clay.

5. $OCR = f(S_{uFV}, \sigma'_{vo})$

Reference: Mayne and Mitchell, 1988.

Estimation procedure: $OCR = 3.22*S_{uFV}/\sigma'_{vo}$

Required parameters: field vane undrained shear strength, S_{uFV} , and effective overburden pressure σ'_{vo} , both in kPa. Their quotient ranges between 0.1 to 10.

Applicability: intact clays.

Reliability: low to medium. $r^2 = 80.6\%$.

Comments: the relation was based on a sample of 96 intact clays (209 observations).

12. Correlations for the estimation of effective preconsolidation stress, σ'_p .

1. $\sigma'_p = f(LI)$

Reference: Stas and Kulhawy, 1984.

Estimation procedure: $\sigma'_p / 100 = 10^{(1.11 - 1.62 * LI)} \pm 0.4$.

Required parameters: the liquidity index, LI, ranging between - 0.2 and 1.4.

Applicability: clays with sensitivity between 1 to 10.

Reliability: medium, $r^2 = 74\%$, and $sd = 0.33$.

Comments: the correlation was obtained from 150 data points from clays of various sensitivities.

2. $\sigma'_p = f(S_{uFV})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure:

$$\sigma'_p = 3.54 * S_{uFV} \quad \sigma'_{pmin} = 2 * S_{uFV} \quad \sigma'_{pmax} = 6 * S_{uFV}$$

Required parameters: field vane undrained shear strength, S_{uFV} ranging between 10 and 500 kPa.

Applicability: intact clays.

Reliability: medium. $r^2 = 83.2\%$ (number of data points, $n=205$).

Comments: This correlation was produced using data from clays worldwide. The sample consisted of 96 intact clays and 1 fissured, which deviates significantly from its estimated value.

3. $\sigma'_p = f(N_{SPT})$.

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $\sigma'_p = a \cdot N_{SPT} \cdot p_a$,

$$p_a \approx 100 \text{ kPa.}$$

$$a_{\min} = 0.25, \quad a_{\text{mean}} = 0.47, \quad a_{\max} = 1$$

Required parameters: the number of blows, N_{SPT} .

Applicability: intact clays.

Reliability: low. $r^2 = 69.9$ (number of data points, $n=126$).

Comments: This correlation is useful only as a first order estimator of preconsolidation pressure. 49 intact and 2 fissured clays were tested.

4. $\sigma'_p = f(q_c)$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $\sigma'_p = 0.29 \cdot q_c$, $\sigma'_{p\min} = 0.12 \cdot q_c$, $\sigma'_{p\max} = 0.5 \cdot q_c$

Required parameters: the cone penetration tip resistance, q_c , ranging between 100 and 20000 kPa.

Applicability: intact clays.

Reliability: low to medium, $r^2 = 85.8$ (number of data points, $n=113$).

Comments: 39 intact and 10 fissured clays were tested. Preconsolidation stress of fissured clays can not be estimated from this relation.

5. $\sigma'_p = f(q_T, \sigma'_{vo})$

Reference: Tavenas and Leroueil, 1987.

Kulhawy and Mayne, 1990.

Estimation procedure: $\sigma'_p = 0.33*(q_T - \sigma'_{vo})$.

$$\sigma'_{pmin} = 0.2*(q_T - \sigma'_{vo}) \text{ and } \sigma'_{pmax} = 0.55*(q_T - \sigma'_{vo})$$

Required parameters: the cone penetration tip resistance, q_c , ranging between 60 and 10000 kPa, and the effective overburden pressure, σ'_{vo} (also expressed in kPa).

Applicability: intact clays.

Reliability: medium. $r^2 = 90.4$ (number of data points, $n=102$).

Comments: 26 intact and 9 fissured clays were tested. The preconsolidation stress of fissured clays can not be estimated from this relation.

6. $\sigma'_p = f(p_L)$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $\sigma'_p = 0.45*p_L$ $\sigma'_{pmin} = 0.3*p_L$ $\sigma'_{pmax} = 0.6*p_L$

Required parameters: the limit stress from the self boring pressurimeter test, p_L , ranging between 40 and 5000 kPa.

Applicability: intact clays.

Reliability: medium, $r^2 = 91\%$.

Comments: 35 intact and 6 fissured clays were tested. Preconsolidation stress of fissured clays can not be estimated from this relation.

13. Correlations for estimation of constrained modulus, M .

1. $M = f(\sigma'_{vo}, \Delta\sigma_v, q_c)$.

Reference: Lunne and Christoffersen, 1983.

Estimation procedure: for NC sands:

$$M_o = 4q_c \quad \text{for} \quad q_c < 10 \text{ MPa.}$$

$$M_o = 2q_c + 20 \quad \text{for} \quad 10 \text{ MPa} < q_c < 50 \text{ MPa.}$$

$$M_o = 120 \text{ MPa} \quad \text{for} \quad 50 \text{ MPa} < q_c.$$

$$M = M_o [(\sigma'_{vo} + \Delta\sigma_v) / \sigma'_{vo}]^{0.5}$$

for OC sands:

$$M_o = 5q_c \quad \text{for} \quad q_c < 50 \text{ MPa.}$$

$$M_o = 250 \text{ MPa} \quad \text{for} \quad 50 \text{ MPa} < q_c.$$

Required parameters: the cone resistance q_c , the effective vertical overburden pressure σ'_{vo} and the stress difference $\Delta\sigma_v$. The units for q_c are MPa .

Applicability: sands

Reliability: medium (conservative results).

Comments: the values of M obtained are conservative. The relationship for obtaining M along stress range (σ'_{vo} , $\sigma'_{vo} + \Delta\sigma_v$) is valid only for NC sands. For OC sands the exponent decreases with OCR and becomes 0 for heavily overconsolidated sands (OCR > 4).

2. $M = f(\sigma'_m, \text{OCR}, D_r, q_c)$.

Reference: Jamiolkowski, Ghionna and Lancellotta, 1988.

Estimation procedure: $M = q_c C_o p_a (\sigma'_m / p_a)^{C_1} (\text{OCR})^{C_2} (10)^{C_3 D_r}$

$$C_o = 14.48, C_1 = -0.116, C_2 = 0.313, C_3 = -1.123$$

and $p_a \approx 100 \text{ kPa}$.

Required parameters: the cone resistance q_c , the mean effective stress σ'_m the overconsolidation ratio, OCR and the relative density D_r . The units for q_c are MPa and for σ'_m kPa. D_r should be between 20% and 80%. σ'_m should be between 100 and 500 kPa.

Applicability: sands.

Reliability: medium.

Comments: the correlation was obtained from calibration chamber tests on Ticino sand. The ratio of M/q_c was found to vary between 3 and 22. Relative density can be assessed from the Baldi *et al* [1986] correlation.

3. $M_{dt} = f(D_r, q_c)$.

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: NC sands: $M_{dt} = q_c 10^{1.09 - 0.0075D_r \pm 0.2}$

OC sands: $M_{dt} = q_c 10^{1.78 - 0.0122D_r \pm 0.24}$

Required parameters: the cone resistance q_c and the relative density, D_r .

Applicability: NC and OC sands

Reliability: low to medium. For the first relation, $r^2 = 0.678$, $sd = 0.14$, $n = 9$ sands, and for the second, $r^2 = 0.734$, $sd = 0.18$, $n = 4$ sands.

Comments: both correlations were obtained from data from calibration chamber tests. They do not take into account the effect of stress level. The estimated value of M , corresponds to tangent drained constrained modulus, M_{dt} .

4. $M_{ds} = f(q_T, \sigma'_{vo})$.

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $M_{ds} = 8.25(q_T - \sigma'_{vo})/p_a$ $p_a \approx 100$ kPa

Required parameters: the cone resistance q_c , the effective vertical overburden pressure σ'_{vo} . $(q_T - \sigma'_{vo})$ ranges between 0 and 6000 kPa.

Applicability: clays

Reliability: medium.

Comments: the values of M_{ds} , drained secant constrained modulus, were obtained from 12 sites tested by the piezocone.

Estimation procedures:

after Bjerrum $S_u = \mu_{2D} * S_{uFV}$ where: $\mu_{2D} = 4.76/PI + 0.67 \pm 0.18$

after Azzouz $S_u = \mu_{3D} * S_{uFV}$ where: $\mu_{3D} = 3.86/PI + 0.62 \pm 0.15$

Required parameters: plasticity index PI ranging from 0 to 120.

Applicability: clays, mainly aged NC.

Reliability: medium

Comments: Bjerrum's correlation was produced using data from back analysis of embankment failures and although there is a considerable scatter, it is a very useful and valid correlation. Azzouz et al revised Bjerrum's correlation, using a 3-D method of slope analysis, including end effects. Their correlation is more conservative than Bjerrum's and seems to be preferable, for design purposes (embankments, cuttings).

2. $S_u = f(S_{uFV}, PI, \sigma'_{vo})$

Reference: Aas, Lacasse, Lunne and Hoeg, 1986.

Estimation procedure:

young clays: $PI_{young} = 850(S_{uFV}/\sigma'_{vo})^2 + 83(S_{uFV}/\sigma'_{vo}) - 4.7$

aged clays: $PI_{aged} = 710(S_{uFV}/\sigma'_{vo})^3 - 436.2(S_{uFV}/\sigma'_{vo})^2 + 173.4(S_{uFV}/\sigma'_{vo}) - 14.3$

The calculated values of PI are compared with the original PI value:

if $PI_{aged} \leq PI < PI_{young}$ then the clay is considered as NC

else if $PI > PI_{aged}$ then the clay is considered as OC

If clay is NC, then: $\mu_{NC} = 1/[1.5(S_{uFV}/\sigma'_{vo}) + 0.759]$

If clay is OC, then: $\mu_{OC} = 1/[2.58(S_{uFV}/\sigma'_{vo}) + 0.638]$

and $S_u = \mu * S_{uFV}$

Required parameters: effective overburden pressure, σ'_{vo} in kPa, field vane shear strength, S_{uFV} , in kPa and plasticity index, PI. The quotient S_{uFV} / σ'_{vo} ranges from 0 to 1 and PI ranges from 0 to 100%.

14. Correlations for estimation of the drained elastic modulus, E_d .

1. $E_{dt} = f(\sigma'_1, \sigma'_3, \phi'_{TC}, \text{soil type, grading})$

Reference: Duncan and Chang, 1970.

Estimation procedure:

$$E_{dt} = \kappa p_a (\sigma'_3/p_a)^n [1 - R_f(1 - \sin\phi'_{TC})(\sigma'_1 - \sigma'_3)/(2\sigma'_3 \sin\phi'_{TC})]^2$$

$$\kappa \approx 300(\phi'_{TC} - 25)/(45 - 25)$$

$$R_f = 0.7 \text{ for well-graded sand or gravel}$$

$$0.8 \text{ for poorly-graded sand or gravel}$$

$$n = 1/3 \text{ for gravel}$$

$$1/2 \text{ for sand}$$

$$2/3 \text{ for low plasticity silt}$$

Required parameters: effective major and minor stresses, σ'_1 and σ'_3 , in kPa, the effective friction angle in triaxial compression, ϕ'_{TC} , soil type, with values gravel, sand and silt and grading, with values well-graded and poorly-graded.

Applicability: all coarse soils

Reliability: low

Comments: the validation of the parameters κ , R_f and n was made from Trautmann and Kulhawy [1987] and Kulhawy *et al* [1983]. The estimated value of E_d corresponds to the drained tangent modulus, E_{dt} .

2. $E_d = f(N_{60})$

Reference: Kulhawy and Mayne, 1990.

$$E_d \approx 5p_a(N_{60}) \quad \text{sand with fines}$$

$$E_d \approx 10p_a(N_{60}) \quad \text{clean NC sands}$$

$$E_d \approx 15p_a(N_{60}) \quad \text{clean OC sands}$$

Required parameters: the number of blows from SPT test, corrected for test procedures, N_{60} , $p_a \approx 100$ kPa.

Applicability: sands, NC and OC, silty sands, clayey sands

Reliability: low

Comments: the values of E_d estimated from this correlation are only first order estimators.

15. Correlations for the estimation of shear modulus, G.

1. $G = f(e, \text{OCR}, S, \sigma'_m, \nu)$

Reference: Hardin, 1983.

Estimation procedure: $G_{\max} = p_a S \cdot \text{OCR}^M (\sigma'_m / p_a)^{0.5} / [2(1 + \nu)(0.3 + 0.7e^2)]$

$$p_a \approx 100 \text{ kPa} \quad \sigma'_m = 1/3 * (\sigma'_{vo} + 2\sigma'_{ho}) \quad \text{OCR}^M = 1$$

S ranges between 1200 and 1500 for clean sands

$$G = 5\% \text{ to } 10\% G_{\max}$$

Required parameters: void ratio, e , ranging between 0 and 2, overconsolidation ratio, OCR, stiffness coefficient, S and mean principal effective stress, σ'_m , in kPa.

Applicability: clean sands

Reliability: low.

Comments: the correlation was based on shear wave velocity measurements from the resonant column test. The shear strains in this test are in the range $10^{-4}\%$ to $10^{-1}\%$, where $G \approx G_{\max}$, whereas for shear strains in the order of 1% (static loading), $G = 5\%$ to $10\% G_{\max}$. $\text{OCR}^M = 1$ is an assumption, made by Hardin for convenience of the estimation.

2. $G = f(e, OCR, PI, \sigma'_m)$

Reference: Hardin and Drnevich, 1972.

Estimation procedure: $G_{max} = p_a [321OCR^M (\sigma'_m/p_a)^{0.5} (2.97 - e)^2 / (1 + e)]$

$$p_a \approx 100 \text{ kPa} \quad \sigma'_m = 1/3 * (\sigma'_{vo} + 2\sigma'_{ho})$$

$$M = 0.947857(PI/100) - 0.44642(PI/100)^2$$

$$G = 5\% \text{ to } 10\% G_{max}$$

Required parameters: void ratio, e , ranging between 0 and 2, overconsolidation ratio, OCR, plasticity index, PI and mean principal effective stress, σ'_m , in kPa.

Applicability: clays

Reliability: low

Comments: the correlation was based on shear wave velocity measurements from the resonant column test. The shear strains in this test are in the range $10^{-4}\%$ to $10^{-1}\%$, where $G \approx G_{max}$, whereas for shear strains in the order of 1% (static loading), $G = 5\%$ to $10\% G_{max}$.

2. $G = f(N_{SPT})$

Reference: Wroth, Randolph, Houlsby and Fahey, 1979.

Estimation procedure: $G_{max} = p_a 120 N_{SPT}^{0.77}$ $60 p_a N_{SPT}^{0.71} < G_{max} < 300 p_a N_{SPT}^{0.8}$

$$\text{and } p_a \approx 100 \text{ kPa} \quad G = 5\% \text{ to } 10\% G_{max}$$

Required parameters: the number of blows, N_{SPT} .

Applicability: clays

Reliability: low

Comments: the correlation was based on a number of relationships for G_{max} at dynamic strains versus N_{SPT} . The necessary correction for shear strain effect leads to $G = 5\%$ to $10\% G_{max}$.

16. Correlations for the estimation of compression index, C_c .

1. $C_c = f(LL)$

Reference: Skempton, 1944.

Terzaghi and Peck, 1967.

Estimation procedure:

a. Skempton $C_c = 0.007(LL - 10)$

b. Terzaghi and Peck $C_c = 0.009(LL - 10) \pm 30\%$

Required parameters: liquid limit, LL. In the second relation $0 < LL < 100$.

Applicability: for the second relation inorganic clays with sensitivity less than 4.

Reliability: low

Comments: both relations can only provide with an approximation of the compression index.

2. $C_c = f(PI, G_s)$, or $f(PI)$

Reference: Wroth and Wood, 1978.

Estimation procedure:

$$C_c = 0.5 PI G_s, \quad \text{and for: } G_s=2.7: \quad C_c = PI/74$$

Required parameters: plasticity index, PI, ranging between 0 and 100 and specific gravity, G_s .

Applicability: inorganic clays.

Reliability: low. For the simplified expression $r^2=66.3\%$ and $sd=0.16$.

Comments: no account is taken of the effect of sensitivity to the value of C_c .

17. Correlation for the estimation of recompression index, C_r .

1. $C_r = f(\text{PI})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $C_r = \text{PI} / 385$

Required parameters: plasticity index, PI, ranging between 0 and 100.

Applicability: inorganic clays.

Reliability: low. $r^2 = 44.8\%$ and $sd = 0.051$.

Comments: two assumptions underlying this correlation: critical state parameter $\Lambda = 0.8$ (for natural clays) and specific gravity of solid particles, $G_s = 2.7$.

Part 2: Corrections.

1. Interrelationships between effective stress friction angles from different tests.

1. $\phi'_{TE} = f(\phi'_{TC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: a. for coarse soils: $\phi'_{TE} = 1.12\phi'_{TC}$

b. for intact clays: $\phi'_{TE} = 1.22\phi'_{TC}$

Required parameters: effective angle of friction in triaxial compression, ϕ'_{TC} .

Applicability: first relation: all coarse soils.

second relation: intact clays.

Reliability: first relation: medium

second relation: low, $r^2 = 84.3\%$, $sd = 5.3^\circ$, $n = 70$ (55 intact clays).

2. $\phi'_{PSC} = f(\phi'_{TC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: a. for coarse soils: $\phi'_{PSC} = 1.12\phi'_{TC}$

b. for intact clays: $\phi'_{PSC} = 1.10\phi'_{TC}$

Required parameters: effective angle of friction in triaxial compression, ϕ'_{TC} .

Applicability: first relation: all coarse soils.

second relation: intact clays.

Reliability: first relation: medium

second relation: medium, $r^2 = 87.1\%$, $sd = 2.2^\circ$, $n = 18$ (12 intact clays).

3. $\phi'_{PSE} = f(\phi'_{TC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: a. for coarse soils: $\phi'_{PSE} = 1.25\phi'_{TC}$

b. for intact clays: $\phi'_{PSE} = 1.34\phi'_{TC}$

Required parameters: effective angle of friction in triaxial compression, ϕ'_{TC} .

Applicability: first relation: all coarse soils.

second relation: intact clays.

Reliability: low for both relations.

Comments: these are indirect correlations, and as such they involve double uncertainty. For the first relation $1.25 = 1.12$, (for PSC/TC) x 1.12 (for TE/TC), and similarly for the second $1.34 = 1.10$ x 1.22 .

4. $\phi'_{DS} = f(\phi'_{PSC}, \phi'_{CV})$, or $f(\phi'_{TC}, \phi'_{CV})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $\phi'_{DS} = \text{Arctan}(\tan\phi'_{PSC}\cos\phi'_{CV})$

$\phi'_{DS} = \text{Arctan}[\tan(1.12\phi'_{TC})\cos\phi'_{CV}]$

Required parameters: constant volume effective angle of friction and plain strain compression effective angle, ϕ'_{PSC} , or effective angle of friction in triaxial compression, ϕ'_{TC} .

Applicability: all coarse soils

Reliability: medium

2. Interrelationships between undrained shear strengths from different tests.

1. $S_{uPSC} = f(S_{uCK_0UC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $S_{uPSC} = 1.12 S_{uCK_0UC}$

Required parameters: undrained shear strength from CK_0UC tests.

Applicability: intact clays

Reliability: low, $r^2 = 72.4\%$, $sd = .026$, $n = 10$ (10 intact clays).

2. $S_{uPSE} = f(S_{uCK_0UE})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $S_{uPSE} = 1.29 S_{uCK_0UE}$,

Required parameters: undrained shear strength from Ck_0UE tests.

Applicability: intact clays

Reliability: low, $r^2 = 63.9\%$, $sd = 0.030$, $n = 6$ (6 intact clays).

3. $S_{uDSS} = f(S_{uCK_0UC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $S_{uDSS} = 0.67 S_{uCK_0UC}$

Required parameters: undrained shear strength from Ck_0UC tests.

Applicability: intact clays

Reliability: low, $r^2 = 40.0\%$, $sd = 0.030$, $n = 44$ (40 intact clays).

4. $S_{uCK_0UE} = f(S_{uCK_0UC})$

Reference: Kulhawy and Mayne, 1990.

Estimation procedure: $S_{uCK_0UE} = 0.487 S_{uCK_0UC}$

Required parameters: undrained shear strength from Ck_0UC tests.

Applicability: intact clays

Reliability: low.

3. Corrections for the estimation of the in-situ undrained shear strength, $S_{u \text{ field}}$

1. $S_u = f(S_{uFV}, PI)$

References: Bjerrum, 1972.

Azzouz, Baligh and Ladd, 1983.

Estimation procedures:

after Bjerrum $S_u = \mu_{2D} * S_{uFV}$ where: $\mu_{2D} = 4.76/PI + 0.67 \pm 0.18$

after Azzouz $S_u = \mu_{3D} * S_{uFV}$ where: $\mu_{3D} = 3.86/PI + 0.62 \pm 0.15$

Required parameters: plasticity index PI ranging from 0 to 120.

Applicability: clays, mainly aged NC.

Reliability: medium

Comments: Bjerrum's correlation was produced using data from back analysis of embankment failures and although there is a considerable scatter, it is a very useful and valid correlation. Azzouz et al revised Bjerrum's correlation, using a 3-D method of slope analysis, including end effects. Their correlation is more conservative than Bjerrum's and seems to be preferable, for design purposes (embankments, cuttings).

2. $S_u = f(S_{uFV}, PI, \sigma'_{v0})$

Reference: Aas, Lacasse, Lunne and Hoeg, 1986.

Estimation procedure:

young clays: $PI_{young} = 1495(S_{uFV}/\sigma'_{v0})^2 - 175.8(S_{uFV}/\sigma'_{v0}) + 16.44$

aged clays: $PI_{aged} = 1453.8(S_{uFV}/\sigma'_{v0})^3 - 1386.8(S_{uFV}/\sigma'_{v0})^2 + 544.6(S_{uFV}/\sigma'_{v0}) - 58.44$

The calculated values of PI are compared with the original PI value:

if $PI_{young} \leq PI < PI_{aged}$ then the clay is considered as NC

else if $PI \geq PI_{aged}$ then the clay is considered as OC

If clay is NC, then: $\mu_{NC} = 1/[1.5(S_{uFV}/\sigma'_{v0}) + 0.759]$

If clay is OC, then: $\mu_{OC} = 1/[2.58(S_{uFV}/\sigma'_{v0}) + 0.638]$

and $S_u = \mu * S_{uFV}$

Required parameters: effective overburden pressure, σ'_{v0} in kPa, field vane shear strength, S_{uFV} , in kPa and plasticity index, PI.

Applicability: clays both NC and OC, aged and young.

Reliability: medium to high for the range of application.

Comments: the correction factor μ is referenced to an "average laboratory strength", $S_{uLAB} = [S_{uTC} + S_{uDSS} + S_{uTE}] / 3$, measured on specimens anisotropically consolidated to the initial in-situ stresses (k_0 conditions). The correlation is in good agreement with measured values for excavations (base failures) and embankments.

References to Appendix A.

- Aas, G., Lacasse, S., Lunne, T. and Hoeg, K. (1986), "Use of in-situ tests for foundation design on clay", ASCE Spec. Conf. In-situ '86, Use of in-situ tests in Geotechnical Engineering, Blacksburg Virginia, USA, pp. 1-30.
- Azzouz, A. S., Baligh, M. M., and Ladd, C. C. (1983), "Corrected field vane strength for embankment design", Journal of Geotechnical Engineering, Vol. 109, pp. 730-734.
- Baldi, G., Bellotti, R., Ghionna, V. N., Jamiolkowski, M. and Pasqualini, E. (1986), "Interpretation of CPT's and CPTU's. 2nd part: drained penetration of sands.", Fourth International Geotechnical Seminar, Singapore, pp. 143-156.
- Bazaraa, A. R. S. S. (1967), "Use of the SPT test for estimating settlement on shallow foundations on sand.", PhD Thesis, University of Illinois.
- Been, K. and Jefferies, M. G. (1985), "A state parameter for sands", Geotechnique No 35, pp. 99-112.
- Bjerrum, L., and Simons, N. E. (1960), "Comparison of shear strength characteristics of N.C. clays", Proceedings of the Research Conference on Shear Strength of Cohesive Soils, Oslo, pp. 711-726.
- Bjerrum, L. (1972), "Embankments on soft ground", state-of-the-art report, Proc. of the American Society of Civil Engineers Special Conference on Performance of Earth and Earth-Supported Structures, Lafayette, Ind., USA, Vol. 2, pp. 1-54.

- Bolton, M. D. (1986), "The Strength and Dilatancy of Sands", *Geotechnique*, Vol. 36, No. 1, pp. 65-78.
- Chandler, R. J. (1988), "The In-situ measurement of the Undrained Shear Strength of Clays Using the Field Vane", *Vane Shear Strength Testing in Soils: Field and Laboratory Studies (STP 1014) ASTM*, Philadelphia, pp. 13-44
- Duncan, J. M. and Chang, C. Y. (1970), "Nonlinear Analysis of Stress and Strain in Soils", *Journal of the Soil Mechanics and Foundation Divisions, ASCE*, Vol. 96, No. SM5, Sept. 1970, pp. 1629-1653.
- Durgunoglu, H. T., and Mitchell, J. K. (1975), "*Static penetration resistance of soils.*", *Proceedings of Conference on In-situ Measurement of Soil Properties*, Am. Soc. Civ. Engrs. Raleigh, N. Carolina, June 1975, Vol. 1, pp. 151-188.
- Gibbs, H. J. and Holtz, W. G., (1957), "Research on determining the density of sands by spoon penetration testing", *IV ICSMFE, London* Vol. 1, pp. 35-39.
- Gibson, R. E. (1953), "Experimental determination of the true cohesion and true angle of internal friction in clays", *Proceedings of the 3rd International Conference on Soil Mechanics and Foundation Engineering, Zurich*, pp. 126-130.
- Giuliani, F. and Giuliani F. L. N. (1982), "New analytical correlations between SPT, overburden pressure and relative density", *Proceedings of the Second European Symposium of Penetration testing, Amsterdam, 24 - 27 May 1982.*

- Hardin, B. O. (1983), "Modulus of Subgrade Reaction: New Perspective", *Journal Geotechnical Engineering, ASCE*, Vol. 109, No. 12, Dec. 1983, pp. 1591-1596.
- Hardin, B. O. and Drnevich, V. P. (1972), "Shear Modulus and Damping in Soils: Design Equations and Curves", *Journal of the Soil Mechanics and Foundation Divisions, ASCE*, Vol. 98, No. SM7, July 1972, pp. 667-692.
- Houlsby, G. T. and Wroth, C. P. (1989), "The influence of soil stiffness and lateral stress on the results of in-situ soil test", *12th Int. Conf. of Soil Mechanics and Foundation Engineering. Session 2. Rio de Janeiro, Brazil.*
- Jáky, J. (1944), "The Coefficient of Earth pressure at rest", *Journal of the Society of Hungarian Architects and Engineers, Budapest, Oct. 1944, pp. 355-358.*
- Jamiolkowski, M., Ghionna, V. N. and Lancellotta, R., (1988), "New correlations of penetration tests for design practice", *International Symposium of Penetration Testing ISOPT-1. Orlando, USA. Proc., Vol. 1, pp 263-296.*
- Jamiolkowski, M., Ladd, C. C., Germaine, J. and Lancellotta, R. (1985), "New developments in field and laboratory testing of soils", *11th ICSMFE.*
- Koerner, R. M. (1970), "Effect of Particle Characteristics on Soil Strength", *Journal of the Soil Mechanics and Foundations Division, ASCE*, Vol. 96, No. SM4, July 1970, pp. 1221-1234.
- Kulhawy, F. H. and Mayne, P. W. (1990), "Manual on Estimating Soil Properties for Foundation Design", *Report. EL-6800, Electric Power Res. Inst., Palo Alto.*

- Kulhawy, F. H., Jackson, G. S., and Mayne, P. W. (1989), "First Order Estimation of k_0 in Sands and Clays", Foundation Engineering Current Principles and Practices, (ed. F. H. Kulhawy), ASCE, New York, pp. 121-134.
- Kulhawy, F. H., Trautmann, C. H., Beech, J. F., O'Rourke, T. D., McGuire, W., Wood, W. A. and Capano, C. (1983), "Transmission Line Structure Foundations for Uplift-Compression Loading.", Report EL-2870, Electric Power Res. Inst., Palo Alto.
- Lacasse, S. and Lunne, T. (1988), "Calibration of dilatometer correlations.", Proc. of International Symposium on Penetration Testing ISOPT-1, Orlando, Florida, U.S.A., Vol. 1, pp.539-548.
- Larsson, R. (1977), "Basic Behaviour of Scandinavian Soft Clays", Report 4, Swedish Geotechnical Institute, Linkoping, pp. 125.
- Leroueil, S., Tavenas, F. & Le Bihan, J. P. (1983), "Propriétés caractéristiques des argiles de l' est du Canada" Canadian Geotechnical Journal, Vol. 20 (4), pp. 681-705.
- Lunne, T., Powell, J. J. M., Hauge, E. A., Uglow, I. M. and Mokkelbost, K. H. (1990), "Correlation of dilatometer readings to lateral stress", Speciality Session on Measurement of Lateral Stress, 69th Annual Meeting of the Transportation Research Board, Washington DC, USA.
- Lunne, T. and Christoffersen, H. P. (1983), "Interpretation of some cone penetrometer data for offshore sands", Norwegian Geotechnical Institute, Oslo, Report 52108-15.

