Renewal of a Linear Electrical Network Simulator Into Ada

by

Warren Dean Buckle

VOLUME 1

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 1993

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any other degree or examination in any other University.

28 day of JUNE 1993

ABSTRACT

Renewal is the extraction of the intellectual content (algorithms, data structures) from an existing program, and then building a new more maintainable program using more modern programming methods and languages. A survey of software structure on maintenance highlighted the different hierarchies produced by functional and object-oriented design methods.

Elecsim, a linear circuit simulator written in Pascal, was chosen as the existing program to be renewed. The new version follows the approach of decoupling the user interface and introducing an explicit scheduler. The object-oriented design technique is used extensively. Other issues addressed include online-help and documentation for the program.

Conclusions are drawn which are generally applicable from the specific lessons learnt from the Elecsim/Elector case study.

ACKNOWLEDGEMENTS

I would like to acknowledge the guidance, support and encouragement from my supervisor Professor A.J Walker. I would also like to thank the University of the Witwatersrand for providing the necessary equipment and facilities.

CONTENTS

DECLARATION ii

ABSTRACT iii

ACKNOWLEDGEMENTS iv

CONTENTS v

LIST OF FIGURES ix

LIST OF TABLES x

CHAPTER 1 : INTRODUCTION 1

1.1 Software Deterioration 1
1.2 The Need for Renewal 1
1.3 Computer-Aided Circuit Design CACD 2
1.4 Aim and Scope 3
1.4.1 The Renewal and Maintenance Exercise 3
1.4.2 Program Structure 3
1.4.3 Scheduling and Concurrent Process 4
1.4.4 Decoupling the User Interface 4
1.4.5 Graphical User Interface Following CUA Guidelines 5
1.4.6 Use of a Modern Language 5
1.5 Structure of the Dissertation 6
1.5.1 Main Body 6
1.5.2 User Reference Manual 6
1.5.3 Technical Reference Manual 6

CHAPTER 2 : SOFTWARE STRUCTURE 8

2.1 Introduction 8
2.2 What is Software Structure 8
2.3 Complex Systems 8
2.3.1 Hierarchies 8
2.3.2 Attributes of Complex Systems 9
2.3.3 Classification 9
2.4 The Object Model 10
2.4.1 Definition 10

2.4.2 Relationships . . . 10
2.4.3 Synchronization . . . 11
2.4.4 Modules . . . 11
2.5 The Class Model . . . 12
2.5.1 Definition . . . 12
2.5.2 Relationships between Classes . . . 12
2.6 The Interplay between Classes and Objects . . . 13
2.6.1 Identifying Mechanisms . . . 13
2.7 Architecture of Very-Large Systems . . . 13
2.8 Frameworks . . . 14
2.8.1 What are Frameworks ? . . . 14
2.8.2 Current Research into Frameworks . . . 15
2.8.3 MacApp . . . 15
2.8.4 Turbo Vision . . . 17
2.8.5 Vamp : The Aldus Application Framework . . . 20
2.8.6 ET++ . . . 20
2.9 Extension Hierarchies . . . 21
2.9.1 System Building with Hierarchy Combination . . . 21
2.9.2 Advantages of this Approach . . . 22
2.10 Problems of Incompatible Class Hierarchies . . . 22
2.11 Experiments into Program Structure . . . 24
2.11.1 An Experiment with COBOL Programs . . . 24
2.11.2 LADY . . . 25
2.12 Functional Decomposition versus Object-Oriented . . . 26
2.12.1 Functional Decomposition . . . 26
2.12.2 Object-Oriented Decomposition . . . 26
2.12.3 Adding or Modifying Functions . . . 27
2.13 Obstacles not Addressed by Current Methods . . . 28
2.13.1 Introduction of Composition Filters . . . 28
2.13.2 Problems with Certain Types of Modelling . . . 29
2.14 Restructuring . . . 29
2.14.1 Tools for Restructuring . . . 30
2.14.2 A Comparison between Computer-Assisted and Manual Restructuring . . . 31
2.15 Re-engineering . . . 32
2.15.1 Re-engineering using an Object-Oriented Architecture . . . 32
2.16 Summary . . . 33

CHAPTER 3 : DECOUPLING THE USER INTERFACE 34

3.1 Introduction 34
3.2 The User Interface Embedded in the Application Processes 34
3.3 Using the Object Oriented Approach 34
3.4 Decoupling Low-Level User Interface Features 35
3.5 Module Coded Menu Structure 35
3.6 A Menu Data Structure 36
3.7 Decoupled Menu Structure and Scheduler 37
3.8 Schedulers 39
3.9 Summary 39

CHAPTER 4 : USER INTERFACE CONSTRUCTION 40

4.1 Introduction 40
4.2 The Appearance of Screen Objects 40
4.3 The Routing of Input 41
4.4 The Behaviour of Screen Objects 41
4.5 The Construction Of Screen Objects 41
4.6 Elector's User Interface 43
4.6.1 Subscreens (Dialogue Boxes) 43
4.6.2 Menus 45
4.7 Dos_Environment Package 45
4.7.1 Halo Graphics 45
4.7.2 Assembly Language 47
4.7.3 Tutorial and Help Screens 48
4.7.4 Additional Information 48
4.8 Summary 48

CHAPTER 5 : ELECTOR CONSTRUCTION 49

5.1 Introduction 49
5.2 Implementation Language 49
5.3 Netlist Abstract Data Type 49
5.4 Restructuring The Program 51
5.5 Scheduling 54
5.6 Floating Point Representation 54
5.7 Summary 56

CHAPTER 6 : SOFTWARE RENEWAL 57

6.1 **Introduction** 57
6.1.1 Software Engineering 57
6.1.2 No Standard Components 57
6.1.3 Software Reuse with the Object-Oriented Paradigm 58
6.1.4 Factors Affecting Maintenance and Renewal 58
6.2 **Program Understanding** 59
6.2.1 Factors affecting Program Understanding 59
6.2.2 Understanding Elecsim Code 60
6.2.3 Problems in Understanding Elecsim 60
6.3 **Problems in Constructing Elector** 61
6.3.1 Language and Design Comprehension 61
6.4 **Time Records** 61
6.4.1 COCOMO Estimates 63
6.5 **Summary** 64

CHAPTER 7 : CONCLUSION 65

7.1 **Conclusions for this Project** 65
7.1.1 General Conclusions 65
7.1.2 Renewal 65
7.1.3 Decoupling the User Interface 66
7.1.4 Program Structure 67
7.1.5 Review of Research Questions 67
7.2 **Suggestion for Further Improvements and Research** 68
7.2.1 Implementation Language 68
7.2.2 User Interface 68
7.2.3 Component Entry 68

REFERENCES 69

APPENDIX A : ELECSIM PASCAL FILES A-1

APPENDIX B : ELECTOR ADA SOURCE CODE B-1

APPENDIX C : ELECTOR ASSEMBLER SOURCE CODE C-1

LIST OF FIGURES

Figure 2-1	MacApp Class Hierarchy	16
Figure 2-2	Turbo Vision Class Hierarchy	18
Figure 2-3	View Tree Example	19
Figure 2-4	Factors influencing people's perceptions of software structure. The largest influence is the source code itself.	30
Figure 3-1	Different Menus in Separate Modules	35
Figure 3-2	Menu Hierarchy formed in the Menu Data Structure	36
Figure 3-3	The Menu Object acting as a Scheduler	37
Figure 3-4	A Scheduler Object requesting a menu option from the Menu Object, and then passing control to the appropriate Application Object.	38
Figure 4-1	Ada Code for Forms Screen Object	42
Figure 4-2	Communication between Application and Subscreen Objects	43
Figure 4-3	The Prompt Subscreen Object	43
Figure 4-4	Ada Code for Subscreen Objects	44
Figure 4-5	The Menu Object	45
Figure 4-6	Elector Menu Hierarchy	46
Figure 4-7	Ada Source for Menus Object	47
Figure 5-1	The Main Objects of Elector	49
Figure 5-2	The Netlist Linked List Structure	50
Figure 5-3	Ada Code for Netlists Package	50
Figure 5-4	Ada Code for Method of an Object	51
Figure 5-5	Program Structure of Elecsim (3 Levels)	51
Figure 5-6	Program Structure of Elector	52
Figure 5-7	Ada Code for Scheduler Package	55
Figure 6-1	Time taken for Renewal Activities	62

LIST OF TABLES

Table 2-1	ET++ Current Implementation	20
Table 2-2	Examples of Conflicts in Incompatible Class Hierarchies	22
Table 4-1	Types of Subscreens (Dialogue Boxes)	44
Table 6-1	Time taken for various Activities	62
Table 6-2	COCOMO values for Basic Model	63

CHAPTER 1

INTRODUCTION

1.1 Software Deterioration

Physical systems *wear out* from the effects of dust, vibration, temperature extremes and many other environmental factors. Also, when a physical component of a system wears out or is defective, it is replaced by another component of the same design.

Software and the logic of systems do not wear out. If a software component is defective, there is a defect in the design or in the translation of the design into the implementation language. Software does deteriorate because modifications degrade it's structure. "New capability, often not recognized during the earlier life of the system, is superimposed on an existing structure without redesign of the system as a whole." [Lehman, 1980]

Lehman categorizes programs into three classes *S*, *P* and *E* [Lehman, 1980].

- *S Programs* : Their function is formally defined by and derivable from a specification. A specific problem is stated. eg. Find the lowest common multiple of two integers.
- *P Programs* : The problem statement is imprecise. It is a model of a real-world situation and contains uncertainties, unknowns, arbitrary criteria and continuous variables. The problem statement and its solution approximate the real world situation. The perception of the problem, it's formulation, the model, the program specification and/or the program implementation will need to be changed when the differences between data from observation and computation is no longer acceptable.
- *E Programs* : They mechanise a human or social activity. The program becomes part of the world it models - it becomes embedded in it. eg. computer operating and air-traffic systems.

P and *E* type programs are developed for an environment that is constantly changing. These programs are maintained to keep them functioning in this changing environment. However, each maintenance exercise has the negative effect of degrading the overall structure of the software. A point is reached where the software becomes so difficult to maintain that it is discarded. These software programs become prime candidates for renewal. An attempt can then be made to recover the intellectual and financial investment in these software programs.

1.2 The Need for Renewal

Renewal is more than translation - it is the extraction of algorithms and intellectual content of an existing program, and then the building of a new program using the most modern and effective programming methods and languages.

In the 1950's and 1960's, programmers spent most of their time creating new programs. A large body of software was created covering most engineering and scientific fields [Caspers-Jones, 1984]. This software was designed using functional design concepts which leaves the software hard to change without introducing defects. Very often, the software becomes incomprehensible and is discarded.

Many numerical algorithms were developed long before the existence of computers, including the mathematical modelling of many physical and engineering systems. In early electronic computer systems, many of the software programs have sound numerical algorithms. There should be an attempt to extract such algorithms and save both human and financial resources. "We cannot and should not declare *old* software obsolete or not worth studying. This collection of functions is an important asset, embodying a wealth of experience, and constitutes an inventory of *ideas* for identifying the building blocks for future systems. Even if the code itself is inelegant and possibly not reusable, a study of the specifications and identification of the most frequently used components could reveal a set of generic classes of algorithms and functions which could be reusable in future systems." [Schneidewind, 1987] *P* and *E* type programs have the greatest potential for renewal.

Renewal requires that a program be re-implemented within a new architectural framework, as well as a new implementation language [Walker, 1988]. Renewal is different from maintenance - translation.

- Maintenance is the incremental modification of a program with a view to correcting latent errors or adding further features after the product has been delivered [Schneidewind, 1987]. This process generally degrades the overall program structure.
- Translation from one implementation language to another may achieve the same functionality, but the existing structural deficiencies will be retained.

By renewing a program instead of starting from scratch :

- The existing program is a tangible expression of the specification.
- The existing program is a benchmark for evaluating the performance and results of the renewed program.

1.3 Computer-Aided Circuit Design CACD

CACD packages are examples of *P-programs*. This makes them prime candidates for renewal.

CACD overcomes problems inherent in the manual breadboard approach and is an important tool for the electrical engineer. An electronic circuit without CACD is designed with analytical techniques, design charts and tables. The circuit prototypes are then build on a breadboard. Various tests are undertaken to verify the performance of the circuit. Optimisation by trail and error may be undertaken, but tolerance and worst-case analysis are difficult to perform.

With CACD, the ability to test a whole range of scenarios and ask *What if...?* questions permits the design of circuits with greater stability and applicability [Anderson, 1978], [Blume, 1986]. CACD is also a powerful tool in the education of engineers. Circuit theory is taught with a series of lectures, tutorials and laboratory exercises. There is limited time for practical experimentation, with each circuit taking time to construct on the breadboard. *What if...?* questions are generally not feasible.

A wide variety of CACD packages are available. A well known example is SPICE [Walker and McGregor, 1986]. Elecsim is a circuit simulator that analyses circuits using linear models and methods. It was renewed from Linsim [Edward, 1983a, 1983b] which was build on top of Lincad.

1.4 Aim and Scope

It is the purpose of this dissertation to present an examination and case study into the feasibility of renewing software by decoupling the user interface and using the object-oriented paradigm. The study uses a modern design method (object-oriented) and a modern general-purpose programming language (Ada). Elecsim, a linear circuit simulator written in Pascal, was chosen as the existing program to be renewed. The PC with EGA / VGA and mouse was chosen to be the hardware platform on which the program was to execute. The renewed program is Elector.

1.4.1 The Renewal and Maintenance Exercise

The results of the renewing exercise can be used to determine the productivity of such an exercise. Elecsim consists of about 12000 lines of Pascal code. The results that emerge are more closely related to the renewing of large software systems. The size of the project is enough to get meaningful results with one to two years work, but small enough for a single person to complete. The main questions to which answers are sought include:

- How productive is renewal ?
- To what degree can these findings be extrapolated to provide general lessons ?

1.4.2 Program Structure

A survey of the existing literature into how the structure of the program itself affects maintainability. Using functional decomposition methods, the structure of the program is a pyramid structure. The disadvantage of this is the difficulty in :

- Modifying a function.
- Adding a function.

A strategy worth investigating is the flattening out of the structure so that all the major processes are on the same level and invoked by a scheduler. Adding a function is done by simply adding it to one side of this flattened structure without the need for modification of the structure itself.

1.4.3 Scheduling and Concurrent Process

Investigation into the effect of hierarchy in the scheduling of processes. A hierarchy is an implicit scheduler for programs designed using the functional decomposition design method.

- This implicit scheduler is distributed amongst the modules all down the hierarchy which represent the various processes of the program. Modifying the program to add a new process means tampering with this scheduler. The impact of these modifications is often not visible initially.
- Modifying the program with this implicit scheduler to run on parallel hardware using concurrent programming techniques is difficult.
- To invoke another process in a different part of the hierarchy can usually only be done by returning to the highest level of the hierarchy and then going down it to the desired process. This means that every process going up this hierarchy tree has to be terminated, and every process going down this hierarchy tree invoked before reaching the desired process.

1.4.4 Decoupling the User Interface

In many programs where there is intensive user-program interaction, the user interface *(UI)* is tightly coupled to the hierarchy. Any change in the *UI* means that the program has to be modified with changes usually affecting unrelated processes. Also, any process that is modified or added will have a direct impact on the *UI*.

Coutaz notes "Other principles address the structural relationships between the system and the user interface, particularly on the need for a rigorous separation between the functionality of a computer system and its user interface...Distinguishing the user interface from the applications is a reasonable design principle because a proper modularity permits the user interface and the applications to evolve more independently." [Coutaz, 1985] Hartson adds "..almost everyone now agrees that some kind of separation is needed between the dialogue and computation components." [Hartson, 1989]

Therefore, some method must be found to separate the *UI* from the rest of the program - to become application independent.

1.4.5 Graphical User Interface Following CUA Guidelines

The new version should have a graphical user interface similar to that specified by the IBM [Wheeler and Ganek, 1988], [Dunfee, et al, 1988] Common User Access guidelines. These guidelines are part of the Systems Applications Architecture. The Systems Applications Architecture is a framework in which applications are developed so that they run consistently on major IBM computing systems. Support is provided for cross-system compatibility as well as for distributed applications, including distributed processing and distributed data. SAA uses a generic software structure model as a basic building block. One of the design goals is to build replaceable, extensible applications. This is partly achieved by building on a common base. This base must be identical in all SAA environments. SAA consists of three elements:

- *Common User Access (CUA)* : This interface [Berry, 1988], [Uhlir, 1988] controls the interaction between a person and the system. It ensures that the way things look to the user and the actions required from the user are familiar. This allows skills acquired from one application or system to be transformed to another system. The CUA has pop-up windows, pull-down menus, action-bars, etc.
- *Common Communications Support (CCS)* : This interface deals with how systems work together to get a job done. It controls how systems communicate with one another and how they move information through the communications network.
- *Common Programming Interface (CPI)* : This specifies how a new application is to be written by a programmer. It allows applications to be independent of the underlying system by defining a set of application building blocks consisting of languages and programming services for application programmers.

The benefits for applications that comply are:

- Easier to learn and use due to CUA.
- Execute on a broader range of environments due to CPI.
- Interconnection of different systems as needed due to CCS.
- Utilization of advanced technologies such as relational data and cooperative and distributed processing.

1.4.6 Use of a Modern Language

The renewed package based on Elecsim should be implemented in a modern language that is designed with modern software engineering principles in mind. Such a language is Ada [Gonzalez, 1991], [Savitch and Petersen, 1992]. Ada is a very powerful, general purpose, structured programming language. This language embodies many key software engineering principles, from information hiding and modularity to separate module compilation. This language is the only language to be used in all embedded applications of the Department of Defense of the United States of America. The size of this organization, as well as the intense research being done into

Ada, means that it will have a substantial software base in the future.

Ada is not a true object-oriented language, even though it has features which do support object-oriented programming to a large degree. Two different approaches to rectify this are :

- *Preprocessor* : This translates the *Object-Oriented* Ada into standard Ada. There is a commercial product called Classic-Ada which supports inheritance, dynamic binding and polymorphism.
- *Ada9x* : This is a project which will come up with a new Ada object-oriented language. Research and discussion is been undertaken to make the object-oriented mechanisms of this new language both reliable and consistent with the semantics and style of standard Ada.

By implementing Elector in Ada, a comparison can be made between it and Elecsim in Turbo Pascal 5.0, and possibly Linsim in Fortran.

1.5 Structure of the Dissertation

The dissertation is divided into three main parts.

1.5.1 Main Body

It contains a description and critical examination of the renewal exercise, and the user interface decoupling approach. A survey of studies into renewal, maintenance and software structure is made. Also, a survey of important technical factors which have an influence on program maintenance is made. The method used in understanding the behaviour of the program to be renewed is discussed. Observations on the renewal exercise are presented.

1.5.2 User Reference Manual

It describes how Elector behaves from the user's perspective. It covers a brief history of Elector, followed by an explanation of how the user interface works. Each menu option is discussed in detail. Finally an example is given which covers most of the important features of Elector.

1.5.3 Technical Reference Manual

It describes how Elector was constructed and behaves from a programmer's perspective. It covers four main areas.