- Marchetti, S. (1980), "In-situ tests by flat dilatometer" ASCE. Journal of Geotechnical Engineering. Vol. 106, No. GT3, pp. 299-321.
- Marcuson, W. F., III and Bieganousky, W. A.(1977), "SPT and Relative Density in Coarse Sands", Journal of the Geotechnical Engineering Division, ASCE, Vol. 103, No. GT11, Nov. 1977, pp. 1295-1309.
- Mayne, P. W. (1988), "Determining OCR in Clays from laboratory strength", Journal of Geotechnical Engineering, ASCE, Vol. 114, No. 1, Jan. 1988, pp. 76-92.
- Mayne, P. W. and Kulhawy, F. H. (1982), " k_0 - OCR Relationships in Soil", Journal of the Geotechnical Engineering Division, ASCE, Vol. 108, No. GT6, pp.851-872.
- Mayne, P. W. and Mitchell, J. K. (1988), "Profiling of Overconsolidation Ratio in Clays by Field Vane", Canadian Geotechnical Journal, Vol. 25, No. 1, Feb. 1988, pp. 150-157.
- Mesri, G. (1989), "A Re-evaluation of the $s_{u(mob)} \approx 0.22 \sigma'_p$ using Laboratory Shear Tests", Canadian Geotechnical Journal, Vol. 26, No 1, pp. 162-164.
- Meyerhof, G. G. (1963), "Some recent research on bearing capacity of foundations", Canadian Geotechnical Journal, Vol. 1, pp. 16-26.
- Mitchell, J. K. (1976), "Fundamentals of Soil Behavior", John Wiley and Sons, New York, pp. 422.

Peck, R. B., Hanson, W. E. and Thorburn, T. H. (1974), "Foundation Engineering", John Wiley, London, pp. 514.

Robertson, P. K. and Campanella, R. G. (1983), "Interpretation of Cone Penetration Tests. Part I: Sand", Canadian Geotechnical Journal, Vol. 20, No. 4, Nov. 1983, pp. 718-733

Schmertmann, J. H. (1975), "Measurement of In-Situ Shear Strength", Proceedings, ASCE Speciality conference on In-Situ measurement of Soil Properties, Vol. 2, Raleigh, pp. 57-138.

Schmertmann, J. H. (1978), "Guidelines for cone penetration test: performance and design", U.S. Dept. of Transp., Fed. Highways Admin., Offices of Research and Development, Washington DC, Report FHWA-TS-78-209, July 1978.

Schmertmann, J. H. (1983). Revised procedure for calculating K_o and OCR from DMT's with $I_D > 1.2$ and which incorporates the penetration force measurement to permit calculating the plane strain friction angle. DMT Digest No. 1. GPE INC, Gainesville, Florida, USA.

Skempton, A. W. (1944), "Notes on the Compressibility of Clays.", Quart. Journ. Geol. Soc., 100, pp. 119-135.

Skempton, A. W. and Northey, R. D. (1952), "The sensitivity of clays", Geotechnique, No. 3, pp. 30-53

Skempton, A.W. and Bjerrum, L. (1957), "A contribution to the settlement analysis of foundations on clay" Geotechnique, No. 7, pp. 168-178.

- Skempton, A. W. (1986), "Standard Penetration Test Procedures and the Effects in Sands of Overburden Pressure, Relative density, Particle Size, Ageing, and Overconsolidation", *Geotechnique*, Vol. 36, No. 3, Sept. 1986, pp. 425-447.
- Stas, C. V. and Kulhawy, F. H. (1984), "Critical Evaluation of Design Methods for Foundations Under axial Uplift and Compression Loading", Report EL-3771, Electric Power Research Institute, Palo Alto, Nov 1984, 198 p.
- Stroud, M. A. (1988), "The Standard Penetration Test - Its application and interpretation", Proceedings of the the Institution of Civil Engineers Geotechnology Conference, Birmingham, 6-8 July 1988, pp. 29-51.
- Tavenas, F. and Leroueil, S. (1987), "State-of-the-Art on Laboratory and In-Situ Stress-Strain-Time Behaviour of Soft Clays Proceedings, International Symposium on Geotechnical Engineering of Soft Soils, Mexico City, pp. 1-46.
- Terzaghi, K. and Peck, R. B., (1967), "Soil Mechanics in Engineering Practice", 2nd Ed., John Wiley and Sons, New York, 729 p.
- Thorne, C. P. (1984), "Strength assessment and stability analysis for fissured clays", *Geotechnique* 34, No 3.
- Trautmann, C. H. and Kulhawy, F. H. (1987), "CUFAD - A Computer Program for Compression and Uplift Foundation Analysis and Design", Report EL-4540-CCM, Vol. 16, Electric Power Research Institute, Palo Alto, Oct. 1987, 148 p.

U.S. Navy (1982), "Design Manual: Soil Mechanics, Foundations and Earth Structures", NAVFAC, U.S. Naval Publications and Forms, Alexandria, pp. 355.

Wood, D. M. (1983), "Index Properties and Critical State Soil Mechanics", Proceedings, Symposium on recent Developments in Laboratory and Field Tests and Analysis of Geotechnical Problems, Bangkok, Dec. 1983, pp. 301-309.

Wroth, C. P., Randolph, M. F., Houlsby, G. T. and Fahey, M. (1979), "A Review of the Engineering Properties of Soils with Particular emphasis to the Shear Modulus", CUED/D-SOILS TR 75, University of Cambridge, 79 p.

Wroth, C. P. and Wood, D. M. (1978), "The Correlation of Index Properties with Some Basic Engineering Properties of Soils", Canadian Geotechnical Journal, Vol. 15, No. 2, May 1978, pp. 137-145.

Wroth, C. P. and Houlsby, G. T. (1985), "Soil mechanics - property characterisation and analysis procedures", 11th ICSMFE, Vol. 1, San Francisco, pp. 1-55.

Appendix B

Typical values for ground properties.

Soils

- Peak effective angle of friction, ϕ' , and dry density, d_{dry} for normally consolidated inorganic gravels, sands and non-plastic silts.

Reference: U.S. Navy (1982), Design Manual: "Soil Mechanics, Foundations and Earth Structures", NAVFAC, U.S. Naval Publications and Forms, Alexandria, 355 p.

Soil type	Effective angle of friction ϕ' (in degrees)	Dry density, d_{drv} (Mg/m ³)
gravel	(27, 36, 46)	(1.69, 2.02, 2.36)
well-graded gravel	(28, 37,46)	(1.82, 2.09, 2.36)
very loose well-graded gravel	(28, 29, 30)	(1.82 ,1.86 ,1.95)
loose well-graded gravel	(29.5, 31, 33)	(1.87 ,1.95 ,2.03)
medium dense well-graded gravel	(32.5, 35.5, 38.5)	(1.95, 2.06, 2.18)
dense well-graded gravel	(37, 39.5, 42)	(2.00, 2.13, 2.27)
very dense well-graded gravel	(41, 43.5, 46)	(2.17, 2.26, 2.36)
poorly-graded gravel	(27, 35.5, 44)	(1.69, 1.97, 2.25)
very loose poorly-graded gravel	(27, 28, 29.5)	(1.69, 1.78, 1.87)
loose poorly-graded gravel	(29, 30.8, 32.6)	(1.74, 1.84, 1.95)
medium dense poorly-graded gravel	(31.8, 34.5, 37.6)	(1.81, 1.95, 2.09)
dense poorly-graded gravel	(36, 38.5, 41)	(1.93, 2.07, 2.18)
very dense poorly-graded gravel	(39, 41.5, 44)	(2.00, 2.12, 2.25)
sand	(26, 34, 42)	(1.4, 1.76, 2.07)
well-graded sand	(27, 34, 42)	(1.57, 1.82, 2.07)
very loose well-graded sand	(27, 28, 29)	(1.57, 1.65, 1.73)
loose well-graded sand	(28.6, 30, 32)	(1.61, 1.71, 1.81)

Soil type	Effective angle of friction ϕ' (in degrees)	Dry density, d_{drv} (Mg/m ³)
medium dense well-graded sand	(31, 33.5, 36)	(1.69, 1.81, 1.93)
dense well-graded sand	(35, 37, 39)	(1.79, 1.89, 2.00)
very dense well-graded sand	(38, 40, 42)	(1.87, 1.97, 2.07)
poorly-graded sand	(26, 33.5, 41)	(1.4, 1.7, 2.00)
very loose poorly-graded sand	(26, 27.5, 29)	(1.4, 1.54, 1.68)
loose poorly-graded sand	(28, 29.5, 31.5)	(1.44, 1.59, 1.74)
medium dense poorly-graded sand	(30, 33, 36)	(1.51, 1.68, 1.86)
dense poorly-graded sand	(33.5, 36, 38.5)	(1.61, 1.77, 1.93)
very dense poorly-graded sand	(36, 38.5, 41)	(1.68, 1.84, 2.00)
non-plastic silty sand	(26, 33.5, 41)	(1.4, 1.7, 2.00)
very loose silty sand	(26, 27.5, 29)	(1.4, 1.54, 1.68)
loose silty sand	(28, 29.5, 31.5)	(1.44, 1.59, 1.74)
medium dense silty sand	(30, 33, 36)	(1.51, 1.68, 1.86)
dense silty sand	(33.5, 36, 38.5)	(1.61, 1.77, 1.93)
very dense silty sand	(36, 38.5, 41)	(1.68, 1.84, 2.00)
non-plastic silt	(26, 31, 37)	(1.26, 1.45, 1.65)
very loose silt	(26, 26.8, 27.6)	(1.26, 1.32, 1.38)
loose silt	(27, 28.5, 30)	(1.30, 1.37, 1.44)
medium dense silt	(29, 31, 33)	(1.36, 1.45, 1.54)
dense silt	(32, 33.5, 35)	(1.45, 1.52, 1.60)
very dense silt	(34, 35.5, 37)	(1.52, 1.58, 1.65)

- Peak effective angle of friction (ϕ'), natural dry and bulk density (ρ_d , ρ_{bulk}) of silts and clays.

Reference: Carter, M. and Bentley (1991), S. P., "Correlations of soil properties",
Pentech Press, London.

Soil type	ϕ' (in degrees)	d_{bulk} (Mg/m ³)	d_{drv} (Mg/m ³)
low-plasticity silt	(u, u, 32)	n/a	n/a
high-plasticity silt	(u, u, 25)	n/a	n/a
low plasticity clayey silt	(u, u, 32)	n/a	n/a
high plasticity clayey silt	(u, u, 25)	n/a	n/a
sandy clay	(u, u, 31)	n/a	n/a
low plasticity clay	(26, 31, 36)	n/a	n/a
high plasticity clay	(19, 23.5, 28)	n/a	n/a
very soft clays	n/a	(1.6, 1.65, 1.7)	(0.9, 1.0, 1.1)
soft clays	n/a	(1.7, 1.8, 1.9)	(1.1, 1.25, 1.4)
firm clays	n/a	(1.8, 2.0, 2.2)	(1.3, 1.6, 1.9)
stiff clays	n/a	(2.0, 2.2, 2.4)	(1.7, 1.95, 2.2)

- Compacted density (d_{comp}) and optimum moisture contents (omm) of soils.

Reference: Carter, M. and Bentley (1991), S. P., "Correlations of soil properties",
Pentech Press, London.

Soil type	d_{bulk} (Mg/m ³)	omm (%)
clean gravel or sandy gravel	(1.85, 2.0, 2.15)	(8, 11, 14)
well-graded	(2.0, 2.07, 2.15)	(8, 9.5, 11)
poorly-graded	(1.85, 1.92, 2.0)	(11, 12.5, 14)

Soil type	d_{bulk} (Mg/m ³)	omm (%)
silty gravel	(1.9, 2.02, 2.15)	(8, 10, 12)
clayey gravel	(1.85, 1.92, 2.0)	(9, 11.5, 14)
clean sand	(1.6, 1.85, 2.10)	(9, 14.5, 21)
well-graded	(1.75, 1.92, 2.1)	(9, 12.5, 16)
poorly-graded	(1.6, 1.75, 1.9)	(12, 16.5, 21)
silty sand	(1.75, 1.87, 2.0)	(11, 13.5, 16)
clayey sand	(1.7, 1.85, 2.0)	(11, 15, 19)
inorganic silts	(1.1, 1.5, 1.9)	(12, 26, 40)
low-plasticity	(1.5, 1.7, 1.9)	(12, 17, 24)
high-plasticity	(1.1, 1.3, 1.5)	(24, 32, 40)
inorganic clays	(1.3, 1.6, 1.9)	(12, 24, 36)
low-plasticity	(1.5, 1.7, 1.9)	(12, 17, 24)
high-plasticity	(1.3, 1.5, 1.7)	(19, 27.5, 36)
low plasticity organic silt	(1.3, 1.45, 1.6)	(21, 17, 33)
high-plasticity organic clay	(1.05, 1.32, 1.6)	(21, 33, 45)

- Compressibility and coefficient of volume compressibility, m_v of clays.

Reference: Weltman, A. J. Head, J. M. (1983), "Site Investigation Manual",
Construction Industry Research and Information Association, London,
113 p.

Soil type	Compressibility	m_v (m ² /MN)
inorganic clay	very low to high	(0.05, 0.775, 1.5)
very heavily overconsolidated	very low	(u, u, 0.05)

Soil type	Compressibility	m_v (m^2/MN)
heavily overconsolidated	low	(0.05, 0.075, 0.1)
lightly overconsolidated	medium	(0.1, 0.2, 0.3)
normally consolidated	high	(0.3, 0.9, 1.5)
highly organic clays and peats	very high	(1.5, u, u)

- Compressibility index, C_c of cohesive soils.

Reference: Holtz, R. D. and Kovacs, W. D. (1981), "An Introduction to Geotechnical Engineering.", Prentice-Hall, New Jersey, 733 p.

Soil type	C_c
Normally consolidated medium sensitive clays	(0.2, 0.35, 0.5)
Organic clays	(4, u, u)
Peats	(10, 12.5, 15)
Organic silt and organic clayey silt	(1.5, 2.75, 4)

Rocks

- Dry densities, d_{dry} , of some typical rocks.

References: Davis, S. N. and DeWiest, R. J. M. (1966), "Hydrogeology", ed. Wiley J., New York.

Clark, S. P. (1966) (ed), "Handbook of Physical Constants.", Geological Society of America, Memoir 97.

Rock	d_{dry} (Mg/m ³)	Rock	d_{dry} (Mg/m ³)
Granite	2.65	Coal	0.7-2.0
Diorite	2.85	Dense Limestone	2.7
Gabbro	3.0	Marble	2.75
Gypsum	2.3	Amphibolite	2.99
Rock salt	2.1	Rhyolite	2.37
Basalt	2.77		

- Hydraulic conductivities, k , of some typical rocks.

References: Davis, S. N. and DeWiest, R. J. M. (1966), "Hydrogeology", Wiley, New York.

Brace, W. F., Walsh, J. B. and Frangos, W. T. (1978), "Permeability of granite under high pressure", J. Geoph. Res. 73, pp. 2225-2236.

Rock	k Lab, (cm/s)	k Field, (cm/s)
Sandstone	$(3 \times 10^{-3}, 8 \times 10^{-8})$	$(10^{-3}, 3 \times 10^{-8})$
Rock	k Lab, (cm/s)	k Field, (cm/s)

Shale	$(10^{-9}, 5 \times 10^{-13})$	$(10^{-8}, 10^{-11})$
Limestone, Dolomite	$(10^{-5}, 10^{-13})$	$(10^{-3}, 10^{-7})$
Basalt	10^{-12}	$(10^{-2}, 10^{-7})$
Granite	$(10^{-7}, 10^{-11})$	$(10^{-4}, 10^{-9})$
Schist, intact	10^{-8}	2×10^{-7}
Schist, fissured	$(10^{-4}, 3 \times 10^{-4})$	n/a

- Point load strength index of some typical rocks.

References: Broch, E. and Franklin, J. A. (1972), "The point load strength test",
Int. J. of Rock Mechanics and Mining Science 9, pp. 669-697

Rock	Point load strength index, (MPa)
Tertiary sandstone and claystone	(0.05, 1)
Coal	(0.2, 2)
Limestone	(0.25, 8)
Mudstone, Shale	(0.2, 8)
Volcanic flow rocks	(3.0, 15)
Dolomite	(6.0, 11)

Appendix C

The ProTalk code.

GREP1.ptk

```
/* This file contains the functions and methods for browsing the ground_representation object base
and displaying "typical" values of ground properties. */
```

```
#include <prk/math.pth>
```

```
#include <prk/lib.pth>
```

```
/*-----*/
```

```
function menu()
```

```
{
```

```
  bound inputs;
```

```
  /* Puts the "search" dialog box on screen */
```

```
  parpb("Reset ", slb1@, Ground@);
```

```
  slb2.SelectionItems = Null;
```

```
  slb2.MaxNumOfLines = 1;
```

```
  std.Values = Null;
```

```
  search.PutOnScreen!();
```

```
}
```

```
/*-----*/
```

```
method slb1.React! (?new_value, ?old_value)
```

```
{
```

```
  bound inputs;
```

```
  /* The method attached to the list box displaying ground types in the "search" dialog box*/
```

```
  slb2.SelectionItems = Null;
```

```
  slb2.MaxNumOfLines = 1;
```

```
  ?lpar = all slb1.SelectionItems;
```

```
  select {
```

```
    case: {IsString(?old_value);
```

```
      not {IsString(?new_value);};
```

```
      ?x1 = find direct subclassof ConvertToSymbol(?old_value);
```

```
      or {ObjectModule(?x1) == ground_rep@;
```

```
          ObjectModule(?x1) == expand@;}}
```

```
    {slb1.SelectionItems = Null;
```

```
      for {find ?x == direct subclassof ConvertToSymbol(?old_value);
```

```
          or {ObjectModule(?x) == ground_rep@;
```

```
              ObjectModule(?x) == expand@;}}
```

```
      do slb1.SelectionItems += ConvertToString(?x);
```

```
      ?lpar = DeleteListElmt(?old_value, ?lpar);
```

```
      ?lpar = AppendLists(?lpar, `(?old_value));
```

```
      slb1.UserData = AppendLists(slb1.UserData, `(?lpar));}
```

```
    case: {IsString(?new_value);
```

```
      . find1 scb.Values == ?;}
```

```
    {?new_value = ConvertToSymbol(?new_value);
```

```
      for {find ?new_value@.?sl == ?;
```

```
          IsList(?new_value@.?sl);
```

```
          IsFacet(?new_value@, ?sl, `Required parameters");}
```

```
    do {slb2.SelectionItems += ?sl@.name;}
```

```
    find ?num = count slb2.SelectionItems;
```

```
    select {
```

```
      case: {?num < 4;
```

```

        ?num > 0;}
        slb2.MaxNumOfLines = ?num;
        case: ?num == 0;
        slb2.MaxNumOfLines = 1;
        otherwise: slb2.MaxNumOfLines = 4;}
        est2cr.React!("Dismiss");}

    otherwise: est2cr.React!("Dismiss");}
}
/*-----*/
method spb1.React! (?button)
{
    bound inputs;
    /* The method attached to the, "Back", "Forward" and
       "Reset" push buttons in the "search" dialog box. */

    parpb (?button, slb1@, Ground@);
}
/*-----*/
method spb2.React! (?button)
{
    bound inputs;
    /* The method attached to the, "New ground type", "Delete
       ground type" push buttons in the "search" dialog box. */

    select {
        case: ?button == "New ground type";
        {if IsString(slb1.Values);
         {ngtlb.SelectionItems = slb1.Values;}
         else {ngtlb.SelectionItems = Null;}
         ngt.PutOnScreen!();}

        case: ?button == "Delete ground type";
        delgt();}
}
/*-----*/
method ngtlb.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the list box for removing parent
       objects in the "New ground type" dialog box*/

    if IsString(?new_value);
    {ngtlb.SelectionItems -= ?new_value;}
    else {;}
}
/*-----*/
method ngtpb.React! (?value)
{
    bound inputs;
    /* The method attached to the "Add parent" push button in the "New ground type" dialog box*/

    if IsString(slb1.Values);
    {ngtlb.SelectionItems += slb1.Values;}
    else {;}
}
/*-----*/
method ngtcr.React! (?button)

```

```

{
bound inputs;
/* The method attached to the command row of the "New ground type" dialog box*/

select {

case: ?button == "Create";
  {?ml = `();
  for ?x inlist ModuleClasses(expand);
  do collect ConvertToSymbol(?x) into ?ml;
  ?pl = all ngtlb.SelectionItems;
  if {IsString(ngteb.Values);
    ListLength(FindListElmt(?ml, ConvertToSymbol(ngteb.Values))) == 0;
    ListLength(?pl) > 0;}
  {MakeClass(ConvertToSymbol(ngteb.Values), expand@, ?pl, `());
  SaveApp(ground_rep);
  ngt.TakeOffScreen!();}
  else {;}}

case: ?button == "Cancel";
  ngt.TakeOffScreen!();}
}
/*-----*/
function delgt()
{
bound inputs;
/* This function checks if the selected ground type can be deleted. */

select {

case: not IsString(slbl.Values);
  {msgtd.Values = "\nSelect first the ground type to delete
and then press Delete ground type.\n";
  Msgwin.PutOnScreenAndWait!();}

case: {IsString(slbl.Values);
  ObjectModule(ConvertToSymbol(slbl.Values)) == ground_rep@;}
  {msgtd.Values = "The ground type selected cannot be deleted
because it is system supplied.";
  msgcr.ButtonLabels = "Dismiss";
  Msgwin.PutOnScreenAndWait!();}

case: {IsString(slbl.Values);
  ObjectModule(ConvertToSymbol(slbl.Values)) == expand@;}
  {?ob = ConvertToSymbol(slbl.Values);
  ?chil1 = 0;
  ?chil2 = 0;
  for {find ?obt == subclassof ?ob;
  ObjectModule(?obt) == expand@;}
  do ?chil1 = ?chil1 + 1;
  for {find ?obt == subclassof ?ob;
  ObjectModule(?obt) == est@;}
  do ?chil2 = ?chil2 + 1;
  select {
  case: ?chil1 > 0;
    {msgtd.Values = "\nThe ground type selected to be deleted is a parent
of other ground\ntypes lower in the hierarchy. Therefore it
should not be deleted,\nbefore examinining its descendants

```

```

                                for data that might exist,\nwhich will be lost. \nIf you still
                                want to delete it then delete first all it's descendants.\n";
msgcr.ButtonLabels = "Dismiss";
Msgwin.PutOnScreenAndWait!();
case: {?chil1 == 0;
      or {find1 ?ob.?sl == ?;
          ?chil2 > 0;}}
      {msgtd.Values = "\nThe ground type selected to be deleted has got implemented
                        data\nwhich will be lost if the ground type is deleted.\n
                        For more information on the excisting data press
                        Cancel and\nthen, the Parameter search option.\nOtherwise
                        press Delete ground type.\n";
        msgcr.ButtonLabels = "Cancel";
        msgcr.ButtonLabels += "Delete";
        Msgwin.PutOnScreenAndWait!();}
case: {?chil1 == 0;
      not find1 ?ob.?sl == ?;
      ?chil2 == 0;}
      delgtconf();}}
}
/*-----*/
function delgtconf ()
{
  bound inputs;
  /* This function deletes any existing subclasses of the ground type to be deleted. */

  ?pl = all subclassof ConvertToSymbol(slb1.Values);
  for {?x inlist ?pl;
      ListLength(all subclassof ?x) == 0;}
  do {DeleteObject(?x);}
  delgtcheck();
}
/*-----*/
function delgtcheck ()
{
  bound inputs;
  /* This function puts a message dialog box on screen, which
     confirms the deletion of the selected ground type. */

  ?val = ConvertToSymbol(slb1.Values);
  if ListLength(all subclassof ?val@) > 0;
  {killit();}
  else {spb1.`React!`("Back  ");
        msgtd.Values = AppendStrings("The ground type ", ConvertToString(?val),
                                     "\nhas been deleted");

        DeleteObject(?val@);
        Msgwin.PutOnScreen!();}
}
/*-----*/
method scb.React! (?moused_item, ?selected_values)
{
  bound inputs;
  /* The method attached to the "Display Implemented parameters for
     the selected ground type" push button in the "search" dialog box*/

  if find1 scb.Values == ?;
  {std.Values = "Parameters list. Select to examine";
   ?new_value = find1 slb1.Values;

```

```

?old_value = "";
slb1.React!(?new_value, ?old_value);}
else {std.Values = Null;
      slb2.MaxNumOfLines = 1;
      slb2.SelectionItems = Null;}
}
/*-----*/
method slb2.React! (?new_value, ?old_value)
{
  bound inputs;
  /* Puts on screen the dialog box for displaying the typical values
     of the selected parameter for the selected ground type. */

  if {IsString(?new_value);
      IsString(slb1.Values);
      ?soil = ConvertToSymbol(slb1.Values);}
  {?num = 1;
   find ?sl.name == ?new_value;
   if IsFacet(?sl, format, units);
   {?units = ?sl.format..units;}
   else {?units = "";}
   ?sl = ConvertToSymbol(?sl);
   est2cr.React!("Dismiss");

  select {

    case: IsList(?soil.?sl.."Required parameters");
      {for ?x inlist ?soil.?sl.."Required parameters";
       do {?lb_name = ConvertToSymbol(AppendStrings("est2lb", ConvertToString(?num)));
          MakeDialogBoxControl(ListBox, groundrepUI@, ?lb_name);
          ?lb_name.Identifier = ?lb_name;
          ?lb_name.Title = ConvertToString(?x);
          ?lb_name.DialogBox = estimate2@;
          for ?val inlist ?soil.?sl..?x;
          do {?lb_name.SelectionItems += ?val;}
          if ListLength(?soil.?sl..?x) > 6;
          {?lb_name.MaxNumOfLines = 6;}
          else {?lb_name.MaxNumOfLines = ListLength(?soil.?sl..?x);}
          estimate2@.Contents += ?lb_name@;
          ?lb_name.PositionY = estimate2@.UserData;
          estimate2@.UserData= 18*?lb_name.MaxNumOfLines + estimate2@.UserData + 25;
          ?num = ?num + 1;
          est2cr.UserData += ?lb_name@;}}

    case: {not IsList(?soil.?sl.."Required parameters");
          IsList(?soil.?sl);}
      {;}

    otherwise: fail;}

  MakeDialogBoxControl(TextDisplay, groundrepUI@, est2tdtil);
  est2tdtil.Identifier = est2tdtil;
  est2tdtil.Title = ?new_value;
  if ?units != "";
  {est2tdtil.Values = AppendStrings("(", ?units, ")");}
  else {;}
  est2tdtil.PositionY = estimate2@.UserData;
  estimate2@.UserData= estimate2@.UserData + 30;

```

```

est2tdtil.PositionX = 0;
est2tdtil.DialogBox = estimate2@;
estimate2@.Contents += est2tdtil@;
est2td("min value      :", "min");
est2td("average value  :", "av");
est2td("max value      :", "max");
est2cr.UserData..slot = ?sl;
estimate2@.PutOnScreen!();}
else {;}
}
/*-----*/
function est2td(?str, ?nam)
{
    bound inputs;

    /* Creates the Text Display objects for the min., max. and
       mean values of the parameter that is being estimated. */

    ?td = ConvertToSymbol(AppendStrings("est2td", ?nam));
    MakeDialogBoxControl(TextDisplay, groundrepUI@, ?td);
    ?td.Identifier = ?td;
    ?td.Title = ?str;
    ?td.PositionY = estimate2@.UserData;
    estimate2@.UserData = estimate2@.UserData + 30;
    ?td.PositionX = 0;
    ?td.DialogBox = estimate2@;
    estimate2@.Contents += ?td@;
}
/*-----*/
method est2cr.React! (?button)
{
    bound inputs;
    /* Retrieves the typical values of the estimated parameter */

    ?lblest = Null;
    select {

        case: ?button == "Estimate";
        {?lblest = all ?self.UserData;
         est2tdtil.UserData = Null;
         ?str = slb1.Values;
         for ?i from 0 to ListLength(?lblest)-1;
         do {?x = ListNth(?lblest, ListLength(?lblest)-1-?i);
            select {
                case: ?x.Values == "unknown";
                {?st = "";}
                case: find ?x.Values == ?;
                {?st = AppendStrings(?x.Values, " ");}
                otherwise: ?st = "";}
            ?str = AppendStrings(?st, ?str);}
         ?ob = ConvertToSymbol(?str);
         est2tdtil.UserData = FindObject(?ob);
         ?sl = est2cr.UserData..slot;
         if {est2tdtil.UserData != Null;
            find ?ob.?sl == ?;
            IsList(?ob.?sl);}
         {est2tdmin@.Values = ListFirst(?ob.?sl);
          est2tdmax@.Values = ListNth(?ob.?sl, 2);