- The main objects from which Elector is constructed are implemented in separate compilation modules. Each module is discussed in terms of :
  - The inter-relationship and visibility between the different modules.
  - Components (Large data types and data structures).
  - Methods (Procedures and functions).
- The NETLIST_TYPE abstract data type is discussed in detail. This is the foundation ADT of the application part of Elector and holds all the information about the components in the circuit being analysed.
- The behaviour of the major objects in Elector is discussed using data flow diagrams and program description language.
- The construction and behaviour of the user interface is discussed in several chapters. This arises because the user interface is build as a set of abstract layers. Each *layer* is discussed in a separate chapter. The chapters are :
  - *Dos Environment Package* : Highly implementation dependent and allows access to the underlying computer in a controlled way.
  - *Screen Objects* : Smallest objects which appear on the screen and from which all the other objects are build.
  - *The Help Object* : Provides on-line help information.
  - *The Screens Object* : Provides subscreen objects which get and give information from the user to the application processes of Elector.
  - *The Menus Object* : Provides menus from which the user chooses the next process to execute.

CHAPTER 2

SOFTWARE STRUCTURE

2.1 Introduction

This chapter focuses on software structure and how it affects software maintenance and related activities. Experiments into program structure are discussed. Developments in frameworks, restructuring, re-engineering are also briefly discussed since they all have an effect on maintenance. Functional decomposition and object-oriented design methods are briefly compared.

2.2 What is Software Structure

Ossher gives a good introduction to software structure [Ossher, 1986] :

- "An important part of understanding a program is understanding its structure: new readers of a program generally try to gain an appreciation of global structure before they look at any details. Structure and an understanding of structure are therefore of vital importance when working with programs."
- "In this context, the structure of a program means the *organization* and *interactions* of its parts. Each part is termed a unit. Examples of units are modules, procedures, statements, classes and methods; the precise nature of the units depends on the programming language being used and the granularity of interest."
- "As programs grow large, their structure becomes complex, and difficult to deal with. This is particularly true in the case of modular programs: partitioning of concerns leads to modules that are easy to understand in isolation, but to so many of them that it is difficult to understand their relationships."

2.3 Complex Systems

2.3.1 Hierarchies

Booch notes that there are many different hierarchies present within the same complex system. Each hierarchy is layered with the more abstract classes and objects built upon more primitive ones. What is considered primitive is dependent on the problem at hand.

- "Hierarchy is a ranking or ordering of abstractions" [Booch, 1991]
- "Not only are complex systems hierarchical, but the levels of this hierarchy represent different levels of abstraction, each built upon the other, and each understandable by itself."
- "We choose a given level of abstraction to suit our particular needs."

The two most important hierarchies in a complex system are its class hierarchies (*kind of* hierarchy) and its object structure (*part of* hierarchy). *Kind of* hierarchies specify generalization and specialization relationships. *Part of* hierarchies describe aggregation relationships. In the *part of* hierarchy, a class is at a higher level of abstraction than any of the classes that make up its implementation.

The class structure allows the capture of common properties of system parts in one place. Thus, knowledge does not have to be duplicated for each object.

2.3.2 Attributes of Complex Systems

Booch lists five attributes which are common to all complex systems [Booch, 1991] :

- "Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached."
- "The choice of what components in a system are primitive are relatively arbitrary and is largely up to the discretion of the observer of the system."
- "Intracomponent linkages are generally stronger than intercomponent linkages. This fact has the effect of separating the high-frequency dynamics of the components - from the low-frequency dynamics - involving interaction among components."
- "Hierarchic systems are usually composed of only a few different kinds of subsystems in various combinations and arrangements."
- "A complex system that works is invariably found to have evolved from a simple system that worked....A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a simple system."

2.3.3 Classification

"The incremental and iterative nature of classification directly impacts the construction of class and object hierarchies in the design of a complex software system. In practice, it is common to assert a certain class structure early in a design and then revise this structure over time." [Booch,1991]

"Only at later stages in the design, once clients have been built that use this structure, can we meaningfully evaluate the quality of our structure....On the basis of this experience, we may decide to create new subclasses from existing ones (derivation). We may split a large class into several smaller ones (factorization), or create one larger class by uniting smaller ones (composition)."

There is no such thing as the *perfect* classification, but some classifications are better than others. Any classification is relative to the perspective of the observer doing the classification.

"You do not always design types in a type hierarchy by starting with a supertype and then creating the subtypes. Frequently, you create several seemingly disparate types, realize they are related, and then factor out their common characteristics into one or more supertypes...several passes up and down are usually required to produce a complete and correct program design."

2.4 The Object Model

2.4.1 Definition

"An object has state, behaviour, and identity; the structure and behaviour of similar objects are defined in their common class; the terms instance and object are interchangeable....The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties....Behaviour is how an object acts and reacts, in terms of its state changes and message passing." [Booch, 1991]

The state represents the condition of an object at a certain moment. This condition is expressed in terms of the values of local variables. Events indicate the transitions from one state to another. In object-oriented systems, events are usually initiated by the reception of messages.

Because each object has state, the order in which operations are invoked is important. Each object can be considered to be like a tiny, independent machine (finite state machine). Also, because the object has state, it must take up some amount of space in the physical world or in computer memory.

- *Active Objects* : They contain their own thread of control and are generally autonomous (they can exhibit some behaviour without being operated upon by other objects). They therefore serve as the roots of control.
- *Passive Objects* : They can only undergo a state change when explicitly acted upon.

There is only one active object at a time (one thread of control) in sequential systems, but there may be many in concurrent systems.

2.4.2 Relationships

There are two types of relationships among objects :

- *Using* : Where one object uses the services of another independent object.
- *Containing* : Where the object consists of (is constructed from) smaller objects.

There is a tradeoff related to the use of either relationship.

- *Containing* rather than *using* reduces the number of objects that must be visible at the level of the enclosing object.
- *Using* rather than *containing* reduces coupling among objects which is desirable.

2.4.3 Synchronization

Whenever one object passes a message to another with which it has a using relationship, the two objects must be synchronized in some manner. Subprogram calls synchronize the objects in a completely sequential application.

2.4.4 Modules

In certain languages, (Object Pascal, C++, CLOS, and Ada) the module is a separate language construct which requires a separate set of design decisions. There is often a separation between the interface and implementation of a module. In Ada, the body (implementation) of a module can depend upon modules which are not visible to the module's specification (interface).

Classes and objects form the logical structure of a system, the modules produce the physical structure of the system. Modules can therefore help deal with the complexity of large systems where there are hundreds of classes. They can serve as physical containers in which classes and objects are declared.

"In traditional structured design, modularization is primarily concerned with the meaningful grouping of subprograms, using the criteria of coupling and cohesion. In object-oriented design, the problem is subtly different : the task is to decide where to physically package the classes and objects from the design's logical structure, which are distinctly different from subprograms." [Booch, 1991]

"Finding the right classes and objects and then organizing them into separate modules are entirely independent design decisions. The identification of classes and objects is part of the logical design of the system, but the identification of modules is part of the system's physical design."

Another important point raised by Booch is that of defining the main program from which the application can be invoked from the operating system. Defining the main program is one of the least important decisions in object-oriented design, but in structured design, the main program serves as the root and the binding that holds everything together.

2.5 The Class Model

2.5.1 Definition

"A class is a set of objects that share a common structure and a common behaviour." [Booch, 1991]

2.5.2 Relationships between Classes

There are four types of relationships :

- *Inheritance* relationships (includes single and multiple inheritance)
- *Using* relationships
- *Instantiation* relationships
- *Metaclass* relationships

Inheritance : It uses the same technique people use when trying to understand a new concept. They compare it with something that is already known.

"Designing a suitable inheritance hierarchy among abstractions is largely a matter of intelligent classification." [Booch, 1991]

The problem with single inheritance is apparent when subclasses are derived from two equally attractive classes can be derived from. Because only one class can be chosen, the functionality of the excluded class must be reimplemented. A problem that arises in multiple inheritance is that where class D has two parents B and C, both of which has A as a parent. D would then inherit twice from A. To get around this problem :

- Occurrences of repeated inheritance is illegal.
- Superclasses can be duplicated, but fully qualified names are needed to refer to members of a specific copy.
- Multiple references to the same class are taken to be that class.

Inheritance can violate encapsulation in one of three ways [Liskov, 1988] :

- A subclass might access an instance variable of its superclasses.
- Call a private operation of it's superclass.
- Refer directly to superclasses of its superclass.

C++ offers some protection with 3 parts to the interface of classes:

- *Public* : Visible to all clients.
- *Protected* : Visible only to the class and its subclasses.
- *Private* : Visible only to the class itself.

Using Relationships : A class interface can use another class, or a class's implementation can use another class. In the first case, the used class must also be visible to any clients. Booch maintains that using multiple inheritance to indicate aggregation (an object (class) contains other objects) is a bad idea [Booch, 1991].

"Our rule of thumb is that if an abstraction is greater than the sum of its component parts, then using relationships are more appropriate. If the abstraction is a kind of some other abstraction, or if it is exactly equal to the sum of its components, then inheritance is a better approach."

2.6 The Interplay between Classes and Objects

Generally, classes are static, but objects are created and destroyed at a phenomenal rate during the lifetime of the program.

2.6.1 Identifying Mechanisms

In object-oriented design, the key abstractions are first identified to form a model of reality. Only after this is the behaviour of the abstractions added to produce the observable behaviours of the system.

Designing a class embodies the knowledge of how individual objects behave, but mechanisms describes how collections of objects work together. Thus, key abstractions reflect the vocabulary of the problem domain and mechanisms the soul of the design.

2.7 Architecture of Very-Large Systems

"For very large systems, we find classes, objects and modules provide an essential yet insufficient means of decomposition. Fortunately, the object model scales up. In large systems, we find clusters of abstractions build in layers on top of one another. At any given level of abstraction, we find meaningful collections of objects that cooperate to achieve some higher level behaviour." [Booch, 1991]

At this level, the objects of interest are subsystems and their interactions, rather than data structures and algorithms [Shaw, 1989]. An automatic classification tool called ARCH with a graph editor has been developed to help the architect of the system to visualize and construct it [Schwanke, Altucher, and Platoff, 1989].

Object-oriented techniques have been applied to the development of large, evolving, distributed,

real-time systems [Buhr and Casselman, 1992]. The architecture of such systems can change dynamically. For example, a component could be added to the system while the system is running. The component is given the identifiers of other specific components with which it may need to communicate. Sending information to these components then begins an interchange process that enables temporary point-to-point links to be established between the new component and the existing ones. These links may change over the course of time. These *softlinks* form the basis by which groups of components dynamically collaborate.

It is easiest to view this type of system as consisting of a sea of components that come and go and that have the dynamically-changing groups overlaid on top of them.

2.8 Frameworks

The most promising development in the object-oriented arena seems to be that of frameworks. What follows is a description of what a framework is, and four examples are discussed.

2.8.1 What are Frameworks ?

Wirf-Brock, et al, describes a framework as [Wirf-Brock, et al, 1990] :

> "An object-oriented framework provides a skeleton for implementing an application or application subsystem in some problem domain. A framework is typically composed of a mixture of concrete and abstract classes along with a model of interaction among the classes. A framework is used by configuring or connecting the predefined concrete classes and by deriving new concrete classes from the abstract classes. Normally, the model of interaction and control flow among the classes is not modified when using a framework."

Wirfs-Brock and Johnson give a shorter description [Wirfs-Brock and Johnson, 1990] :

> "A framework is a generalization of the subsystems that can be built from it."

Schmucker describes a framework as such [Schmucker, 1986] :

> "An application framework is only the shell of a real application - a shell that you can easily customize into a true application. This customization process differentiates an application framework from a set of merely useful subroutines."

Frameworks are not simple class libraries. They require more work to design. What is probably of greater reuse than the code is the design of the framework. In other words, the interface and specification of the components.

Frameworks were first used for the user interface construction. Examples of this are the Smalltalk-80's Model/View/Controller and Borland's Turbo Vision. However, their application has become more extensive. MacApp has functionality for complete, generic applications.

"Because the basic set of classes and their models of interaction are shared by all users of a framework, it is appropriate to think of a framework as a reusable design. Designing a framework is difficult because the designer must develop a solution that spans an entire problem domain and then implement that solution in an extensible and reusable manner."

2.8.2 Current Research into Frameworks

The design of a framework is in itself research. A theory of the problem domain is developed and is expressed with an object-oriented design. Wirfs-Brock and Johnson have identified three main areas of research :

- *Designing Frameworks* : What are the characteristics of a good framework and how is one designed.
- *Using Frameworks* : How does one configure a particular application based on a framework.
- *Describing Frameworks* : What notation is needed, other than that applicable to object-oriented design in general.

Research is also being done into constructing frameworks for specific application domains and operating system domains.

2.8.3 MacApp

MacApp [Schmucker, 1986] is an application framework that knows generally what a Macintosh application is supposed to do. It knows how the menus work and how to scroll windows, where and when to pass control to an activated desk accessory, and so on. In other words, it knows about all the things that are common to Macintosh applications.

This framework is more than just skeleton with a fixed number of *pluggable slots* for commands and selection. You can override every major and minor decision. To get this flexibility, the framework is set up as a group of classes or class library. If your application is to behave in a unique way, new classes and methods can be added.

The Basic Structure of MacApp

MacApp contains more than 30 different classes and over 450 methods. The class *TApplication* takes care of the things that are the responsibility of the application as a whole :

- Launching an application

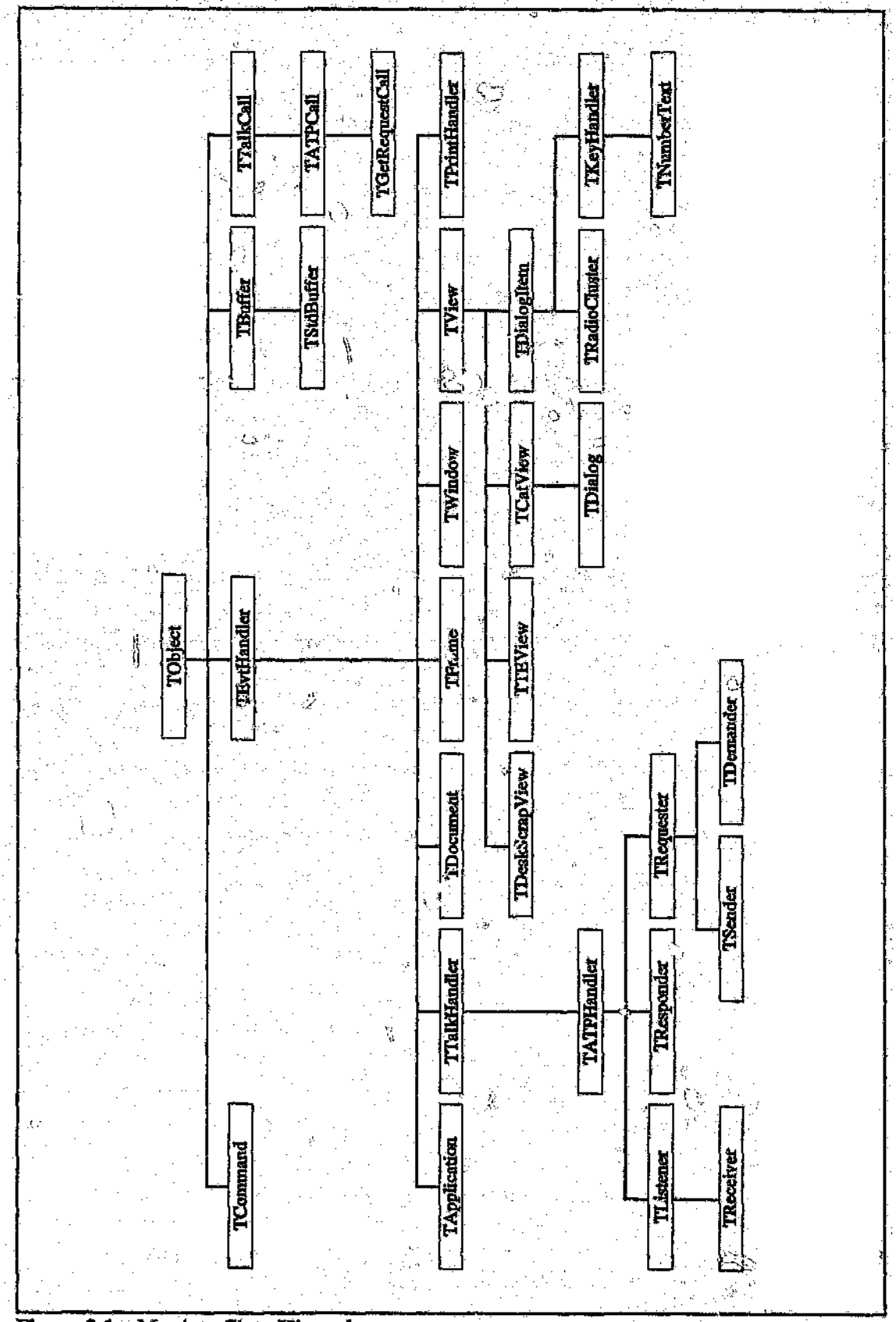


Figure 2-1 MacApp Class Hierarchy

- Setting up the menu bar
- Deciding which documents to display in the "Open Which Document?" dialogue box.

The new application will be a special subclass of *TApplication* with the methods that need to be modified overridden. The class *TView* handles everything inside the windows. It knows when a part of a window needs to be redrawn. It doesn't know exactly how to do this and relies on you to override the methods that supply these behaviours in your subclasses of *TView*.

The Benefits and Costs of Using MacApp
Schmucker notes the time needed for application development :

> "Early studies indicate that MacApp can reduce application development time by a factor of four to five. MacApp also decreases the amount of source code you need, again by a factor of four to five."

He also notes that the consistency of the Macintosh user-interface standard is maintained. Error handling and an interactive debugging facility is provided. Probably the most important thing MacApp does is provide a conceptual framework that lets you concentrate on your application rather than on Macintosh internal details.

2.8.4 Turbo Vision

Turbo Vision [Borland, 1990] is an object-oriented application framework for windowing programs. It consists of :

- Multiple, resizeable, overlapping windows
- Pull-down menus
- Dialogue boxes
- Buttons, scroll bars, input boxes, check boxes and radio buttons
- Mouse support
- Standard handling of keystrokes and mouse clicks
- Built-in colour installation

"In traditional structured programming, when a tool such as a menu manager doesn't quite suit your needs, you modify the tool's source code until it does. Furthermore, changing proven source code (especially code written by someone else) is a fine way to introduce obnoxious new bugs into a proven subsystem, bugs that could propagate far beyond your area of original concern."

"With Turbo Vision, you never have to modify the actual source code. You *change* Turbo Vision by extending it....You add to it by deriving new object types, and change what you need to by overriding the inherited methods with new methods that you write for you new objects."

Turbo Vision is a hierarchy, not a disjoint box of tools. "There is a single architectural vision behind every component of Turbo Vision, and they all work together in many subtle, interlocking ways."