```

```

    est2tdav@.Values = ListNth(?ob.?sl, 1);}
else { est2tdmin@.Values = u;
      est2tdmax@.Values = u;
      est2tdav@.Values = u;}}

case: ?button == "Dismiss";
{est2cr.UserData = Null;
for ?death = find estimate2@.Contents;
do {if ?death != est2cr@;
    {estimate2@.Contents -== ?death@;
    DeleteObject(?death@);}
    else {;}}
est2cr.UserData.slot = Null;
estimate2@.UserData = 10;
estimate2@.TakeOffScreen!();}}
}
/*-----*/
method scr.React! (?button)
{
  bound inputs;
  /* Performs the actions incorporated in the "search" dialog box command row. */

  select {

  case: ?button == "Parameter\n search";
  {for find ?win.IsOnScreen == True;
  do ?win.TakeOffScreen!();
  par();}

  case: {?button == " Update\nParameters";
  IsString(slb1.Values);
  ?ob = ConvertToSymbol(slb1.Values);
  find1 ?ob@.?sl == ?;
  IsFacet(?ob@, ?sl, "Required parameters");}
  {upsr0.Contents -== upsr0pb1 @;
  upsr0.Contents -== upsr0pb2 @;
  upsr0lb.SelectionItems = Null;
  for ?sl inlist ObjectSlots(?ob);
  do { if {IsList(?ob@.?sl);
        IsFacet(?ob@, ?sl, "Required parameters");}
      {upsr0lb.SelectionItems += ?sl.name;}}
  upsrcr0.ButtonLabels = "Dismiss";
  upsrcr0.ButtonLabels += " Add\nParameter";
  upsrcr0.ButtonLabels += " Update";
  upsr0.PutOnScreen!();}

  case: {?button == " Update\nParameters";
  IsString(slb1.Values);}
  {parpb ("Reset ", upsr0lb@, Parameters@);
  upsr0.Contents += upsr0pb1 @;
  upsr0.Contents += upsr0pb2 @;
  upsrcr0.ButtonLabels = "Dismiss";
  upsrcr0.ButtonLabels += " Update";
  upsr0.PutOnScreen!();}

  case: {?button == " Update\nParameters";
  not IsString(slb1.Values);}
  {msgtd.Values = "Please select a ground type first";

```



```

        Msgwin.PutOnScreenAndWait!();}

    case: ?button == "Dismiss";
        {for find ?win.IsOnScreen == True;
         do {?win.TakeOffScreen!();}
         mmen.PutOnScreen!();}

    case: ?button == "Help";
        {;}
}
/*-----*/
method msgcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "Information" dialog box*/

    select {

        case: ?button == "OK";
            step1OK("OK");

        case: ?button == "Cancel";
            {;}

        case: {?button == "Remove Parameter";
            IsSymbol(Msgwin.UserData);}
            {warncr.UserData = Msgwin.UserData;
             ?sl = Msgwin.UserData;
             warntd.Values = AppendStrings("Do you really want to remove the variable,\n",
             corpre.UserData.?sl..parameter);

             warncr.ButtonLabels -= "Continue";
             warncr.ButtonLabels += "Delete";
             Warningwin.PutOnScreenAndWait!();}

        case: ?button == "Delete";
            {killit();
             msgcr.ButtonLabels = "Dismiss";}

        case: ?button == "Remove";
            {select {
             case: apretd2.Values == high;
                 ?sl = High_Applicability;
             case: apretd2.Values == medium;
                 ?sl = Medium_Applicability;
             otherwise: ?sl = Low_Applicability;}
             ?val = ListNth(apreocr.UserData, ListLength(apreocr.UserData)-1);
             if ListLength(?val) > 1;
             {for ?x inlist ListRest(?val);
              do {if ListLength(all appli0cr.UserData.?sl..?x) < 2;
                 {DeleteFacet(appli0cr.UserData, ?sl, ?x);}
                 else {for ?y inlist all appli0cr.UserData.?sl..?x;
                     do {if ListFirst(?y) == ConvertToSymbol(apretd.Values)@;
                         {appli0cr.UserData.?sl..?x -= ?y;}
                     else {;}}}}
             if ListLength(all appli0cr.UserData.?sl) > 1;
             {appli0cr.UserData.?sl -= ?val;}
             else {appli0cr.UserData.?sl = Null;}}
             else {if ListLength(all appli0cr.UserData.?sl) > 1;

```

```

        {appli0cr.UserData.?sl -== ?val;}
        else {appli0cr.UserData.?sl = Null;}
        aprecr.UserData = DeleteListElmt(ListNth(aprecr.UserData, ListLength(aprecr.UserData) -
1), aprecr.UserData);
        apre1b.SelectionItems = Null;
        apretd.Values = Null;
        apretd2.Values = Null;}

    case: ?button == "Correlate";
    {?inst = ModuleInstances(correl1 @);
    for ?x1 inlist ?inst;
    do {DeleteObject(?x1);}
    dbcrr1b.SelectionItems = Null;
    dbcrr1b.SelectionItems = Null;
    ?list = `();
    for ?x inlist corspb.UserData;
    do {?list = AppendLists(?list, `(?x.Parameters_needed));}
    ?list = SortList(?list, Alphabetize);
    for ?i from 0 to ListLength(?list)-1;
    do {dbcrr1b.SelectionItems += ListNth(?list, ListLength(?list)-1-?i);}
    dbcrrcr.ButtonLabels = "Dismiss";
    dbcrrcr.ButtonLabels += "Correlate";
    for find ?win.IsOnScreen == True;
    do {?win.TakeOffScreen!();}
    dbcrrcr.UserData = cors@;
    dbcrrcr.PutOnScreen!();}

    case: ?button == "Update ";
    {for find ?win.IsOnScreen == True;
    do {?win.TakeOffScreen!();}
    Step2.UserData..control = cors@;
    upcor(ListFirst(corspb.UserData));}

    case: ?button == "Update";
    {dbcrr1b.SelectionItems = Null;
    ?list = `();
    for ?x inlist corspb.UserData;
    do {?list = AppendLists(?list, `(?x.Parameters_needed));}
    ?list = SortList(?list, Alphabetize);
    for ?i from 0 to ListLength(?list)-1;
    do {dbcrr1b.SelectionItems += ListNth(?list, ListLength(?list)-1-?i);}
    dbcrrcr.ButtonLabels = "Dismiss";
    dbcrrcr.ButtonLabels += "Update";
    for find ?win.IsOnScreen == True;
    do {?win.TakeOffScreen!();}
    dbcrrcr.UserData = cors@;
    dbcrrcr.PutOnScreen!();}
}

Msgwin.TakeOffScreen!();
msgcr.ButtonLabels = "Dismiss";
Msgwin.UserData = Null;
}

```

GREP2.ptk

```
/* This file contains the functions and methods associated with the "Parameters search" dialog box. */
```

```
#include <prk/math.pth>
```

```
#include <prk/lib.pth>
```

```
/*-----*/
```

```
function par ()
```

```
{
```

```
  bound inputs;
```

```
  /* Puts the "Parameters search" dialog box on screen. */
```

```
  parlb1.SelectionItems = Null;
```

```
  parlb2.SelectionItems = Null;
```

```
  parlb2.MaxNumOfLines = 1;
```

```
  parpb2.ButtonLabel = Null;
```

```
  for ?x inlist ObjectSlots(Ground@);
```

```
  do parlb1.SelectionItems += ConvertToString(?x);
```

```
  par.PutOnScreen!();
```

```
}
```

```
/*-----*/
```

```
method parcr.React! (?button)
```

```
{
```

```
  bound inputs;
```

```
  /* The method attached to the command row of the "Parameters search" dialog box. */
```

```
  select {
```

```
    case: ?button == "Ground tree";
```

```
      {for find ?win.IsOnScreen == True;
```

```
        do ?win.TakeOffScreen!();
```

```
        menu();}
```

```
    case: ?button == "Dismiss";
```

```
      {for find ?win.IsOnScreen == True;
```

```
        do ?win.TakeOffScreen!();
```

```
        mmen.PutOnScreen!();}
```

```
    case: ?button == "Help";
```

```
      {;}
```

```
}
```

```
/*-----*/
```

```
method parpb1.React! (?button)
```

```
{
```

```
  bound inputs;
```

```
  /* Executes a search for the specified ground types (either basic or specific). */
```

```
  parlb2.SelectionItems = Null;
```

```
  parlb2.MaxNumOfLines = 1;
```

```
  if {IsString(parlb1.Values);
```

```
      IsNumber(pareb1.Values);}
```

```
    {?sl = ConvertToSymbol(parlb1.Values);
```

```
      ?num = pareb1.Values;
```

```

if ?button == "Basic ground types";
{?module = ground_rep@;}
else {?module = expand@;}

select {

case: parrb1.Values == "In range";
  {for {find ?gt.?sl == ?;
      ObjectModule(?gt) == ?module;
      IsList(?gt.?sl);
      IsNumber(ListFirst(?gt.?sl));
      IsNumber(ListNth(?gt.?sl, 2));
      ListFirst(?gt.?sl) <= ?num;
      ListNth(?gt.?sl, 2) >= ?num;}
   do parlb2.SelectionItems += ConvertToString(?gt);}

case: parrb1.Values == "Lower than";
  {for {find ?gt.?sl == ?;
      ObjectModule(?gt) == ?module;
      IsList(?gt.?sl);
      IsNumber(ListFirst(?gt.?sl));
      ListFirst(?gt.?sl) <= ?num;}
   do parlb2.SelectionItems += ConvertToString(?gt);}

case: parrb1.Values == "Greater than";
  {for {find ?gt.?sl == ?;
      ObjectModule(?gt) == ?module;
      IsList(?gt.?sl);
      IsNumber(ListNth(?gt.?sl, 2));
      ListNth(?gt.?sl, 2) >= ?num;}
   do parlb2.SelectionItems += ConvertToString(?gt);}}

  mnol();}
else {fail;}
}
/*-----*/
function mnol()
{
  bound inputs;
  /* Sets the number of lines of the ground types display list box. */

  find ?num1 = count parlb2.SelectionItems;
  select {

  case: {?num1 < 4;
        ?num1 > 0;}
    parlb2.MaxNumOfLines = ?num1;

  case: ?num1 == 0;
    parlb2.MaxNumOfLines = 1;

  otherwise:parlb2.MaxNumOfLines = 4;}

  if find parlb2.SelectionItems == ?;
  {parpb2.ButtonLabel = "More Detail";}
  else {parpb2.ButtonLabel = Null;}
}
/*-----*/

```

```

method parpb2.React! (?value)
{
  bound inputs;
  /* The method attached to the "More Detail", "Back" push button. */

  select {

    case: {?value == "More Detail";
      IsString(parlb2.Values);
      {parpb2.UserData = AppendLists^(parlb2.Values), all parlb2.SelectionItems);
      ?a = parlb2.Values;
      childval(?a);
      mnol();
      parpb2.ButtonLabel = "Back";}

    case: ?value == "Back";
      {?start = ListFirst(parpb2.UserData);
      parpb2.UserData = DeleteListElmt(?start, ListRest(parpb2.UserData));
      parlb2.SelectionItems = Null;
      for ?x inlist parpb2.UserData;
      do parlb2.SelectionItems += ?x;
      parlb2.SelectionItems += ?start;
      mnol();}}

  }
  /*-----*/
function childval(?a)
{
  bound inputs;
  /* This function searches for ground types that meet the specified criteria
  and belong to the est module */

  ?pl = `();
  parlb2.SelectionItems = Null;
  parlb2.MaxNumOfLines = 1;

  if {IsString(parlb1.Values);
      IsNumber(pareb1.Values);}
  {?sl = ConvertToSymbol(parlb1.Values);
  for {find ?ob = direct subclassof ConvertToSymbol(?a);
      ObjectModule(?ob) == est@;}
  do {collect ?ob into ?pl;}

  for ?ob1 inlist ?pl;
  do ?pl = AppendLists(?pl, all subclassof ?ob1);
  ?num = pareb1.Values;

  select {

    case: parrb1.Values == "In range";
      {for {?gt inlist ?pl;
          IsList(?gt.?sl);
          IsNumber(ListFirst(?gt.?sl));
          IsNumber(ListNth(?gt.?sl, 2));
          ListFirst(?gt.?sl) <= ?num;
          ListNth(?gt.?sl, 2) >= ?num;}
      do parlb2.SelectionItems += ConvertToString(?gt);}

    case: parrb1.Values == "Lower than";

```

```

    { for {?gt inlist ?pl;
        IsList(?gt.?sl);
        IsNumber(ListFirst(?gt.?sl));
        ListFirst(?gt.?sl) <= ?num;}
      do parlb2.SelectionItems += ConvertToString(?gt);}

    case: parrb1.Values == "Greater than";
    { for {?gt inlist ?pl;
        IsList(?gt.?sl);
        IsNumber(ListNth(?gt.?sl, 2));
        ListNth(?gt.?sl, 2) >= ?num;}
      do parlb2.SelectionItems += ConvertToString(?gt);}}

    else { fail;}
  }
  /*-----*/

```

UPTV.ptk

/ This file contains the functions and methods for the implementation procedures for "typical" values of ground properties. */*

```

#include <prk/math.pth>
#include <prk/lib.pth>
/*****

```

```

method upsr0lb.React! (?new_value, ?old_value)
{
  bound inputs;
  /* The method attached to the list box for
  specifying the parameter to update. */

  parlb(?new_value, ?old_value, upsr0lb@);
}
/*-----*/
method upsr0pb1.React! (?button)
{
  bound inputs;
  /* The method attached to "Back" "Forward" "Reset" push buttons. */

  parpb (?button, upsr0lb@, Parameters@);
}
/*-----*/
method upsr0pb2.React! (?button)
{
  bound inputs;

  /*The method attached to "New quantitative parameter"
  and "New qualitative parameter" push buttons. */

  select {

    case: ?button == "New quantitative parameter";
    {pba1("Quantitative ");
     nip.Title = "Quantitative Parameter Definition";
     nip.PutOnScreenAndWait!();}

```

```

    case: ?button == "New qualitative parameter";
        {pbal("Qualitative ");
         vip.Title = "Qualitative Parameter Definition";
         vip.PutOnScreenAndWait!();}
    }
/*-----*/
method upsrcr0.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the dialog box
       for specifying the parameter to update. */

    select {

        case: {?button == " Update";
            IsString(upsr0lb.Values);
            find ?obj.name == upsr0lb.Values;
            ListLength(all subclassof ?obj) == 0;}
        {upsr1.TakeOffScreen!();
         upsr1lb1.SelectionItems = Null;
         for {?y inlist SlotFacets(upsr1lb1, UserData);
              ?y != NoBPFacetNames;}
         do DeleteFacet(upsr1lb1, UserData, ?y);
         for {find ?ob.format == Qualitative;
              not find1 superclassof ?ob == "Field test parameters";}
         do {upsr1lb1.SelectionItems += ?ob.name;}
         if {IsString(slb1.Values);
              ?gt = ConvertToSymbol(slb1.Values)@;
              ?par = ConvertToSymbol(?obj);
              IsList(?gt.?par.."Required parameters");}
         {for ?x inlist ?gt.?par.."Required parameters";
          do {MakeFacet(upsr1lb1, UserData, ?x);
              upsr1lb1.UserData..?x = ?gt.?par..?x;}}
         else {;}
         upsr1.PutOnScreen!();}

        case: {?button == " Update";
            not IsString(upsr0lb.Values);}
        {msgtd.Values = "Please select a parameter to implement/update.";
         Msgwin.PutOnScreenAndWait!();}

        case: {?button == " Update";
            IsString(upsr0lb.Values);
            find ?obj.name == upsr0lb.Values;
            ListLength(all subclassof ?obj) > 0;}
        {msgtd.Values = "Only parameters (not parameter categories)\ncan be
implemented/updated";
         Msgwin.PutOnScreenAndWait!();}

        case: ?button == " Add\nParameter";
            addpar();

        case: ?button == "Dismiss";
            upsr0.TakeOffScreen!();}
    }
/*-----*/

```

```

function addpar ()
{
    bound inputs;
    /*The function attached to the "Add Parameter" push button. */

    upsr0.TakeOffScreen!();
    parpb ("Reset ", upsr0lb@, Parameters@);
    upsr0.Contents +== upsr0pb1@;
    upsr0.Contents +== upsr0pb2@;
    upsrcr0.ButtonLabels = "Dismiss";
    upsrcr0.ButtonLabels +== " Update";
    upsr0.PutOnScreen!();
}
/*****

method upsr1lb1.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the "Select required parameters" list box. */

    uplb(?new_value, ?old_value);
}
/*-----*/
function uplb(?new_value, ?old_value)
{
    bound inputs;

    rp.UserData = Null;
    rplb.SelectionItems = Null;
    rplb.UserData = Null;
    rppb.ButtonLabel = Null;
    rpeb.DefaultValues = Null;

    if {IsString(?new_value);
        find ?rep.name == ?new_value;}
    {?rep = ConvertToSymbol(?rep);
    rp.UserData = ?rep;
    rp.UserData..name = ?new_value;
    rp.Title = ?new_value;
    if IsFacet(upsr1lb1, UserData, ?rep);
    {for ?i from 0 to ListLength(upsr1lb1.UserData..?rep);
    do {rplb.SelectionItems +== ListNth(upsr1lb1.UserData..?rep,
                                        ListLength(upsr1lb1.UserData..?rep)-1-?i);}

    for {?pv = find ?rep@.format..per_val;
        ?pv != upsr1lb1.UserData..?rep;}
    do {rplb.UserData +== ?pv;}
    rplb.UserData = all rplb.UserData;
    rplb.UserData = AppendLists(rplb.UserData,
                                `(upsr1lb1.UserData..?rep));
    if ListLength(rplb.UserData) > 1;
    {rppb.ButtonLabel = "Show alternative set";}
    else{;}}

    else {?y = find1 ?rep@.format..per_val;
        for ?i from 0 to ListLength(?y)-1;
        do {rplb.SelectionItems +== ListNth(?y, ListLength(?y)-1-?i);
            rplb.SelectionItems +== "unknown";}
        for {?pv = find ?rep@.format..per_val;

```



```

        ?pv != ?y;}
    do {rplb.UserData += AppendLists(?pv, `("unknown"));}
    rplb.UserData = all rplb.UserData;
    rplb.UserData = AppendLists(rplb.UserData, `(?y));
    if ListLength(rplb.UserData) > 1;
    {rppb.ButtonLabel = "Show alternative set";}
    else{;}

    rp.PutOnScreenAndWait!();}
}
/*****
method rppb1.React! (?button)
{
    bound inputs;
    /* The method attached to the "Delete selected" push button. */

    if ListLength(all rplb.SelectionItems) > 1;
    {rplb.SelectionItems -= rplb.Values;}
    else {rplb.SelectionItems = Null;}
}
/*-----*/
method rppb.React! (?value)
{
    bound inputs;
    /* The method attached to the "Show alternative" push button. */

    rplb.SelectionItems = Null;
    ?l = ListLength(ListFirst(rplb.UserData));
    for ?i from 0 to ?l-1;
    do {rplb.SelectionItems += ListNth(ListFirst(rplb.UserData), ?l-?i-1);}
    rplb.UserData = ListRest(rplb.UserData);
    rplb.UserData = AppendLists(rplb.UserData, `(all rplb.SelectionItems));
}
/*-----*/
method rpeb.React! (?new_value)
{
    bound inputs;
    /* The method attached to the "Add a new value in the list" entry box. */

    rplb.SelectionItems += ?new_value;
    rpeb.Values = rpeb.DefaultValues;
}
/*-----*/
method rpcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "required parameter" dialog box. */

    select {

        case: {?button == "Assign";
            ListLength(all rplb.SelectionItems) > 0;}
        {if find1 ?name.name == rp.Title;
            {?name = ConvertToSymbol(?name);
            upsr11b1@.UserData..?name = SortList(all rplb.SelectionItems, Alphabetize);
            rp.TakeOffScreen!();}
            else{fail;}}
    }
}

```

```

case: {?button == "Assign";
    ListLength(all rplb.SelectionItems) == 0;}
{msgtd.Values = "Please define values for the required parameter";
Msgwin.PutOnScreenAndWait!();
fail;}

case: ?button == "Detach";
{if find1 ?name.name == rp.Title;
{?name = ConvertToSymbol(?name);
if IsFacet(upsr1lb1@, UserData,?name);
{DeleteFacet(upsr1lb1@, UserData,?name);
if ListLength(all uplb.SelectionItems) > 1;
{uplb.SelectionItems -== uplb.Values;}
else {uplb.SelectionItems = Null;}}
else {fail;}}
else {fail;}}
rp.TakeOffScreen!();}

case: ?button == "Cancel";
rp.TakeOffScreen!();}
}
/*****/
method upsrcr1.React! (?button)
{
    bound inputs;
/* The method attached to the command row of the "Required parameters" dialog box. */

select {
case: ?button == "Show";
{upsrpreviewcr.ButtonLabels = "Dismiss";
show(0);}

case: {?button == "Update";
    not IsString(slb1.Values);}
{msgtd.Values = "Please select first a ground type";
Msgwin.PutOnScreenAndWait!();
fail;}

case: {?button == "Update";
    not IsString(upsr0lb.Values);}
{msgtd.Values = "Please select first a parameter to implement/update";
Msgwin.PutOnScreenAndWait!();
fail;}

case: {?button == "Update";
    IsString(upsr0lb.Values);
    find ?par.name == upsr0lb.Values;
    ListLength(all subclassof ?par) > 0;}
{msgtd.Values = "Only parameters (not parameter categories)
    can be implemented/updated";
.Msgwin.PutOnScreenAndWait!();
fail;}

case: {?button == "Update";
    IsString(slb1.Values);
    IsString(upsr0lb.Values);}
{?ob = ConvertToSymbol(slb1.Values);
if find ?sl.name == upsr0lb.Values;
{?sl = ConvertToSymbol(?sl);}
else {?sl = u;

```

```

fail;}

if or {find1 upsr1lb1.UserData..?fac == ?;
      {IsSlot(?ob, ?sl);
       IsList(?ob.?sl..`"Required parameters");
       ListLength(?ob.?sl..`"Required parameters") > 0;}}
{upsrpreviewcr.ButtonLabels = "Reset";
 upsrpreviewcr.ButtonLabels += "Continue";

if IsSlot(?ob, ?sl);
{for ?f inlist ?ob.?sl..`"Required parameters";
 do {upsr1lb1.UserData..?f = ?ob.?sl..?f;}}
else {;}
show(0);

if upsr1lb1.UserData == 1;
{upsr1lb1.UserData = Null;
 fail;}
else {upsr1lb1.UserData = Null;}

if IsSlot(?ob, ?sl);
{;}
else {MakeSlot(Ground@, ?sl);
      C:{PrkSetSlotInheritance(Ground@, ?sl, PrkSVNoInheritance);}
      C:{PrkMakeFacet(Ground@, ?sl, `"Required parameters",
                    PrkMakeRawFacetData(PrkNull, PrkDefault, PrkSVNoInheritance));}}

if IsList(?ob.?sl..`"Required parameters");
{for {?x inlist SlotFacets(upsr1lb1, UserData);
     ?x != NoBPFacetNames;}
 do {if ListLength(FindListElmt(?ob.?sl..`"Required parameters", ?x)) > 0;
     {;}
     else {?ob.?sl..`"Required parameters" = AppendLists(?ob.?sl..`"Required parameters",
                                                         `(?x));}}}

else {?list = `();
      for {?x inlist SlotFacets(upsr1lb1, UserData);
         ?x != NoBPFacetNames;}
 do {?list = AppendLists(?list, `(?x));}
   ?ob.?sl..`"Required parameters" = ?list;}

for {?x inlist SlotFacets(upsr1lb1, UserData);
     ?x != NoBPFacetNames;}
do { if IsFacet(Ground@, ?sl, ?x);
     {;}
     else {C:{PrkMakeFacet(Ground@, ?sl, ?x,
                          PrkMakeRawFacetData(PrkNull, PrkDefault, PrkSVNoInheritance));}}
        Ground.?sl..?x = Null;
        ?ob.?sl..?x = upsr1lb1.UserData..?x;}

find1 ?param.name == upsr0lb.Values;
if {IsFacet(?param, format, units);
   ?param.format..units != "";}
{updatewin.Title = AppendStrings(ConvertToString(?param),
                                  " (in ", ?param.format..units, ")");}
else {updatewin.Title = ConvertToString(?param);}
sdef(?ob, ?sl);
updatewincr.UserData = ?ob;
updatewintd.Values = ?ob;

```

```

updatewin.PutOnScreenAndWait!();
upsr1lb1.UserData.."Required parameters" = ?ob.?sl.."Required parameters";
testup2(?ob, ?sl);
test3a();
SaveApp(ground_rep);}

else {if IsSlot(?ob, ?sl);
    {;}
    else {MakeSlot(Ground@, ?sl);
        C:{PrkSetSlotInheritance(Ground@, ?sl, PrkSVNoInheritance);}
        C:{PrkMakeFacet(Ground@, ?sl, "Required parameters",
            PrkMakeRawFacetData(PrkNull, PrkDefault, PrkSVNoInheritance));}}

    find1 ?param.name == upsr0lb.Values;
    if {IsFacet(?param, format, units);
        ?param.format..units != "";}
    {updatewin.Title = AppendStrings(ConvertToString(?param),
        " (in ", ?param.format..units, ")");}
    else {updatewin.Title = ConvertToString(?param);}
    sdef(?ob, ?sl);
    updatewind.Values = ?ob;
    updatewinincr.UserData = ?ob;
    updatewin.PutOnScreenAndWait!();
    test3a();
    SaveApp(ground_rep);}

case: ?button == "Cancel";
    upsr1.TakeOffScreen!();

case: ?button == "HELP";
    {;}
}
}
}
/*-----*/
function show (?v)
{
    bound inputs;
    /* This function makes the appropriate settings for the
        "Preview" dialog box and outs it on screen. */

    uplb.SelectionItems = Null;
    for {?fac inlist SlotFacets(upsr1lb1, UserData);
        ?fac != NoBPFacetNames;}
    do {?f = ?fac@.name;
        uplb.SelectionItems += ?f;}
    upsrpreview@.PutOnScreenAndWait!();
}
/*-----*/
method uplb.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the list box of the
        "Preview" dialog box. */

    uplb(?new_value, ?old_value);
}
/*-----*/

```

```

method upsrpreviewcr.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "Preview" dialog box. */

  select {

    case: ?button == "Dismiss";
      upsr1lb1.UserData = 0;
    case: ?button == "Continue";
      upsr1lb1.UserData = 0;
    otherwise: upsr1lb1.UserData = 1;
  }
  upsrpreviewcr.ButtonLabels = "Dismiss";
  upsrpreview@.TakeOffScreen!();
}
/*-----*/
method updatewincr.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "Update window" dialog box. */

  select {

    case: {?button == "OK";
      or{IsNumber(updatewineb1.Values);
        updatewineb1.Values == u;}
      or{IsNumber(updatewineb2.Values);
        updatewineb2.Values == u;}
      or{IsNumber(updatewineb3.Values);
        updatewineb3.Values == u;}}
    {select {

      case: {IsNumber(updatewineb1.Values);
        IsNumber(updatewineb2.Values);
        updatewineb1.Values > updatewineb2.Values;}
        {msgtd.Values = "The specified minimum value is greater than the maximum";
        Msgwin.PutOnScreenAndWait!();}

      case: {IsNumber(updatewineb1.Values);
        IsNumber(updatewineb3.Values);
        updatewineb1.Values > updatewineb3.Values;}
        {msgtd.Values = "The specified minimum value is greater than the mean";
        Msgwin.PutOnScreenAndWait!();}

      case: {IsNumber(updatewineb2.Values);
        IsNumber(updatewineb3.Values);
        updatewineb3.Values > updatewineb2.Values;}
        {msgtd.Values = "The specified mean value is greater than the maximum";
        Msgwin.PutOnScreenAndWait!();}

      otherwise: {?ob = updatewincr.UserData;
        updatewincr.UserData = Null;
        if find ?sl.name == upsr0lb.Values;
        {?sl = ConvertToSymbol(?sl);}
        else {?sl = u;
          fail;}
        ?ob.?sl = `(updatewineb1.Values, updatewineb3.Values, updatewineb2.Values);

```

```

        updatewin.TakeOffScreen!();}}

case: ?button == "Cancel";
    updatewin.TakeOffScreen!();

otherwise: {msgtd.Values =
    AppendStrings("the min, max and mean values of the implemented parameter\n",
        "should be numbers or the symbol u (u for unknown)");
    Msgwin.PutOnScreen!();}
}
}
/*-----*/
function testup2 (?ini_ob, ?sl )
{
    bound inputs;
    /* The function that creates the object hierarchy for the representation of "typical" values. */

    ?parents_list = `(?ini_ob);
    for {find ?ob = subclassof ?ini_ob;
        ObjectModule(?ob) == est@;}
    do ?parents_list = AppendLists(?parents_list, `(?ob));

    for {?x inlist ?parents_list;
        IsList(?x.?sl.."Required parameters");
        ListLength(?x.?sl.."Required parameters") > 0;}
    do {?fac = ListFirst(?x.?sl.."Required parameters");
        ?rp = ListRest(?x.?sl.."Required parameters");

        for ?y inlist ?ini_ob.?sl..?fac;
        do {select {

            case: ?y == "unknown";
                ?x.?sl.."Required parameters" = ?rp;

            case: ?y != "unknown";
                {?name = ConvertToSymbol(AppendStrings(?y, " ", ConvertToString(?x)));
                ?sublist = `();
                for ?sub = find subclassof ?ini_ob;
                do { ?sublist = AppendLists(?sublist, `(ConvertToSymbol(?sub)));}
                if ListLength(FindListElmt(?sublist, ?name)) > 0;
                {;}
                else {MakeClass(?name, est@, `(?x), `());}
                ?name.?sl.."Required parameters" = ?rp;
                for {?fa inlist SlotFacets(?name, ?sl);
                    ?fa != "Required parameters";}
                do {?name.?sl..?fa = "";}

                /* Update Facets. */

                ?nam = AppendStrings(ConvertToString(?ini_ob), "\n\n");
                if {find1 ?parent = direct superclassof ?name;
                    ObjectModule(?parent) == est@;}
                {for {?z inlist SlotFacets(?parent, ?sl);
                    ?z != "Required parameters";}
                do {?name.?sl..?z = ?parent.?sl..?z;
                    if {IsFacet(upsr1lb1, UserData, ?z);
                        ?z != NoBPFacetNames;
                        find ?parent.?sl..?z == ;}