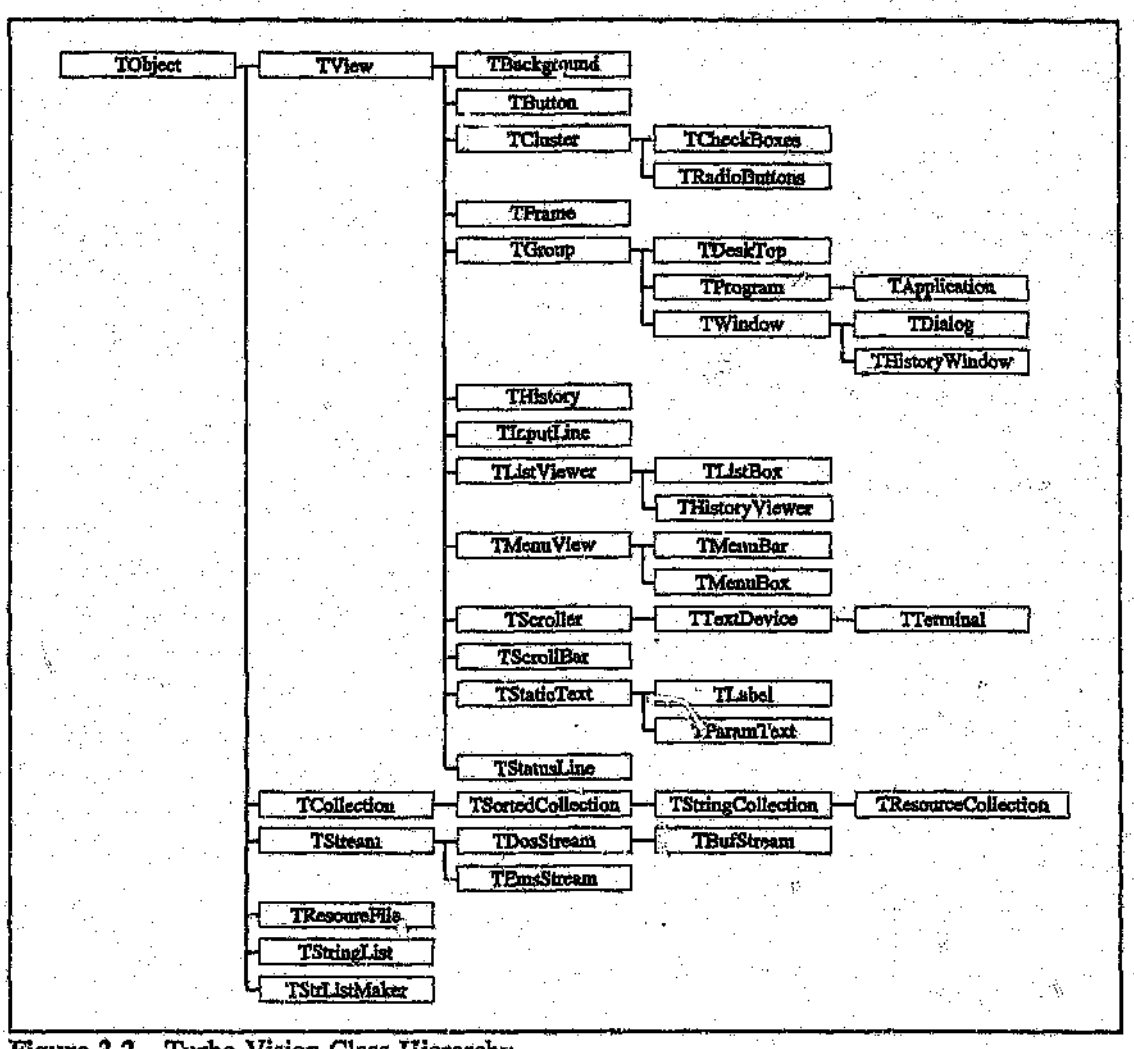


Figure 2-2 Turbo Vision Class Hierarchy

Flow of Execution and Events

The Turbo Vision framework deals with user input in a different manner when compared to conventional programs. The central ideas are that of events and routing of events. Events are not objects because they do not perform any actions themselves. They only convey information to objects - they are some sort of occurrence to which your application must respond. Events come from the keyboard, from the mouse, or from other parts of Turbo Vision.

Event-driven programs separate the control structures that read and evaluate user input (and other events) from procedures and functions that act on that input. In other words, the function of getting input is separated from the function of responding to that input.

- "Calling program typically contain many blocks of code, each of which involves getting some input, deciding which code gets that input, calling the appropriate routine(s) to process the input, then doing the same thing again. In addition, the code that finishes with the input must then know where to give control for the next round of input."
- "Event-driven programs, on the other hand, have a central event-dispatching mechanism, so the bulk of your program does not have to worry about fetching input and deciding what to do with it. Your routines simply wait for the central dispatcher to hand them input to process."

To get events to the right places, routing of events is done. Some events are broadcast all over the application, while others are directed to particular parts of the program.

Relationships between Views

A view is any program element that is visible on the screen - and all such elements are objects. There are two distinct relationships between views :

- They are members of the Turbo Vision object hierarchy. These relationships are as parents and children. (See Figure 2-2)
- They are members of the view tree. The view tree represents the relationship between instances of classes - between owner and subview.

Example : In the view tree diagram (See Figure 2-3) the *TDialog* owns the *TButton*. This relationship is not between hierarchical object types (*TDialog* is not an ancestor of *TButton*). A *TButton* object interacts with its owner in the view tree - *TDialog*, but will draw upon attributes inherited from it's ancestor (*TView*).

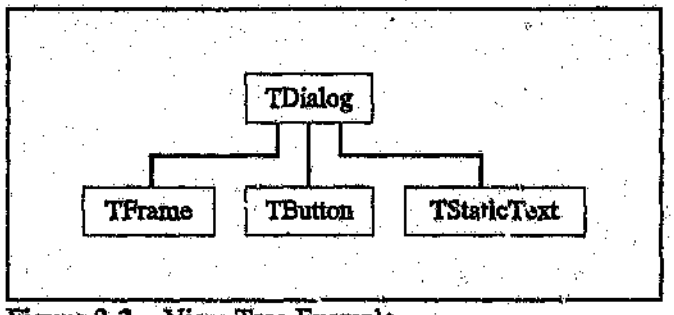


Figure 2-3 View Tree Example

"As your Turbo Vision application opens and closes windows, the view tree grows and shrinks as object instances are inserted and removed." This statement illustrates the difference in structure between an object-oriented design and a functional decomposition design. The view tree would be part of the structure or architecture of the program if it was developed by functional decomposition methods. This view tree would have been hard-coded. However, by using object-oriented techniques, this structure is dynamic which grow and shrinks during the program execution lifetime.

2.8.5 Vamp : The Aldus Application Framework

An application framework called Vamp [Ferrel and Meyer, 1989] was developed by Aldus as a practical experiment in solving several software engineering and project management problems. Vamp is written in C++ to run on the Macintosh and Microsoft Windows. It is an event-driven collection of classes. The main objective was to develop a baseline set of limited *foundation classes* upon which to build further software projects. It therefore addressed architectural issues rather than providing a fully functional application.

Few conclusions were drawn from the project other than it now provided a conceptual and design framework from which future products could be build.

2.8.6 ET++

ET++ is another object-oriented application framework [Weinand, Gamma, and Marty, 1988] implemented in C++ for UNIX environment with a conventional window system. A collection of user interface building blocks, basic data structures as well as the MacApp application framework were used for the architecture of this framework.

The main aim of this project was to design and implement an object-oriented application framework for a UNIX environment with about the same functionality as that of MacApp and the user interface elements of the Macintosh toolbox. Portability was also a major issue.

The ET++ Architecture

"The basic building blocks of the ET++ architecture are a class hierarchy and a small device independent layer mainly mapping an abstract window system interface to the underlying real window system."

Foundation classes represent basic data structures (array, list, set, etc). Graphical elements such as menus, dialogs, scrollbars are implemented in the user interface classes. Application classes are high level classes that factor out the common control structure of applications running in a highly graphical environment.

ET++	Lines of Code	Classes	Methods
Foundation Classes	6000	34	403
Application Framework	13200	97	1071
Device Interface	6800 (3000)	9	179

Table 2-1 ET++ Current Implementation

Conclusion

The ET++ application framework provides the same type of application architecture for UNIX environments that is found in MacApp environments. This should provide greater consistency and portability of applications.

2.9 Extension Hierarchies

An approach in which extensions of all kinds are clearly separated from the base hierarchy is proposed [Ossher and Harrison, 1992]. Usually extending existing classes is done by editing the code associated with those classes, which is undesirable because bugs can be introduced in functioning code. Extensions, including extensions to existing classes, are written in separate, sparse extension hierarchies. The entire system is obtained by combining the extension hierarchies with the base hierarchy. This differs from subclassing which is a form of extension-by-addition. Extending existing classes is important when :

- Objects created by existing code need to exhibit some extended behaviour. The creation code generally names specific classes, but to obtain extended behaviour without changing the creation code, one must extend those classes.
- Existing, persistent objects need to exhibit some extended behaviour. The classes to which those objects belong need to be extended.
- Instances of all existing subclasses of an existing class need to exhibit some extended behaviour.

Sequences of successive extensions can be combined using an extension operator, and parallel extensions are combined with a merge operator. The difference between extension and merge has to do with overriding. When extending a base hierarchy, extensions are allowed to override some details of the base. When merging two independent extensions, the one cannot override the other because then some of the changes made will be lost.

2.9.1 System Building with Hierarchy Combination

The use of hierarchy combination leads to a new approach to system building, which involves two levels.

- *Higher Level* : Existing base and extension hierarchies are combined using the extension and merge operators. The extensions are kept in a library, and the environment provides assistance with locating suitable extensions and checking that their combination is valid.
- *Lower Level* : Writing of individual extensions for cases where there is not suitable extension in the library.

2.9.2 Advantages of this Approach

- Extension of existing classes and changes to the superclass structure, as well as addition of new classes. Extension of existing classes includes introducing new instance variables and methods, and overriding existing methods.
- Extensions are well separated, allowing distribution of extensions and selection of the desired extensions from a library.
- In the absence of conflicts, it supports integration of separately developed extensions and upgrading of the base hierarchy.
- It is a linguistic approach with precise but easily-understood semantics.
- It depends only on a few standard properties of inheritance hierarchies, such as class names, descriptions and superclasses.
- Implementations can support combinations of hierarchies whose source form is not available.

2.10 Problems of Incompatible Class Hierarchies

Berlin addresses the issue of reusable class libraries which are not compatible due to architectural mismatches such as error handling and composition conventions [Berlin, 1990]. In the article, five pragmatic dimensions along which combinations of sub-systems must match are discussed.

Pragmatic Issue	Example Conflicts	
	Caller Expects	Callee Expects
Argument Validation	Interactive validation	Declarative validation
Error Handling	Error value	Non-local jump (throw)
Composite Objects	Top-down creation. Outside-in event handling	Bottom-up creation. Inside-out event handling
Control and Communication	Responsibility for event loop. Responsibility for cleanup.	Responsibility for event loop. Do cleanup / notification.
Group and Compound Operations	Single setup / cleanup for a group operation	Each element encapsulates it's own setup / cleanup

Table 2-2 Examples of Conflicts in Incompatible Class Hierarchies

Issue 1 : The problem of argument validation. Argument validation in the user interface requires co-operation among many layers of an application. There are three ways for the user to specify arguments :

- *Declaratively* : All arguments are validated together. This results in simplified module

interfaces. However, the user is informed of incorrect input only after all the input has been entered.

- *Interactively* : The user gets feedback on each argument. This requires that the interface knows more semantics of the underlying application.
- *Implicitly* : The arguments are validated according to the user's preferences, the operation or the context of the program.

Issue 2 : The problem of error handling. Error handling involves detecting resource failures and incorrect values. The type of problem must be identified and handled or passed up (with some cleanup if appropriate). Problems that occur here are :

- Passing the problem, but the higher layers are expecting the problem to be handled by the layer in which it occurred.
- Not passing the buck. Here, the higher layers know how to handle the problem, but are not given a chance.
- Incompatible conventions. Layers do not understand one anothers error conventions.

Issue 3 : The problem of which element gets input in a composite object. Composite objects are structured objects who elements are other objects. Problems occur when an element tries to exercise bottom-up control in a top-down composite (tries to get input even though there are other objects specifically designed to do this), or if an element expects to be given instructions and so fails to get them.

- *Creation Style* : Objects that expect to be given input have outside-in event handling. Objects that go and get input have inside-out event handling.
- *Event-Handling* : Problems occur when events are incorrectly intercepted, or dropped if the events are not the appropriate ones. The order in which events are handled can also cause problems.

Issue 4 : The problem of who is in control, who is a messenger and who coordinates. As an example, user interface functions can be called by the application (internal control) or the user interface can call the application when the user gives a command.

Issue 5 : The problem of inefficient and redundant actions, extra error possibilities and complex undo actions in case of error. These problems are caused by encapsulation at the wrong level, but that the correct level is not possible to foresee.

The following conclusions were drawn in the article :

- "Independently sensible architectural decisions can lead to fundamental conflicts among complementary reusable components"
- "These pragmatic decisions are pervasive yet neglected by Object-Oriented methodology."
- "Software descriptions should include not only the semantics but also the global pragmatic

decisions of the systems architecture, since these decisions must be compatible across subsystems."

The pragmatic decisions discussed need to be explicitly stated for reusable software. This will allow developers to build systems from subsystems whose architectures are compatible.

2.11 Experiments into Program Structure

2.11.1 An Experiment with COBOL Programs

Gibson and Senn conducted an experiment into the effects of program (system) structure on maintenance [Gibson and Senn, 1989]. An old, ill-structured program was improved in two stages. This resulted in three versions of the program. The program was an ISAM file update system consisting of about 2000 lines of Cobol code.

The second system had the following improvements over first system.

- Long jumps removed.
- Knots removed.
- If nesting added.
- Intricacy of PERFORMed hierarchy added.

The third system had the following improvements over the second system.

- Redundancies removed.
- Increased and higher abstraction.
- Intricacy of PERFORMed hierarchy added.

In the study, four variables were measured.

- *Time* required to implement changes to the system.
- *Accuracy* of the modifications.
- *Confidence* that the programmers have in the correctness of their work.
- *Perceptions* of programmers to the relative complexity of the system.

Three different maintenance tasks were performed by experienced professional programmers to each of the three versions of the programme. The study found that:

- The second version produced only marginal benefits in terms of programmer performance (time) over the first version. However, ripple effect errors were almost reduced in half.
- The third version improved performance in every measure.
- Programmers could not separate the complexity of the program from the complexity of the task at hand. Programmers were inconsistent in their ranking of programs according to complexity.
- The six metric measures that were used produced objective ranking which were consistent

with the actual performance of the programmers.

This study shows that modifying a program's structure to improve it's maintenance does indeed seem to help maintenance programmers. This study was conducted only on programs that had been constructed using the top-down functional decomposition method. They therefore have hierarchies that relate processes to one another.

2.11.2 LADY

The study using LADY (LAnguage for Distributed sYstems) [Rombach, 1987], [Rombach and Ulery, 1989] was to investigate the effect that a set of structural concepts had on the maintainability of the resulting software using the language.

- *Goal 1* : Determine differences in the maintenance behaviour of systems implemented in LADY and a traditional Pascal-like implementation language due to differences in the structural concepts and features (language designer's perspective)
- *Goal 2* : Determine measurable structural criteria (complexity metrics) that allow the explanation or prediction of the maintenance behaviour of a LADY system (language user's perspective).

A controlled experiment was used so that the impact of structural software aspects on maintainability could be analyzed in isolation. Three classes of complexity metrics were used :

- *Internal* (or code oriented) - length, structure and intensity of embedding.
- *External* (or system oriented) - information flow between units with or without implicit flows.
- *Hybrid* (combinations of metrics from the other two classes)

The conclusion drawn from this study were :

- "Although it is not yet known what constitutes a good distribution, it is evident that the distribution of structural complexity across different levels influences maintenance (especially comprehensibility)."
- "LADY was found to improve the maintainability of software products by structuring them according to application inherent cohesion and coupling."

In the same article by Rombach and Ulery, they also describe how a controlled experiment was conducted to identify the impact of specific Ada features such as packaging, generics, and tasking on maintainability. They say that even thought the significance of the results varied, the results supported many of the conclusions derived from the LADY-language evaluation project.

The emphasis in the LADY study was on what good structure concepts in a language affect maintainability.

2.12 Functional Decomposition versus Object-Oriented

The software building blocks (entities) and the relationships between them are assembled to form the basic architecture of the application. Different design methods produce different software building blocks and assemble them in different ways. The two design methods that are of interest are that of functional decomposition and of object-oriented design [Korson and McGregor, 1990]. Of particular interest in this comparison is the structure each produces and how effective is it towards maintenance. Using a natural language analogy to compare these two methods :

> If procedures and functions are verbs and pieces of data are nouns, a procedure-oriented program is organized around verbs while an object-oriented program is organised around nouns.

In languages that support the module construct, a complete separate structure can be developed specifying the visibility of the different modules. This structure need not be a hierarchical tree structure, but a graph.

2.12.1 Functional Decomposition

(1) The fundamental buildings blocks are procedures and functions which perform some task and/or transformation.
(2) These building blocks are assembled using a single hierarchy. This hierarchy specifies how the processes relate to one another. In other words, the hierarchy forms an implicit scheduler with a single thread of control.
(3) This hierarchy of processes is static and hard-coded.
(4) A tree-type structure is formed with the highest module being the main program.
(5) By scanning this hierarchy (structure chart) from left to right, a rough indication of how the program executes and what processes are involved can be obtained.
(6) The system is decomposed by identifying the processes in the problem domain.
(7) It is very difficult to define a system as a set of communicating tasks or a set of independent program units having limited visibility to one another.
(8) There is only procedural abstraction. Data structures are not encapsulated.
(9) The only way to deal with exception conditions is to pass a control flag back to the calling module(s).
(10) The design works best for one-off systems that do not have to be modified much.

2.12.2 Object-Oriented Decomposition

(1) The fundamental building blocks are classes and objects (which directly model some physical object).

(2) These building blocks are assembled using two different hierarchies:
- *Kind of* hierarchy - Classes are structured using inheritance. This hierarchy specifies how classes are related to one another. It is an example form of the cognitive skill of classification which organizes knowledge.
- *Part of* hierarchy - This specifies how objects use other objects in order to exhibit certain behaviour.

(3) The *kind of* hierarchy is static, but the *part of* hierarchy can change dynamically during the life of the program. Objects can be created and destroyed continuously, each using services of objects who themselves have limited lifespans.

(4) Both of these hierarchies need not be strict trees, but can be more like graphs.
- *Kind of* hierarchy with multiple inheritance.
- *Part of* hierarchy with independent, autonomous objects.

(5) Because there is not a hierarchical structuring of processes, many processes can executing simulataneously in autonomous objects.

(6) Many classes can be related back directly to the original problem (real-world modelling). However, some classes are only found in the application (eg Window Managers).

(7) Inheritance coupled with polymorphism and dynamic binding minimizes the amount of existing code that must be changed when extending a system.

(8) There is both procedural and data abstraction. Data structures are encapsulated.

2.12.3 Adding or Modifying Functions

The research question of 1.1.2 can be answered using the comparison between functional decomposition and object-oriented. Object-oriented systems tend to be built in layers of abstraction. In each layer there is a collection of objects, called a subsystem, with restricted visibility to other layers. Booch notes that the components which make a subsystem tend to be structurally flat instead of strictly hierarchical and deeply nested [Booch, 1986].

Adding a function is merely adding a method to an object or class. All methods of a particular object/class are on the same level of abstraction. A new method is now available to other objects/classes that use this d object/class. Adding new methods does not alter existing methods, and the additional co serted within the class definition (declaration) of that object and is thus localized to that class. Modifying the implementation of a function means modifying only the code within the class definition to which the function belongs. The logical properties of the object in which the function is a method are unaltered.