```

```

        ?parent.?sl.?z != "");}
        {?str = Substring(?z@.name, 0, FindSubstring(?z@.name, ", "));
        ?nam = AppendStrings(?nam, ?str, ": ", ?parent.?sl.?z, "\n");}
        else {;}}
    else {;}
    ?name.?sl.?fac = ?y;
    ?str = Substring(?fac@.name, 0, FindSubstring(?fac@.name, ", "));
    ?nam = AppendStrings(?nam, ?str, ": ", ?y);

    sdef(?name, ?sl);
    updatewincr.UserData = ?name@;

    wincall(?nam);}
    }
    }
    if ?x.?sl.."Required parameters" == ?rp;
    {;}
    else {?x.?sl.."Required parameters" = `();}
    }
    testup3 (?ini_ob, ?sl);
}
/*-----*/
function testup3 (?ini_ob, ?sl)
{
    bound inputs;
    /* This function checks if the testup2 function should be invoked again or not. */

    ?pl = AppendLists(`(?ini_ob), all subclassof ?ini_ob);
    ?con = 0;
    for {?x inlist ?pl;
        or {?x == ?ini_ob;
            ObjectModule(?x) == est@;}
        IsList(?x.?sl.."Required parameters");}
    do {if ListLength(?x.?sl.."Required parameters") > 0;
        {?con = ?con + 1;}
        else {?con = ?con + 0;}}
    if ?con > 0;
    {testup2 (?ini_ob, ?sl);}
    else{?ini_ob.?sl.."Required parameters" = upsr1lb1.UserData.."Required parameters";
        updatewind.Values = Null;
        DeleteFacet(upsr1lb1, UserData, "Required parameters");}
}
/*-----*/
function wincall (?name)
{
    bound inputs;
    /* This function calls the "Update window" dialog box. */

    updatewind.Values = ?name;
    updatewin.PutOnScreenAndWait!();
}
/*-----*/
function test3a ()
{
    bound inputs;
    /* This function deletes the estimation objects that do not contain any "typical" values. */

    for {?x inlist ModuleClasses(est@);}

```

```

do {if {ListLength(all subclassof ?x) < 1;
      not {find1 ?x.?sl == ?;}}
    {DeleteObject(?x);}
  else {;}}
}
/*-----*/
function sdef (?ob, ?sl)
{
  bound inputs;
  /* This function is used for displaying already implemented
     "typical" values in the "Update window" dialog box. */

  if {IsList(?ob.?sl);}
  {updatewineb1.DefaultValues = ListFirst(?ob.?sl);
   updatewineb2.DefaultValues = ListNth(?ob.?sl, 2);
   updatewineb3.DefaultValues = ListNth(?ob.?sl, 1);}
  else {updatewineb1.DefaultValues = Null;
        updatewineb2.DefaultValues = Null;
        updatewineb3.DefaultValues = Null;}
}

```


TESTCOR.ptk

```
/* This file contains the functions and methods for checking the variables of a correlation. */
```

```
#include <prk/lib.pth>
```

```
#include <prk/math.pth>
```

```
/*-----*/
```

```
method Correlations.Data_Check! ()
```

```
{
```

```
  bound inputs;
```

```
  /* The method attached to the Data_Check! slot in every correlation object. */
```

```
  ?error_string = "";
```

```
  ?warning_string = "";
```

```
  ?self.error = Null;
```

```
  ?self.error..warnings = Null;
```

```
  ?var = ?self.Parameter..parameter;
```

```
  ?self.?var = Null;
```

```
  for find ?self.?slIP.IP_description == ?;
```

```
  do {?self.?slIP = Null;}
```

```
  nv_check(?self, BV_description);
```

```
  if find ?self.error == ?;
```

```
  {?error_list = all ?self.error;
```

```
  for ?x1 inlist ?error_list;
```

```
  do {?error_string = AppendStrings(?error_string, "\n", ?x1);}
```

```
  errorfd.Values = ?error_string;
```

```
  Errorwin.PutOnScreen!();
```

```
  fail;}
```

```
  else {;}
```

```
  nv_check1(?self, BV_description);
```

```
  nv_check2(?self, BV_description);
```

```
  select {
```

```
    case: find ?self.error == ?;
```

```
    {?error_list = all ?self.error;
```

```
    for ?x1 inlist ?error_list;
```

```
    do {?error_string = AppendStrings(?error_string, "\n", ?x1);}
```

```
    errorfd.Values = ?error_string;
```

```
    Errorwin.PutOnScreen!();
```

```
    fail;}
```

```
    case: find ?self.error..warnings == ?;
```

```
    {?warning_list = all ?self.error..warnings;
```

```
    for ?x1 inlist ?warning_list;
```

```
    do {?warning_string = AppendStrings(?warning_string, "\n", ?x1);}
```

```
    warnfd.Values = ?warning_string;
```

```
    warncr.UserData = `(?self, 1);
```

```
    Warningwin.PutOnScreenAndWait!();}
```

```
  otherwise: {datacheck2 (?self);}
```

```
}
```

```
/*-----*/
```

```

function datacheck2 (?self)
{
    bound inputs;
    /* The driving data-check function. */

    ?error_string = "";
    ?warning_string = "";
    ?self.error = Null;
    ?self.error..warnings = Null;

    if find ?self.?sloo..IV_description == Quantitative;
    {?self.?sloo = Null;
    calculatef(?self, `Parameters_needed);
    nv_check(?self, IV_description);
    nv_check1(?self, IV_description);
    nv_check2(?self, IV_description);}
    else {calculatef(?self, `Parameter);
        fail;}

    select {

        case: find ?self.error == ?;
            {?error_list = all ?self.error;
            for ?x1 inlist ?error_list;
            do {?error_string = AppendStrings(?error_string, "\n", ?x1);}
            errord.Values = ?error_string;
            Errorwin.PutOnScreen!();
            fail;}

        case: find ?self.error..warnings == ?;
            {?warning_list = all ?self.error..warnings;
            for ?x1 inlist ?warning_list;
            do {?warning_string = AppendStrings(?warning_string, "\n", ?x1);}
            warntd.Values = ?warning_string;
            warncr.UserData = `(?self, 2);
            Warningwin.PutOnScreenAndWait!();}

        otherwise: calculatef(?self, `Parameter);}
    }
    /*****
function nv_check(?self, ?fac)
{
    bound inputs;
    /* Perorms format checks for quantitative basic variables. */

    ?vlist = `();
    for find ?self.?sl..?fac == Quantitative;
    do {?y = 0;
        if {IsList(?self.?sl);
        for ?x inlist ?self.?sl;
        do {if IsNumber(?x);
            {;}
            else{?y = ?y + 1;}}
        ?y == 0;}
    {;}
    else {?self.error += AppendStrings("The value(s) of ", ?self.?sl..variable,
        " should be in parentheses ()",
        "\n separated by commas ,.");}}

```

```

}
/*****
function nv_check1(?self, ?fac)
{
  bound inputs;
  /* Checks quantitative variables' values are in range. */

  ?vlist = `();
  for find ?self.?sl.?fac == Quantitative;
  do { ?max = ?self.?sl.max_value;
      ?min = ?self.?sl.min_value;

      select {

        case: {IsNumber(?self.?sl.max_value);
              IsNumber(?self.?sl.min_value);}
          {for ?x inlist ?self.?sl;
            do {if {?x <= ?max;
                  ?x >= ?min;}
              {;}
            else {?self.error..warnings +=
                  AppendStrings("The value ", ConvertToString(?x), " of the ",
                                ConvertToString(?self.?sl.variable)," is not between ",
                                ConvertToString(?min), " and ", ConvertToString(?max));}}}

        case: {IsNumber(?self.?sl.max_value);
              not {IsNumber(?self.?sl.min_value);}}
          {?vlist = ?self.?sl;
            for ?x inlist ?vlist;
            do {if ?x <= ?max;
              {;}
            else {?self.error..warnings += AppendStrings("The value ", ConvertToString(?x),
                  " of the ", ConvertToString(?self.?sl.variable)," is greater than ", ConvertToString(?max));}}}
          case: {not {IsNumber(?self.?sl.max_value);
                  IsNumber(?self.?sl.min_value);}
              {?vlist = ?self.?sl;
                for ?x inlist ?vlist;
                do {if ?x >= ?min;
                  {;}
                else {?self.error..warnings += AppendStrings("The value ", ConvertToString(?x),
                      " of the ", ConvertToString(?self.?sl.variable)," is less than ", ConvertToString(?min));}}}

          otherwise: {;}}
      }
}
/*****
function nv_check2(?self, ?fac)
{
  bound inputs;
  /* Checks the number of values for each variable. */

  ?input_list = `();
  if ?fac == BV_description;
  {for find ?self.?sl.?fac == Quantitative;
    do {collect ListLength(?self.?sl) into ?input_list;}
    for ?i from 0 to ListLength(?input_list)-1;
    do {if ListNth(?input_list, ?i) != ListFirst(?input_list);
        {?self.error += "The number of values of each Quantitative\nBasic Variable is not the
same";}
}
}

```

```

else {;}}
else {for find or {?self.?sl.?fac == Quantitative;
        ?self.?sl..BV_description == Quantitative;}
    do {collect ListLength(?self.?sl) into ?input_list;}
    for ?i from 0 to ListLength(?input_list)-1;
    do {if ListNth(?input_list, ?i) != ListFirst(?input_list);
        {?self.error += "The number of values of each Quantitative\nVariable is not the
same";}
        else {;}}
    }
}

```

CORRSEARCH.ptk

/ This file contains the functions and methods for searching the **correlation** object base. */*

```
#include <prk/math.pth>
```

```
#include <prk/lib.pth>
```

```
/******
```

```
function sstart()
```

```
{
    bound inputs;
    /* Puts the "correlation search" dialog box on screen. */
```

```
    for {?x inlist SlotFacets(corslb1, UserData);
```

```
        ?x != NoBPFacetNames;}
    do {corslb1.UserData..?x = Null;}
    cors.PutOnScreen!();
}

```

```
/******
```

```
method corslb1.React! (?new_value, ?old_value)
```

```
{
    bound inputs;
    /* The method attached to the list box of the "correlations search" dialog box. */

    select {

        case: ?new_value == "the parameter(s) to estimate";
            {corsplb1.SelectionItems = Null;
              ?list = `();
              corsplb1.DefaultValues = Null;
              for find ?x.Parameter == ?;
              do {?list = AppendLists(?list, `(?x.Parameter));}
              setlbd(?list, corsplb1@, par);
              corssp.PutOnScreenAndWait!();}

        case: ?new_value == "the variable(s) of the correlation/correction";
            {setcorv(Quantitative, corsv1b1@);
              setcorv(Qualitative, corsv1b2@);
              corsv.PutOnScreenAndWait!();}

        case: ?new_value == "the applicable ground type(s)";
            {?list = `();
              corsalb1.DefaultValues = Null;
              corsalb1.SelectionItems = Null;
              for ?x inlist `(High_Applicability, Medium_Applicability, Low_Applicability);
```

```

do {for find ?y.?x == ?;
  do {for ?z inlist all ?y.?x;
    do {?list = AppendLists(?list, `(ConvertToString(ListFirst(?z))))};}}
setlbd(?list, corslb1@, app);
corsa.PutOnScreenAndWait!(); }

case: ?new_value == "the reliability of the correlation/correction";
{corsrlb.DefaultValues = Null;
if IsList(corslb1.UserData..rel);
{for ?x inlist corslb1.UserData..rel;
do {corsrlb.DefaultValues +== ?x;}}
else {};}
corsr.PutOnScreenAndWait!();}

case: ?new_value == "the reference of the correlation/correction";
{corsrelb.SelectionItems = Null;
?list = `();
corsrelb.DefaultValues = Null;
for find ?x.wintitle..author == ?;
do {?list = AppendLists(?list, `(?x.wintitle..author));}
setlbd(?list, corsrelb@, ref);
corsre.PutOnScreenAndWait!();}

otherwise: {};}
}
/*****
function setlbd(?list, ?lb, ?fac)
{
  bound inputs;
  /* This function retrieves all the possible values for each search
  criterion (except variables) and places them into a list box. */

  ?list = SortList(?list, Alphabetize);
  for ?i from 0 to ListLength(?list)-1;
  do {?lb.SelectionItems +== ListNth(?list, ListLength(?list)-1-?i);}
  if IsList(corslb1.UserData..?fac);
  {for ?x inlist corslb1.UserData..?fac;
  do {?lb.DefaultValues +== ?x;}}
  else {};}
}
/*****
function setcorv(?for, ?lb)
{
  bound inputs;
  /* This function retrieves all the possible values
  for variables and places them into a list box. */

  ?lb.SelectionItems = Null;
  ?list = `();
  ?lb.DefaultValues = Null;
  for ?x inlist all subclassof Correlations@;
  do {for find ?x.?sl..BV_description == ?for;
    do {?list = AppendLists(?list, `(?x.?sl.variable));}}
  ?list = SortList(?list, Alphabetize);
  for ?i from 0 to ListLength(?list)-1;
  do {?lb.SelectionItems +== ListNth(?list, ListLength(?list)-1-?i);}
  if IsList(corslb1.UserData..var);
  {for ?x inlist corslb1.UserData..var;

```

```

do {?lb.DefaultValues += ?x;}
else {?lb.DefaultValues = Null;}
}
/*****
method corspcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "parameters display" dialog box. */

    critre(?button, corsplb1@, corsp@, par);
}
/*****
method corsvcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "variables display" dialog box. */

    select {

        case: ?button == "Update";
            {?list = AppendLists(all corsvlb1.Values, all corsvlb2.Values);
             if ListLength(?list) > 0;
             { corslb1.UserData..var = ?list;}
             else {;}
             corsv.TakeOffScreen!();}

        case: ?button == "Cancel";
            {corsv.TakeOffScreen!();}
    }
}
/*****
method corsacr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the
       "applicable ground types display" dialog box. */

    critre(?button, corsalb1@, corsa@, app);
}
/*****
method corsrcr.React! (?button)
{
    bound inputs;

    /* The method attached to the command row of the "reliability display" dialog box. */

    critre(?button, corsrlb@, corsr@, rel);
}
/*****
method corsrecre.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "references display" dialog box. */

    critre(?button, corsrelb@, corsre@, ref);
}
/*****
function critre(?button, ?lb, ?win, ?fac)
{

```

```

bound inputs;
select {

  case: ?button == "Update";
  {?list = all ?lb.Values;
   if ListLength(?list) > 0;
   {corslb1.UserData..?fac = ?list;}
   else {corslb1.UserData..?fac = Null;}
   ?win.TakeOffScreen!();}

  case: ?button == "Cancel";
  {?win.TakeOffScreen!();}
}
/*****
method corspb.React! (?button)
{
  bound inputs;
  /* The method attached to the "Search", "Preview search settings"
     and "Clear all settings" push buttons. */

  select {

    case: ?button == "Search";

      {msgtd.Values = "";

       ?lp = `();
       if IsList(corslb1.UserData..par);
       {for ?x inlist corslb1.UserData..par;
        do {find ?ob.Parameter == ?x;
           ?lp = AppendLists(?lp, `(?ob));}
          ?lp = obchk(?lp);
          msgtd.Values = AppendStrings(msgtd.Values, "Parameters search results: ");
          inseres (?lp);}
          else {?lp = c;}

        ?lv = `();
        if IsList(corslb1.UserData..var);
        {for ?x inlist corslb1.UserData..var;
         do {for ?y inlist all subclassof Correlations@;
            do {find ?y.?sl.variable == ?x;
               ?lv = AppendLists(?lv, `(?y));}}
           ?lv = obchk(?lv);
           msgtd.Values = AppendStrings(msgtd.Values, "Variables search results: ");
           inseres (?lv);}
           else {?lv = c;}

          ?la = `();
          if IsList(corslb1.UserData..app);
          {for ?x inlist corslb1.UserData..app;
           do {for ?sl inlist `(High_Applicability, Medium_Applicability,
                               Low_Applicability);
              do {for ?y inlist all subclassof Correlations@;
                 do {for ?z inlist all ?y.?sl;
                    do {?gt = ListFirst(?z);
                       if {or {?gt == ConvertToSymbol(?x)@;
                               find subclassof ?gt == ConvertToSymbol(?x)@;}}
                      {?la = AppendLists(?la, `(?y));}}}}}}

```

```

?la = obchk(?la);
msgtd.Values = AppendStrings(msgtd.Values, "Applicability search results: ");
inseres (?la);}
else {?la = c;}

?lr = `();
if IsList(corslb1.UserData..rel);
{for ?x inlist corslb1.UserData..rel;
do {find ?ob.Reliability == ?x;
?lr = AppendLists(?lr, `(?ob));}
?lr = obchk(?lr);
msgtd.Values = AppendStrings(msgtd.Values, "Reliability search results: ");
inseres (?lr);}
else {?lr = c;}

?lt = `();
if IsList(corslb1.UserData..ref);
{for ?x inlist corslb1.UserData..ref;
do {find ?ob.wintitle..author == ?x;
?lt = AppendLists(?lt, `(?ob));}
?lt = obchk(?lt);
msgtd.Values = AppendStrings(msgtd.Values, "References search results: ");
inseres (?lt);}
else {?lt = c;}

seresult(?lp, ?lv, ?la, ?lr, ?lt);
Msgwin.PutOnScreenAndWait!();}

case: ?button == "Preview search settings";
{msgtd.Values = "";
for {?x inlist SlotFacets(corslb1, UserData);
?x != NoBPFacetNames;}
do {select {
case: ?x == par;
?str = AppendStrings(msgtd.Values, "Parameter(s):\n");
case: ?x == var;
?str = AppendStrings(msgtd.Values, "Variable(s):\n");
case: ?x == app;
?str = AppendStrings(msgtd.Values, "Ground type(s):\n");
case: ?x == rel;
?str = AppendStrings(msgtd.Values, "Reliability score(s):\n");
case: ?x == ref;
?str = AppendStrings(msgtd.Values, "Reference(s):\n");}
precri(?str, ?x);}

Msgwin.PutOnScreenAndWait!();}

case: ?button == "Clear all settings";
{for {?x inlist SlotFacets(corslb1, UserData);
?x != NoBPFacetNames;}
do {corslb1.UserData..?x = Null;}}
}
/*****
function obchk (?l)
{
bound inputs;
/* The method attached to the "Correlations only", "Corrections only" and "Both" radio buttons. */

```



```

if corsrb.Values == "Both";
{;}
else {if corsrb.Values == "Correlations only";
    {for ?x inlist ?l;
    do {if ObjectModule(?x) == correction@;
        {if ListFirst(?l) == ?x;
            {?l = ListRest(?l);}
            else {?l = DeleteListElmt(?x, ?l);}}}
    else {for ?x inlist ?l;
        do {if ObjectModule(?x) == correlation@;
            {if ListFirst(?l) == ?x;
                {?l = ListRest(?l);}
                else {?l = DeleteListElmt(?x, ?l);}}}
    }
    return ?l;
}
}
/*****
function inseres (?l)
{
    bound inputs;
    /* This function counts the number of hits for individual criteria. */

    corspb.UserData = Null;
    for ?x inlist ?l;
    do {corspb.UserData += ?x;}
    ?l = all corspb.UserData;
    corspb.UserData = Null;
    ?n = ListLength(?l);
    if ?n > 0;
    {msgtd.Values = AppendStrings(msgtd.Values, ConvertToString(?n),
        " hit(s)\n");}
    else {msgtd.Values = AppendStrings(msgtd.Values, "no hits\n");}
}
}
/*****
function seresult (?lp, ?lv, ?la, ?lr, ?lt)
{
    bound inputs;
    /* This function counts the number of hits for the combination of all criteria. */

    corspb.UserData = Null;
    ?resl = `();
    ?sr = c;
    ?ol = `();
    for ?x inlist `(?!p, ?lv, ?la, ?lr, ?lt);
    do {select {

        case: {IsList(?x);
            ListLength(?x) > 0;}
            {?ol = AppendLists(?ol, `(?x));}

        case: {IsList(?x);
            ListLength(?x) == 0;}

        ?sr = "no hits";}}

if IsString(?sr);

{msgtd.Values = AppendStrings(msgtd.Values, "Overall search results: ", ?sr);
fail;}

```

```

if ListLength(?ol) == 0;
{ ?n = ConvertToString(ListLength(all subclassof Correlations@));
msgtd.Values = AppendStrings(msgtd.Values, "Overall search results: ",
                             ?n, " hits");
corspb.UserData = all subclassof Correlations@;}

else {for ?x inlist ListFirst(?ol);
      do {?count = 0;
          for ?y inlist ListRest(?ol);
          do {if ListLength(FindListElmt(?y, ?x)) > 0;
              {?count = ?count + 1;}}
          if ?count == ListLength(ListRest(?ol));
          {?resl = AppendLists(?resl, `(?x));}}
          ?n = ConvertToString(ListLength(?resl));
          msgtd.Values = AppendStrings(msgtd.Values, "\nOverall search results: ",
                                       ?n, " hit(s)");

          corspb.UserData = ?resl;}
if ListLength(corspb.UserData) > 0;
{select {

  case: mmenlb.Values == "Estimate ground parameters from correlations/corrections";
      {msgtd.Values = AppendStrings(msgtd.Values, "\n\nTo use the identified set of correlations
                                             and/or corrections,\npress Correlate\n");
      msgcr.ButtonLabels += "Correlate";}

  case: {mmenlb.Values == "Update implemented correlations/corrections";
        ListLength(corspb.UserData) == 1;}
      {msgtd.Values = AppendStrings(msgtd.Values, "\n\nTo update the identified
                                             correlation/correction,\npress Update");
      msgcr.ButtonLabels += "Update "};

  case: {mmenlb.Values == "Update implemented correlations/corrections";
        ListLength(corspb.UserData) > 1;}
      {msgtd.Values = AppendStrings(msgtd.Values, "\n\nTo update any of the identified
                                             correlations/corrections,\npress Update");
      msgcr.ButtonLabels += "Update";}}
}
/*****
function precri(?str, ?fac)
{
  bound inputs;
  /* This function creates the appropriate text for the presentation of the search results. */

  if IsList(corslb1.UserData..?fac);
  {for ?x inlist corslb1.UserData..?fac;
    do {?str = AppendStrings(?str, " ", ?x, "\n");}}
  else {?str = AppendStrings(?str, " ", "unspecified\n");}
  msgtd.Values = ?str;
}
/*****
method corscr.React! ()
{
  bound inputs;
  /* The method attached to the command row of the "correlation search" dialog box. */
  cors.TakeOffScreen!();
  mmen.PutOnScreen!();
}

```

CORRUI.ptk

```
/* This file contains the functions and methods for creating and using correlation dialog boxes. */
```

```
#include <prk/math.pth>
#include <prk/lib.pth>
#builtin Format (?x) ((ConvertToFloat(ConvertToFixnum(10*?x + .5)))/10)
#builtin Format3 (?x) ((ConvertToFloat(ConvertToFixnum(1000*?x + .5)))/1000)
#builtin Format2 (?x) ((ConvertToFloat(ConvertToFixnum(100*?x + .5)))/100)
#builtin Format0 (?x) (ConvertToFloat(ConvertToFixnum(1*?x + .5)))
/*****/

function startcorr()
{
  bound inputs;
  /* This function puts the "Correlation Display" dialog box on screen. */

  ?inst = ModuleInstances(correl1@);
  for ?x1 inlist ?inst;
  do {DeleteObject(?x1);}
  dbcorr1b.SelectionItems = Null;
  for ?x inlist all subclassof Correlations@;
  do {dbcorr1b.SelectionItems += ?x.Parameters_needed;}
  dbcorr1b.PutOnScreen!();
}
/*****/

method dbcorr1b.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "Correlation Display" dialog box. */

  select {

    case: ?button == "Correlate";
    {?posx = 200;
    ?posy = 150;
    for {?x2 inlist all dbcorr1b.Values;
    find ?obje.Parameters_needed == ?x2;}
    do {if {find1 ?ob.UserData == ?obje;
    ObjectModule(?ob) == correl1@;}
    {?ob.PositionX = ?posx;
    ?ob.PositionY = ?posy;
    ?ob.PutOnScreen!();}
    else {create_db(?obje, ?posx, ?posy);}
    ?posx = ?posx + 30;
    ?posy = ?posy + 30;}}

    case: ?button == "Dismiss";
    {dbcorr1b.TakeOffScreen!();
    ?ob = dbcorr1b.UserData;
    ?ob.PutOnScreen!();}

    case: ?button == "Update";
    {select {

      case: ListLength(all dbcorr1b.Values) > 1;
```

```

        {msgtd.Values = "Only one correlation can be updated each time.\nPlease deselect all
others, but the one needs\nto br updated.";}
        Msgwin.PutOnScreenAndWait!();}

    case: ListLength(all dbcorr1b.Values) < 1;
        {msgtd.Values = "Please select first the correlation to update.";}
        Msgwin.PutOnScreenAndWait!();}

    otherwise: {find ?ob.Parameters_needed == dbcorr1b.Values;
                dbcorr.TakeOffScreen!();
                Step2.UserData..control = dbcorr@;
                upcor(?ob);}}}}
}
/*****
function create_db(?obje, ?posx, ?posy)
{
    bound inputs;
    /* This function creates the dialog boxes and their components for each selected correlation. */

    ?winname = ?obje.winname;
    MakeDialogBox(correl1 @, ?winname);
    ?winname.Identifier = ?winname;
    ?winname.Title = ?obje.wintitle;
    ?winname.PositionX = ?posx;
    ?winname.PositionY = ?posy;

    for find ?obje.?s1NBV..BV_description == Quantitative;
    do {make_eb(?obje, ?s1NBV, ?winname@);}

    for find ?obje.?s1VBV..BV_description == Qualitative;
    do {?num_of_lines = ListLength(?obje.?s1VBV..per_val);
        if ?num_of_lines > 4;
        {?num_of_lines = 4;}
        else {;}
        make_lb(?obje, ?s1VBV, ?num_of_lines, ?winname@);}

    for find ?obje.?s1NIV..IV_description == Quantitative;
    do {?td_name = ?obje.?s1NIV..tdname;
        MakeDialogBoxControl(TextDisplay, correl1 @, ?td_name);
        ?td_name.Identifier = ?obje.?s1NIV..ebname;
        if ?obje.?s1NIV..units != "";
        {?title = AppendStrings(?obje.?s1NIV..variable, " (", ?obje.?s1NIV..units, ")");}
        else {?title = ?obje.?s1NIV..variable;}
        string_man(?title, ?td_name, Title, 40);
        ?td_name.DialogBox = ?winname@;
        ?winname@.UserData += ?td_name@;
        ?td_name.UserData = `";}

    for find ?obje.?s1NIP..IP_description == Quantitative;
    do {make_td(?obje, ?s1NIP, tdname, ?winname@);
        make_td(?obje, ?s1NIP, tadmin, ?winname@);
        make_td(?obje, ?s1NIP, tdmx, ?winname@);
        make_td(?obje, ?s1NIP, tdav, ?winname@);
        make_td(?obje, ?s1NIP, tdmam, ?winname@);}

    for find ?obje.?s1NIP..IP_description == Qualitative;
    do {make_td(?obje, ?s1NIP, tdname, ?winname@);}
    if ?obje.Parameter..format == Quantitative;

```

```

{make_td(?obje, Parameter, tname, ?winname@);
make_td(?obje, Parameter, tmin, ?winname@);
make_td(?obje, Parameter, tmax, ?winname@);
make_td(?obje, Parameter, tdav, ?winname@);
make_td(?obje, Parameter, tdmam, ?winname@);}
else {make_td(?obje, Parameter, tname, ?winname@);}

?cr = ?obje.Parameter.cr;
MakeDialogBoxControl(CommandRow, correll@, ?cr);
?cr.Identifier = ?cr;
?cr.ButtonLabels = "Dismiss";
for ?z inlist `("Comments", "Reliability", "Applicability", "Estimate");
do {?cr.ButtonLabels += ?z;}
?cr.DefaultButton = "Estimate";
?cr.DialogBox = ?winname@;
?cr.React! = `?activate.React!;
?winname@.UserData += ?cr@;
?cr.UserData = `"";
for ?con inlist all ?winname@.UserData;
do {?winname@.Contents += ?con;}
?winname@.UserData = ?obje;
?winname@.PutOnScreen!();
}
/*-----*/
function make_eb(?obje, ?sl, ?winname)
{
bound inputs;
/* This function creates the entry boxes (used for representing
basic quantitative variables) for each correlation. */

?eb = ?obje.?sl.ebname;
MakeDialogBoxControl(EntryBox, correll@, ?eb);
?eb.Identifier = ?eb;
if {IsFacet(?obje, ?sl, BV_description);
?sl@.format.units != "";}
{?str = AppendStrings(?obje.?sl.variable, "(", ?sl@.format.units, ")");}
else {?str = ?obje.?sl.variable;}
string_man(?str, ?eb, Title, 40);
?eb.DefaultValues = ?obje.?sl;
?eb.DialogBox = ?winname@;
?eb.ReturnType = PrkType;
?eb.Width = 30;
?winname@.UserData += ?eb@;
?eb.UserData = ?sl;
}
/*-----*/
function make_lb(?obje, ?sl, ?num_of_lines, ?winname)
{
bound inputs;
/* This function creates the list boxes (used for representing
basic qualitative variables) for each correlation. */

?lb = ?obje.?sl.lname;
MakeDialogBoxControl(ListBox, correll@, ?lb);
?lb.Identifier = ?lb;
string_man(?obje.?sl.variable, ?lb, Title, 40);
for ?xv inlist ?obje.?sl.per_val;
do {?lb.SelectionItems += ?xv;}
}

```

```

?lb.DefaultValues = ?obje.?sl;
?lb.DialogBox = ?winname@;
?lb.MaxNumOfLines = ?num_of_lines;
?winname@.UserData += ?lb@;
?lb.UserData = ?sl;
}
/*-----*/
function make_td(?obje, ?sl, ?td, ?winname)
{
    bound inputs;
    /*This function creates the necessary text displays for each correlation. */

    select {

        case: ?sl == Parameter;
            ?par = ?obje.Parameter;
        otherwise: ?par = ?obje.?sl.parameter;}

    select {

        case: ?td == tadmin;
            ?title = "min: ";
        case: ?td == tdmx;
            ?title = "max: ";
        case: ?td == tday;
            ?title = "average:";
        case: ?td == tdmam;
            ?title = "Overall min, mean, max:";
        otherwise: {?title = AppendStrings("\n", ?par);}

    }

    if ?sl == Parameter;
    {?td_name == ?obje.Parameter..?td;}
    else {?td_name == ?obje.?sl..?td;}
    MakeDialogBoxControl(TextDisplay, correll @, ?td_name);
    ?td_name.Identifier = ?td_name;
    string_man(?title, ?td_name, Title, 40);
    ?td_name.MaxNumLines = 15;
    ?td_name.DialogBox = ?winname@;
    ?winname@.UserData += ?td_name@;
    ?td_name.UserData = `"";
}
/*-----*/
method activate.React!(?button)
{
    bound inputs;
    /* The method attached to the command row of each correlation dialog box window. */

    ?corr_obj = ?self.DialogBox.UserData;
    select{

        case: ?button == "Estimate";
            {for ?x inlist all ?self.DialogBox.Contents;
            do {if IsSlot(?corr_obj, ?x.UserData, `SV);
            { ?slo = ?x.UserData;
            ?corr_obj.?slo = find ?x.Values;}
            else {;}
            }
            ?corr_obj.`"Data_Check!";
}

```

```

for find ?corr_obj.?sloo..IV_description == Quantitative;
do {?ebname = ?corr_obj.?sloo..tdname;
    ?ebname.Values = ?corr_obj.?sloo;}
for find ?corr_obj.?sloo..IP_description == Quantitative;
do {par_result(?corr_obj, ?sloo);}
for find ?corr_obj.?sloo..IP_description == Qualitative;
do {?tdname = ?corr_obj.?sloo..tdname;
    ?tdname.Values = all ?corr_obj.?sloo;}
if ?corr_obj.Parameter..format == Quantitative;
{par_result(?corr_obj, Parameter);}
else {?tdname = ?corr_obj.Parameter..tdname;
    ?sl1 = ?corr_obj.Parameter..parameter;
    ?tdname.Values = all ?corr_obj.?sl1;}}

case: ?button == "Dismiss";
    ?self.DialogBox.TakeOffScreen!();

case: ?button == "Comments";
    {corcomm.Title = "Comments";
    com_creation (?corr_obj);}

case: ?button == "Applicability";
    {applipre(?self.DialogBox.UserData);
    corcomtd.Values = msgtd.Values;
    corcomm.Title = "Applicability";
    corcomm.PutOnScreen!();}

case: ?button == "Reliability";
    {corcomm.Title = "Reliability";
    corcomtd.Values = AppendStrings("Reliability score: ", ?corr_obj.Reliability);
    if find ?corr_obj.Reliability..r2 == ?;
        {corcomtd.Values = AppendStrings(corcomtd.Values, "\n\n", "Coefficient of fit: ",
            ConvertToString(?corr_obj.Reliability..r2), "%");}
    else{;}
    if find ?corr_obj.Reliability..sd == ?;
        {corcomtd.Values = AppendStrings(corcomtd.Values, "\n\n", "Standard deviation: +-",
            ConvertToString(?corr_obj.Reliability..sd));}
    else{;}
    if find ?corr_obj.Reliability..n == ?;
        {corcomtd.Values = AppendStrings(corcomtd.Values, "\n\n", "Number of data points: ",
            ConvertToString(?corr_obj.Reliability..n));}
    else{;}
    corcomm.PutOnScreen!();}
}
/*****
function par_result(?corr_obj, ?sloo)
{
    bound inputs;
    /* This function retrieves the correlation results from the parameters
        slots and places them inside the correlation dialog box window. */

    ?tdmin = ?corr_obj.?sloo..tdmin;
    ?tdmax = ?corr_obj.?sloo..tdmax;
    ?tdav = ?corr_obj.?sloo..tdav;
    ?tdmam = ?corr_obj.?sloo..tdmam;
    ?tdmin@.Values = Null;
    ?tdmax@.Values = Null;
    ?tdav@.Values = Null;
}

```

```

if ?sloo == Parameter;
{?slo = ?corr_obj.Parameter..parameter;}
else{?slo = ?sloo;}
?val_list = all ?corr_obj.?slo;

for ?i from 0 to ListLength(?val_list)-1;
do {?tdmin@.Values += ListFirst(ListNth(?val_list, ListLength(?val_list)-1-?i));
    ?tdmax@.Values += ListNth(ListNth(?val_list, ListLength(?val_list)-1-?i), 2);
    ?tdav@.Values += ListNth(ListNth(?val_list, ListLength(?val_list)-1-?i), 1);}

find ?av = sum ?tdav.Values;
find ?num = count ?tdav.Values;

if ListLength(all ?tdmin@.Values) > 0;
{?tdmin@.Values = all ?tdmin@.Values;}
else {?tdmin@.Values = Null;}

if ListLength(all ?tdmax@.Values) > 0;
{?tdmax@.Values = all ?tdmax@.Values;}
else {?tdmax@.Values = Null;}

if ListLength(all ?tdav@.Values) > 0;
{?tdav@.Values = all ?tdav@.Values;}
else {?tdav@.Values = Null;}

if ListLength(all ?tdmin@.Values) > 0;
{?min_list = Sort(?tdmin.Values, ">");
?max_list = Sort(?tdmax.Values, "<");

select {

case: ?corr_obj.?sloo..num_of_dec == 0;
{?av = ?av/?num;
?av = Format0(?av);}

case: ?corr_obj.?sloo..num_of_dec == 1;
{?av = ?av/?num;
?av = Format(?av);}

case: ?corr_obj.?sloo..num_of_dec == 2;
{?av = ?av/?num;
?av = Format2(?av);}

case: ?corr_obj.?sloo..num_of_dec == 3;
{?av = ?av/?num;
?av = Format3(?av);}

otherwise: ?av = ?av;
}
?tdmam.Values = `(ListFirst(?min_list), ?av, ListFirst(?max_list));}
else {?tdmam.Values = Null;}
}
/*-----*/
function com_creation (?obje)
{
bound inputs;
/* This function creates the comments in the "comments" dialog box. */

```



```

?string = "";
?string l = "";

for find or {?obje.?sl..BV_description == Quantitative;
             ?obje.?sl..IV_description == Quantitative;}
do {
  select {

    case: {IsNumber(?obje.?sl..max_value);
          IsNumber(?obje.?sl..min_value);
          IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be between ",
                                ConvertToString(?obje.?sl..min_value), " and ",
                                ConvertToString(?obje.?sl..max_value), " ",
                                ?obje.?sl..units, ". ");}

    case: {IsNumber(?obje.?sl..max_value);
          IsNumber(?obje.?sl..min_value);
          not IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be between ",
                                ConvertToString(?obje.?sl..min_value), " and ",
                                ConvertToString(?obje.?sl..max_value), ". ");}

    case: {IsNumber(?obje.?sl..max_value);
          not IsNumber(?obje.?sl..min_value);
          not IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be less than ",
                                ConvertToString(?obje.?sl..max_value), ". ");}

    case: {not IsNumber(?obje.?sl..max_value);
          not IsNumber(?obje.?sl..min_value);
          not IsString(?obje.?sl..units);}
        {?string = "";}

    case: {not IsNumber(?obje.?sl..max_value);
          IsNumber(?obje.?sl..min_value);
          not IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be more than ",
                                ConvertToString(?obje.?sl..min_value), ". ");}

    case: {IsNumber(?obje.?sl..max_value);
          not IsNumber(?obje.?sl..min_value);
          IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be less than ",
                                ConvertToString(?obje.?sl..max_value), " ",
                                ?obje.?sl..units, ". ");}

    case: {not IsNumber(?obje.?sl..max_value);
          IsNumber(?obje.?sl..min_value);
          IsString(?obje.?sl..units);}
        {?string = AppendStrings(?obje.?sl..variable, " should be more than ",
                                ConvertToString(?obje.?sl..min_value), " ",
                                ?obje.?sl..units, ". ");}

    case: {not IsNumber(?obje.?sl..max_value);
          not IsNumber(?obje.?sl..min_value);
          IsString(?obje.?sl..units);}
  }
}

```

```

                {?string = AppendStrings(?obje.?sl..variable, "is expressed in ",
                ?obje.?sl..units, ". ");}
            }
            string_man(?string, step3td, Values, 50);
            ?string1 = AppendStrings(?string1, step3td.Values, "\n");
        }
    if ?string1 != "";
    {?string1 = AppendStrings(?string1, "\n");}
    else {;}

    if ListLength(all ?obje.Comments) > 0;
    {for ?x inlist all ?obje.Comments;
    do {?string1 = AppendStrings(?string1, ?x);}
    else {if ?string1 != "";
        {?string1 = Substring(?string1, 0, StringLength(?string1)-2);}
        else {;}}

    corcomtd.Values = ?string1;
    corcomm.PutOnScreen!();
}
/*****

```

UPSTEP1.ptk

```

/* This file contains the functions and methods relevant to the first stage of implementing
correlations. */

```

```

#include <prk/math.pth>
#include <prk/lib.pth>
/*****
function upcor(?name)
{
    bound inputs;
    /* This function initiates the updating procedure for an already existing correlation. */

    PrintLine(?name);
    Step2.UserData..name = ConvertToSymbol(?name);
    RenameObject(ConvertToSymbol(?name), correl_temp);
    PrintLine(?name);
    Step2aueb.DefaultValues = correl_temp.wintitle..author;
    Step2.PutOnScreen!();
}
/*****
function startup()
{
    bound inputs;
    /* This function initiates the procedure for implementing a new correlation. */

    startup1();
    Step2.UserData..control = Step1@;
    Step1.PutOnScreen!();
}
/*****
function startup1 ()
{
    bound inputs;

```

```
/* The second stage in the initiation of the procedure for implementing a new correlation. */
```

```
msgcr.ButtonLabels = "Dismiss";
Step_1_lb_1.UserData..other = Null;
Step2.UserData = 0;
Step2lb1.SelectionItems = Null;
Step2lb1.UserData = Null;
Step2lb2.SelectionItems = Null;
Step2lb2.UserData = Null;
lbslo(Step2lb1@, BV_description, Quantitative, variable);
lbslo(Step2lb2@, BV_description, Qualitative, variable);
lbslo(nviplb@, IP_description, Quantitative, parameter);
lbslo(nviplb@, IP_description, Qualitative, parameter);
s2lb12var (Step2lb1@, Quantitative, qtim, qtot);
s2lb12var (Step2lb2@, Qualitative, qlim, qlot);
lbst1();

for {find ?ob = subclassof Parameters@;
    ListLength(all superclassof ?ob) > 1;}
do {if not {ListLength(FindListElmt(all Step_1_lb_1.SelectionItems,
    ?ob.name)) > 0;}
    {Step_1_lb_1.UserData..other += ?ob.name;}
    else {;}}
Step_1_lb_1.UserData..other = SortList(all Step_1_lb_1.UserData..other, Alphabetize);
}
/*****
function lbslo (?lb, ?type, ?format, ?vp)
{
    bound inputs;
    /* Sets the lists displaying the implemented variables
        and parameters (quantitative and qualitative). */

    ?list = `();
    for find ?obje.winname == ?;
    do {for find ?obje.?slo..?type == ?format;
        do {?list = AppendLists(?list, `(?obje.?slo..?vp));}}
    ?list = SortList(?list, Alphabetize);

    select {

        case: {or {?lb == Step2lb1@;
            ?lb == Step2lb2@;}}
            {for ?i from 0 to ListLength(?list)-1;
                do {?lb.SelectionItems += ListNth(?list, ListLength(?list)-1-?i);}}

        otherwise: {if ?format == Quantitative;
            {?lb.UserData..qtim = ?list;}
            else {?lb.UserData..qlim = ?list;}}
    }
/*****
function s2lb12var (?lbbv, ?for, ?facim, ?facot)
{
    bound inputs;
    /* Sets the lists displaying the remaining variables (quantitative and qualitative). */

    ?list = `();
    for {find ?ob = subclassof Parameters@;
        ListLength(all superclassof ?ob) > 1;
```

```

        ?ob.format == ?for;}
do {if not {ListLength(FindListElmt(all ?lbbv.SelectionItems,
                                ?ob.name)) > 0;}
    {?lbbv.UserData += ?ob.name;}
    else {;}
if not {ListLength(FindListElmt(nviplb.UserData.?facim,
                                ?ob.name)) > 0;}
    {?list = AppendLists(?list, `(?ob.name));}
    else {;}}
?lbbv.UserData = SortList(all ?lbbv.UserData, Alphabetize);
nviplb.UserData.?facot = ?list;
}
/*****
function lbst1 ()
{
    bound inputs;
    /*Sets the lists displaying the implemented and the remaining
       basic parameters (quantitative and qualitative). */

    ?list = `();
    Step_1_lb_1.SelectionItems = Null;
    for {find ?ob.Parameter == ?;
        or {ObjectModule(?ob) == correlation@;
            ObjectModule(?ob) == correction@;}}
    do {?list = AppendLists(?list, `(?ob.Parameter));}
    ?l = SortList(?list, Alphabetize);
    for ?i from 0 to ListLength(?l)-1;
    do {Step_1_lb_1.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}
}
/*****
method Step_1_lb_1.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the list box of the "Step1" dialog box. */

    select{

    case:{IsString(?new_value);
        find1 ?ob.name == ?new_value;
        ?ob.format == Quantitative;}
        {msgtd.Values = AppendStrings(?ob.name,
            "\nFormat: ", ConvertToString(?ob.format),
            ".\nUnits: ", ?ob.format.units,
            ".\nNumber of decimal points: ",
            ConvertToString(?ob.format.num_of_dec),".");

        msgcr.ButtonLabels = "Cancel";
        msgcr.ButtonLabels += "OK";
        Msgwin.PutOnScreenAndWait!();}

    case:{IsString(?new_value);
        find1 ?ob.name == ?new_value;
        ?ob.format == Qualitative;}
        {msgtd.Values = AppendStrings(?ob.name,
            "\nFormat: ", ConvertToString(?ob.format));

        msgcr.ButtonLabels = "Cancel";
        msgcr.ButtonLabels += "OK";
        Msgwin.PutOnScreenAndWait!();}
}

```

```

/*****
method Step1rb1.React! (?moused_item, ?old_item)
{
  bound inputs;
  /* The method attached to the radio buttons of the "Step1" dialog box. */

  select {

    case: ?moused_item == "Show implemented";
      lbst1();

    case: ?moused_item == "Show other";
      {Step_1_lb_1.SelectionItems = Null;
       ?l = Step_1_lb_1.UserData.other;
       for ?i from 0 to ListLength(?l)-1;
       do {Step_1_lb_1.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}}
  }
}
*****/
method Step1cr.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "Step1" dialog box. */

  select {

    case: ?button == "OK";
      step1OK("OK");

    case: ?button == "Help";
      {Step1Help ();
       helptd.Values = Step1cr.UserData;
       help.PutOnScreen!();}

    otherwise: {Step1.TakeOffScreen!();
               mmen.PutOnScreen!();}
               help.TakeOffScreen!();}
  }
}
*****/
function step1OK (?button)
{
  bound inputs;
  /* This function performs the actions associated with the "OK"
  button of the command row of the "Step1" dialog box. */

  if Step_1_rb_1.Values == "New Correlation";
  {?module = correlation@;}
  else {?module = correction@;}

  if {IsString(Step_1_lb_1.Values);
     find1 ?par.name == Step_1_lb_1.Values;}
  {if ?par.format == Quantitative;

   {?dec = ?par.format.num_of_dec;
    ?units = ?par.format.units;
    Step2lb1.SelectionItems -= ?par.name;

   if ListFirst(Step2lb1.UserData) == ?par.name;
   {Step2lb1.UserData = ListRest(Step2lb1.UserData);}
   else {Step2lb1.UserData = DeleteListElmt(?par.name, Step2lb1.UserData);}
  }
  }
}

```

```

if ListFirst(nviplb.UserData..qtim) == ?par.name;
{nviplb.UserData..qtim = ListRest(nviplb.UserData..qtim);}
else {nviplb.UserData..qtim = DeleteListElmt(?par.name,
nviplb.UserData..qtim);}

if ListFirst(nviplb.UserData..qtot) == ?par.name;
{nviplb.UserData..qtot = ListRest(nviplb.UserData..qtot);}
else {nviplb.UserData..qtot = DeleteListElmt(?par.name,
nviplb.UserData..qtot);}}

else {?dec = no;
Step2lb2.SelectionItems -== ?par.name;

if ListFirst(Step2lb2.UserData) == ?par.name;
{Step2lb2.UserData = ListRest(Step2lb2.UserData);}
else {Step2lb2.UserData = DeleteListElmt(?par.name,
Step2lb2.UserData);}

if ListFirst(nviplb.UserData..qlim) == ?par.name;
{nviplb.UserData..qlim = ListRest(nviplb.UserData..qlim);}
else {nviplb.UserData..qlim = DeleteListElmt(?par.name,
nviplb.UserData..qlim);}

if ListFirst(nviplb.UserData..qlot) == ?par.name;
{nviplb.UserData..qlot = ListRest(nviplb.UserData..qlot);}
else {nviplb.UserData..qlot = DeleteListElmt(?par.name,
nviplb.UserData..qlot);}}

else {msgtd.Values = "Please, select a Basic Parameter.";
Msgwin.PutOnScreenAndWait!();
fail;}

MakeClass(correl_temp, ?module, Correlations@, `());
MakeMultiValueSlot(correl_temp, ConvertToSymbol(?par));
correl_temp.Parameter = Step_1_lb_1.Values;
correl_temp.Parameter..format = ?par.format;
correl_temp.Parameter..parameter = ConvertToSymbol(?par);

if {?dec != no;}
{correl_temp.Parameter..num_of_dec = ?dec;
correl_temp.Parameter..units = ?units;}
else {;}

Step1.TakeOffScreen!();
help.TakeOffScreen!();
Step2aueb.DefaultValues = Null;
Step2aueb.Values = Null;
Step2.UserData..control = Step1@;
Step2.PutOnScreen!();
}
/*****
method Step1pb1.React! (?button)
{
bound inputs;
/* The method attached to the "Quantitative " and
"Qualitative " push buttons of the "Step1" dialog box. */

```

```

pba1(?button);
select {

    case: ?button == "Quantitative ";
        {nip.Title = "New Quantitative Parameter Definition";
        nip.PutOnScreenAndWait!();}

    case: ?button == "Qualitative ";
        {vip.Title = "New Qualitative Parameter Definition";
        vip.PutOnScreenAndWait!();}}
}
/*****
function pba1 (?button)
{
    bound inputs;
    /* Sets the dialog boxes for specifying a quantitative or
    qualitative new parameter (basic parameter of the correlation). */

    nlb2ma();
    select {

        case: ?button == "Quantitative ";
            {nip.Contents += newparlb1@;
            nip.Contents += nipom@;
            newparlb1.DialogBox = nip@;
            nipom.DialogBox = nip@;
            newparlb1.PositionY = 210;
            nipom.PositionY = 160;
            nipeb1.DefaultValues = Null;
            nipeb2.DefaultValues = Null;
            nipeb3.DefaultValues = Null;}

        case: ?button == "Qualitative ";
            {vip.Contents += newparlb1@;
            newparlb1.DialogBox = vip@;
            newparlb1.PositionY = 110;
            vipeb1.DefaultValues = Null;
            vipeb2.DefaultValues = Null;}}
    }
/*****
function nlb2ma ()
{
    bound inputs;
    /* Sets the lists with the parameter categories. */

    newparlb1.SelectionItems = Null;
    for find ?x = direct subclassof Parameters@;
    do {newparlb1.SelectionItems += ?x.name;}
}
/*****
function ipcheck (?lb1, ?eb2, ?eb1, ?ob)
{
    bound inputs;
    /* Performs checks in the user supplied data for the creation of a new basic parameter. */

    select {

        case: not {IsString(?lb1);}

```

```

        {msgtd.Values = "A category for the new parameter\nshould be specified.";
        Msgwin.PutOnScreenAndWait!();
        ?ob.UserData = 1;}
case:?eb2 == "";
    {msgtd.Values = "A shorthand description for the new parameter\nshould be specified.";
    Msgwin.PutOnScreenAndWait!();
    ?ob.UserData = 1;}

case:?eb1 == "";
    {msgtd.Values = "The full name of the new parameter\nshould be specified.";
    Msgwin.PutOnScreenAndWait!();
    ?ob.UserData = 1;}

    otherwise: ?ob.UserData = 0;}
}
/*****/
function Step1Help()
{
    bound inputs;
    /* The help text for the "Step1" dialog box. */

    Step1cr.UserData = AppendStrings(

"This dialog box is the first step in the procedure of defining a\n",
"new correlation, or correction.\n\n",
"In the case of a new correlation, select New Correlation (default value),\n",
"otherwise click on New Correction.\n\n",
"The next step is to define the basic parameter of the correlation\n",
"correction. This can be done by selecting one of the parameters\n",
"displayed in the list box. If a parameter is selected an Information\n",
>window, containing information about the selected parameter, will\n",
>appear on screen.\n\n",
>Initially the list box contains all the parameters that have been used\n",
>as basic parameters in implemented correlations and corrections.\n",
>In order to view all the other parameters, click on Show other and\n",
>select the desired parameter from the list box. If the desired parameter\n",
>is not contained in the list box, then click on Quantitative, to define a\n",
>new parameter of quantitative format, or on qualitative to define a\n",
>new parameter of qualitative format.");
}

```

UPSTEP2.ptk

```

/* This file contains the functions and methods relevant to the second stage of implementing
correlations. */

```

```

#include <prk/math.pth>
#include <prk/lib.pth>
/*****/

method Step2lb1.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the quantitative basic variables display dialog box. */

    nbv.UserData = Null;
}

```



```

setdef1();
if IsString(?new_value);
{find1 ?sl.name == ?new_value;
nbv.UserData = ?sl;
nbv.UserData..name = ?new_value;
nbv.UserData..units = ?sl.format..units;
nbvtd.Values = AppendStrings(?new_value, "\n\nUnits: ",
?sl.format..units);

if {?sl1 = ConvertToSymbol(?sl);
find1 ?ob.?sl1..variable == ?new_value;}
{nbveb3.DefaultValues = ?ob.?sl1..min_value;
nbveb4.DefaultValues = ?ob.?sl1..max_value;}
else {nbveb3.DefaultValues = Null;
nbveb4.DefaultValues = Null;}
nbv.PutOnScreenAndWait!();}
else {;}
}
/*****/
method rblb1.React! (?moused_item, ?old_item)
{
bound inputs;
/* The method attached to the "implemented", "other"
push buttons for quantitative basic variables. */

select {

case: {?moused_item == "implemented";
?old_item == "other";}
lbudo(Step2lb1);

case: {?moused_item == "other";
?old_item == "implemented";}
lbudo(Step2lb1);}
}
/*****/
method pblb1.React! (?value)
{
/* The method attached to the "Create a new Quantitative" basic variable push button. */

nbv.UserData = Null;
setdef1();
pba1("Quantitative ");
nip.Title = "New Quantitative Basic Variable Definition";
nip.PutOnScreenAndWait!();
}
/*****/
function setdef1 ()
{
bound inputs;
/* Sets the dialog box for specifying a new basic quantitative variable. */

nbv.Contents -== newparlb1@;
nbv.Contents -== nipom@;
nbvcr.ButtonLabels -== "Remove Variable";
}
/*****/
method Step2lb2.React! (?new_value, ?old_value)
{

```

```

bound inputs;
/* The method attached to the qualitative basic variables display dialog box. */

vbv.UserData = Null;
setdef();
if IsString(?new_value);
{find1 ?sl.name == ?new_value;
vbv.UserData = ?sl;
vbv.UserData..name = ?new_value;
vbvtd.Values = ?new_value;
if {?sl1 = ConvertToSymbol(?sl);
    find1 ?ob.?sl1..variable == ?new_value;}
{for ?i from 0 to ListLength(?ob.?sl1..per_val)-1;
do {vbvlb.SelectionItems += ListNth(?ob.?sl1..per_val,
    ListLength(?ob.?sl1..per_val)-1-?i);}

for {?pv = find ?sl.format..per_val;
    ?pv != ?ob.?sl1..per_val;}
do {vbvlb.UserData += ?pv;}
vbvlb.UserData = all vbvlb.UserData;
vbvlb.UserData = AppendLists(vbvlb.UserData,
    `(?ob.?sl1..per_val));

if ListLength(vbvlb.UserData) > 1;
{vbvpb.ButtonLabel = "Show alternative set";}
else{;}}
else {?y = find1 ?sl.format..per_val;
    for ?i from 0 to ListLength(?y)-1;
do {vbvlb.SelectionItems += ListNth(?y, ListLength(?y)-1-?i);}
for {?pv = find ?sl.format..per_val;
    ?pv != ?y;}
do {vbvlb.UserData += ?pv;}
vbvlb.UserData = all vbvlb.UserData;
vbvlb.UserData = AppendLists(vbvlb.UserData, `(?y));
if ListLength(vbvlb.UserData) > 1;
{vbvpb.ButtonLabel = "Show alternative set";}
else{;}}
vbv.PutOnScreenAndWait!();}
else {;}
}
/*****/
method rblb2.React! (?moused_item, ?old_item)
{
bound inputs;
/* The method attached to the "implemented", "other"
push buttons for qualitative basic variables. */

select {

case: {?moused_item == "implemented";
    ?old_item == "other";}
lbudo(Step2Ib2);

case: {?moused_item == "other";
    ?old_item == "implemented";}
lbudo(Step2Ib2);}
}
/*****/
method pblb2.React! (?value)

```

```

{
  bound inputs;
  /* The method attached to the "Create a new Qualitative" basic variable push button. */

  vbv.UserData = Null;
  setdef();
  pba1("Qualitative ");
  vip.Title = "New Qualitative Basic Variable Definition";
  vip.PutOnScreenAndWait!();
}
/*****
function setdef ()
{
  /* Sets the dialog box for specifying a new basic qualitative variable. */

  vbv1b.UserData = Null;
  vbv1b.SelectionItems = Null;
  vbv1b.ButtonLabel = Null;
  vbv1b3.DefaultValues = Null;
  vbv1b.Contents -== newpar1b1 @;
  vbv1bcr.ButtonLabels -== "Remove Variable";
}
/*****
function lbudo (?lb)
{
  bound inputs;
  /* Performs list settings for the "implemented", "other"
  push buttons for qualitative basic variables. */

  ?list = all ?lb.SelectionItems;
  ?lb.SelectionItems = Null;
  ?l = SortList(?lb.UserData, Alphabetize);
  for ?i from 0 to ListLength(?l)-1;
  do {?lb.SelectionItems +== ListNth(?l, ListLength(?l)-1-?i);}
  ?lb.UserData = ?list;
}
/*****
method Step2pb1.React! (?button)
{
  /* The method attached to the push button for creating intermediate variables. */

  niv.PutOnScreenAndWait!();
}
/*****
method Step2pb2.React! (?button)
{
  bound inputs;
  /* The method attached to the push buttons for specifying intermediate parameters. */

  nviplb.SelectionItems = Null;
  select {

  case: ?button == "Quantitative";
  {nvip.Title = "Quantitative Intermediate Parameter Definition";
  for ?x inlist nviplb.UserData..qtim;
  do {nviplb.SelectionItems +== ?x;}
  nvippb.Title = "New Quantitative Intermediate Parameter:";}
}
}