This is not the case with functional decomposition methods. Adding or changing a function usually results in the adding or changing of a transformation or mapping of some input data to some output data. Other processes which are expecting this output data to be in a particular format will not behave as expected. Thus adding and modifying functions in functional decomposition methods impact on other unrelated processes. Object-oriented methods avoid this.

2.13 Obstacles not Addressed by Current Methods

2.13.1 Introduction of Composition Filters

Aksit and Bergmans have identified a number of important obstacles that are not addressed by current methods of object-oriented design [Aksit and Bergmans, 1992].

- *Identification of Problem-Domain Structures* : Many problems do not have any clear structure. Often, underlying theories of large systems are not clearly understood.
- *Dealing with Excessive Domain Objects* : Integrating domain knowledge ... ecifications can create an excessive number of objects, although only a few of the ... may be relevant to the problem being analyzed.
- *Early Decomposition* : Subsystems can be identified more accurately if structural relations and object interactions have been determined. This is because subsystem boundaries are largely determined by inter-object relations and interactions. The problem that arises is that if subsystems are not identified before starting with object identification, the project will probably become unmanageable. On the other hand, if subsystems are identified prior to object-identification, then the defined subsystem boundaries may not be optimal.
- *Commonality versus Partitioning* : In order to identify class hierarchies features of objects must be compared. Since subsystems partition the system, classes belonging to the same hierarchy can be scattered over different subsystems. This can make finding the correct inheritance hierarchy very difficult.
- *Subsystem Identification Using Object-Interactions* : Only intuitive techniques for subsystem identification are provided by most existing methods. This is not sufficient for large systems, since interactions can be too complex and subject to changes.
- *Multiple Views* : Not all view provided by an object are of interest to other objects that use its services. Therefore it is desirable to define views of an object that could differentiate between clients.
- *Coordinated Behaviour* : The message send model is considered as being too low-level, because it can only specify communications that involve two objects at a time and its semantics cannot be easily extended

Possible Solutions

To address the identified problems, they have introduced a new concept called composition filters. The basic object model is extended by introducing input and output composition filters that affect the received and sent messages, respectively. Output composition filters support the abstract communication types that abstract patterns of communication and large scale synchronization among objects.

2.13.2 Problems with Certain Types of Modelling

Harrison, et al, describes their experiences building an extensible software development environment $RPDE_{Pascal}$ using the object-oriented paradigm [Harrison, Shilling, and Sweeney, 1989]. They wanted to capture all the phases of the software development process for their environment. In their paper, they partially support the claim that object-oriented systems are easily extensible.

"A software system is easily extensible if the system can be modified to extend its functionality without needing to make major modifications to the existing implementation....Our experience has taught us that object instances provide a natural way to model program constructs, and to capture complex relationships between different aspects of the software system being constructed."

However, they mention some extensions which were difficult to incorporate into the system.

"A natural approach is to refine an object type so that each phase is represented as a distinct interface, the distinctions between interfaces are retained, and the instance variables of an object are encapsulated according to the interfaces....We would like the object to retain one identity in all the phases, but in each phase to play a different role that demonstrates different behaviour."

They mention that the phases in which an object may participate is a result of the software development process itself and is not known in advance.

"The conventional object-oriented paradigm does not provide the support needed to naturally model all the phases of the software development process. The conventional OOP provides one interface, one set of instance variables, and all the instance variables are visible to all the methods of the interface".

2.14 Restructuring

Arnold provides a tutorial in software restructuring and defines it as [Arnold, 1989] :

> "Software restructuring is the modification of software to make the software easier to understand and to change, or less susceptible to error when future changes are made. *Software* includes external and internal documentation concerning source code, as well as the source code itself....Software restructuring connotes taking existing information and refashioning it so it can be more easily understood."

Software restructuring's main goal is improving maintainability. Techniques include modifying:

- Code :
  - Pretty printing, coding style standardization.

  - Using reusable code and packages to replace poorly structured code.
  - Control flow including removal of goto statements, creating case statements.
- Documentation
- Programming Environments/Tools
- Software Engineers
- Management and Policies

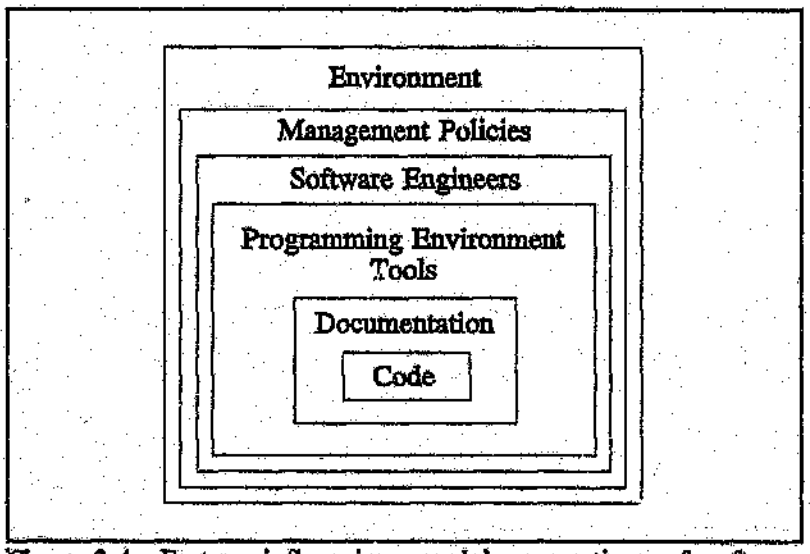


Figure 2-4 Factors influencing people's perceptions of software structure. The largest influence is the source code itself.

For software restructuring, *structure* is determined by two things : the software and the perceiver. Arnold also says that if the programmer is not trained to look for certain aspects of structure, his or her perceptions of structure will be influenced. Because perceptions differ, the notion of software structure is dynamic. At higher levels, software restructuring merges with software renewal.

Studies into restructuring are concerned with improving the structure of software to improve maintenance. However, the thrust is with unstructured COBOL software. Little attention is given to the restructuring of data.

2.14.1 Tools for Restructuring

Tools for restructuring COBOL code have been developed. One tool of this type is called STRUCTURED RETROFIT [Schneidewind, 1987].

> "It is claimed that structured retrofit has been able to restructure 60 percent of programs offered automatically, another 20 percent with some manual intervention, and 20 percent cannot be restructured cost-effectively."

2.14.2 A Comparison between Computer-Assisted and Manual Restructuring

Griswold and Notkin developed a restructuring tool with which they used to investigate how people can benefit from the use of such tools [Griswold and Notkin, 1992].

The Restructuring Tool

This tool defines a collection of composable source-level structural transformations that are guaranteed to preserve the meaning of programs to which they are applied. This set of transformations includes :

- Inlining an expression, which replaces the uses of a variable definition with a defining expression.
- Abstracting a binding (approximately the inverse of inlining), which moves an expression into a scope binding, assigns the result of the expression to the new binding variable, and puts a reference to the variable in the old location of the expression.
- Removing an embedded expression from a function definition and abstracting it as a parameter of the function, so that the removed expression must be passed as an argument to calls of the function.
- Extracting a function, which turns a sequence of expressions into a function, and replaces the extracted statements with a call to the function.
- Replacing repeated sequences with a call to the same function.
- Data-oriented manipulations, such as combining a group of variables into a record or changing a single variable into a list of variables, and making the necessary updates to the uses of the modified structures.

The tool does not automatically determine what transformations to apply. The engineer makes the subjective decisions about restructuring and then uses the tool to apply these changes. The tool restructures Scheme programs.

The Experiment

An experiment using this tool was conducted to compare how people using standard editing tools restructure a program against that which the tool supports. Six subjects were given a matrix multiplicative program, a description of the four modifications. Each subject's times were taken for editing, thinking, and debugging.

Conclusions

A number of conclusions were drawn :

- Subjects used a mixture of the copy/paste paradigm (where a copy of the original source is made, moved and modified) and the cut/paste paradigm (where the original source is moved in its entirety.)
- People make mistakes, even with small programs (fits into a single window). One cost of making errors during restructuring is that it increases the time to do the restructuring.
- Manual restructuring is haphazard compared to the computer-assisted process. When making

changes in one part of the source, it is tempting to make other nearby changes to save time scrolling back and forth in the program, even if they are not semantically linked to the first set of changes.

The tool simplifies the restructuring process from *change definition and then change all uses* to *change definition*. It allows the engineer to focus more clearly on the difficult, subjective aspects of modifying software.

2.15 Re-engineering

2.15.1 Re-engineering using an Object-Oriented Architecture

Re-engineering is the redesign of only part of the system using a different software technique. It therefore fits in between renewal and maintenance. There are three different forms of re-engineering:

- A complete change of implementation technique with no change in functionality.
- A partial change of implementation technique with no change in functionality.
- A partial change in functionality.

Re-engineering = Reverse engineering + System Change + Forward engineering

In a system which has part of it implemented in a new design technique, a communication interface between the new and old parts needs to be developed. A question that needs to be answered is : "Where should the border between the old and the new system be?". Often, well delimited system parts are candidates for re-engineering.

The article by Jacobson and Lindström describes a practical method for re-engineering based on object-oriented modelling [Jacobson and Lindström, 1991]. The model is represented by a directed graph which is constructed from information extracted from the existing system. Three case studies were presented using ObjectOry (an industrial software development process) :

- Military system for handling spare parts.
- Telecommunications system.
- Traffic control system.

Unfortunately, the advantages and/or disadvantages using this method of re-engineering were not discussed.

2.16 Summary

This chapter has introduced the concept of software structure and has highlighted key aspects which are involved in maintainable and reusable software. The characteristics of complex systems was mentioned. The Object and Class modelling of software produces software entities which resemble real world entities.

Frameworks are reusable designs which are constructed using object oriented methods. Several existing frameworks were examined. Extension hierarchies which are another form of software construction were also discussed. Object oriented and functional decomposition methods were compared with respect to software structure.

The next chapter presents an approach to decouple the User Interface from the rest of the application program.

CHAPTER 3

DECOUPLING THE USER INTERFACE

3.1 Introduction

The construction of the User Interface has a large influence on the rest of the program structure. In programs that were developed using the top-down functional decomposition, the User Interface often does the major scheduling of the program. The User Interface needs to know how the rest of the program is constructed to be able to do this scheduling. It is therefore tightly coupled to the rest of the program. Also, this scheduling is *hard-coded* into the hierarchical program structure. This makes it difficult to change the flow of control in a program, especially if concurrent processing is desired.

This chapter expands on the points raised in section 1.1.3 and 1.1.4. The approach presented here uses the object-oriented approach, but not the inheritance and polymorphism features. It is best used with languages like Ada which do not have inheritance and polymorhism features. It can therefore be considered an example of object-based programming with abstract data types.

3.2 The User Interface Embedded in the Application Processes

When the User Interface *(UI)* is embedded in the application processes, a process that needs input or output simply calls the actual input/output device itself. The code for the *UI* is interspersed amongst the different application processes. There is no distinct scheduler, but an implicit scheduler is formed by the structure of the program itself (if the functional decomposition method was used). This leads to duplication of *UI* code, which often results in inconsistent appearance and behaviour of the *UI*. Any change in the *UI* could be extremely difficult as it is scattered amongst the program hierarchy. Adding or modifying processes can be difficult because of the lack of a centralised *UI*.

3.3 Using the Object Oriented Approach

Object-oriented concepts include the concepts of data encapsulation, information hiding and localization. Localization is important in that all data structures and code dealing with the User Interface can be found in one or a few classes (objects). This reduces interaction between unrelated parts of the program. Modifications to the User Interface will only affect a limited number of modules and so reduce inconsistent behaviour and appearance. consistent.

The User Interface is divided into two main sections :

- *Low-Level Objects* : These objects deal with input and output requested by application objects.
- *High-Level Objects* : These objects deal with mechanisms to get application processing options (usually menus) and often perform a scheduling function.

3.4 Decoupling Low-Level User Interface Features

Decoupling the low-level facets of the User Interface is relatively easily. Instead of sending output directly to the output device, or getting input from an input device, a request is made to the appropriate User Interface method. This allows the low-level implementation of the User Interface to be easily modified. A consistent appearance and behaviour can be maintained because one set of methods do all the low-level activities.

3.5 Module Coded Menu Structure

With this approach, each menu is coded in a module. For a large number of menus, there would be a large number of modules. Code for displaying and handling input to the menu would be duplicated in each module. Since the menus are scattered amongst many modules, the behaviour and appearance of the menu could be different from one menu to another. This can cause confusion to the user and the maintenance programmer. Modifying menus requires effort because an option that is moved from one menu to another will also require the code (calling the process related to that option) to be moved to the new module.

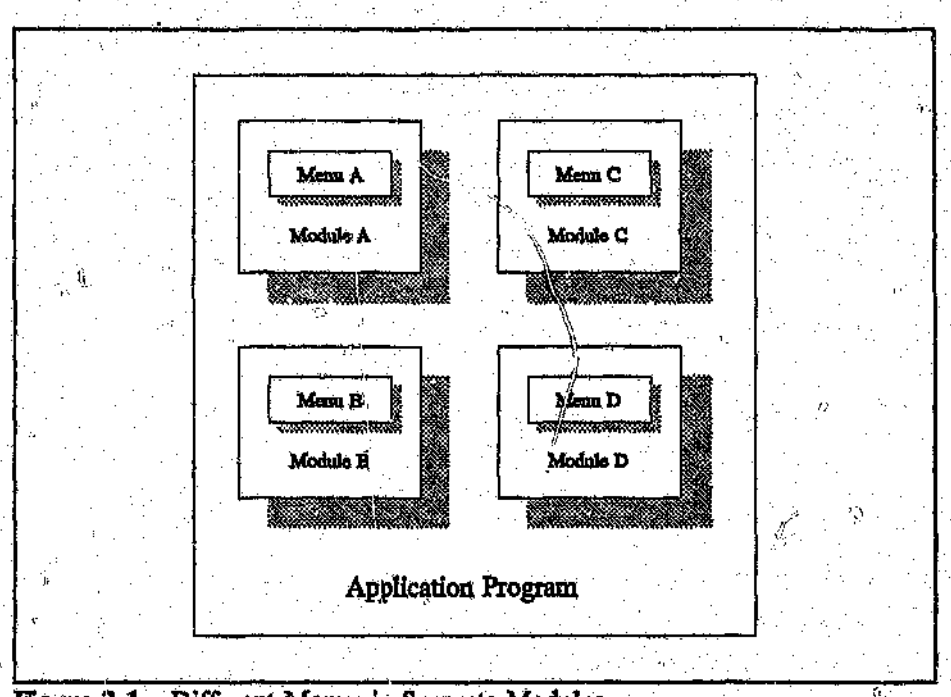


Figure 3-1 Different Menus in Separate Modules

3.6 A Menu Data Structure

The program structure of the menus reflects the relationship between the different menus. This menu hierarchy is realised in the actual code which produces a menu structure and menu behaviour that is rigid and which cannot be easily modified. To solve the problem of modifiability, the menu structure is realised as a data structure. This allows the menu's structure and behaviour to be modified by simply changing a value of a component in the menu data structure.

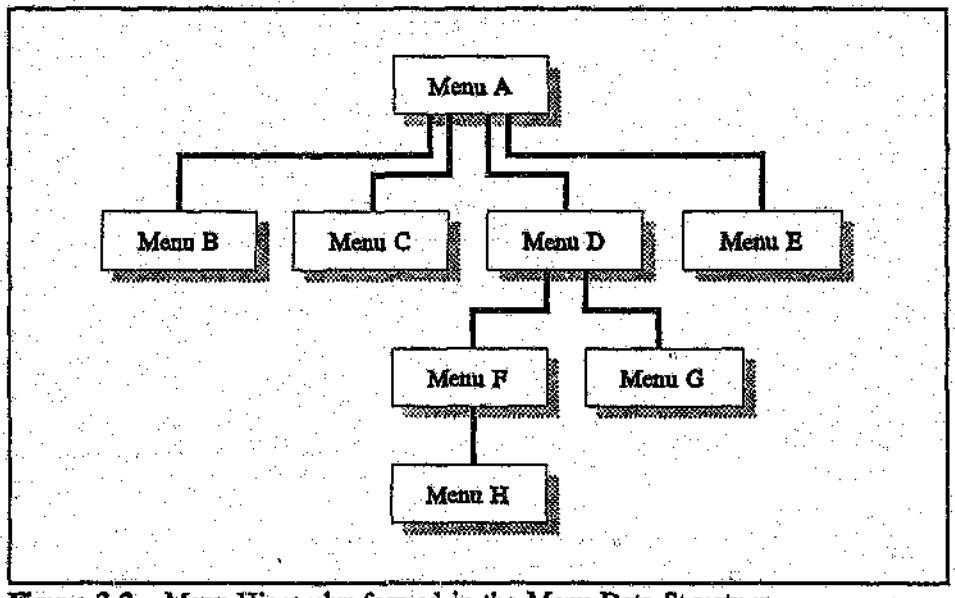


Figure 3-2 Menu Hierarchy formed in the Menu Data Structure

One procedure can be used for drawing the menus, and another for handling input to the menus. These procedures extract information from the menu data structure which determines the appearance and behaviour of the current menu. By changing the values in the menu's data structure, the menu's appearance and behaviour can be modified. However, the procedures for drawing the menu, and handling the input to the menu are unaltered. In other words, the different menus are differentiated by different values in their data structures, not different methods (program code) for each menu. This ensures consistency in menu appearance and behaviour.

For a large menu data structure with many menus, it would be possible to move through many menus, even back to the original menu, without leaving the procedure that deals with the menu input. When an option is chosen, the procedure is terminated and the required process is called.

Also, a change in program status caused by the execution of a process can be reflected in all the menus directly. For example, when Elector is started, only a few menu options are available. However, after recovering a circuit from the disk, or inserting a component, many more options become available. This is reflected in the menus by displaying the options that are now selectable in a different colour to those that are not selectable.

3.7 Decoupled Menu Structure and Scheduler

Very often, the program is controlled directly by the User Interface. For this to occur, the User Interface must have some knowledge of the application objects and is therefore tightly coupled to them. Any change in the application will force a change in the User Interface.

To completely decouple the User Interface from the application objects, the scheduling properties of the User Interface need to be extracted and made into a distinct independent object. The *UI* would then only be concerned with getting a menu option from the user and passing it back to the scheduler. The scheduler is thus considered to be the highest level object of the program.

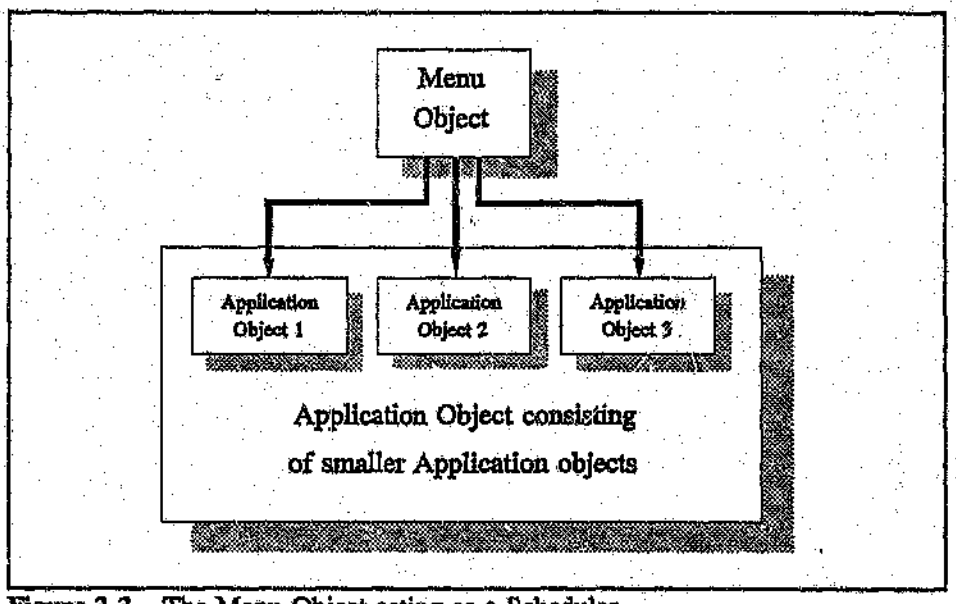


Figure 3-3 The Menu Object acting as a Scheduler

There is a corresponding OPERATION code for every menu option. The OPERATION code is stored in the same record data structure as the other information needed by the menu object to display and process a menu. When an option is chosen on the menu, the procedure (that gets a menu option from the user) will return the corresponding OPERATION code back to the scheduler. The scheduler will then decide if the desired process should be invoked. If there is no OPERATION code for a menu option, the scheduler won't be able to do anything.

To move a menu option from one menu to another only requires changing the initialisation values of the menu data structure. The scheduler object does not need to be modified. To add or delete an option is almost as simple. The initialisation values of the menu data structure are changed to include the new option, or the option to be deleted is removed. The scheduler object is modified slightly (usually requires only adding or deleting an option in a case statement), so that it will recognize the new OPERATION code (corresponding to the new menu option). This is not the situation for the previously discussed cases.

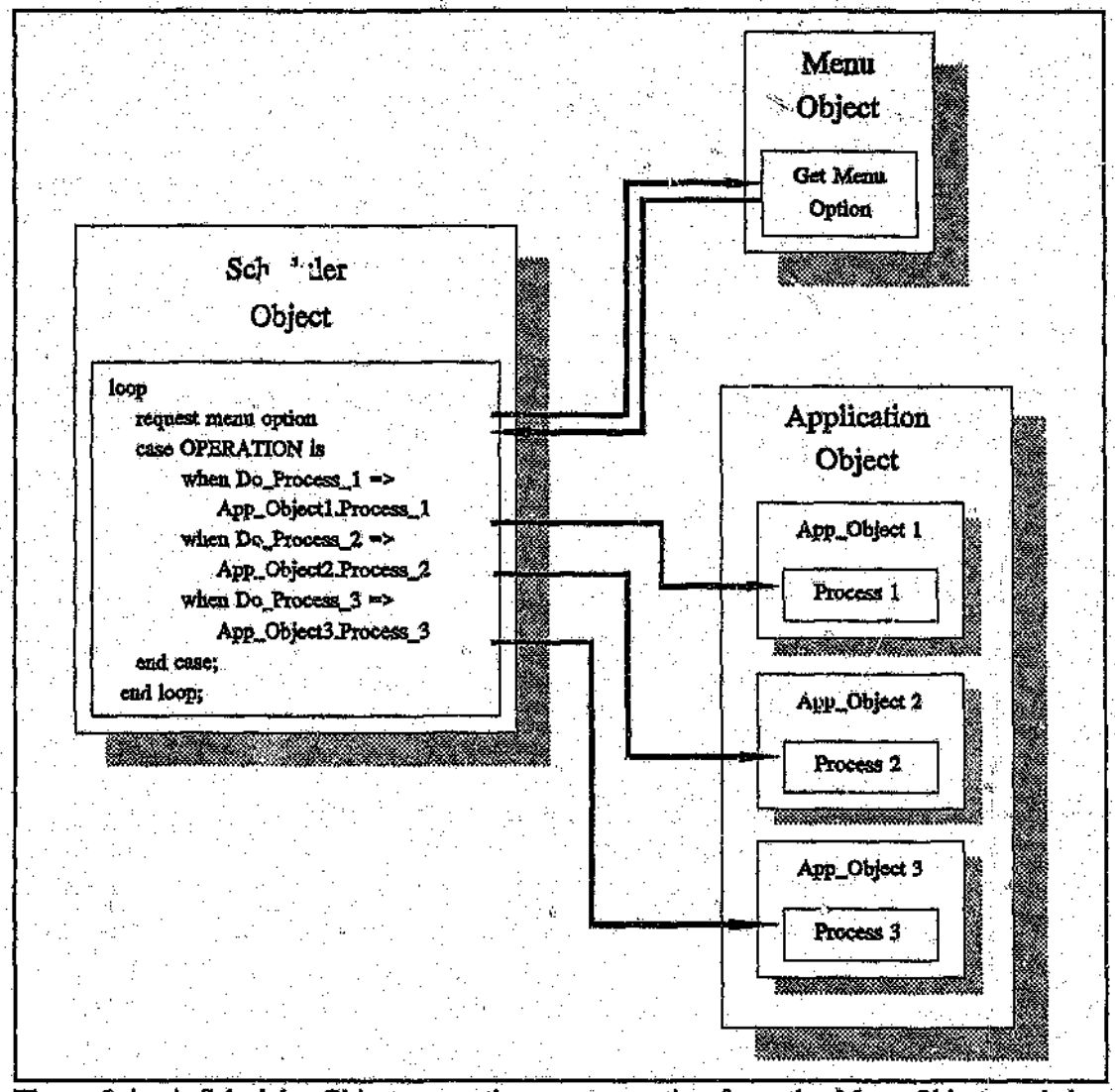


Figure 3-4 A Scheduler Object requesting a menu option from the Menu Object, and then passing control to the appropriate Application Object.

For the deleted menu options, it is good practice to modify the scheduler so that it will not recognise the deleted OPERATION code. This is not absolutely necessary because the OPERATION code for the deleted menu option can never be returned by the menu procedure as the option is not available for the user to choose. However, it avoids confusion for the maintenance programmer who will look for the corresponding option in the menu structure, but not find it.

Figure 3-4 shows that the scheduler object calls the menu object to get a menu option from the user. Once the user has chosen an option, the OPERATION code corresponding to that option is returned to the scheduler object. The scheduler object then determines which process must then be invoked. Control passes from the scheduler to the object in which the process is found. The process is then invoked. When the process is finished, control is passed back to the scheduler.

Note the difference in the operation of the menu in this case as compared to the other cases. In the second case (Figure 3-3), the menu object is the highest-level object and it exercises control over the program. Here, the menu object is asked for an option by the scheduler. This option can be found on any menu, but only one option can be returned. The scheduler object exercises control over the program, and the menu is subordinate to it.

3.8 Schedulers

A distinct scheduler needs to be introduced when flattening the program structure of a functional decomposed program, or when there are a number of totally independent objects in an object-oriented program. The scheduling object initiates a process in another object by passing a message to that object. When that object has finished execution of the process, a message is send back to the scheduling object. In a sequential implementation, where there is a single thread of control, the passing of a message to an object is the same as the passing of control to that object. In a concurrent implementation, where there are many threads of control, the message can be some form of notification to an object to start the execution of a particular process.

There can be more than one scheduler in a program, depending on the number of independent objects and the number of abstract layers. If an object or an abstract layer has several complex processes which need to be synchronized and/or coordinated, an embedded scheduler object could be created. This scheduler would make the scheduling of different processes in independent objects explicit. Scheduling algorithms and data structures could be encapsulated in a single object. For example, a service provided by one object might in turn require the services from many other independent objects. An embedded scheduler object within this object could then synchronize the requests for different services from other objects.

3.9 Summary

An approach was presented that decouples the User Interface from the application objects. The scheduling properties of the User Interface are extracted and made into an independent scheduler object. The scheduler objects asks the menu object to get a menu option from the user. Once the user has chosen an option, the corresponding code is returned to the sheduler which will envoke the chosen operation. The menu object encapsulates all menus, with the menu hierarchy realized as a tree data structure.

The next chapter describes the major concepts involved in the construction of the User Interface.

CHAPTER 4

USER INTERFACE CONSTRUCTION

4.1 Introduction

The User Interface that was developed for Elector consists of many objects. Each of these objects can be constructed from yet smaller objects. Input is passed to these objects instead of having them get it themselves. Each object is responsible for two things :

- It's appearance.
- Input that is routed to it.

This is in contrast to many programs where the User Interface is a flat display area that can be considered a single object.

The previous chapter focused on the structure of the connection between the User Interface and the application objects. These issues are mainly of concern to the maintenance programmer. This chapter focuses on the objects of the User Interface itself and use the guidelines mentioned in section 1.1.5.

4.2 The Appearance of Screen Objects

The appearance of an object is determined by it's *Draw* method. The object must be able to draw itself at anytime. This is important because the object will need to update it's appearance when it's internal state changes.

Objects are not in a strict display hierarchy, but all objects need the window object to appear on the screen. The window object can appear anywhere on the screen, and objects can appear anywhere in the window object. Objects can be drawn on top of one another (in the same place), but this will only confuse the user. For such a case, the last object to draw itself will appear on top of all the others. However, when the input record is routed from one object to another, the input will be handled by the first object that drew itself there, not the last.

The construction of useful Subscreen (dialogue box) objects will require the use of many screen objects. There always has to be a Window object, and usually 2 to 4 Button objects. Depending on the purpose of the Subscreen, there will also be a Pick_List object, Form object or an Input_Line object.

4.3 The Routing of Input

Objects in the User Interface are passive when it comes to getting input. They wait until input is sent to them. If any of the input concerns them, they will act upon it.

Input is gathered by a special Inputter object that checks the mouse (if one is present) and the keyboard. An input record is filled with the present status of the input devices. This record is then routed to the different screen objects. The first object to receive the record is the Window object, which checks if it should switch to *moving* mode. The input record is then routed to the next object, usually a button or a set of buttons. From there, the input record could be routed to Input_Line objects, Pick_List objects and so on. After an object handles input concerning it, it will set a field in the input record to a value which other objects will recognise as *the input was directed to some other object and it has been taken care of.* Other objects that see this value will ignore the input record.

Objects can also communicate with one another by means of the input record. A field of the input record is set to a predefined value which other objects examine. A situation where this occurs is when the Window object has moved. It needs to tell the objects inside it that it has moved and give them the new global coordinates. Most objects are not affected by the move, but the Input_Line and Form objects are concerned because they use the hardware cursor which is set according to global coordinates, not local.

4.4 The Behaviour of Screen Objects

The behaviour of a screen object is determined by it's *Handle_Input* method. When the object is passed an input record, it is this method that checks to see if any inputs are directed to this object. If there are any inputs, the object responds with an appropriate action to the input. After this, the input record is marked to tell other objects that the input has been handled.

4.5 The Construction Of Screen Objects

The package construct and private types are used extensively in the construction of screen objects and Subscreens. The Screen_Objects package is a container which holds the smaller objects. Within the Screen_Objects package are a number of packages for Buttons, Boolean_Buttons, Input_Line, Pick_List, Form and Window objects.

Within each of these smaller packages is a private type, subprograms and possibly exceptions. The private type defines the structure of the data. The internal structure of such a type cannot be accessed or modified outside the package. Data encapsulation is thus enforced. The subprograms

```
package Screen_Objects is

   ------------------------------------------------------------

   package Forms is

      type FORM_TYPE is private;

      INTEGER_ERROR          : exception;
      INTEGER_MIN_ERROR      : exception;
      INTEGER_MAX_ERROR      : exception;
      INTEGER_EQUAL_ERROR    : exception;
      REAL_ERROR             : exception;
      REAL_MIN_ERROR         : exception;
      REAL_MAX_ERROR         : exception;

      ERROR_FIELD            : INTEGER_TYPE;

      procedure Create (FORM : in out FORM_TYPE;
         FORM_DATA      : in FIELDS_TYPE;
         OFFSET_ROW     : in ROW_TYPE;
         OFFSET_COLUMN  : in COLUMN_TYPE;
         TEXT_COLUMN    : in COLUMN_TYPE;
         VALUE_COLUMN   : in COLUMN_TYPE;
         ACTIVE_COLOUR  : in SCREEN_COLOUR_TYPE;
         TEXT_COLOUR    : in SCREEN_COLOUR_TYPE;
         FIELD_COLOUR   : in SCREEN_COLOUR_TYPE);

      procedure Draw (FORM : in FORM_TYPE;
         FORM_DATA : in FIELDS_TYPE);

      procedure Handle_Input (FORM : in out FORM_TYPE;
         FORM_DATA : in out FIELDS_TYPE;
         INPUTS    : in out INPUT_TYPE);

   private

      type FORM_TYPE is
         record
            ACTIVE_FIELD    : POSITIVE;
            CURSOR_ROW      : ROW_TYPE;
            CURSOR_COLUMN   : COLUMN_TYPE;
            TEXT_COLUMN     : COLUMN_TYPE;
            VALUE_COLUMN    : COLUMN_TYPE;
            ACTIVE          : BOOLEAN;
            INSERTING       : BOOLEAN;
            ACTIVE_COLOUR   : SCREEN_COLOUR_TYPE;
            FIELD_COLOUR    : SCREEN_COLOUR_TYPE;
            TEXT_COLOUR     : SCREEN_COLOUR_TYPE;
         end record;

   end Forms;

   ------------------------------------------------------------

end Screen_Objects;
```

Figure 4-1 Ada Code for Forms Screen Object

are the methods for that type, and only they can be used on the private type. The exceptions are defined for abnormal situations that could occur in instantions of the type. Together, these three parts in a package form an abstract data type.

4.6 Elector's User Interface

The user interface can be broken down into Subscreens and Menus.

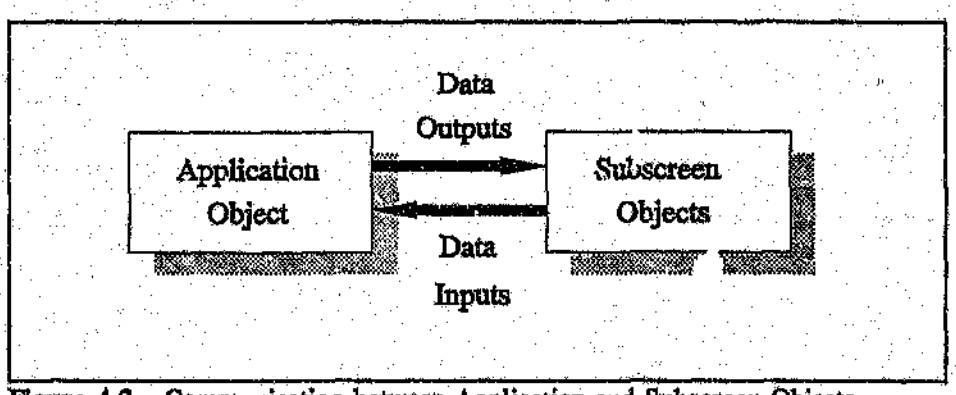


Figure 4-2 Communication between Application and Subscreen Objects

4.6.1 Subscreens (Dialogue Boxes)

The Subscreens decouple the User Interface from the application objects at a low-level. They get input from the user for the various application objects, as well as displaying output from the application objects to the user. The application objects do not have any knowledge of how the information is acquired or presented. These two types of objects are only related to each other in that they pass information to each other. Otherwise, their data structures, methods, etc are totally independent. These objects are separated both physically and logically. There are different Subscreen objects, each dealing with a different type of data.

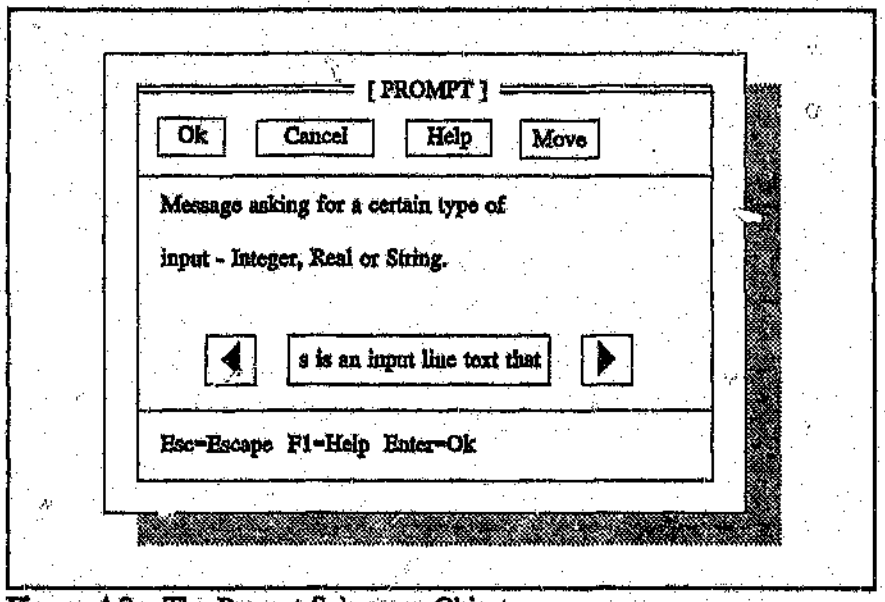


Figure 4-3 The Prompt Subscreen Object

Subscreen	Purpose
Errors	Displays a message that an error has occurred in an application object.
Prompts	Asks the user for some input and then get's the input. The input can be boolean, character, integer, real and string values. Lists and forms can also be displayed. For exclusive lists, only one option can be chosen. For inclusive lists, options can be selected or de-selected. For forms, related integer and real values can be entered.
Status	Displays a status message to the user about the state of a particular process or a related item.

Table 4-1 Types of Subscreens (Dialogue Boxes)

```
package Screens is
   ------------------------------------------------------------------
   package Prompts is

      procedure Add_To_List (FILE_NAME : in STRING);

      procedure Read (FORM : in out FIELDS_TYPE;
         LIST      : in out LIST_ARRAY_TYPE;
         EXCLUSIVE : in BOOLEAN := TRUE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (FORM : in out FIELDS_TYPE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (LIST : in out LIST_ARRAY_TYPE;
         EXCLUSIVE : in BOOLEAN := TRUE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (FILE_NAME : out STRING;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_FILES);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_BOOLEAN;
         ITEM      : out BOOLEAN);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_INTEGER;
         ITEM      : out INTEGER_TYPE);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_REAL;
         ITEM      : out REAL_TYPE);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO   : in  INTEGER_TYPE := HP_SUB_PROMPTS_STRING;
         ITEM        : out STRING;
         ITEM_LENGTH : out POSITIVE);

   end Prompts;
   ------------------------------------------------------------------
end Screens;
```

Figure 4-4 Ada Code for Subscreen Objects

4.6.2 Menus

The high-level part of the User Interface, the menus, are completely encapsulated in the Menus package. To facilitate communication between the menus and the scheduler, a set of operation codes is defined in the Menus package specification. These operation codes correspond to actual menu options. The menu structure is build according to the last case that was discussed in Chapter 3 - separate Scheduler and Menus objects. The menu structure is a dynamic data structure that is initialised only once at the start of the program. Only two subprograms (methods) are available. The *Get_Menu_Operation* gets the option that was chosen on one of the menus. The *Change_Menu_Option* allows the Scheduler object to set a menu option to be selectable or non-selectable.