```

```

    case: ?button == "Qualitative";
        {nvip.Title = "Qualitative Intermediate Parameter Definition";
        for ?x inlist nvipb.UserData..qlim;
        do {nvipb.SelectionItems += ?x;}
        nvipb.Title = "New Qualitative Intermediate Parameter:;"}

    nvip.PutOnScreenAndWait!();
}
/*****
method Step2cr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "Step2" dialog box. */

    select {

        case: ?button == "OK";
            {if findI correl_temp.?sl..BV_description == ?;
            {;}
            else {msgtd.Values = "Please select or define at least one\nBasic Variable (Quantitative or
Qualitative).";
            Msgwin.PutOnScreenAndWait!();
            fail;}
            if StringLength(Step2aueb.Values) > 0;
            {;}
            else {msgtd.Values = "Please specify the correlation/correction's reference.";
            Msgwin.PutOnScreenAndWait!();
            fail;}
            ?numlist = `();
            for {find ?ob.Parameter..parameter == correl_temp.Parameter..parameter;
            IsSymbol(?ob.winname);
            ?ob != correl_temp@;}
            do {?num = ConvertToNumber(Substring(ConvertToString(?ob.winname),
StringLength(ConvertToString(?ob.Parameter..parameter)))));
            collect ?num into ?numlist;}

            if ListLength(?numlist) > 0;
            {SortList(?numlist, "<");

                                correl_temp.winname =
ConvertToSymbol(AppendStrings(ConvertToString(correl_temp.Parameter..parameter),
ConvertToString(ListFirst(?numlist)+1)));}
                                {correl_temp.winname =
ConvertToSymbol(AppendStrings(ConvertToString(correl_temp.Parameter..parameter),
                                ConvertToString(1)));}

            ?numlist = `();}
            ?name = ConvertToString(correl_temp.winname);
            for find correl_temp.?nbv..BV_description == Quantitative;
            do {correl_temp.?nbv..ebname = ConvertToSymbol(AppendStrings(?name,
                                ConvertToString(?nbv)));}
            for find correl_temp.?vbv..BV_description == Qualitative;
            do {correl_temp.?vbv..lbname = ConvertToSymbol(AppendStrings(?name,
                                ConvertToString(?vbv)));}
            for find correl_temp.?niv..IV_description == Quantitative;
            do {correl_temp.?niv..tdname = ConvertToSymbol(AppendStrings(?name,
                                ConvertToString(?niv)));}
            for find correl_temp.?nip..IP_description == Quantitative;

```

```

do { correl_temp.?nip..tdmax = ConvertToSymbol(AppendStrings(?name,
                                                ConvertToString(?nip), "max"));
    correl_temp.?nip..tdmin = ConvertToSymbol(AppendStrings(?name,
                                                            ConvertToString(?nip), "minim"));
    correl_temp.?nip..tdmam = ConvertToSymbol(AppendStrings(?name,
                                                            ConvertToString(?nip), "mam"));
    correl_temp.?nip..tdav = ConvertToSymbol(AppendStrings(?name,
                                                            ConvertToString(?nip), "av" ));
    correl_temp.?nip..tdname = ConvertToSymbol(AppendStrings(?name,
                                                             ConvertToString(?nip), "nam"));}
for find correl_temp.?vip..IP_description == Qualitative;
do { correl_temp.?vip..tdname = ConvertToSymbol(AppendStrings(?name,
                                                             ConvertToString(?vip), "nam"));}

if correl_temp.Parameter..format == Quantitative;
{ correl_temp.Parameter..tdmax = ConvertToSymbol(AppendStrings(?name, "max"));
  correl_temp.Parameter..tdmin = ConvertToSymbol(AppendStrings(?name, "minim"));
  correl_temp.Parameter..tdmam = ConvertToSymbol(AppendStrings(?name, "mam"));
  correl_temp.Parameter..tdav = ConvertToSymbol(AppendStrings(?name, "av"));}
else {};
correl_temp.Parameter..tdname = ConvertToSymbol(AppendStrings(?name, "nam"));
correl_temp.Parameter..cr = ConvertToSymbol(AppendStrings(?name, "cr"));
correl_temp.Comments..tdname = ConvertToSymbol(AppendStrings("com", ?name, "td"));
?obname = correl_temp.winname;
correl_temp.wintitle = AppendStrings(ConvertToString(correl_temp.Parameter..parameter),
                                     " = f(");
correl_temp.Parameters_needed = AppendStrings(ConvertToString(
                                             correl_temp.Parameter..parameter), " from ");
for find correl_temp.?bv..BV_description == ?;
do { ?obname = AppendStrings(?obname, "_", ConvertToString(?bv));
    correl_temp.wintitle = AppendStrings(correl_temp.wintitle, ConvertToString(?bv), ", ");
    if StringLength(correl_temp.Parameters_needed) > 40;
    { correl_temp.Parameters_needed = AppendStrings(correl_temp.Parameters_needed,
                                                    ConvertToString(?bv), ", ");}
    else { correl_temp.Parameters_needed = AppendStrings(correl_temp.Parameters_needed,
                                                         correl_temp.?bv..variable, ", ");}
correl_temp.Parameters_needed = AppendStrings(correl_temp.Parameters_needed,
                                             Step2aueb.Values);
correl_temp.wintitle = AppendStrings(Substring(correl_temp.wintitle, 0,
                                             StringLength(correl_temp.wintitle)-2), ", ", Step2aueb.Values);
correl_temp.wintitle..author = Step2aueb.Values;
correl_temp.Parameters_needed = AppendStrings( correl_temp.Parameters_needed, " (",
                                             ConvertToString(ListFirst(?numlist)+1), ").");
?obname = ConvertToSymbol(?obname);
RenameObject(correl_temp, ?obname);

Step2.TakeOffScreen!();
for ?z inlist `(appli0cr@, reliacr@,step3cr@);
do { ?z.UserData = ?obname@;}
parpb("Reset ", appli0lb1 @, Ground@);
parpb("Reset ", appli1lb1 @, Parameters@);
corrpre.UserData = Null;
calestwin(?obname@);
}

case: ?button == "Preview";
{corrpre.TakeOffScreen!();
  corrpre.UserData = correl_temp@;
  corrpre.Contents = precr@;
}

```

```

corrpre.Contents += ppre@;
ppre.SelectionItems = correl_temp.Parameter;
for ?x inlist `(nbvpre@, vbvpre@, nivpre@, nippre@, vippre@);
do {?x.SelectionItems = Null;
    ?x.MaxNumOfLines = 1;}
for find correl_temp.?nbv..BV_description == Quantitative;
do {nbvpre.SelectionItems += correl_temp.?nbv..variable;
    nbvpre.MaxNumOfLines = nbvpre.MaxNumOfLines + 1;}
for find correl_temp.?vbv..BV_description == Qualitative;
do {vbvpre.SelectionItems += correl_temp.?vbv..variable;
    vbvpre.MaxNumOfLines = vbvpre.MaxNumOfLines + 1;}
for find correl_temp.?niv..IV_description == Quantitative;
do {nivpre.SelectionItems += correl_temp.?niv..variable;
    nivpre.MaxNumOfLines = nivpre.MaxNumOfLines + 1;}
for find correl_temp.?nip..IP_description == Quantitative;
do {nippre.SelectionItems += correl_temp.?nip..parameter;
    nippre.MaxNumOfLines = nippre.MaxNumOfLines + 1;}
for find correl_temp.?vip..IP_description == Qualitative;
do {vippre.SelectionItems += correl_temp.?vip..parameter;
    vippre.MaxNumOfLines = vippre.MaxNumOfLines + 1;}
for {?x inlist `(nbvpre@, vbvpre@, nivpre@, nippre@, vippre@);
    ?x.MaxNumOfLines > 1;}
do {corrpre.Contents += ?x;}
if nbvpre.MaxNumOfLines == 1;
{vbvpre.PositionY = nbvpre.PositionY;}
else {vbvpre.PositionY = nbvpre.PositionY + (nbvpre.MaxNumOfLines-1)*20 + 30;}

if vbvpre.MaxNumOfLines == 1;
{if nbvpre.MaxNumOfLines == 1;
    {nivpre.PositionY = nbvpre.PositionY;}
else {nivpre.PositionY = nbvpre.PositionY + (nbvpre.MaxNumOfLines-1)*20 + 30;}}
else {nivpre.PositionY = vbvpre.PositionY + (vbvpre.MaxNumOfLines-1)*20 + 30;}
if nivpre.MaxNumOfLines == 1;
{if vbvpre.MaxNumOfLines == 1;
    {nippre.PositionY = vbvpre.PositionY;}
else {nippre.PositionY = vbvpre.PositionY + (vbvpre.MaxNumOfLines-1)*20 + 30;}}
else {nippre.PositionY = nivpre.PositionY + (nivpre.MaxNumOfLines-1)*20 + 30;}
if nippre.MaxNumOfLines == 1;
{if nivpre.MaxNumOfLines == 1;
    {vippre.PositionY = nivpre.PositionY;}
else {vippre.PositionY = nivpre.PositionY + (nivpre.MaxNumOfLines-1)*20 + 30;}}
else {vippre.PositionY = nippre.PositionY + (nippre.MaxNumOfLines-1)*20 + 30;}
for {?x inlist all corrpre.Contents;
    ?x != precr@;
    ?x != ppre@;}
do {?x.MaxNumOfLines = ?x.MaxNumOfLines - 1;}
corrpre.PutOnScreenAndWait!();}

case: ?button == "Cancel";
{if IsFacet (Step2@, UserData, name);
    {RenameObject(correl_temp, Step2.UserData..name);
    DeleteFacet(Step2@, UserData, name);}
else {DeleteObject(correl_temp);}
Step2.TakeOffScreen!();
?ob = Step2.UserData..control;
?ob.PutOnScreen!();}
}
}

```

```

/*****/
method ppre.React! (?new_value, ?old_value)
{
  /* The method attached to "parameter preview" list box. */

  lbfuc(?new_value);
}
/*****/
method nbvpre.React! (?new_value, ?old_value)
{
  bound inputs;

  /* The method attached to "quantitative variables preview" list box. */

  find1 corrpre.UserData.?sl..variable == ?new_value;
  nbv.UserData = ?sl;
  nbv.UserData..name = ?new_value;
  nbv.UserData..units = corrpre.UserData.?sl..units;
  nbvtd.Values = AppendStrings(?new_value, "\n\nUnits: ",
                                corrpre.UserData.?sl..units);
  nbveb3.DefaultValues = corrpre.UserData.?sl..min_value;
  nbveb4.DefaultValues = corrpre.UserData.?sl..max_value;
  nbvcr.ButtonLabels += "Remove Variable";
  nbv.PutOnScreenAndWait!();
}
/*****/
method vbvpre.React! (?new_value, ?old_value)
{
  bound inputs;
  /* The method attached to "qualitative variables preview" list box. */

  find1 corrpre.UserData.?sl..variable == ?new_value;
  vbv.UserData = ?sl;
  vbv.UserData..name = ?new_value;
  vbvtd.Values = ?new_value;
  vbvlb.SelectionItems = Null;
  for ?i from 0 to ListLength(corrpre.UserData.?sl..per_val)-1;
  do { vbvlb.SelectionItems += ListNth(corrpre.UserData.?sl..per_val,
                                       ListLength(corrpre.UserData.?sl..per_val)-1-?i);}

  vbvcr.ButtonLabels += "Remove Variable";
  vbv.PutOnScreenAndWait!();
}
/*****/
method nivpre.React! (?new_value, ?old_value)
{
  bound inputs;
  /* The method attached to "intermediate variables preview" list box. */

  find1 corrpre.UserData.?sl..variable == ?new_value;
  nbv.UserData = ?sl;
  nbv.UserData..name = ?new_value;
  nbv.UserData..units = corrpre.UserData.?sl..units;
  nbvtd.Values = AppendStrings(?new_value, "\n\nUnits: ",
                                corrpre.UserData.?sl..units);
  nbveb3.DefaultValues = corrpre.UserData.?sl..min_value;
  nbveb4.DefaultValues = corrpre.UserData.?sl..max_value;
  nbvcr.ButtonLabels += "Remove Variable";
}

```

```

    nbv.PutOnScreenAndWait!();
}
/*****
method nipre.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to "quantitative interemediate parameters preview" list box. */

    ipdel(?new_value, ?old_value);
}
/*****
method vipre.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to "qualitative interemediate parameters preview" list box. */

    ipdel(?new_value, ?old_value);
}
/*****
function ipdel (?new_value, ?old_value)
{
    bound inputs;
    /* The function for removing variables and parameters. */

    find1 corrpri.UserData.?sl.parameter == ?new_value;
    Msgwin.UserData = ?sl;
    msgcr.ButtonLabels += "Remove Parameter";
    select{
        case:{IsString(?new_value);
            find1 ?ob.name == ?new_value;
            ?ob.format == Quantitative;}
            {msgtd.Values = AppendStrings(?ob.name,
                "\nFormat: ", ConvertToString(?ob.format),
                "\nUnits: ", ?ob.format.units,
                "\nNumber of decimal points: ",
                ConvertToString(?ob.format.num_of_dec),".");
            Msgwin.PutOnScreenAndWait!();}
        case:{IsString(?new_value);
            find1 ?ob.name == ?new_value;
            ?ob.format == Qualitative;}
            {msgtd.Values = AppendStrings(?ob.name,
                "\nFormat: ", ConvertToString(?ob.format));
            Msgwin.PutOnScreenAndWait!();}
    }
/*****
method warncr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "warning" dialog box. */

    select {

        case: {?button == "Continue";
            IsList(warncr.UserData);}
            {?ob = ListFirst(warncr.UserData);
            if ListNth(warncr.UserData, 1) == 1;
            {datacheck2 (?ob);}
            else {calculatef(?ob, `Parameter);}}
    }

```



```

case: {?button == "Delete";
    IsSymbol(warncr.UserData);
    ?sl = warncr.UserData;
    corrpre.UserData.?sl..BV_description == Quantitative;}
sldel(?sl, Step2lb1@, nbv@, nbvpre@);

case: {?button == "Delete";
    IsSymbol(warncr.UserData);
    ?sl = warncr.UserData;
    corrpre.UserData.?sl..BV_description == Qualitative;}
sldel(?sl, Step2lb2@, vbv@, vbvpre@);

case: {?button == "Delete";
    IsSymbol(warncr.UserData);
    ?sl = warncr.UserData;
    corrpre.UserData.?sl..IV_description == Quantitative;}
{DeleteSlot(corrpre.UserData, ?sl);
nbv.TakeOffScreen!();
if ListLength(all nivpre.SelectionItems) > 1;
{nivpre.SelectionItems -== nbv.UserData..name;}
else {nivpre.SelectionItems = Null;}}

case: {?button == "Delete";
    IsSymbol(warncr.UserData);
    ?sl = warncr.UserData;
    corrpre.UserData.?sl..IP_description == Quantitative;}
{sldelp(?sl, nipre@);}

case: {?button == "Delete";
    IsSymbol(warncr.UserData);
    ?sl = warncr.UserData;
    corrpre.UserData.?sl..IP_description == Qualitative;}
{sldelp(?sl, vipre@);}
}

Warningwin.TakeOffScreen!();
warncr.ButtonLabels -== "Delete";
warncr.ButtonLabels +== "Continue";
warncr.UserData = Null;
}
/*****
function sldel (?sl, ?lb, ?win, ?lb1)
{
    bound inputs;
    /* The function that deletes the slot representing the selected for removal variable. */

    ?name = corrpre.UserData.?sl..variable;
    find ?ob.name == ?name;
    restor(?sl, ?name, ?ob, ?lb);
    DeleteSlot(corrpre.UserData, ?sl);
    ?win.TakeOffScreen!();
    if ListLength(all ?lb1.SelectionItems) > 1;
    {?lb1.SelectionItems -== ?win.UserData..name;}
    else {?lb1.SelectionItems = Null;}
}
/*****
function sldelp (?sl, ?lb)

```

```

{
  bound inputs;
  /* The function that deletes the slot representing the selected for removal parameter. */

  ?name = corrpre.UserData.?sl..parameter;
  ?si = corrpre.UserData.?sl..parameter;
  find ?ob.name == ?name;
  DeleteSlot(corrpre.UserData, ?sl);
  if ?ob.format == Quantitative;
  { ?lb1 = Step2lb1@; }
  else { ?lb1 = Step2lb2@; }
  restor(?sl, ?name, ?ob, ?lb1);
  if ListLength(all ?lb.SelectionItems) > 1;
  { ?lb.SelectionItems -== ?si; }
  else { ?lb.SelectionItems = Null; }
}
/*****
function restor (?sl, ?name, ?ob, ?lb)
{
  bound inputs;
  /* This function restores the names of the removed variables/parameters
     to the appropriate lists, so that they can be reselected. */

  if {?x = find subclassof Correlations@;
      IsSlot(?x, ?sl, `SV);}
  {if find1 ?lb.SelectionItems == "other";
   { ?lb.SelectionItems +== ?name; }
   else { ?lb.UserData = AppendLists(?lb.UserData, `(?name));}}
  else {if find1 ?lb.SelectionItems == "other";
   { ?lb.UserData = AppendLists(?lb.UserData, `(?name));}
   else { ?lb.SelectionItems +== ?name;}}

  if {?x = find subclassof Correlations@;
      IsSlot(?x, ?sl, `MV);
      IsFacet(?x, ?sl, IP_description);}
  {if ?ob.format == Quantitative;
   {nviplb.UserData..qtim = AppendLists(`(?name),
                                         nviplb.UserData..qtim);}
  else {nviplb.UserData..qlim = AppendLists(`(?name),
                                             nviplb.UserData..qlim);}}

  else {if ?ob.format == Quantitative;
   {nviplb.UserData..qtot = AppendLists(`(?name),
                                         nviplb.UserData..qtot);}
  else {nviplb.UserData..qлот = AppendLists(`(?name),
                                             nviplb.UserData..qлот);}}
}
/*****

```

UPSTEP25.ptk

```
/* This file contains the functions and methods relevant to the implementation of variables and
parameters for correlations. */

#include <prk/math.pth>
#include <prk/lib.pth>
/*****

method nbvcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the
       "New Quantitative Basic Variable" dialog box. */

    select {

        case: {?button == "OK";
            nbv.UserData != 3;}
            {if {IsNumber(nbveb3.Values);
                IsNumber(nbveb4.Values);
                nbveb3.Values >= nbveb4.Values;}
                {msgtd.Values = "The specified min value is >= to the max value!";
                Msgwin.PutOnScreenAndWait!();
                fail;}
            else {nvc(nbv.UserData, nbv.UserData..name, nbv.UserData..units, BV_description,
nbveb3@, nbveb4@, nbv@, im);
                if {IsFacet(correl_temp@, nbv.UserData,BV_description);
                    IsFacet(correl_temp@, nbv.UserData,IV_description);}
                    {DeleteFacet(correl_temp@, nbv.UserData,BV_description);}
                    else {;}}

        case: ?button == "Remove Variable";
            {warncr.UserData = nbv.UserData;
            warntd.Values = AppendStrings("Do you really want to remove the variable,\n",
nbv.UserData..name);
            warncr.ButtonLabels -== "Continue";
            warncr.ButtonLabels +== "Delete";
            Warningwin.PutOnScreenAndWait!();}

        otherwise: nbv.TakeOffScreen!();}
    }
/*****

function nvc (?sl, ?name, ?units, ?des, ?eb3, ?eb4,?ob, ?imne)
{
    bound inputs;
    /* Creates a new quantitative basic or intermediate variable. */

    ?sl = ConvertToSymbol(?sl);
    if find correl_temp@.?sl..variable == ?;
    {;}
    else {MakeSlot(correl_temp@, ?sl);}

    correl_temp@.?sl..?des = Quantitative;
    if ?imne == im;
    {correl_temp@.?sl..variable = ?name;}
    else {if ?des == IV_description;
```

```

        {correl_temp@.?sl..variable = ?name;}
    else {correl_temp@.?sl..variable = AppendStrings(?name, ", ",
                                                    ConvertToString(?sl));}}

correl_temp@.?sl..units = ?units;
if find correl_temp@.?sl..variable == ?;
{;}
else {MakeFacet(correl_temp@, ?sl, max_value);
      MakeFacet(correl_temp@, ?sl, min_value);
      MakeFacet(correl_temp@, ?sl, ebname);}

if IsNumber(?eb3.Values);
{correl_temp@.?sl..min_value = ?eb3.Values;}
else {?eb3.Values = Null;}
if IsNumber(?eb4.Values);
{correl_temp@.?sl..max_value = ?eb4.Values;}
else {?eb4.Values = Null;}

if {?ob == nbv@;
    ListLength(FindListElmt(all Step2lb1.SelectionItems,
                           correl_temp@.?sl..variable)) > 0;}
{Step2lb1.SelectionItems -= correl_temp@.?sl..variable;
 if ListFirst(nviplb.UserData..qtim) == correl_temp@.?sl..variable;
 {nviplb.UserData..qtim = ListRest(nviplb.UserData..qtim);}
 else {nviplb.UserData..qtim = DeleteListElmt(correl_temp@.?sl..variable,
                                              nviplb.UserData..qtim);}

if ListFirst(nviplb.UserData..qtot) == correl_temp@.?sl..variable;
{nviplb.UserData..qtot = ListRest(nviplb.UserData..qtot);}
else {nviplb.UserData..qtot = DeleteListElmt(correl_temp@.?sl..variable,
                                              nviplb.UserData..qtot);}}

else{;}

nbv.TakeOffScreen!();
nip.TakeOffScreen!();
}
/*****/
method nivcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "New Intermediate Variable" dialog box. */

    select {

    case: ?button == "OK";
        {if {IsNumber(niveb3.Values);
            IsNumber(niveb4.Values);
            niveb3.Values >= niveb4.Values;}
        {msgtd.Values = "The specified min value is >= to the max value!";
         Msgwin.PutOnScreenAndWait!();
         fail;}
        else {?list = `();
            for find ?correl_temp.?slo..IV_description == Quantitative;
            do {?slot = ConvertToString(?slo);
                ?x = ConvertToNumber(Substring(?slot, StringLength(?slot) -1));
                collect ?x into ?list;}

            if ListLength(?list) > 0;

```

```

        {SortList(?list, "<");
        ?sl = ConvertToSymbol(AppendStrings("IV_", ListFirst(?list)));}
        else {?sl = IV_1;}

        nvc(?sl, niveb2@.Values, niveb5@.Values,IV_description,
            niveb3@, niveb4@, niv@, ne);
        nivestcr.UserData = ?sl;
        callnivest(ConvertToSymbol(?sl));
        niv.TakeOffScreen!();}

        otherwise: niv.TakeOffScreen!();}
    }
    /*****
method vbvpb2.React! (?value)
{
    bound inputs;
    /* Deletes the selected permissible value. */

    vbvlb.SelectionItems -= vbvlb.Values;
}
    /*****
method vbveb3.React! (?new_value)
{
    bound inputs;
    /* Adds a new permissible value. */

    vbvlb.SelectionItems += ?new_value;
    vbveb3.Values = vbveb3.DefaultValues;
}
    /*****
method vbvpb.React! (?value)
{
    bound inputs;
    /* Displays alternative sets of permissible values. */

    vbvlb.SelectionItems = Null;
    ?l = ListLength(ListFirst(vbvlb.UserData));
    for ?i from 0 to ?l-1;
    do {vbvlb.SelectionItems += ListNth(ListFirst(vbvlb.UserData), ?l-?i-1);}
    vbvlb.UserData = ListRest(vbvlb.UserData);
    vbvlb.UserData = AppendLists(vbvlb.UserData, `(all vbvlb.SelectionItems));
}
    /*****
method vbvcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the
        "New Qualitative Basic Variable" dialog box. */

    select {

        case: {?button == "OK";
            vbv.UserData != 4;}
            {if ListLength(all vbvlb.SelectionItems) > 0;
                {vbvc();}
            else {msgtd.Values = "The permissible values list box\nshould contain at least one value.";
                Msgwin.PutOnScreenAndWait!();}}
    }
}

```

```

    case: ?button == "Remove Variable";
        {warncr.UserData = vbv.UserData;
         warntd.Values = AppendStrings("Do you really want to remove the variable,\n",
vbv.UserData.name);
         warncr.ButtonLabels -== "Continue";
         warncr.ButtonLabels +== "Delete";
         Warningwin.PutOnScreenAndWait!();}

    case: {?button == "OK";
         vbv.UserData == 4;}
        {ipcheck(newparlb1.Values, vbveb2.Values, vbveb1.Values, vbv@);
         if vbv.UserData == 1;
         {fail;}
         else {vbv.UserData = 4;}
         slcheck(vbveb2@, "variable");
         if vbveb2.UserData == error;
         {fail;}
         else {;}
         if find1 vbvlb.SelectionItems == ?;
         {vbvmak(?ob);
          callapwin(?ob@);}
         else {msgtd.Values = "The permissible values list box\nshould contain at least one value.";
          Msgwin.PutOnScreenAndWait!();}}

    otherwise: vbv.TakeOffScreen!();}
}
/*****
function vbvc ()
{
    bound inputs;
    /* Creates a new qualitative basic variable. */

    ?sl = ConvertToSymbol(vbv.UserData);
    if find correl_temp@.?sl.variable == ?;
    {;}
    else {MakeSlot(correl_temp@, ?sl);}

    correl_temp@.?sl.BV_description = Qualitative;
    correl_temp@.?sl.variable = vbv.UserData.name;
    if find correl_temp@.?sl.variable == ?;
    {;}
    else {MakeMultiValueFacet(correl_temp@, ?sl, per_val);
         MakeFacet(correl_temp@, ?sl, lbnam);}

    correl_temp@.?sl.per_val = SortList(all vbvlb.SelectionItems, Alphabetize);
    if ListLength(FindListElmt(all Step2lb2.SelectionItems,
                             correl_temp@.?sl.variable)) > 0;
    {Step2lb2.SelectionItems -== correl_temp@.?sl.variable;
     if ListFirst(nviplb.UserData..qlim) == correl_temp@.?sl.variable;
     {nviplb.UserData..qlim = ListRest(nviplb.UserData..qlim);}
     else {nviplb.UserData..qlim = DeleteListElmt(correl_temp@.?sl.variable,
                                                  nviplb.UserData..qlim);}

     if ListFirst(nviplb.UserData..qplot) == correl_temp@.?sl.variable;
     {nviplb.UserData..qplot = ListRest(nviplb.UserData..qplot);}
     else {nviplb.UserData..qplot = DeleteListElmt(correl_temp@.?sl.variable,
                                                  nviplb.UserData..qplot);}}
    else {;}
}

```

```

vbv.TakeOffScreen!();
vip.TakeOffScreen!();
}
/*****
function slcheck(?eb2, ?var)
{
    bound inputs;
    /*Checks the user supplied data for the implementation of a qualitative parameter. */

    ?npar = ConvertToSymbol(makesymbol(?eb2.Values));
    Update_info.Parameter = FindObject(?npar);
    if Update_info.Parameter != Null;
    {msgtd.Values = AppendStrings("The shorthand description of the new ", ?var,
        "\nis already used from another parameter.\n",
        "Please use an alternative expression");
    Msgwin.PutOnScreenAndWait!();
    ?eb2.UserData = error;}
    else {?eb2.UserData = Null;}
}
/*****
method nipcr.React! (?button)
{
    bound inputs;
    /*The method attached to the command row of the
        "New Quantitative Intermediate Parameter" dialog box. */

    ?npar = ConvertToSymbol(makesymbol(nipeb2.Values));
    select {

        case: ?button == "OK";
            {ipcheck(newparlb1.Values, nipeb2.Values, nipeb1.Values, nipcr@);
            if nipcr.UserData == 1;
            {fail;}
            else {;}
            slcheck(nipeb2@, "parameter");
            if nipeb2.UserData == error;
            {fail;}
            else {nipfun();

                select {

                    case: nip.Title == "New Quantitative Parameter Definition";
                        {if Step_1_rb_1.Values == "New Correlation";
                        {?module = correlation@;}
                        else {?module = correction@;}
                        MakeClass(correl_temp, ?module, `(Correlations@), `());
                        MakeMultiValueSlot(correl_temp@, ?npar);
                        correl_temp@.Parameter..format = Quantitative;
                        correl_temp@.Parameter..parameter = ?npar;
                        correl_temp@.Parameter..units = nipeb3.Values;
                        correl_temp@.Parameter..num_of_dec = nipom.Values;
                        correl_temp@.Parameter = ?npar.name;
                        nip.TakeOffScreen!();
                        Step1.TakeOffScreen!();
                        Step2aueb.DefaultValues = Null;
                        Step2aueb.Values = Null;
                        Step2.PutOnScreen!();}
                }
            }
        }
}

```

```

        case: nip.Title == "New Quantitative Parameter";
        {appli4eb1.DefaultValues = Null;
         appli4eb2.DefaultValues = Null;
         appli4td.Values = nipec3.Values;
         appli4.PutOnScreenAndWait!();}

        case: nip.Title == "New Quantitative Intermediate Parameter Definition";
        {MakeMultiValueSlot(correl_temp@, ?npar);
         correl_temp@.?npar..IP_description = Quantitative;
         correl_temp@.?npar..parameter = ?npar.name;
         correl_temp@.?npar..units = nipec3.Values;
         correl_temp@.?npar..num_of_dec = nipom.Values;
         nvip.TakeOffScreen!();}

        case: nip.Title == "New Quantitative Basic Variable Definition";
        {nbv.UserData = ?npar@;
         nbv.UserData.name = ?npar@.name;
         nbv.UserData.units = ?npar.format.units;
         setdef1();
         nbveb3.DefaultValues = Null;
         nbveb4.DefaultValues = Null;
         nbvtd.Values = AppendStrings(?npar@.name, "\n\nUnits: ",
                                     ?npar.format.units);

         nbv.PutOnScreenAndWait!();
         fail;}}}}

    nip.TakeOffScreen!();
}
/*****/
function nipfun ()
{
    bound inputs;
    /* Creates the slot used for the representation of a
       new quantitative intermediate parameter. */

    find1 ?ob.name == newparlb1.Values;
    ?npar = ConvertToSymbol(makesymbol(nipeb2.Values));
    MakeClass(?npar, GPar@, `(?ob), `());
    ?npar.name = AppendStrings(nipeb1.Values, ", ", nipeb2.Values);
    ?npar.format = Quantitative;
    ?npar.format.units = nipeb3.Values;
    ?npar.format.num_of_dec = nipom.Values;
}
/*****/
method vipcr.React! (?button)
{
    bound inputs;
    /*The method attached to the command row of the
       "New Qualitative Intermediate Parameter" dialog box. */

    ?npar = ConvertToSymbol(makesymbol(vipeb2.Values));
    select {

        case: ?button == "OK";
        {ipcheck(newparlb1.Values, vipeb2.Values, vipeb1.Values, vipcr@);
         if vipcr.UserData == 1;
         {fail;}}
    }
}