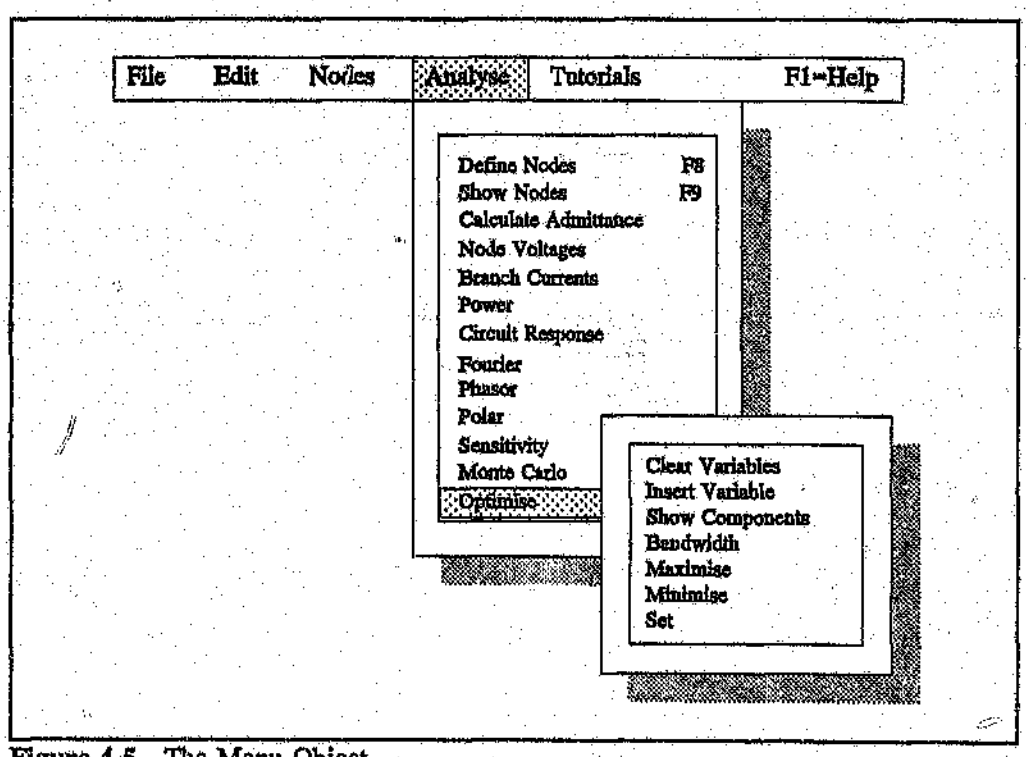


Figure 4-5 The Menu Object

Menus can appear spread out with options next to one another, or stacked with options under one another. Options can be selected with the highlighting bar, shortcut keys, hotkeys and the mouse.

4.7 Dos_Environment Package

4.7.1 Halo Graphics

Initially, it was anticipated that the Halo Professional Graphics Library would be used for the

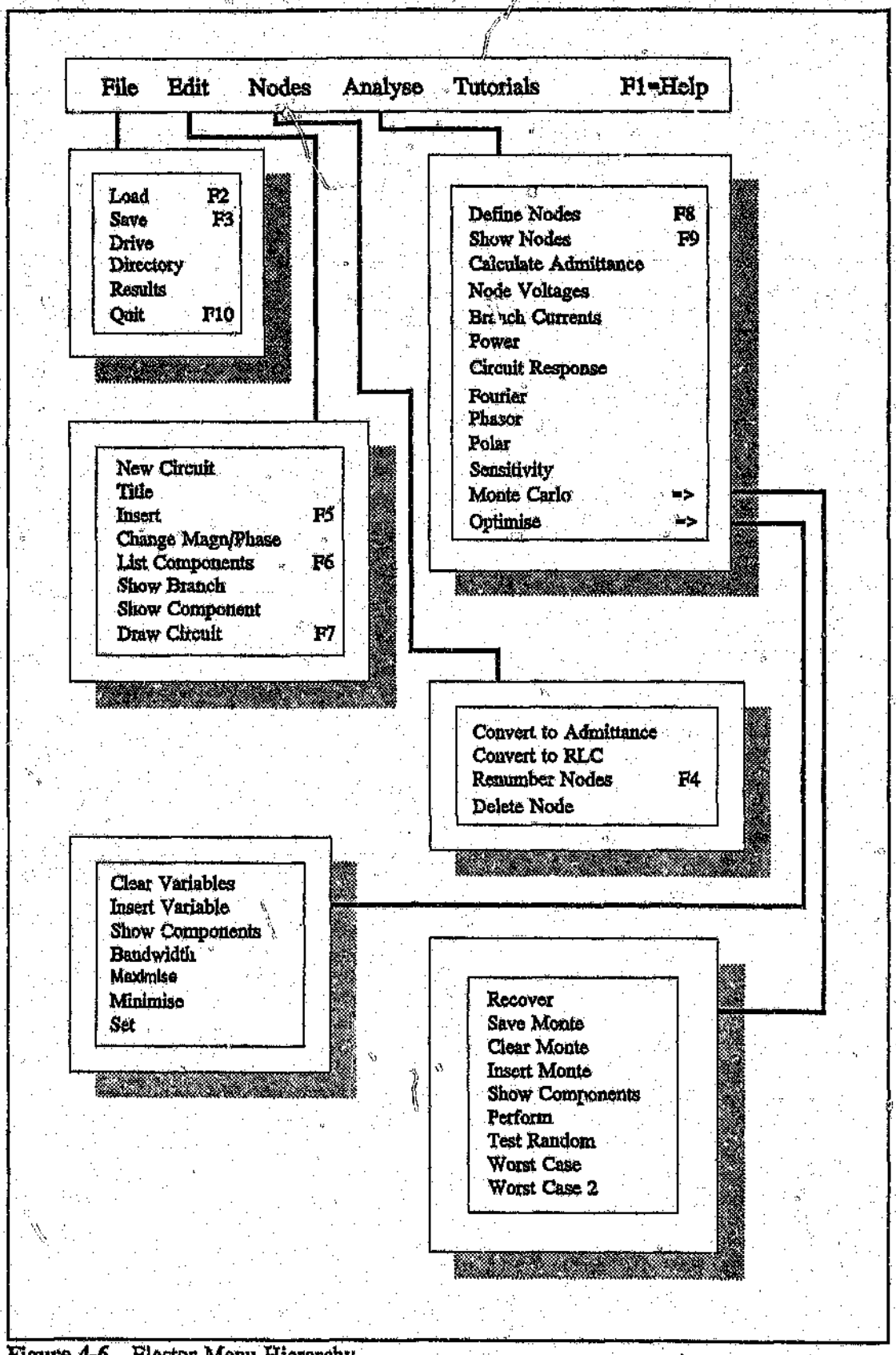


Figure 4-6 Elector Menu Hierarchy

```
package Menus is
   ------------------------------------------------------------------------

   NO_SUCH_OPERATION_IN_MENUS : exception;

   OP_ADMITTANCE          : constant := 20;
   OP_BRANCH_CURRENTS     : constant := 21;
   -- Any number of menu options can be defined here.

   procedure Get_Menu_Operation (OPERATION : out OPERATION_TYPE);

   procedure Change_Menu_Option (OPERATION : in OPERATION_TYPE;
      STATE : in BOOLEAN);

   ------------------------------------------------------------------------
end Menus;
```

Figure 4-7 Ada Source for Menus Object

graphics. Halo consists of a run-time graphics code library that is loaded separately from the program that user it. When this package was installed, it was found that two errors were present.

- In the language bindings, a type declaration for floating point numbers used attributes that deal with scalar values only. This caused compilation errors to occur. This is relatively easy to correct.
- Halo uses 32 bit floating point values. The Meridian Ada compilers use only 64 bit floating point values. This means that the two software packages are incompatible at this low level.

4.7.2 Assembly Language

Since Halo Graphics Library could not be u some other approach had to be taken. The Ada subprograms of Meridian were limited, slow . . bulky. An indication of the slow speed is 5 minutes to draw a particular circuit schematic using the Meridian subprograms. The same circuit takes 3 seconds using Assembly Language.

The Menus, Subscreen and Help objects pop-up onto the screen, and then disappear. To implement these objects, a system of buffer is used. There are two different methods of implementation. The method chosen uses a large buffer for the Main screen, and a smaller buffer for the Menu, Subscreen and Help objects. The transfer of large amounts of data from the buffers to screen needs to be quick.

The computer and operating system features were accessed using the BIOS system because :

- *Portability* at the assembly language level. All these functions are build into the computer.
- *Understandability* : These functions are well documented in many books and manuals.
- *Reliability* : The manufacturer will have tested and made sure that these functions work for their machines.

For information on the low-level assembly language implementation which forms the foundation on which the Ada implementation is build, see the Dos Environment chapter in the Technical Reference Manual.

4.7.3 Tutorial and Help Screens

Elector is meant to be a teaching aid to understand linear electrical circuits. Thus special tutorial screens were developed which give a summary on certain parts of electrical engineering theory as well as the design of circuits on Elector. A context-sensitive help system was developed giving more information on certain aspects of the program.

4.7.4 Additional Information

For more information on the appearance, behaviour or structure of Elector's User Interface, see the following chapters in the Technical Reference Manual :

- Screen Objects
- Help Object
- Screens Object
- Menus Object

4.8 Summary

The main concepts in the construction of the User Interface for Elector was discussed. This included the appearance and behaviour of Screen Objects and the routing of input to these objects. The two main types of objects are the Subscreen (Dialogue Box) and Menu objects. The implementation of these objects, including the use of Assembly Language to access the computer resources, was discussed.

The next chapter describes the construction of the Scheduler and Application objects of Elector.

CHAPTER 5

ELECTOR CONSTRUCTION

5.1 Introduction

The previous chapter focussed on the construction of the User Interface of Elector. This chapter focuses on the remaining two parts of Elector - the Scheduler and Application objects. Part of the discussion will be a comparison between Elecsim (the original program) and Elector.

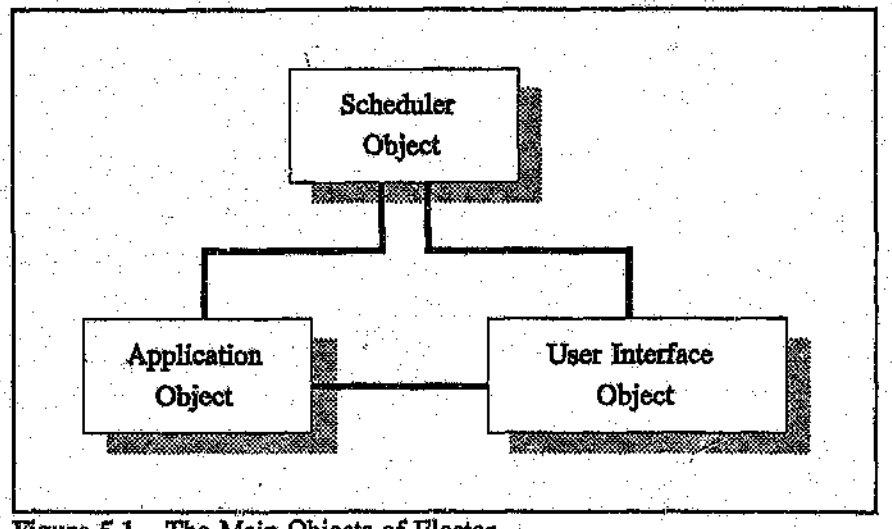


Figure 5-1 The Main Objects of Elector

5.2 Implementation Language

Elector is implemented in *Ada* using Meridian Ada version 4.1. Elecsim is implemented in *Pascal* using Turbo Pascal version 5. Many features available in Ada and not in Pascal were used. This includes packages and private types which make it possible to enforce abstract data types.

5.3 Netlist Abstract Data Type

The Netlists are the core application object of Elector and holds all information for the components in a circuit. The backbone of the Netlist is a dynamic single-linked list.

In Elecsim, the structure of the Netlist data type is visible throughout the application part of the program. This means that any procedure can change individual component values within the Netlists. Several instances of uncontrolled access and modification is found in the Monte-Carlo and Optimisation modules.

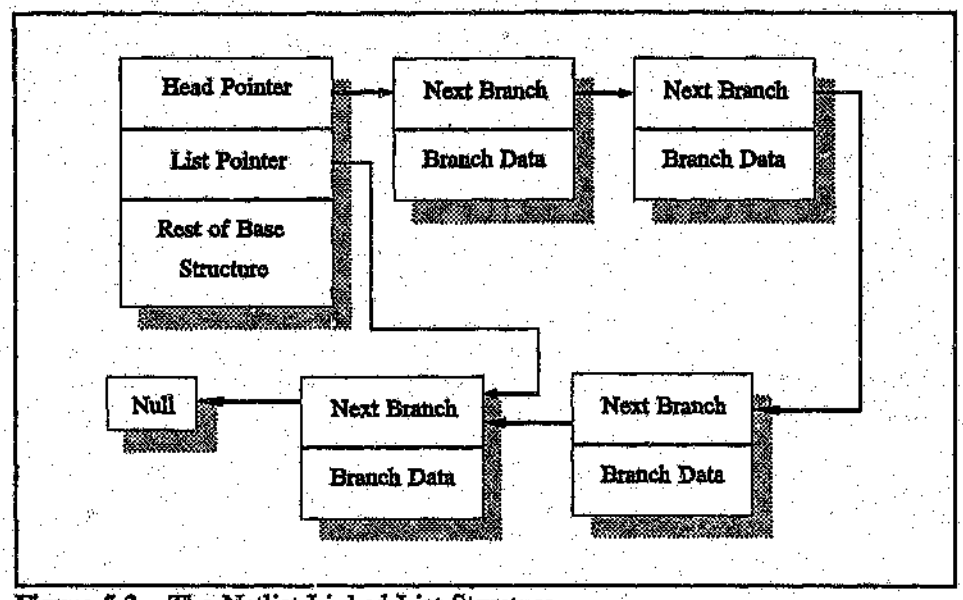


Figure 5-2 The Netlist Linked List Structure

To avoid this, as well as enforce the object oriented approach, the Netlists were declared as an *abstract data type*. All processes which Lad made use of the internal structure of the Netlists had to be modified. This included making additional methods in the Netlists package to provide additional services.

```
package Netlists is
   ------------------------------------------------------------------
   type NETLIST_TYPE is private;
   Declaration of visible types, constants and variables.
   ------------------------------------------------------------------
   package Screen_Supporter is
      Methods for use with low-level user interface.
   end Screen_Supporter;
   ------------------------------------------------------------------
   package Inner is
      Methods concerning NETLIST_TYPE only.
   end Inner;
   ------------------------------------------------------------------
   package Outer is
      Methods concerning NETLIST_TYPE and low-level user interface.
   end Outer;
   ------------------------------------------------------------------
   List of exceptions which could be raised.
private
   Declaration of NETLIST_TYPE internal structure which is hidden.
end Netlists;
```

Figure 5-3 Ada Code for Netlists Package

To actually implement the Netlists abstract data type, the package and the private type were used. The methods that could operate on this type were subprograms that were declared in the package specification. Each method has one parameter of the Netlists abstract data type. This is used to specify which instantiation of the Netlists abstract data type is to be operated on.

```
procedure Convert_Admittance_To_RLC (FREQUENCY : in REAL_TYPE;
            NETLIST_OBJECT : in out NETLIST_TYPE);
```

Figure 5-4 Ada Code for Method of an Object

5.4 Restructuring The Program

Many of the procedures in Elecsim were put into separate compilation modules just for the sake of putting them into separate modules. This led to the case where many modules that were fairly closely related, had only one procedure in them. In Elector, procedures were put into modules that more closely modelled the different parts of the simulator. This restructuring also allowed the large data structures to be hidden in the bodies of packages instead of being visible throughout the program.

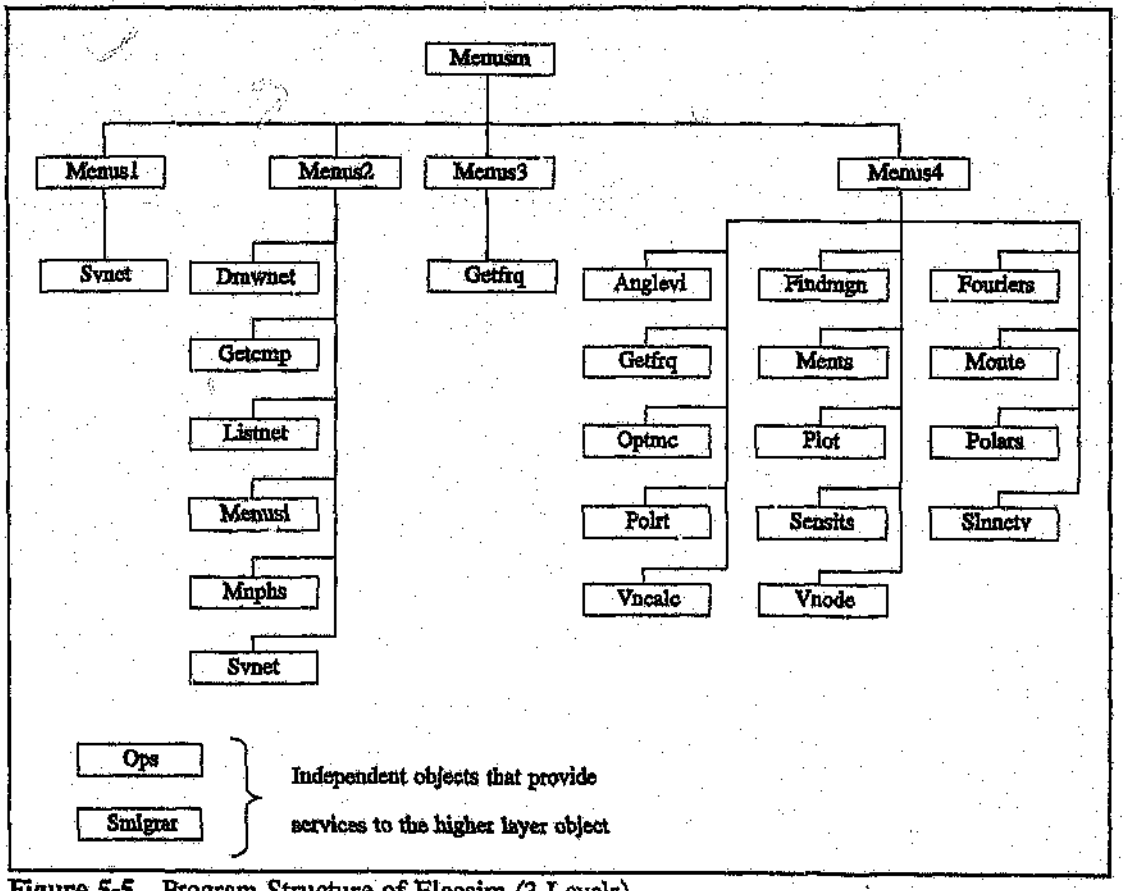


Figure 5-5 Program Structure of Elecsim (3 Levels)

Elector has a layered structure with several independent objects in each layer. There is limited visibility, with the upper layers able to use the services of lower layers, but not the way around. Only objects in layer one use each others services. Objects in all other layers use only the services of the layers below them.

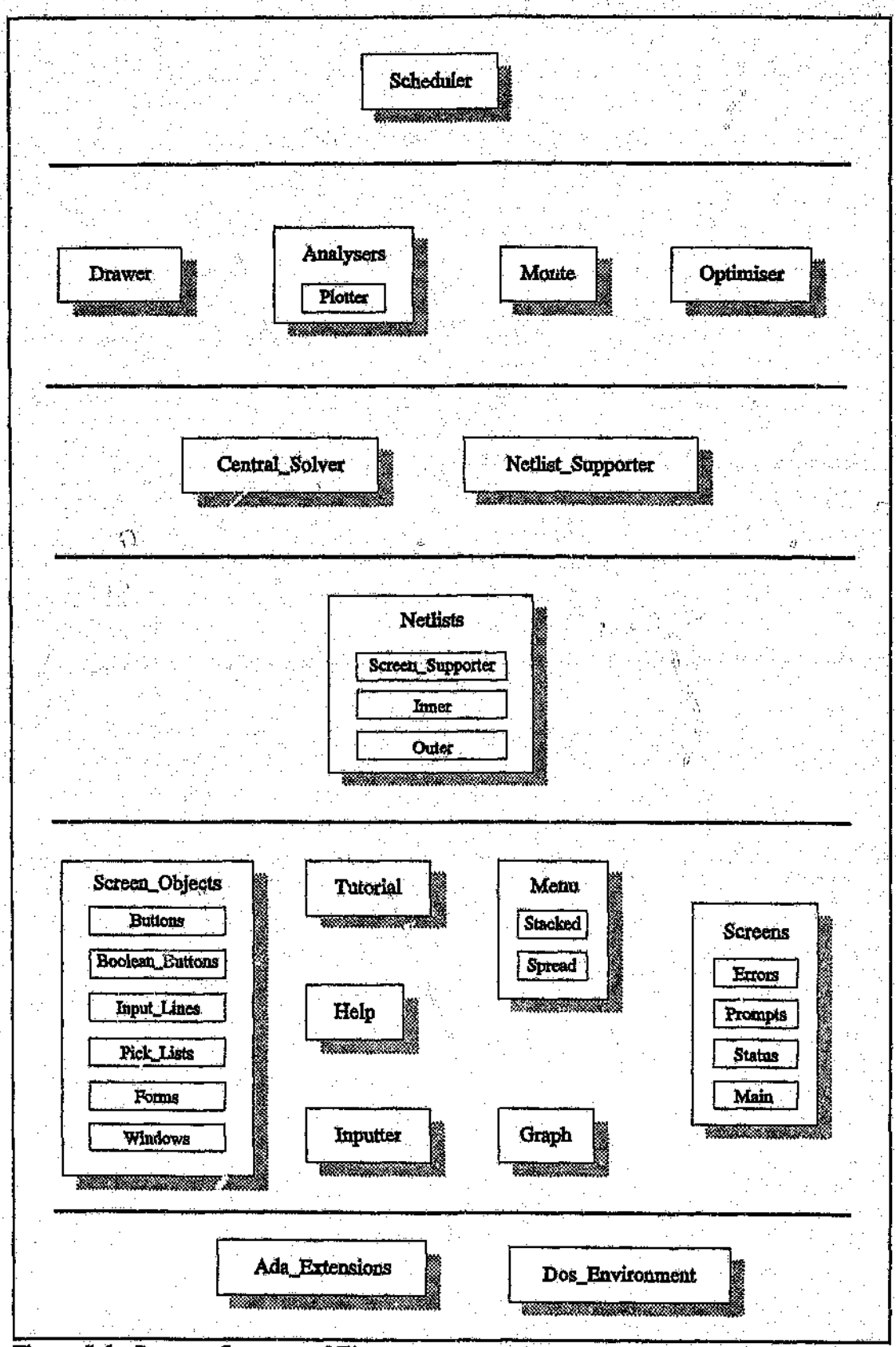


Figure 5-6 Program Structure of Elector

The first layer has two objects which provide low-level services.

- *Ada_Extensions* add string and mathematical functions to extend the Ada language.
- *Dos_Environment* has subprograms to access the computer hardware and operating system.

The second layer has seven objects which all deal with the User Interface.

- *Screen_Objects* contains the smallest objects from which the User Interface is constructed.
- *Screens* contains the Subscreen objects which request and present data to the user.
- *Inputter* gets all input from all input devices.
- *Menu* contains the different menus for Elector.
- *Graph* contains procedures to access the graphic screen.
- *Help* gives context-sensitive help for the User Interface.
- *Tutorial* gives brief text on how to use Elector.

The third layer has only one object - the *Netlists*. This holds all the data for the components of circuits.

The fourth layer has two objects which use the services of the *Netlists* object in order to provide services to other objects.

- *Central_Solver* is the main *solver* which determines all node voltages.
- *Netlist_Supporter* provides additional *Netlists* procedures which do not use the internal representation of the type.

The fifth layer has four objects which do all the analysis, simulation, optimising and drawing of the circuit.

- *Drawer* draws the circuit schematic on the graphic screen.
- *Analysers* calculate various circuit characteristics such as voltages, current.
- *Monte* performs Monte-Carlo analysis on the circuit.
- *Optimiser* performs optimisation on parts of the circuit.

The sixth layer is contains the *Scheduler* object which coordinates and schedules all the objects in Elector.

The following objects contain smaller objects :

- *Screen_Objects*
- *Screens*
- *Menus*
- *Netlists*
- *Analysers*

5.5 Scheduling

The Scheduling object is built according to the approach described in Chapter 3. It is built around a large case statement from which it calls processes that must execute. The scheduling object also contains variables which keep the main *state* of the program. It uses these variables to determine if a requested process is allowed to execute. For example, a request to find node voltages of an empty circuit should be ignored. Another example would be changing the layout of the circuit. If a node has been deleted or inserted, the topology of the circuit must first be checked before any analysis operations are undertaken. If the topology is not satisfactory, the user must be alerted. Any request for an analysis operation must be ignored until the topology is satisfactory.

The Scheduling object has an initializing subprogram because it has *state* variables which must be initialised. This procedure can also call any other initializing procedures for other objects which it schedules. The Scheduling object also has procedures which tell the Menus object (of the User Interface) which menu options can be selected and which cannot. There is also a main scheduling procedure that is called by a higher level object after the scheduler has been initialized (initializing procedure has been called). The higher level object can be considered to be the main program in most cases.

5.6 Floating Point Representation

In Elecsim, 4 bytes are used for floating point types. Meridian Ada supports only 8 byte floating point types. This forces a tradeoff. By using 8 bytes, there is greater accuracy, but data structures are twice the size. However, the 64K limit means that a single data structure that is > 32K in Elecsim cannot be implemented directly. Most of the large data structures have their sizes determined from the maximum number of nodes that are allowed in the circuit. For 100 nodes, the least number of bytes required would be :

$$100 * 100 * 4 \approx 40K \text{ (Elecsim)}$$

$$100 * 100 * 8 \approx 80K \text{ (Elector)}$$

The data structure in Elector is greater than the 64K limit. To still be able to implement this data structure, the maximum number of allowed nodes in a circuit must be reduced. The largest permissable number is 90 nodes which gives data structures of 64K.

A second tradeoff is the optimum maximum number of allowed nodes in a circuit. The larger the number, the larger the circuit that can be analyzed. However, it is also more likely that the program will run out of memory and not be able to implement these data structures at all. The best number appears to be 75, though this can be easily changed by changing the MAX_NODES

```
package body Scheduler is
   Methods for communicating with menus object about selectable options.
   Method for initializing scheduler object and other smaller objects.
   ------------------------------------------------------------------
   procedure Scheduling is
   begin
      loop
         Get an option from the menus object.
         begin
            begin
               -- Preprocessing section.
               if (the previous operation was a Mo[illegible]) and
                  (this one is not) then
                  Destroy the monte data structure.
                  Update menus object.
               end if;
               if (the circuit must be checked) and
                  (an analysis operation is chosen) then
                  Check the circuit topology.
               end if;
            exception
               when (some form of exception) =>
                  Display an appropriate error message.
            end;
            -- Actual scheduling section.
            begin
               -- Calls to appropriate object to execute process.
               case OPERATION is
                  when (a chosen operation) =>
                     Inform necessary object to perform chosen operation.
               end case;
            exception
               when (some form of exception) =>
                  Display an appropriate error message.
            end;
         exception
            when (some form of exception not handled) =>
               Display an appropriate error message.
         end;
         -- Postprocessing section.
         if (the circuit nodes must be checked) then
            Find the maximum number of circuit nodes.
            if (there are no nodes and there were) then
               Inform menus object about selectable options.
            elsif (there are circuit nodes) then
               Inform menus object about selectable options.
            end if;
         end if;
      end loop;
   end Scheduling;
-------------------------------------------------------------------
end Scheduler;
```

Figure 5-7 Ada Code for Scheduler Package

constant in the Netlist package declaration to the new maximum node value. With 75 nodes, 2850 different branches can be constructed. Memory will run out well before all the branches have been inserted. This is about the same size as the data structures in Elecsim.

[illegible] * 75 * 8 ≈ 45K

5.7 Summary

The construction of the Scheduler and Application objects of Elector has been described. The Scheduler object is the *highest* object in the program. The Application objects are build in layers, with restricted visibility between layers - objects at a higher level can use the services of those at a lower level. The most important application object is the Netlists. It holds all the circuit information. Floating-point representation and program restructuring were also discussed.

The next chapter focuses on the renewal exercise and the method used to renew Elecsim.

CHAPTER 6

SOFTWARE RENEWAL

6.1 Introduction

The need for software renewal was highlighted in Chapter 1.2. The study of software maintenance and renewal falls within the broader subject of software engineering. Any new program that is created from scratch or renewed should be done with proper adherence to the goals and principles of Software Engineering. Failure to do so will lead to software that will be very difficult to maintain, and, not of much greater use than the original program.

6.1.1 Software Engineering

Software engineering is concerned with the production of quality software that meets it's requirements, is within budget and is delivered on time [Schach, 1990]. Because of this, software engineering covers not only areas such as mathematics and computer science, but areas such as economics, management and psychology. Software production is not like traditional engineering. There are four general properties which serve as goals for software engineering [Booch, 1983]:

- Modifiability
- Efficiency
- Reliability
- Understandability

The following principles are applied to the design of software so that the goals of software engineering can be realised as closely as possible:

- Abstraction
- Information Hiding
- Modularity
- Localization
- Uniformity
- Completeness
- Confirmability

6.1.2 No Standard Components

Instead of assembling software from a library of existing components, it is usually custom build.

- This means that every component must be developed from scratch increasing software development costs and time.
- It is more difficult to understand the software because components cannot be recognized

immediately. Even components that are very similar to established library components must be closely inspected to determine where they differ from the library components.

When designing hardware (by comparison), catalogues of IC components are used that give information on each IC in terms of:

- A defined and validated function.
- A well defined interface.
- A standard set of integration guidelines.

There is a movement to build software libraries of reusable components. The object-oriented approach concept of inheritance provides a powerful language mechanism for using class libraries of reusable components.

6.1.3 Software Reuse with the Object-Oriented Paradigm

Henry, et al, determined quantitatively that the object-oriented paradigm is better than a procedural approach in terms of software maintenance [Henry, Humphrey and Lewis, 1990]. However, it found the subjects viewed the object-oriented techniques as more difficult to accomplish. The objective data supported object-oriented design in that it resulted in fewer maintenance tasks and reduced effort.

The study by Lewis, et al, supports this enhanced reuse [Lewis, et al, 1991]. A controlled experiment was conducted to evaluate the impact of the object-oriented paradigm on software reuse. Some of the findings were ;

- Productivity is substantially improved using the object-oriented paradigm, although a significant part of this improvement is due to the effort of reuse.
- Software reuse improved productivity no matter which language was used.
- The differences in the languages were far more important when programmers reused than when they did not.
- The object-oriented paradigm has a particular affinity for reuse.

6.1.4 Factors Affecting Maintenance and Renewal

Part of the reason for renewing a software program will be to increase it's maintainability. Factors that affect the maintenance of a software program must therefore be identified and acted upon in the new design and development. Identified factors [Sommerville, 1985] affecting program maintenance include :

- *Module Independence* : Modifications of one module do not affect the rest of the program.
- *Programming Language* : Programs implemented in high-level programming languages are easier to understand than programs implemented in low-level languages.

- *Programming Style* : Clearly, well set-out programs increase understanding.
- *Program Validation and Testing* : The more time spent on testing and validation means less time spent on corrective maintenance because there are less errors.
- *Quality and Quantity of Program Documentation* : Correct, clear, concise and complete documentation drastically increases program understanding.

Using the object-oriented approach and Ada as the implementation language, the first two factors are addressed. Ada *pretty-printer* formatters help with the third factor. By thoroughly testing modules, the maintenance effort can be reduced. Development of additional quality documentation will improve comprehension and understanding of the program.

6.2 Program Understanding

The first step in any renewal effort is the understanding of the behaviour (how it works) of the existing program to be renewed. From this understanding, the relevant algorithms and data structures can be extracted. Without proper understanding, many of the negative properties of the existing program will be transferred to the new one. At worst, algorithms and data structures will not behave as they should in the new program.

6.2.1 Factors affecting Program Understanding

The key to effective maintenance or renewal is program understanding. Changing a program or developing a new one is likely to result in errors if the original program is not properly understood [Sommerville, 1985].

- The person trying to understand the program must have a thorough knowledge and understanding of the programming language in which the program is implemented.
- The type of application should be understood. For example, compilers have various parts : lexical, syntactic and semantic analysis parts for the front end, and code generation and optimization for the back end. When trying to understand this type of program, the first step would be to look for these parts. In other words, recognition of major components of the system. Without this, each component must be studied in detail to get an understanding of it's function and properties.
- Once the major program components have been identified, how these components are activated and used can be found by one of two ways:
  * The purpose of each component is determined, and then how and where the component should be used is deduced.
  * The program code is studied and traced to find where and how the components are used. The purpose of the components can then be determined.
- A strategy to determine how a program component works is to study the component's:

* Input and output statements to discover the interaction of the component with it's environment.
* Data declarations.
* Procedural declarations.
* Control statements to discover the component's operation.

The order in which each part of the component is studied is up to the programmer.

- Low-level comments should be added at the place of discovering the meaning and function of a certain piece of code by the programmer.

6.2.2 Understanding Elecsim Code

The following approach was taken in understanding Elecsim.

(1) Reading the *User and Reference Manual* version 1.0. This manual documents the behaviour of each function. The explanation of the functions are grouped according to the menu on which they are found. The explanation of each function is divided into:
   - Function's Purpose.
   - How to use it.
   - Remarks.
   - Error Messages.

(2) Next, the program was executed using several different circuits for analysis. Most of the functions were tried and some understanding of their behaviour was grasped.

(3) A review of Lipworth's dissertation showed the following sections to be useful [Lipworth, 1989] :
   - Data flow diagrams and the corresponding PDL that described the behaviour of some Elecsim processes.
   - Discussion on the netlist data structure.
   - Simulating circuits using Elecsim.

(4) Examining the source code to find the overall structure of Elecsim and the dependencies between the units. (See Appendix A).

(5) Examining all the major data structures, especially the netlists.

(6) With the aid of the data flow diagrams, an understanding of the processes from the highest level of abstraction downwards was developed.

(7) The procedures and functions were then examined. Often, this required code walk-throughs of subprograms, calling subprograms and called subprograms. This was done in an effort to find the *context* in which a subprogram was functioning.

6.2.3 Problems in Understanding Elecsim

- *Redundant Variable Declarations* : Often in the Pascal source, variables were declared and

never used. These probably came about when the code was changed, but the variable and constants were not modified accordingly. This makes the code more complex than it need be.

- *Large Arrays* : Some of the large arrays were larger than they should have been. In *Monte-Carlo* and *Plotter* packages, these arrays were reduced to the size that was needed.
- *Compound Statements* : The way that Pascal deals with compound statements often leads to confusion. Also, it is sometimes difficult to determine which *end* belongs to which *begin*.
- *Inconsistent Code Layout* : The Pascal code for Elecsim was not set out in a consistent style. Variables, language statements, etc. were often written in mixed case. These mixed case names changed from one instance to the next.
- *Indentation* : Often, *if-then-else*, *repeat-until* etc. indentation levels did not match up. This sometimes led to designing sections of code that were put into the wrong *if* statement section or the wrong loop.
- *Comments* : There were limited comments interspersed at random intervals in some of the Pascal units. It took some time to determine the purpose of each module.

6.3 Problems in Constructing Elector

6.3.1 Language and D gn Comprehension

There were a few errors that were caused when aspects of the Pascal language were confused with that of Ada. One of these aspects was a decrementing count in a *for* loop. Pascal requires that the starting limit be declared first. Ada needs a range specification, which means that the starting value is declared as the second value.

Other types of problems were :

- No floating point emulation for version 2.1 of the Meridian compiler.
- Compiler hanging for certain programs.
- Exception handling problems when passing back *out* parameters.
- Access values produce false values for large dynamic arrays.
- Unpredictable behaviour when using the Debugger.
- Compilation library was not properly updated when compiling.
- Optimiser introduced certain errors when compiling and linking.

6.4 Time Records

A detailed time record was kept in the initial stages. However, there were too many problems with the compiler software. Almost half of the total time on the project was spent trying to get around a number of programming problems. Also, in December 1991, a new compiler was

obtained and code had to be modified and adapted to this compiler. The Halo graphic library software which was supposed to be incompatible. However, approximate times were kept and are displayed in Table 6.1 and Figure 6.1.