```



```

else {};
slcheck(vipeb2@, "parameter");
if vipeb2.UserData == error;
{fail;}
else {select {

    case: vip.Title == "New Qualitative Parameter Definition";
    {vipfun();
    if Step_1_rb_1.Values == "New Correlation";
    {?module = correlation@;}
    else {?module = correction@;}
    MakeClass(correl_temp, ?module, `(Correlations@), `());
    MakeMultiValueSlot(correl_temp@, ?npar);
    correl_temp@.Parameter..format = Qualitative;
    correl_temp@.Parameter..parameter = ?npar;
    correl_temp@.Parameter = ?npar.name;
    vip.TakeOffScreen!();
    Step1.TakeOffScreen!();
    Step2aueb.DefaultValues = Null;
    Step2aueb.Values = Null;
    Step2.PutOnScreen!();}

    case: vip.Title == "New Qualitative Parameter";
    {vipfun();
    appli3lb.SelectionItems = Null;
    appli3eb.Values = Null;
    appli3.PutOnScreenAndWait!();}

    case: vip.Title == "New Qualitative Intermediate Parameter Definition";
    {vipfun();
    MakeMultiValueSlot(correl_temp@, ?npar);
    correl_temp@.?npar..IP_description = Qualitative;
    correl_temp@.?npar..parameter = ?npar.name;
    nvip.TakeOffScreen!();}

    case: vip.Title == "New Qualitative Basic Variable Definition";
    {vipfun();
    vbv.UserData = ?npar@;
    vbv.UserData..name = ?npar@.name;
    setdef();
    vbvtd.Values = ?npar@.name;
    vbv.PutOnScreen!();} }}

case: {?button == "OK";
    vip.Title == "Qualitative Basic Variable Definition";}
{ipcheck("miscellaneous", vipeb2.Values, vipeb1.Values, vipcr@);
if vipcr.UserData == 1;
{fail;}
else {};
find ?ob.name == nbvpre.Values;
RenameSlot(correl_temp@, ConvertToSymbol(?ob), ?npar);
correl_temp@.?npar..parameter = ?npar.name;
vbv.UserData = ?npar@;
vbv.UserData..name = ?npar@.name;
setdef();
vbvtd.Values = ?npar@.name;
vbv.PutOnScreenAndWait!();
fail;}

```

```

case: {?button == "OK";
    vip.Title == "Qualitative Intermediate Parameter Definition";}
{ipcheck("miscellaneous", vipeb2.Values, vipeb1.Values, vipcr@);
if vipcr.UserData == 1;
    {fail;}
    else {};
    find ?ob.name == nipre.Values;
    RenameSlot(correl_temp@, ConvertToSymbol(?ob), ?npar);
    correl_temp@.?npar..parameter = ?npar.name;
    nvip.TakeOffScreen!();}}

vip.TakeOffScreen!();
}
/*****
function vipfun ()
{
    bound inputs;
    /* Creates the slot used for the representation of a
       new qualitative intermediate parameter. */

    find1 ?ob.name == newparlb1.Values;
    ?npar = ConvertToSymbol(makesymbol(vipeb2.Values));
    MakeClass(?npar, GPar@, `(?ob), `());
    ?npar.name = AppendStrings(vipeb1.Values, ", ", vipeb2.Values);
    ?npar.format = Qualitative;
    MakeMultiValueFacet(?npar@, format, per_val);
}
/*****
method nviplb.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the list box displaying intermediate parameters. */

    lbfuc(?new_value);
}
/*****
method nviprb.React! (?moused_item, ?old_item)
{
    bound inputs;
    /* The method attached to the "Show implemented", "Show other"
       push buttons in the intermediate parameters dialog box. */

    select {

    case: {?moused_item == "Show implemented";
        nvip.Title == "Quantitative Intermediate Parameter Definition";}
        {nviplb.SelectionItems = Null;
        ?l = SortList(nviplb.UserData..qtim, Alphabetize);
        for ?i from 0 to ListLength(?l)-1;
        do {nviplb.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}}

    case: {?moused_item == "Show other";
        nvip.Title == "Quantitative Intermediate Parameter Definition";}
        {nviplb.SelectionItems = Null;
        ?l = SortList(nviplb.UserData..qtot, Alphabetize);
        for ?i from 0 to ListLength(?l)-1;
        do {nviplb.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}}
}

```

```

case: {?moused_item == `Show implemented";
    nvip.Title == "Qualitative Intermediate Parameter Definition";}
{nviplb.SelectionItems = Null;
?l = SortList(nviplb.UserData..qlim, Alphabetize);
for ?i from 0 to ListLength(?l)-1;
do {nviplb.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}}

case: {?moused_item == "Show other";
    nvip.Title == "Qualitative Intermediate Parameter Definition";}
{nviplb.SelectionItems = Null;
?l = SortList(nviplb.UserData..qlot, Alphabetize);
for ?i from 0 to ListLength(?l)-1;
do {nviplb.SelectionItems += ListNth(?l, ListLength(?l)-1-?i);}}

}
/*****
method nvippb.React! (?button)
{
    bound inputs;
    /* The method attached to the push buttons for
       defining new intermediate parameters. */

    pbal(?button);
    select {

        case: nvip.Title == "Quantitative Intermediate Parameter Definition";
            {newparlb1.PositionY = 210;
             nip.Title = "New Quantitative Intermediate Parameter Definition";
             nip.PutOnScreenAndWait!();}

        case: nvip.Title == "Qualitative Intermediate Parameter Definition";
            {newparlb1.PositionY = 110;
             vip.Title = "New Qualitative Intermediate Parameter Definition";
             vip.PutOnScreenAndWait!();}}
    }
/*****
method nvipcr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of
       the intermediate parameters dialog box. */

    select {

        case: ?button == "OK";
            {if IsString(nviplb.Values);
             {varcre();}
             else {msgtd.Values = "Please select an intermediate parameter.";
                  Msgwin.PutOnScreenAndWait!();}}

        case: ?button == "Cancel";
            nvip.TakeOffScreen!();}
    }
/*****
function varcre ()
{
    bound inputs;

```

```

/* Creates the slot used for the representation of an intermediate parameter. */

find1 ?npar.name == nviplb.Values;
if nvip.Title == "Quantitative Intermediate Parameter Definition";
{MakeMultiValueSlot(correl_temp, ?npar);
correl_temp.?npar..IP_description = Quantitative;
correl_temp.?npar..parameter = ?npar.name;
correl_temp.?npar..units = ?npar.format..units;
correl_temp.?npar..num_of_dec = ?npar.format..num_of_dec;
Step2lb1.SelectionItems -== ?npar.name;
if ListFirst(Step2lb1.UserData) == ?npar.name;
{Step2lb1.UserData = ListRest(Step2lb1.UserData);}
else {Step2lb1.UserData = DeleteListElmt(?npar.name, Step2lb1.UserData);}
if ListFirst(nviplb.UserData..qtim) == ?npar.name;
{nviplb.UserData..qtim = ListRest(nviplb.UserData..qtim);}
else {nviplb.UserData..qtim = DeleteListElmt(?npar.name, nviplb.UserData..qtim);}
if ListFirst(nviplb.UserData..qtot) == ?npar.name;
{nviplb.UserData..qtot = ListRest(nviplb.UserData..qtot);}
else {nviplb.UserData..qtot = DeleteListElmt(?npar.name, nviplb.UserData..qtot);}
else {MakeMultiValueSlot(correl_temp, ?npar);
correl_temp.?npar..IP_description = Qualitative;
correl_temp.?npar..parameter = ?npar.name;
Step2lb2.SelectionItems -== ?npar.name;
if ListFirst(Step2lb2.UserData) == ?npar.name;
{Step2lb2.UserData = ListRest(Step2lb2.UserData);}
else {Step2lb2.UserData = DeleteListElmt(?npar.name, Step2lb2.UserData);}
if ListFirst(nviplb.UserData..qlim) == ?npar.name;
{nviplb.UserData..qlim = ListRest(nviplb.UserData..qlim);}
else {nviplb.UserData..qlim = DeleteListElmt(?npar.name, nviplb.UserData..qlim);}
if ListFirst(nviplb.UserData..qplot) == ?npar.name;
{nviplb.UserData..qplot = ListRest(nviplb.UserData..qplot);}
else {nviplb.UserData..qplot = DeleteListElmt(?npar.name, nviplb.UserData..qplot);}}
nvip.TakeOffScreen!();
}
/*****

```

ESTP.ptk

```

/* This file contains the functions and methods for implementing estimation procedures for
intermediate variables and correlations. */

#include <prk/lib.pth>
#include <prk/math.pth>
/*****/
method funcpb1.React! (?button)
{
bound inputs;
/* The method attached to the first row of the push buttons in the "View Functions" dialog box. */

if or {?button == "=";
?button == "(";
?button == ")";
?button == "+";
?button == "-";
?button == "*";
?button == "/";

```

```

        ?button == "Pow(?x, ?y";
        ?button == "Exp(?x)";
        ?button == "Log(?x)";
        ?button == "Log10(?x)";}
    {funcpb2.Values = Null;}
}
/*****/
method funcpb2.React! (?button)
{
    bound inputs;
    /* The method attached to the second row of the push
        buttons in the "View Functions" dialog box. */

    if or {?button == "sin(?x)";
        ?button == "cos(?x)";
        ?button == "tan(?x)";
        ?button == "Asin(?x)";
        ?button == "Acos(?x)";
        ?button == "Atan(?x)";}
    {funcpb1.Values = Null;}
}
/*****/
method funcrcr.React! (?button)
{
    bound inputs;
    /*The method attached to the command row of the "View Functions" dialog box. */

    if IsString(funcpb1.Values);
    {?val = funcpb1.Values;}
    else {?val = funcpb2.Values;}
    select {

        case: {?button == "Export";
            or {IsString(funcpb1.Values);
                IsString(funcpb2.Values);}}
            {if funcrcr.UserData == nivist@;
                {nivisteb.Values = AppendStrings(nivisteb.Values, ?val);}
                else {estwineb.Values = AppendStrings(estwineb.Values, ?val);}}

        case: ?button == "Cancel";
            func.TakeOffScreen!();}
}
/*****/
function calestwin (?ob)
{
    bound inputs;
    /* This function puts on screen the "Estwin" dialog box. */

    estwinlb1.SelectionItems = ?ob.Parameter;
    estwinlb2.SelectionItems = Null;
    estwinlb2.UserData = Null;
    for find ?ob.?slo..IP_description == Qualitative;
    do {estwinlb1.SelectionItems += ?ob.?slo..parameter;}
    for find ?ob.?slo..IP_description == Quantitative;
    do {estwinlb1.SelectionItems += ?ob.?slo..parameter;}
    for find ?ob.?slo..IV_description == Quantitative;
    do {estwinlb1.SelectionItems += ?ob.?slo..variable;}
    for find ?ob.?slo..BV_description == Qualitative;

```

```

do {estwinlb1.SelectionItems += ?ob.?slo..variable;}
for find ?ob.?slo..BV_description == Quantitative;
do {estwinlb1.SelectionItems += ?ob.?slo..variable;}
estwin.UserData = ?ob;
?list = `();

if {IsFacet(?ob, Parameter, estp);
    find1 ?ob.Parameter..estp == ?;}
{?string = "";
for ?x inlist all ?ob.Parameter..estp;
do {?string = AppendStrings(?string, ?x);}
if Substring(?string, 0, 1) == "\n";
{?string = Substring(?string, 1);}
else {;}

if Substring(?string, StringLength(?string)- 1) == "\n";
{;}
else {?string = AppendStrings(?string, "\n");}

duo(?string);
?list = estwinlb2.UserData;
estwinlb2.UserData = Null;
for ?i from 0 to ListLength(?list)-1;
do {estwinlb2.SelectionItems += ListNth(?list, ListLength(?list)-?i-1);}

if find estwinlb2.SelectionItems == ?;
{estwinlb2.UserData = all estwinlb2.SelectionItems;}
else{;}}

estwin.PutOnScreenAndWait!();
}
/*****/
function duo(?string)
{
bound inputs;
/* This function sorts the lines of code in the estimation procedure. */

?list = `();
while StringLength(?string) > 0;
do {?list = AppendLists(?list, `(Substring(?string, 0,
                                FindSubstring(?string, "\n"))));
    ?string = Substring(?string, FindSubstring(?string, "\n")+1);}
for ?x inlist ?list;
do {PrintLine(?x);}
estwinlb2.UserData = ?list;
}
/*****/
method estwinlb1.React! (?new_value, ?old_value)
{
bound inputs;
/* This method is attached to the variables and parameters
display list box of the "Estwin" dialog box. */

?ob = estwin.UserData;
if {IsString(?new_value);
    or {find ?ob.?sl..variable == ?new_value;
        find ?ob.?sl..parameter == ?new_value;
        find ?ob.?sl == ?new_value;}}

```

```

{select {
    case: {IsFacet(?ob,?sl, BV_description);
        ?ob.?sl.BV_description == Qualitative;}
    {msgtd.Values = AppendStrings(?ob.?sl.variable, " with values:\n      ");
    for ?x inlist ?ob.?sl.per_val;
    do {msgtd.Values = AppendStrings(msgtd.Values, "\", ?x,\"\"\", ", ");}
    msgtd.Values = Substring(msgtd.Values, 0, StringLength(msgtd.Values)-2);
    ?str = AppendStrings("?", ConvertToString(?sl));
    Msgwin.PutOnScreen!();}

    case: ?sl != Parameter;
        ?str = AppendStrings("?", ConvertToString(?sl));

        otherwise: ?str = AppendStrings("?", ConvertToString(?ob.Parameter.parameter));}
    estwineb.Values = AppendStrings(estwineb.Values, ?str);
    estwinlb1.Values = Null;}
}
/*****
method estwineb.React! (?new_value)
{
    bound inputs;
    /* This method is attached to the entry box of the "Estwin" dialog box. */

    if find1 estwinlb2.UserData == ?;
    {;}
    else {estwinlb2.UserData = `();}
    ?i1 = ListFirst(all estwinlb2.UserData);
    ?i2 = ListFirst(ListRest(all estwinlb2.UserData));
    if {?new_value != "";}
    {estwinlb2.UserData = AppendLists(?i1, `(new_value), ?i2);}
    else {estwinlb2.UserData = AppendLists(?i1, ?i2);}
    estwinlb2.SelectionItems = Null;
    for ?i from 0 to ListLength(estwinlb2.UserData) - 1;
    do {estwinlb2.SelectionItems += ListNth(estwinlb2.UserData,
                                           ListLength(estwinlb2.UserData)-1-?i);}

    estwineb.Values = Null;
}
/*****
method estwinlb2.React! (?new_value, ?old_value)
{
    bound inputs;
    /* This method is attached to the code display list box of the "Estwin" dialog box. */

    if ListLength(all estwinlb2.UserData) == 1;
    {if {IsString(?new_value);
        estwineb.Values == "";}
    {estwinlb2.SelectionItems = Null;
    ?i2 = FindListElmt(estwinlb2.UserData, ?new_value);
    if ListFirst(?i2) == ListFirst(estwinlb2.UserData);
    {estwinlb2.UserData = `();}
    else {for ?x inlist estwinlb2.UserData;
        do {if ListLength(FindListElmt(?i2, ?new_value)) > 0;
            {if ListLength(estwinlb2.UserData) > 1;
                {DeleteListElmt(?new_value, estwinlb2.UserData);}
            else {estwinlb2.UserData = `();}}
        else{;}}}}
}

```

```

?l1 = estwinlb2.UserData;
estwineb.Values = ListFirst(?l2);
?l2 = ListRest(?l2);
estwinlb2.UserData = ?l2;
estwinlb2.UserData += ?l1;
?l = AppendLists(?l1, ?l2);
for ?i from 0 to ListLength(?l) - 1;
do { estwinlb2.SelectionItems += ListNth(?l, ListLength(?l)-1-?i); } }
}
/*****
method estwincr.React! (?button)
{
    bound inputs;
    /* The function attached to the command row of the "Estwin" dialog box. */

    ?ob = estwin.UserData;
    select {

        case: ?button == "Update";
            { ?string = "";
              ?para = ?ob.Parameter;
              if {IsFacet(?ob, Parameter, estp);}
              {;}
              else{ MakeMultiValueFacet(?ob, Parameter, estp);}
              for ?x inlist all estwinlb2.SelectionItems;
              do { ?string = AppendStrings(?string, ?x, "\n");}
              ?str = ?string;
              if StringLength(?string) > 200;
              { ?ob.Parameter..estp = Substring(?string, 0, 200);
                ?string = Substring(?string, 200);
                ?l = ConvertToFloat(StringLength(?string))/200;
                ?i = 0;
                while ?i < ?l;
                do { ?ob.Parameter..estp += Substring(?string, 0, 200);
                    ?string = Substring(?string, 200);
                    ?i = ?i+1;}
                ?list = all ?ob.Parameter..estp;
                ?ob.Parameter..estp = Null;
                for ?z inlist ?list;
                do { ?ob.Parameter..estp += ?z;}
                else { ?ob.Parameter..estp = ?string;}
                ?num1 = ListLength(ModuleInstances(AR_correlation));
                createest(?ob, Parameter, ?str);
                ?ob.Parameter = ?para;
                ?num2 = ListLength(ModuleInstances(AR_correlation));
                if or { ?num2 -2 >= ?num1;
                    ?num2 -1 >= ?num1;
                    ?num2 == ?num1;}
                { msgtd.Values = "Installation of formula was successfull.";
                  Msgwin.PutOnScreenAndWait!();
                  estwin.TakeOffScreen!();
                  func.TakeOffScreen!();
                  appli0.PutOnScreen!();}
                else { msgtd.Values = "Installation of the formula was not\unsuccessfull. Please reconsider
the formula.";
                  Msgwin.PutOnScreenAndWait!();} }
            case: ?button == "Cancel";
                {RenameObject(ConvertToSymbol(?ob), correl_temp);

```



```

    for ?z inlist `(appli0cr@, reliacr@,step3cr@);
    do {?z.UserData = Null;}
    estwin.TakeOffScreen!();
    Step2.PutOnScreen!();}

case: ?button == "View Functions";
{funcr.UserData = estwin@;
func.PutOnScreen!();}

case: ?button == "Help";
{;}
}
/*****
function createest(?self, ?sl, ?str)
{
    bound inputs;
    /* This function creates the slot formula for the representation of the estimation procedure. */

    ?string = "";

    for find ?self.?s1..BV_description == Qualitative;
    do {?string = AppendStrings(?string, "?", ConvertToString(?s1), " = ?self.",
        ConvertToString(?s1), "\n");}

    if {find1 ?self.?s2..BV_description == Quantitative;}
        {?string = AppendStrings(?string, "for ?i from 0 to ListLength(?self.",
            ConvertToString(?s2), ") - 1;\ndo {"");
            for find ?self.?s3..BV_description == Quantitative;
            do {?s3 = ConvertToString(?s3);
                ?string = AppendStrings(?string, "?", ?s3, "= ListNth(?self.", ?s3,
                    ", ListLength(?self.",?s3, ") - 1 - ?i);\n");}}
    else {find1 ?self.?s4..IV_description == Quantitative;
        ?string = AppendStrings(?string, "?list = `();\nfor ?i from 0 to ListLength(?self.",
            ConvertToString(?s4), ") - 1;\ndo {"");}

    for find ?self.?s5..IV_description == Quantitative;
    do {?s5 = ConvertToString(?s5);
        ?string = AppendStrings(?string, "?", ?s5, "= ListNth(?self.", ?s5,
            ", ListLength(?self.",?s5, ") - 1 - ?i);\n");}

    ?string = AppendStrings(?string, ?str);
    if find ?self.?s6..IP_description == Quantitative;
    {for find ?self.?s7..IP_description == Quantitative;
    do {?string = AppendStrings(?string, "?self.", ConvertToString(?s7),
        " += format("?,ConvertToString(?s7),
        ", ?self.", ConvertToString(?s7),
        "..num_of_dec);\n");}}

    else {;}

    if find ?self.?s8..IP_description == Qualitative;
    {for find ?self.?s9..IP_description == Qualitative;
    do {?string = AppendStrings(?string, "?self.", ConvertToString(?s9),
        " += ConvertToString("?,ConvertToString(?s9),
        ");\n");}}

    else {;}

    ?par = ConvertToString(?self.Parameter..parameter);
    ?string = AppendStrings(?string, "\n?self.", ?par, " += format("?, ?par,

```

```

        ", ?self.Parameter.num_of_dec);",
        "\n?value = ?self.?slot;");

PrintLine(?string);

C:{PrkARMakeFormula(?self, `Parameter, ?string);}
disablef();
}
/*****
function disablef ()
{
    bound inputs;
    C:{PrkSendMsg(AR_correlation_FormulaClass@, `Disable!);}
}
/*****
function calculatef (?self, ?sl)
{
    bound inputs;
    C:{PrkARCalculateFormula(?self, ?sl, FALSE);}
}
/*****
function deletef (?self)
{
    bound inputs;
    C:{PrkARDeleteFormula(?self, `Parameter);}
}
/*****
function cleanupf ()
{
    bound inputs;
    C:{PrkARCleanupFormulas();}
}
/*****
function callninvest (?sl)
{
    bound inputs;
    /*This function puts on screen the dialog box for implementing
       estimation procedures for intermediate variables ("IPEP"). */

    ninvestlb1.SelectionItems = Null;
    ninvestlb3.SelectionItems = Null;
    ninvestlb3.UserData = Null;

    for find correl_temp@.?slo..BV_description == Quantitative;
    do {ninvestlb1.SelectionItems += correl_temp@.?slo..variable;}
    ninvestlb2.SelectionItems = correl_temp@.?sl..variable;

    if {IsFacet(correl_temp@, ?sl, estp);
        IsString(correl_temp@.?sl..estp);}
    {?string = correl_temp@.?sl..estp;
    if Substring(?string, 0, 1) == "\n";
    {?string = Substring(?string, 1);}
    else {;}

    for ?i from 0 to StringLength(?string)-1;
    do {if Substring(?string, ?i, ?i+1) == "\n";
        {ninvestlb3.SelectionItems += Substring(?string, 0, ?i);}
        else{;}}
    ninvest.PutOnScreenAndWait!();
}

```

```

}
/*****
method nvestlb1.React! (?new_value, ?old_value)
{
    bound inputs;
    /* This method is attached to the variables display list box of the "IPEP" dialog box. */

    if {IsString(?new_value);
        find correl_temp@.?sl.variable == ?new_value;}
    {?str = AppendStrings("?", ConvertToString(?sl));
    nvesteb.Values = AppendStrings(nvesteb.Values, ?str);}
}
/*****
method nvestlb2.React! (?new_value, ?old_value)
{
    bound inputs;
    /* This method is attached to the intermediate variable display list box of the "IPEP" dialog box. */

    if {IsString(?new_value);
        find correl_temp@.?sl.variable == ?new_value;}
    {?str = AppendStrings("?", ConvertToString(?sl));
    nvesteb.Values = AppendStrings(nvesteb.Values, ?str);}
}
/*****
method nvesteb.React! (?new_value)
{
    bound inputs;
    /*This method is attached to the entry box of the "IPEP" dialog box. */

    if find1 nvestlb3.UserData == ?;
    {;}
    else {nvestlb3.UserData = `();}
    ?11 = ListFirst(all nvestlb3.UserData);
    ?12 = ListFirst(ListRest(all nvestlb3.UserData));
    if {?new_value != "";}
    {nvestlb3.UserData = AppendLists(?11, `(?new_value), ?12);}
    else {nvestlb3.UserData = AppendLists(?11, ?12);}
    nvestlb3.SelectionItems = Null;
    for ?i from 0 to ListLength(nvestlb3.UserData) - 1;
    do {nvestlb3.SelectionItems += ListNth(nvestlb3.UserData,
                                        ListLength(nvestlb3.UserData)-1-?i);}

    nvesteb.Values = Null;
}
/*****
method nvestlb3.React! (?new_value, ?old_value)
{
    bound inputs;
    /*This method is attached to the code display list box of the "IPEP" dialog box. */

    if {IsString(?new_value);
        nvesteb.Values == "";}
    {nvestlb3.SelectionItems = Null;
    ?12 = FindListElmt(nvestlb3.UserData, ?new_value);
    if ListFirst(?12) == ListFirst(nvestlb3.UserData);
    {nvestlb3.UserData = `();}
    else {for ?x inlist nvestlb3.UserData;
        do {if ListLength(FindListElmt(?12, ?new_value)) > 0;
            {if ListLength(nvestlb3.UserData) > 1;

```

```

        {DeleteListElmt(?new_value, nivestlb3.UserData);}
    else {nivestlb3.UserData = `();}
    else{;}}

?i1 = nivestlb3.UserData;
nivesteb.Values = ListFirst(?i2);
?i2 = ListRest(?i2);
nivestlb3.UserData = ?i2;
nivestlb3.UserData += ?i1;
?i = AppendLists(?i1, ?i2);
for ?i from 0 to ListLength(?i) - 1;
do {nivestlb3.SelectionItems += ListNth(?i, ListLength(?i)-1-?i);}
}
/*****
method nivestcr.React! (?button)
{
    bound inputs;
    /* The function attached to the command row of the "IPEP" dialog box. */

    ?sl = nivestcr.UserData;
    select {

        case: ?button == "Update";
        {?string = "";
        for ?x inlist all nivestlb3.SelectionItems;
        do {?string = AppendStrings(?string, "\n", ?x);}
        correl_temp@.?sl.estp = ?string;
        createfor(correl_temp@, ?sl, ?string);
        if IsFacet(correl_temp@, ?sl, WhenChangedFacet);
        {msgtd.Values = "Installation of formula was successfull.";
        nivest.TakeOffScreen!();
        niv.TakeOffScreen!();
        Msgwin.PutOnScreenAndWait!();}
        else { msgtd.Values = "Installation of the formula was not\nsuccessfull. Please reconsider
the formula.";
                Msgwin.PutOnScreenAndWait!();}}

        case: ?button == "Cancel";
        {DeleteSlot(correl_temp@, ConvertToSymbol(niveb2.Values));
        nivest.TakeOffScreen!();
        niv.TakeOffScreen!();}

        case: ?button == "View Functions";
        {funcr.UserData = nivest@;
        func.PutOnScreen!();}

        case: ?button == "Help";
        {;}
    }
}
/*****
function createfor (?self, ?sl, ?str)
{
    bound inputs;
    /* This function creates the slot formula for the representation of the IP estimation procedure. */

    ?list = `();
    find1 ?self.?sl1..BV_description == Quantitative;
    ?string = AppendStrings("?list = `();\nfor ?i from 0 to ListLength(?self.",

```

```

        ConvertToString(?sl1), ") - 1;\ndo {"");
for find ?self.?s..BV_description == Quantitative;
do {collect ConvertToString(?s) into ?list;}
PrintLine(?list);
for ?y inlist ?list;
do {?string = AppendStrings(?string, "?", ?y, "= ListNth(?self.", ?y,
        ", ListLength(?self.",?y, "-1 -?i);\n");}
?string = AppendStrings(?string, ?str, "\n?list = AppendLists(`?",
        ConvertToString(?sl), "), ?list);}\n?self.",
        ConvertToString(?sl), " = ?list;\n?value = ?self.?slot;");
PrintLine(?string);
C:{PrkARMakeFormula(?self, `Parameters_needed, ?string);}
disablef();
}

```

```

/*****

```

UPSTEP3.ptk

```

/* This file contains the functions and methods relevant to the implementation of applicability for
correlations. */

```

```

#include <prk/math.pth>

```

```

#include <prk/lib.pth>

```

```

/*****

```

```

method appli0lb1.React! (?new_value, ?old_value)

```

```

{

```

```

    bound inputs;

```

```

    /* The method attached to the ground types display list box in the "appli0" dialog box. */

```

```

    ?lpar = all appli0lb1.SelectionItems;

```

```

    select {

```

```

        case:{IsString(?old_value);

```

```

            not {IsString(?new_value);}

```

```

            ?x1 = find direct subclassof ConvertToSymbol(?old_value);

```

```

            or {ObjectModule(?x1) == ground_rep@;

```

```

                ObjectModule(?x1) == expand@;}}

```

```

        {appli0lb1.SelectionItems = Null;

```

```

        for {find ?x == direct subclassof ConvertToSymbol(?old_value);

```

```

            or {ObjectModule(?x) == ground_rep@;

```

```

                ObjectModule(?x) == expand@;}}

```

```

        do appli0lb1.SelectionItems += ConvertToString(?x);

```

```

        ?lpar = DeleteListElmt(?old_value, ?lpar);

```

```

        ?lpar = AppendLists(?lpar, `(?(old_value));

```

```

        appli0lb1.UserData = AppendLists(appli0lb1.UserData, `(?(lpar));}}

```

```

    }

```

```

/*****

```

```

method appli0pb1.React! (?button)

```

```

{

```

```

    bound inputs;

```

```

    /* The method attached to the "Select", "Back", "Forward"
and "Reset" push buttons in the "appli0" dialog box. */

```

```

    if {?button == "Select";

```

```

        IsString(appli0lb1.Values);}
    {appli2rb.Title = appli0lb1.Values;
    appli2.PutOnScreenAndWait!();}
    else {parpb(?button, appli0lb1@, Ground@);}
}
/*****
method appli0pb2.React! (?button)
{
    bound inputs;
    /* The method attached to the "Define new ground type"
    push buttons in the "appli0" dialog box. */

    if ?button == "Define new ground type";
        {if IsString(appli0lb1.Values);
        {ngtlb.SelectionItems = appli0lb1.Values;}
        else {ngtlb.SelectionItems = Null;}
        ngt.PutOnScreen!();}
}
/*****
method appli0cr.React! (?button)
{
    bound inputs;
    /*The method attached to the command row of the "appli0" dialog box. */

    select {

        case: ?button == "OK";
            {if or {ListLength(all appli0cr.UserData.High_Applicability) >0;
                ListLength(all appli0cr.UserData.Medium_Applicability) >0;
                ListLength(all appli0cr.UserData.Low_Applicability) >0;}
            {appli2cr.UserData = Off;
            appli2.TakeOffScreen!();
            relia1b.DefaultValues = appli0cr.UserData.Reliability;
            reliaeb1.DefaultValues = appli0cr.UserData.Reliability..r2;
            reliaeb2.DefaultValues = appli0cr.UserData.Reliability..sd;
            reliaeb3.DefaultValues = appli0cr.UserData.Reliability..n;
            appli0.TakeOffScreen!();
            relia.PutOnScreen!();}
            else {msgtd.Values = "Applicability should be defined, at least\nin terms of ground
type(s)";
                Msgwin.PutOnScreenAndWait!();}}

        case: ?button == "Cancel";
            {appli0.TakeOffScreen!();
            appli2.TakeOffScreen!();
            calestwin(appli0cr.UserData);}

        case: ?button == "Preview";
            {apreocr.UserData = `();
            ?ob = appli0cr.UserData;
            for ?sl inlist `(High_Applicability, Medium_Applicability, Low_Applicability);
            do {for ?x inlist all ?ob.?sl;
                do {apreocr.UserData = AppendLists(apreocr.UserData, `(?x));}}
            apreocr.`React!`("Show next");
            apre.PutOnScreenAndWait!();}

        case: ?button == "Help";
            {;}
}

```

```

}
/*****
method appli2cr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "appli2" dialog box. */

    select {

        case: ?button == "Update";
            {for {?x inlist SlotFacets(appli1cr@, UserData);
                ?x != NoBPFacetNames;}
            do {DeleteFacet(appli1cr@, UserData, ?x);}
            if appli2rb.Values == NO;
            {appli5.PutOnScreenAndWait!();}
            else {appli4eb1.DefaultValues = Null;
                appli4eb2.DefaultValues = Null;
                parpb("Reset ", appli1lb1@, Parameters@);
                appli1.PutOnScreenAndWait!();}}

        case: ?button == "Cancel";
            appli2.TakeOffScreen!();}
    }
/*****
method appli1lb1.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the parameters display list box of the "appli1" dialog box. */

    parlb(?new_value, ?old_value, appli1lb1@);
}
/*****
method appli1pb1.React! (?button)
{
    bound inputs;
    /* The method attached to the "Select", "Back", "Forward"
       and "Reset" push buttons in the "appli1" dialog box. */

    select {

        case: {?button == "Select";
            IsString(appli1lb1.Values);
            find1 ?ob.name == appli1lb1.Values;
            ListLength(all superclassof ?ob) > 1;}
            {if ?ob.format == Qualitative;
                {appli3cr.UserData = new;
                    appli3.UserData = 2;
                    applipb2f1(?ob);
                    appli3.PutOnScreenAndWait!();}
                else {appli4cr.UserData = new;
                    appli4.Title = AppendStrings(appli1lb1.Values, " restrictions");
                    appli4td.Values = ?ob.format..units;
                    appli4.PutOnScreenAndWait!();}}

        case: {?button == "Select";
            IsString(appli1lb1.Values);
            find1 ?ob.name == appli1lb1.Values;
            ListLength(all superclassof ?ob) == 1;}
    }
}

```

```

        {msgtd.Values = "Only parameters, not parameters categories\ncan be selected";
        Msgwin.PutOnScreenAndWait!();}

    otherwise: parpb(?button, appli1lb1@, Parameters@);}
}
/*****/
method appli1pb2.React! (?button)
{
    bound inputs;
    /* The method attached to the "Define new parameter"
       push buttons in the "appli1" dialog box. */

    select {

        case: ?button == "New quantitative parameter";
        {appli1lb1.Values = Null;
        appli4cr.UserData = new;
        pba1("Quantitative ");
        nip.Title = "New Quantitative Parameter";
        nip.PutOnScreenAndWait!();}

        case: ?button == "New qualitative parameter";
        {appli3cr.UserData = new;
        appli3.UserData = 4;
        pba1("Qualitative ");
        vip.Title = "New Qualitative Parameter";
        vip.PutOnScreenAndWait!();}}
    }
/*****/
method appli1cr.React! (?button)
{
    bound inputs;
    /*The method attached to the command row of the "appli1" dialog box. */

    select {

        case: ?button == "OK";
        appli5.PutOnScreenAndWait!();
        case: ?button == "Cancel";
        appli1.TakeOffScreen!();
        case: ?button == "Preview";
        {;}
        case: ?button == "Help";
        {;}
    }
}
/*****/
function callapwin (?ob)
{
    bound inputs;
    /* Sets the "appli3" dialog box on screen. */

    appli3lb.SelectionItems = Null;
    appli3.UserData = 1;
    appli3.Title = AppendStrings(ConvertToString(?ob), " restrictions");
    for ?x inlist all vbv1b.SelectionItems;
    do {appli3lb.SelectionItems += ?x;}
    appli3.Contents -= appli3pb@;
    appli3.Contents -= appli3eb@;
}

```



```

    appli3.PutOnScreenAndWait!();
}
/*****
method appli3pb.React! (?button)
{
    bound inputs;
    /* The method attached to the "Delete", "Show alternative"
       push buttons of the "appli3" dialog box. */

    select {

        case: ?button == `Delete;
            {if {ListLength(all appli3lb.Values) < 2;
                IsString(appli3lb.Values);}
                {appli3lb.SelectionItems -== appli3lb.Values;}}

        case: ?button == `Show alternative";
            {appli3lb.SelectionItems = Null;
              ?l = ListLength(ListFirst(appli3lb.UserData));
              for ?i from 0 to ?l-1;
              do {appli3lb.SelectionItems += ListNth(ListFirst(appli3lb.UserData),
                                                         ?l-?i-1);}

            appli3lb.UserData = ListRest(appli3lb.UserData);
            appli3lb.UserData = AppendLists(appli3lb.UserData,
                                             `(all appli3lb.SelectionItems));}}
    }
/*****
method appli3eb.React! (?new_value)
{
    bound inputs;
    /* Adds a new permissible value in the list. */

    appli3lb.SelectionItems += ?new_value;
    appli3eb.Values = appli3eb.DefaultValues;
}
/*****
method appli3cr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "appli3" dialog box. */

    select {

        case: {?button == "OK";
            appli3.UserData == 2;
            appli3cr.UserData == new;}
            {if ListLength(all appli3lb.Values) > 0;
              {find ?ob.name == appli1lb1.Values;
                if ListLength(all ?ob.format..per_val) < 1;
                {?ob.format..per_val = SortList(all appli3lb.SelectionItems,
                                                  Alphabetize);}
              else {?ob.format..per_val += SortList(all appli3lb.SelectionItems,
                                                    Alphabetize);}

              find1 ?ob.name == appli1lb1.Values;
              ?fac = ConvertToSymbol(?ob);
              appli1cr.UserData..?fac = all appli3lb.Values;}
            else {msgtd.Values = "At least one value of the parameter\nshould be selected";
              Msgwin.PutOnScreenAndWait!();
            }
        }
    }
}
/*****

```

```

fail;}}

case: {?button == "OK";
    appli3.UserData == 4;
    appli3cr.UserData == new;}
{if ListLength(all appli3lb.Values) > 0;
{find1 ?ob.name == AppendStrings(vipeb1.Values, ", ", vipeb2.Values);
if ListLength(all ?ob.format..per_val) < 1;
{?ob.format..per_val = SortList(all appli3lb.SelectionItems,
                                Alphabetize);}
else {?ob.format..per_val += SortList(all appli3lb.SelectionItems,
                                Alphabetize);}

?fac = ConvertToSymbol(?ob);
appli1cr.UserData..?fac = all appli3lb.Values;
vip.TakeOffScreen!();}
else {msgtd.Values = "At least one value of the parameter\nshould be selected";
Msgwin.PutOnScreenAndWait!();
fail;}}

case: ?button == "Cancel";
vip.TakeOffScreen!();

otherwise: {?fac = ConvertToSymbol(aprelb.Values);
?ob = ConvertToSymbol(apretd.Values);
?sl = aprelb.UserData..sl;
?list = SortList(all appli3lb.Values, Alphabetize);
if ListLength(?list) > 0;
{;}
else {msgtd.Values = "At least one value of the parameter\nshould be selected";
Msgwin.PutOnScreenAndWait!();
fail;}
if AppendLists^(?ob@), ?list) == aprelb.UserData;
{;}
else {if ListLength(all ?fac@.format..per_val) < 1;
{?fac@.format..per_val = SortList(all appli3lb.SelectionItems,
                                Alphabetize);}
else {?fac@.format..per_val += SortList(all appli3lb.SelectionItems,
                                Alphabetize);}

appli0cr.UserData.?sl..?fac += AppendLists^(?ob@), ?list);
appli0cr.UserData.?sl..?fac -= aprelb.UserData;}}

appli3lb.DefaultValues = Null;
appli3.TakeOffScreen!();
}
/*****
function applipb2f1(?ob)
{
bound inputs;
/* Makes the settings for the "appli3" dialog box. */

appli3lb.UserData = Null;
appli3lb.SelectionItems = Null;
appli3pb.ButtonLabels = "Delete";
appli3.UserData = 2;
appli3.Title = AppendStrings(appli1lb1.Values, " restrictions");
appli3.Contents += appli3pb@;
appli3.Contents += appli3eb@;
if ?y = find1 ?ob.format..per_val;

```

```

    {for ?i from 0 to ListLength(?y)-1;
      do {appli3lb.SelectionItems += ListNth(?y, ListLength(?y)-1-?i);
        applipb2f2(?ob, ?y);}
      else {;}
    }
  }
  /***/
function applipb2f2(?ob, ?y)
{
  bound inputs;
  /* Makes additional settings for the "appli3" dialog box. */

  for {?pv = find ?ob.format.per_val;
    ?pv != ?y;}
  do {appli3lb.UserData += ?pv;}
  appli3lb.UserData = all appli3lb.UserData;
  appli3lb.UserData = AppendLists(appli3lb.UserData, (?y));
  if ListLength(appli3lb.UserData) > 1;
  {appli3pb.ButtonLabels += "Show alternative";}
  else {;}
}
/***/
method appli4cr.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "appli4" dialog box. */

  ?list = `();
  if not {IsNumber(appli4eb1.Values);}
  {appli4eb1.Values = Null;}
  else {;}

  if not {IsNumber(appli4eb2.Values);}
  {appli4eb2.Values = Null;}
  else {;}

  select {

    case: {?button == "OK";
      not {IsNumber(appli4eb1.Values);}
      not {IsNumber(appli4eb2.Values);}}
    {msgtd.Values = "At least one of the min, max values\nshould be specified!";
     Msgwin.PutOnScreenAndWait!();
     fail;}

    case: {?button == "OK";
      IsNumber(appli4eb1.Values);
      IsNumber(appli4eb2.Values);}
    {if appli4eb1.Values >= appli4eb2.Values;
     {msgtd.Values = "The specified min value is greater than\nor equal to the max value!";
      Msgwin.PutOnScreenAndWait!();
      fail;}
     else{?list = `(appli4eb1.Values, appli4eb2.Values);}}

    case: {?button == "OK";
      IsNumber(appli4eb1.Values);
      not {IsNumber(appli4eb2.Values);}}
    {?list = `(appli4eb1.Values, u);}
  }
}

```

```

case: {IsNumber(appli4eb2.Values);
      not {IsNumber(appli4eb1.Values);}}
      {?list = `(u, appli4eb2.Values);}

case: ?button == "Cancel";
      {appli4.TakeOffScreen!();
       fail;}}

if appli4cr.UserData == new;
{or {find1 ?ob.name == appli1lb1.Values;
    find1 ?ob.name == AppendStrings(nipeb1.Values, ", ", nipeb2.Values);}
?fac = ConvertToSymbol(?ob);
appli1cr.UserData..?fac = ?list;}
else {?fac = ConvertToSymbol(aprelb.Values);
      ?ob = ConvertToSymbol(apretb.Values);
      ?sl = aprelb.UserData..sl;
      if AppendLists(`(?ob@), ?list) == aprelb.UserData;
      {;}
      else {appli0cr.UserData.?sl..?fac += AppendLists(`(?ob@), ?list);
            appli0cr.UserData.?sl..?fac -= aprelb.UserData;}}

nip.TakeOffScreen!();
appli4.TakeOffScreen!();
}
/*****
method appli5cr.React! (?button)
{
  bound inputs;
  /* The method attached to the command row of the "appli5" dialog box. */

  select {

  case: ?button == "Update";
    {select {
      case: appli5lb.Values == high;
        ?sl = High_Applicability;
      case: appli5lb.Values == medium;
        ?sl = Medium_Applicability;
      case: appli5lb.Values == low;
        ?sl = Low_Applicability;
      otherwise: {?sl = unsigned;
                  fail;}}
      ?ob = ConvertToSymbol(appli2rb.Title);
      ?list = `(?ob@);
      for {?x inlist SlotFacets(appli1cr, UserData);
           ?x != NoBPFacetNames;}
      do {?list = AppendLists(?list, `(?x);
          if IsFacet(appli0cr.UserData, ?sl, ?x);
          {appli0cr.UserData.?sl..?x += AppendLists(`(?ob@), appli1cr.UserData..?x);}
          else {MakeMultiValueFacet(appli0cr.UserData, ?sl, ?x);
                appli0cr.UserData.?sl..?x = AppendLists(`(?ob@), appli1cr.UserData..?x);}}
      appli0cr.UserData.?sl += ?list;
      appli5.TakeOffScreen!();
      appli1.TakeOffScreen!();
      appli2.TakeOffScreen!();}

case: ?button == "Cancel";
      appli5.TakeOffScreen!();}

```

```

}
/*****
function applipre(?ob)
{
    bound inputs;
    /* the applicability preview function. */

    ?string = "";
    for ?sl inlist `(High_Applicability, Medium_Applicability, Low_Applicability);
    do {select {
        case: { ?sl == High_Applicability;
            ListLength(all ?ob.?sl) > 0; }
            ?string = AppendStrings(?string, "\nHigh Applicability:\n");
        case: { ?sl == Medium_Applicability;
            ListLength(all ?ob.?sl) > 0; }
            ?string = AppendStrings(?string, "\nMedium Applicability:\n");
        case: { ?sl == Low_Applicability;
            ListLength(all ?ob.?sl) > 0; }
            ?string = AppendStrings(?string, "\nLow Applicability:\n"); }
    for ?x inlist all ?ob.?sl;
    do { ?soil = ListFirst(?x);
        ?string = AppendStrings(?string, "\n    ", ConvertToString(?soil));
        if ListLength(ListRest(?x)) < 1;
        {;}
        else { ?string = AppendStrings(?string, " with:");
            for ?y inlist ListRest(?x);
            do { ?list = `();
                for ?z inlist all ?ob.?sl..?y;
                do { if ListFirst(?z) == ?soil;
                    { ?list = ListRest(?z); }
                else {;} }
                if ?y.format == Quantitative;
                { ?min = ListFirst(?list);
                    ?max = ListFirst(ListRest(?list));
                    select {
                        case: { IsNumber(?min);
                            IsNumber(?max); }
                            { ?string = AppendStrings(?string, "\n                ",
                                ConvertToString(?y), " between ",
                                ConvertToString(?min), " and ",
                                ConvertToString(?max), " ",
                                ?y.format..units); }
                        case: { IsNumber(?min);
                            not { IsNumber(?max); } }
                            { ?string = AppendStrings(?string, "\n                ",
                                ConvertToString(?y), " greater than ",
                                ConvertToString(?min), ?y.format..units); }
                        case: { not { IsNumber(?min); }
                            IsNumber(?max); }
                            { ?string = AppendStrings(?string, "\n                ",
                                ConvertToString(?y), " lower than ",
                                ConvertToString(?max), ?y.format..units); } }
                    }
                }
            }
        }
        else { ?str = "";
            for ?w inlist ?list;
            do { ?str = AppendStrings(?str, ConvertToString(?w), " or "); }
            ?str = Substring(?str, 0, StringLength(?str)-3);
            ?string = AppendStrings(?string, "\n            ", ConvertToString(?y),
                ": ", ?str); } } }
}

```

```

        ?string = AppendStrings(?string, "\n");}
        ?string = AppendStrings(?string, "\n");}
?string = Substring(?string, 1);
?string = Substring(?string, 0, StringLength(?string)-3);
msgtd.Values = ?string;
}
/*****/
method aprelb.React! (?new_value, ?old_value)
{
    bound inputs;
    /* The method attached to the parameter display list
       box of the "Applicability preview" dialog box. */

    if IsSymbol(?new_value);
    { ?ob = ConvertToSymbol(apretd.Values)@;
      select {

        case: apretd2.Values == high;
          ?sl = High_Applicability;

        case: apretd2.Values == medium;
          ?sl = Medium_Applicability;

        otherwise: ?sl = Low_Applicability;}

    ?obje = appli0cr.UserData;
    ?list = `();
    for ?x inlist all ?obje.?sl..?new_value;
    do {if ListFirst(?x) == ?ob;
        { ?list = ListRest(?x);}
        else{;}}
    aprelb.UserData..sl = ?sl;
    aprelb.UserData = AppendLists(?ob, ?list);
    if ?new_value.format == Quantitative;
    {appli4.Title = AppendStrings(aprelb.Values, " restrictions");
     appli4eb1.DefaultValues = ListFirst(?list);
     appli4eb2.DefaultValues = ListFirst(ListRest(?list));
     appli4td.Values = ?new_value.format..units;
     appli4cr.UserData = up;
     appli4.PutOnScreenAndWait!();}
    else {appli3cr.UserData = up;
          appli3lb.UserData = Null;
          appli3lb.SelectionItems = Null;
          appli3pb.ButtonLabels = `Delete";
          appli3.UserData = 2;
          appli3.Title = AppendStrings(ConvertToString(aprelb.Values),
                                         " restrictions");

          appli3.Contents +== appli3pb@;
          appli3.Contents +== appli3eb@;
          MakeMultiValueSlot(?new_value@, `123456");
          for ?x inlist all ?new_value@.format..per_val;
          do {?new_value@.`123456" = "";
              for ?y inlist ?x;
              do {?new_value@.`123456" +== ?y;}
              for ?z inlist ?list;
              do {?new_value@.`123456" +== ?z;}
              if ListLength(all ?new_value@.`123456") == ListLength(?x)+1;
              {?new_value@.`123456"..suc = ?x;}

```

```

        else{;}
        for ?t from 0 to ListLength(?new_value@.`"123456"..suc) - 1;
            do {appli3lb.SelectionItems += ListNth(?new_value@.`"123456"..suc,
ListLength(?new_value@.`"123456"..suc)-1-?t);}
            DeleteSlot(?new_value@, `"123456");
            for ?tr inlist ?list;
                do {appli3lb.DefaultValues += ?tr;}
                appli3lb2f2(?new_value@, all appli3lb.SelectionItems);
                appli3.PutOnScreenAndWait!();}}
    }
    /*****
method aprecre.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "Applicability preview" dialog box. */

    select {

        case: ?button == "Update";
            {;}

        case: {?button == "Show next";
            ListLength(aprecre.UserData) < 1;}
            {apreclb.SelectionItems = Null;
            apretd.Values = Null;
            apretd2.Values = Null;}

        case: {?button == "Show next";
            ListLength(aprecre.UserData) == 1;}
            {apreshn();}

        case: {?button == "Show next";
            ListLength(aprecre.UserData) > 1;}
            {apreshn();
            aprecre.UserData = AppendLists(ListRest(aprecre.UserData),
            `(ListFirst(aprecre.UserData)));}

        case: ?button == "Remove";
            {if apretd.Values != Null;
            {msgtd.Values = "Are you sure you want to remove the displayed ground type\nwith its
associated parameter restrictions from the\ndefinition of applicability?";
            msgcr.ButtonLabels += "Remove";
            Msgwin.PutOnScreenAndWait!();}
            else {fail;}}

        case: ?button == "Dismiss";
            apre.TakeOffScreen!();

        case: ?button == "Help";
            {;}
    }
    /*****
function apreshn ()
{
    bound inputs;
    /* Displays the next applicability set. */

    apreclb.SelectionItems = Null;

```

```

?x = ListFirst(aprecr.UserData);
apretd.Values = ConvertToString(ListFirst(?x));
for ?y from 1 to ListLength(?x);
do aprelb.SelectionItems += ListNth(?x, ?y);

select {

case: find1 appli0cr.UserData.High_Applicability == ?x;
    apretd2.Values = high;
case: find1 appli0cr.UserData.Medium_Applicability == ?x;
    apretd2.Values = medium;
otherwise: apretd2.Values = low;}
}
/*****/

```

UPSTEP4.ptk

```

/* This file contains the functions and methods relevant to the implementation of reliability and
comments for correlations. */

```

```

#include <prk/math.pth>
#include <prk/lib.pth>
/*****/

```

```

method reliacr.React! (?button)
{
    bound inputs;
    /* The method attached to the command row of the "Reliability" dialog box. */

    select {

case: {?button == "OK";
    IsString(relialb.Values);}
    {appli0cr.UserData.Reliability = relialb.Values;
    if {IsNumber(reliaeb1.Values);
    reliaeb1.Values >= 0;
    reliaeb1.Values <= 100;}
    {appli0cr.UserData.Reliability..r2 = reliaeb1.Values;}
    else {if reliaeb1.Values == PrkEmptyString;
    {;}
    else {msgtd.Values = "The coefficient of fit should be between 0 and 100.";
    Msgwin.PutOnScreenAndWait!();
    fail;}}
    if {IsNumber(reliaeb2.Values);
    reliaeb2.Values >= 0;}
    {appli0cr.UserData.Reliability..sd = reliaeb2.Values;}
    else {if reliaeb2.Values == PrkEmptyString;
    {;}
    else {msgtd.Values = "Standard deviation should be a positive number.";
    Msgwin.PutOnScreenAndWait!();
    fail;}}
    if {IsFixnum(reliaeb3.Values);
    reliaeb3.Values >= 0;}
    {appli0cr.UserData.Reliability..n = reliaeb3.Values;}
    else {if reliaeb3.Values == PrkEmptyString;
    {;}

```



```

        else {msgtd.Values = "The number of points should be a positive integer.";
              Msgwin.PutOnScreenAndWait!();
              fail;}
    defcom(step3cr.UserData, Comments);
    relia.TakeOffScreen!();
    Step3.PutOnScreen!();
    case: {?button == "OK";
          relialb.Values == Null;}
    {msgtd.Values = "Please select a value for the reliability score";
      Msgwin.PutOnScreenAndWait!();
      fail;}

    case: ?button == "Cancel";
    {relia.TakeOffScreen!();
      appli0.PutOnScreen!();}
}
/*****
function defcom (?ob, ?sl)
{
    bound inputs;
    /* Displays the comments of an existing correlation. */

    if find1 ?ob.?sl == ?;
    {?string = "";
     for ?x inlist all ?ob.?sl;
     do {?string = AppendStrings(?string, ?x);}
     if Substring(?string, 0, 1) == "\n";
     {?string = Substring(?string, 1);}
     else{;}

    for ?i from 0 to StringLength(?string)-1;
    do {select {

        case: Substring(?string, ?i, ?i+2) == "-\n";
            ?string = AppendStrings(Substring(?string, 0, ?i),
                                   Substring(?string, ?i+2));

        case: Substring(?string, ?i, ?i+1) == "\n";
            ?string = AppendStrings(Substring(?string, 0, ?i), " ",
                                   Substring(?string, ?i+1));}}

    step3eb.Values = Null;
    step3eb.DefaultValues = ?string;
    PrintLine(step3eb.Values);
    step3eb.React!(?string);}
    else {step3eb.Values = Null;
          step3eb.DefaultValues = Null;
          step3td.Values = Null;}
}
/*****
method step3cr.React! (?button)
{
    /* The method attached to the command row of the "Comments" dialog box. */

    select {

        case: ?button == "OK";
            {?string = step3td.Values;

```

```

if IsString(?string);
{if StringLength(?string) > 200;
{step3cr.UserData.Comments = Substring(?string, 0, 200);
?string = Substring(?string, 200);
?l = ConvertToFloat(StringLength(?string))/200;
?i = 0;
while ?i<?l;
do {step3cr.UserData.Comments += Substring(?string, 0, 200);
?string = Substring(?string, 200);
?i = ?i+1;}
?list = all step3cr.UserData.Comments;
step3cr.UserData.Comments = Null;
for ?z inlist ?list;
do {step3cr.UserData.Comments += ?z;}}
else {step3cr.UserData.Comments = ?string;}}
else {;}
step3cr.UserData = Null;
appli0cr.UserData = Null;
relia.UserData = Null;
if IsFacet(Step2@, UserData, name);
{DeleteFacet(Step2@, UserData, name);}
else {;}
SaveApp(correlation@);
estwin.UserData = Null;
Step3.TakeOffScreen!();
mmem.PutOnScreen!();}

case: ?button == "Cancel";
relia.PutOnScreen!();}

Step3.TakeOffScreen!();
}
/*****/
function makesymbol (?string)
{
bound inputs;
/* This function is used for replacing strings containing space characters
to symbols, by substituting them with underscores ( _). */

if IsString(?string);
{?l = StringLength(?string);
for ?i from 1 to ?l-1;
do {if Substring(?string, ?i, ?i+1) == " ";
{select {

case: ?i == 0;
{?s1 = "";
?s2 = "";
?s3 = Substring(?string, ?i+1);}

case: ?i+1 == ?l;
{?s1 = Substring(?string, 0, ?i);
?s2 = "";
?s3 = "";}

otherwise: {?s1 = Substring(?string, 0, ?i);
?s2 = "_";
?s3 = Substring(?string, ?i+1);}

```

```

    }
    ?string = AppendStrings(?s1, ?s2, ?s3);}
    else {?string = ?string;}}
else {Print("Argument is not a STRING\n");}
return(?string);
}
/*****
method step3eb.React! (?new_value)
{
    bound inputs;
    /* The method attached to the entry box of the "Comments" dialog box. */

    string_man(?new_value, step3td, Values, 50);
}
/*****
function string_man(?string, ?obje, ?sl, ?num)
{
    bound inputs;
    /* This function arranges long strings to lines containing
       approximately a specified number of characters. */

    ?a1 = 0;
    step3eb.UserData = Null;
    ?str = "";
    ?i = 0;
    ?l = ConvertToFloat(StringLength(?string))/?num;

    while ?i<?l;
    do {step3td.UserData = Null;
        select {

            case: StringLength(?string) >= ?num+4;
            {for ?i1 from ?num-4 to ?num+3;
                do {if Substring(?string, ?i1-1, ?i1) == " ";
                    {step3td.UserData += ?i1;}
                    else {;}}
                if find step3td.UserData == ?;
                {select {

                    case: find step3td.UserData == ?num;
                    ?a1 = ?num;
                    case: find step3td.UserData == ?num-1;
                    ?a1 = ?num-1;
                    case: find step3td.UserData == ?num+1;
                    ?a1 = ?num+1;
                    case: find step3td.UserData == ?num-2;
                    ?a1 = ?num-2;
                    case: find step3td.UserData == ?num+2;
                    ?a1 = ?num+2;
                    case: find step3td.UserData == ?num-3;
                    ?a1 = ?num-3;
                    case: find step3td.UserData == ?num+3;
                    ?a1 = ?num+3;
                    otherwise: ?a1 = ?num-4;}

                ?sub = Substring(?string, 0, ?a1-1);
                ?string = Substring(?string, ?a1);
                step3eb.UserData += ?sub;}

```

```

else {?sub = Substring(?string, 0,?num);
      ?string = Substring(?string, ?num);
      step3eb.UserData += AppendStrings(?sub, "-");}}

otherwise: {step3eb.UserData += ?string;}
}
?i = ?i+1;}

?str_list = all step3eb.UserData;
for ?y from 0 to ListLength(?str_list)-1;
do {?ys = ListNth(?str_list, ListLength(?str_list)-1-?y);
    ?str = AppendStrings(?str, ?ys, "\n");}

?obje.?sl = Substring(?str, 0, StringLength(?str)-1);
}
/*****/

```