Time Spent (in weeks)	Activity
5	Dos_Environment (Assembler programming)
9	User Interface development (Design and Ada programming)
3	Graphic Library (Design, Assembler and Ada programming)
3	Modification of code from Meridian 2.1 compiler to Meridian 4.1 compiler.
4	Test cases (Design and Ada programming)
6	Understanding Elecsim source code and additional information.
15	Redesign and coding. (Elector)

Table 6-1 Time taken for various Activities

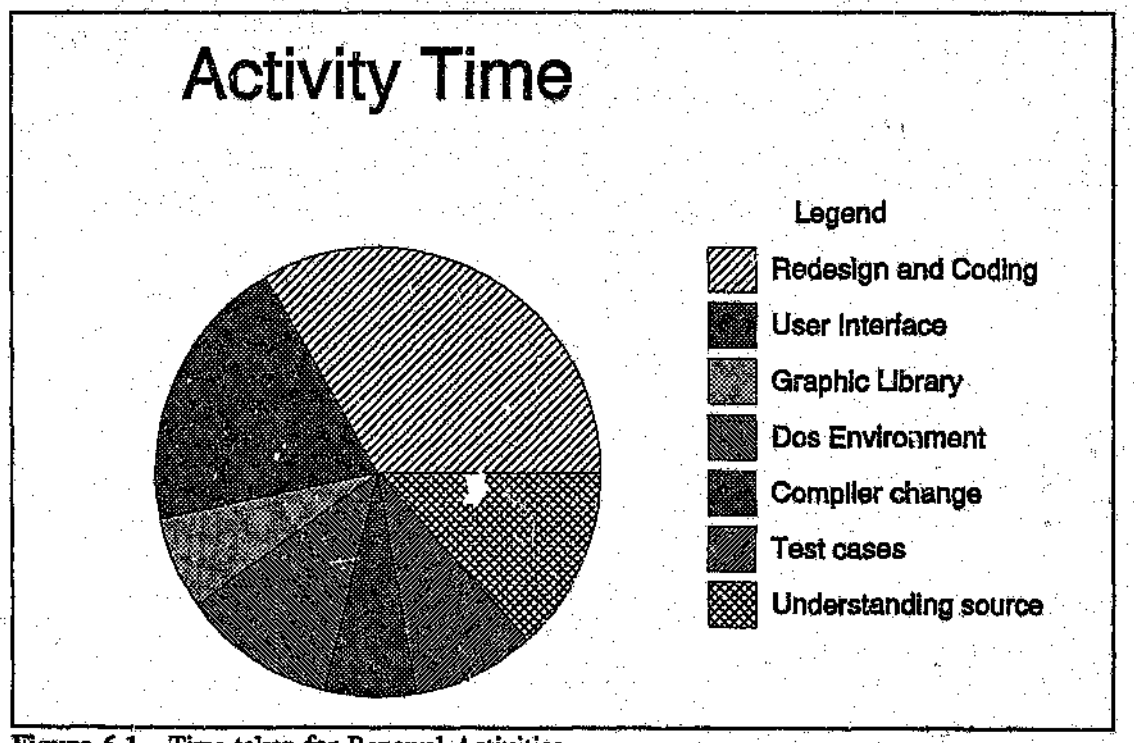


Figure 6-1 Time taken for Renewal Activities

6.4.1 COCOMO Estimates

The basic COCOMO (COnstructive COst MOdel) is a static single-valued model that calculates the software development effort as a function of the program size measured in estimated lines of code (LOC). To use the equations in Equation 6-1, the program must be classified as being one of the following :

- *Organic Mode* : Relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements.
- *Semi-Detached Mode* : Intermediate in size and complexity with teams of mixed experience levels and mixed software requirements.
- *Embedded Mode* : Software project with tight hardware, software and operational requirements.

$$E = a_b\ (KLOC)\ \exp\ (b_b)$$

$$D = c_b\ (E)\ \exp\ (d_b)$$

KLOC : *Estimated number of lines of code in thousands*
D : *Development time in chronological months*
E : *Effort applied in person-months*

Equation 6-1 COCOMO Equation for Basic Model

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 6-2 COCOMO values for Basic Model

Elector is of the *organic* type and consists of 7464 Ada statements. Using lines of code (LOC) for Ada is not very accurate because a subprogram call with one parameter will take one line, but a subprogram with five parameters could take five lines. Both subprogram calls are one statement.

Using Equation 6-1, the effort *E* is about **51** person-months and *D* is about **181** chronological months. Elector took under 11 months. The benefit of renewal can be seen when comparing the estimates and the time required to develop Elector. It must be kept in mind that software in commercial environments is usually constructed in teams. Team communication and coordination will play a major role in software development and increase the person-months time.

6.5 Summary

The importance of Software Engineering goals and principles was highlighted, followed by the problem of software reuse and reusable components. Factors affecting renewal and maintenance were enumerated. A theoretical approach to understanding the program to be renewed was presented. The practical application of this approach was then presented. Once the functionality and construction of Elecsim was understood, Elector was constructed. Some of the difficulties encountered during this construction were discussed.

Time records for the renewal process were presented. A comparision with the COCOMO empirical estimation model was made.

The next chapter draws conclusions from the work presented in this dissertation and makes a few suggestions for further work.

CHAPTER 7

CONCLUSION

7.1 Conclusions for this Project

7.1.1 General Conclusions

Some of the problems found in Elecsim are instances of more general problems.

- *Menu Hierarchy* : The tight coupling of the menus to the rest of the program.
- *Standard Reusable Libraries* : Well-defined S-type components of the program are still created from scratch. Components cannot be readily identified.
- *Documentation* : Scarce explanation on the interaction of major objects (modules) of the program. Documentation has focused on the construction and behaviour of individual objects, and not on the collective behaviour of the objects acting within a group.

The widespread use of functional decomposition (top-down design) has lead to large numbers of software products which are difficult to maintain. Two major factors are the tight coupling of the user interface to the rest of the program, and the hierarchical tree-like program structure.

Object-oriented design techniques address these problems. However, to fully realize object-oriented designs requires object-oriented languages and supporting tools. If these tools are not available for a particular application or domain area (eg. microcontrollers), a flat program structure with a decoupled user interface using a general purpose language is advisable.

Renewal of software is definitely profitable if the software is of the *P*-type class defined in 1.1. The case studies Elecsim and Elector are both *P*-type programs. If the code itself cannot be used, major components of the existing system can be identified for use in future systems.

7.1.2 Renewal

In the case study of Elector, the problems with the compiler are not part of the *renewal process*. As such, the renewal part of the project took only 8 months. This includes the time used to develop a window type user interface which can be used in many other software packages. The basic COCOMO empirical estimation model gives 51 person-months for such a project. The renewal effort is thus six times better.

Some of the algorithms that were extracted are complex. To develop these from scratch would take months. Additional time spent developing these new algorithms does not bring the equivalent benefits. There is not much value in re-inventing the wheel.

The approach taken to understand Elecsim has been successful. Because Elecsim was constructed along object-based lines, major objects within the program were readily identified. Other procedures were incorporated into newly developed objects.

Documentation for the actual program consisted of data-flow diagrams for some processes of Elecsim and PDL for some Netlist procedures. The purpose of many objects and modules could only be determined by closely examining the program code and doing code walk-throughs.

Software tools can help extensively. In the early phases, pretty-printers make code much easier to read and cross-referencers and data dictionaries can help in finding obscure relationships between different parts of the program.

Greater knowledge of Ada could have reduced some of the errors encountered. Ada has basic constructs which are very close to Pascal and so the transition to write simple programs is fairly easy. However, because of the size of Ada and the expressive power it has, there are many situations where an idea can be expressed in many different ways. It takes a bit of time and experience to know which one or which few are the best for a particular situation.

7.1.3 Decoupling the User Interface

There are many benefits from the approach taken to decouple the user interface. Modifications to the user interface or the rest of the program can be done independently of one another.

- *Decoupling* : The user interface is completely decoupled from the application part of the program. It is accomplished by means of a communication channel which returns OPERATION codes corresponding to the option chosen on the menu.
- *Menu Data Structure* : All appearance and behaviour information for the different menus is kept in a data structure instead of being hard coded into menu procedures. Changing any aspect of a menu's appearance or behaviour only requires changing a value in the menu data structure.
- *Localisation* : There is one major data structure that keeps the information for all the menus. There is one method for drawing the menu, one for handling input to the menu, etc. instead of many duplicated procedures. This ensures consistency in the appearance and behaviour of the menus.
- All the menus are accessed using a single method. This allows complete transversal of the menu structure. The result of a process in the application object can be reflected in the menu structure by making certain options selectable or non-selectable.

7.1.4 Program Structure

Restructuring Elecsim has resulted in fewer, larger objects in which the methods (processes) were on the same abstract level. Greater encapsulation and enforced abstract data types has been obtained. The construction of a distinct scheduler for a subsystem seems useful, especially where there are many independent objects which interact with one another in a complex manner.

- *Distinct Independent Scheduler* : The sole purpose of this scheduler is to schedule the program operation. There is no direct user interface involvement, but a request is made to the user interface to get the next menu option.
- *Encapsulation and Localisation* : All the scheduling decisions are confined to a single object. This includes data structures and variables which provide *state* information which the scheduler uses to decide which process must be invoked next. All data structures and scheduling algorithms are hidden. This allows a data structure or algorithm to be replaced without any impact on any other objects.
- *Concurrent Scheduling* : This will not be difficult to accomplish as all scheduling is done by one object. If the implementation is Ada, case statements can be replaced with select statements which rendezvous with tasks.

7.1.5 Review of Research Questions

The initial questions posed at the start of the disertation (1.1.2 and 1.1.3) are manifestations of the functional decomposition design method. The idea of flattening the program structure so that all the process can be on the same level comes about if the program structure is a hierarchy of processes. It it is not, flattening the hierarchy will have a detrimental effect on other parts of the program which are unrelated to the scheduling of processes.

Thus, for programs which were designed using functional decomposition, the flattening of the program structure is favoured. But for programs which were designed with object-oriented decomposition, the flattening of program structure amounts to the flattening of the *kind of* hierarchy, or the *part of* hierarchy. Since these hierarchies do not relate processes to one another, flattening them will interfere with other structural relations in the program .

Since object-oriented methods produce software structure that is deemed to be more suitable for maintenance, the flattening of program structure is not applicable.

7.2 Suggestion for Further Improvements and Research

7.2.1 Implementation Language

Since the selection of Ada as the implementation language, work on a new language standard (Ada9x) incorporating object-oriented features has been evolving. At the same time, the C++ language which is a superset of C and incorporates object-oriented features, has become the industry standard. The widespread use of this language in the telecommunications field provides a large base of software. Many C systems will need to be upgraded and even renewed. A research direction could be examining the renewing of software systems from C to C++.

7.2.2 User Interface

The popularity of Graphical User Interfaces, and especially Windows for DOS, Presentation Manager for OS/2, X-Windows for Unix, means more programs are use them for their user dialogue. Elector has a mainly text-based User Interface with only the graphs and circuit schematics in high-resolution.

A major improvement would be the implementation of a new user interface using Windows or Presentation Manager. The study could determine what the impact would be of adapting the program to the new user interface.

7.2.3 Component Entry

At present, components are entered into Elector using a menu. An enhancement would be the entry of components using a mouse. A component would be selected by moving the mouse pointer to component icons. From there, the icon is dragged to its correct place in the circuit.

REFERENCES

Aksit, M. and Bergmans, L. (1992) Obstacles in Object-Oriented Software Development, *OOPSLA'92 Conference Proceedings*, ACM Press, October 1992, pp. 341-358

Anderson, L.H. (1978) Linear Circuit Analysis, *BYTE*, Vol 3, No 10, October 1978, pp. 100-118

Arnold, R.S. (1989) Software Restructuring, *Proceedings of the IEEE*, Vol 77, No 4, April 1989

Berlin, L. (1990) When Objects Collide : Experiences with Reusing Multiple Class Hierarchies, *Proceedings of OOPSLA/ECOOP'90*, ACM Press, Ottawa, Canada, 1990, pp. 181-193

Berry, R.E. (1988) Common User Access - A Consistent and Usable Human-Computer Interface for the SAA Environments, *IBM Systems Journal*, Vol 27, No 3, 1988, pp. 281-300

Blume, W.F. (1986) Computer Circuit Simulation, *BYTE*, Vol 11, No 7, July 1986, pp. 165-178

Booch, G. (1983) *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, United States of America, 1983, pp. 25-27

Booch, G. (1986) Object-Oriented Development, *IEEE Transactions on Software Engineering*, February 1986, pp. 211-221

Booch, J. (1991) *Object-Oriented Design*, Benjamin/Cummings, United States of America, 1991

Borland, (1990) *Turbo Pascal V6 : Turbo Vision Guide*, Borland, United States, 1990

Buhr, R.J.A. and Casselman, R.S. (1992) Architecture with Pictures, *OOPSLA'92 Proceedings*, ACM Press, 1992, pp. 466-483

Capers-Jones, T.C. (1984) Reusability in Programming : A Survey of the State of the Art, *IEEE Transactions in Software Engineering*, Vol 10, No 5, September 1984, pp. 488-494

Coutaz, J. (1985) Abstractions for User Interface Design, *Computer*, September 1985, pp. 21-34

Dunfee, W.P., McGehe, J.D., Rauf, R.C., Shipp, K.O. (1988) Designing SAA Applications and User Interfaces, *IBM System Journal*, Vol 27, No 3, 1988, pp. 325-347

Edward, L.N.M. (1983a) LINSIM : A Linear Electrical Network Simulation and Optimisation Program, *Int. J. Elect. Enging Educ.*, Vol 20, No 2, February 1983, pp. 151-168

Edward, L.N.M. (1983b) BIMRIM : A Mutual Inductance Algorithm for use with Nodal Analysis Programs, *Int. J. Elect. Engng Educ.*, Vol 20, No 2, February 1983, pp. 341-350

Ferrel, P.J. and Meyer, R.F. (1989) Vamp : The Aldus Application Framework, *OOPSLA '89 Proceedings*, ACM Press, New Orleans, United States, October 1989, pp. 185-189

Gibson, V.R. and Senn, J.A. (1989) System Structure and Software Maintenance Performance, *Communications of the ACM*, Vol 32, No 3, March 1989, pp. 347-358

Gonzalez, D.W. (1991) *Ada Programmer's Handbook and Language Reference Manual*, Benjamin/Cummings, United States of America, 1991

Griswold, W.G. and Notkin, D. (1992) Computer-Aided vs. Manual Program Restructuring, *ACM SIGSOFT Software Engineering Notes*, Vol 17, No 1, Jan 1992, pp. 33-41

Harrison, W.H., Shilling, J.J., Sweeney, P.F. (1989) Good News, Bad News : Experience Building a Software Development Environment Using the Object-Oriented Paradigm, *Conference Proceedings of OOPSLA'89*, ACM Press, New Orleans, United States of America, 1989, pp. 85-94

Hartson, R. (1989) User-Interface Management Control and Communication, *IEEE Software*, January 1989, pp. 62-70

Henry, S.M., Humphrey, M., Lewis, J.A. (1990) Evaluation of the Maintainability of Object-Oriented Software, *Proceedings of Conference on Computer and Communications Systems*, Vol 1, Hong Kong, September 1990, pp. 404-409

Jacobson, I. and Lindström, F. (1991) Re-engineering of Old Systems to an Object-Oriented Architecture, *OOPSLA'91 Conference Proceedings*, ACM Press, 1991, pp. 340-355

Korson, T. and McGregor, J.D. (1990) Understanding Object-Oriented : A Unifying Paradigm, *Communications of the ACM*, Vol 33, No 9, September 1990, pp. 40-60

Lehman, M.M. (1980) Programs, Life Cycles, and Laws of Software Evolution, *IEEE Proceedings*, Vol 68, No 9, September 1980, pp 1060-1076

Lewis, J.A., Henry, S.M., Kafura, D.G., Shulman, R.S. (1991) An Empirical Study of the Object-Oriented Paradigm and Software Reuse, *OOPSLA '91 Conference Proceedings*, 1991, pp. 184-196

Lipworth, A.D. (1989) *Elecsim - An Integrated Interactive Linear Electric Network Simulation and Optimisation Package*, MSc Dissertation, University of the Witwatersrand, Johannesburg, South Africa, 1989

Liskov, B. (1988) Data Abstraction and Hierarchy, *OOPSLA'87 Addendum to the Proceedings*, Orlando, Florida, Vol 23, No 5, May 1988, ACM Press, pp. 17-34

Ossher, H.L. (1986) A Mechanism for Specifying the Structure of Large, Layered Object-Oriented Programs, *SIGPLAN Notices*, Vol 21, No 10, October 1986, pp. 143-151

Ossher, H. and Harrison, W. (1992) Combination of Inheritance Hierarchies, *OOPSLA'92 Conference Proceedings*, ACM Press, October 1992, pp. 25-40

Rombach, H.D. (1987) A Controlled Experiment on the Impact of Software Structure on Maintainability, *IEEE Transactions on Software Engineering*, Vol 13, No 3, March 1987, pp. 344-354

Rombach, H.D. and Ulery, B.T. (1989) Improving Software Maintenance Through Measurement, *Proceedings of the IEEE*, Vol 77, No 4, April 1989, pp. 581-595

Savitch, W.J. and Petersen, C.G. (1992) *Ada : An Introduction to the Art and Science of Programming*, Benjamin/Cummings, United States of America, 1992

Schach, S.R. (1990) *Software Engineering*, Alssen Associates, United States of America, 1990

Schmucker, K.J. (1986) MacApp : An Application Framework, *BYTE*, August 1986, pp. 189-193

Schneidewind, N.F. (1987) The State of Software Maintenance, *IEEE Transactions on Software Engineering*, Vol 13, No 3, March 1987

Schwanke, R.W., Altucher, R.Z., Platoff, M.A. (1989) Discovering, Visualizing, and Controlling Software Structure, *ACM SIGSOFT Engineering Notes*, Vol 14, No 3, May 1989, pp. 147-150

Shaw, M. (1989) Larger Scale Systems Require Higher-Level Abstractions, *ACM SIGSOFT Engineering Notes*, Vol 14, No 3, May 1989, pp. 143-146

Sommerville, I. (1985) *Software Engineering*, 2nd Ed, Addison-Wesley, Finland, 1985, pp. 220-221, 224-225

Uhlir, S. (1988) Enabling the User Interface, *IBM Systems Journal*, Vol 27, No 3, 1988, pp. 306-314

Walker, A.J. (1988) A Software Challenge for the 90's, *Elektron*, South Africa, May 1988, pp. 9-13

Walker, M.G. and McGregor, J. (1986) Computer-Aided Engineering for Analog Circuit Design, *Computer*, Vol 19, No 4, April 1986, pp. 100-108

Weinand, A., Gamma, E., Marty, R. (1988) ET++ - An Object Oriented Application Framework in C++, *Proceedings of OOPSLA'88*, San Diego, United States, September 1988, pp. 46-57

Wheeler, E.F. and Ganek, A.G. (1988) Introduction to Systems Application Architecture, *IBM Systems Journal*, Vol 27, No 3, 1988, pp. 250-263

Wirf-Brock, A., Johnson, R., Cunningham, W., Linton, M. (1990) Panel : Designing Reusable Designs - Experiences Designing Objected Oriented Frameworks, *OOPSLA / ECOOP '90 Proceedings*, Ottawa, Canada, October 1990, ACM Press, pp. 234

Wirfs-Brock, R.J. and Johnson, R.E. (1990) Surveying Current Research in Object-Oriented Design, *Communication of the ACM*, Vol 33, No 9, September 1990, pp. 104-124

APPENDIX A

These are the source files that the Pascal version of Elecsim consists of. Many of these files are include files that were created because of the 64K limit imposed by the Turbo Pascal editor on single files.

3DW	PAS	15385	1-03-90	12:59p	(Include file)
ADMTNET	PAS	3459	11-23-89	5:39p	(Include file)
ANGLEVI	PAS	3055	1-29-90	10:30a	
BIMRIMM	PAS	33995	6-27-88	3:12p	(Include file)
CHECKOP	PAS	1384	1-29-90	10:30a	(Include file)
CHINIT	PAS	2245	1-29-90	10:30a	
CNTCMP	PAS	2879	4-26-88	10:28a	(Include file)
DISPNET	PAS	2026	1-29-90	10:31a	
DRAWNET	PAS	55718	1-29-90	10:31a	
ELEC5	PAS	1333	2-26-90	9:23a	(Main Program)
ELECV	PAS	2206	1-29-90	10:31a	
EXITING	PAS	591	3-23-90	1:37p	(Include file)
FINDMGN	PAS	2485	1-29-90	10:32a	
FOURIERS	PAS	12775	3-20-90	11:42a	
FRECV	PAS	7424	1-29-90	10:24a	Include file)
GETCMP	PAS	3554	1-29-90	10:32a	
GETFRQ	PAS	1444	1-29-90	10:32a	
GRTSR	PAS	846	2-20-90	4:24p	
HEXX	PAS	552	1-29-90	10:32a	
KILL	PAS	5232	1-29-90	10:33a	(Include file)
LBRNET	PAS	9048	1-29-90	10:33a	
LFNETT	PAS	5395	4-26-88	10:28a	(Include file)
LISTNET	PAS	4525	1-29-90	10:33a	
MENTS	PAS	61931	3-19-90	4:49p	
MENTS1	PAS	6247	11-22-89	12:33p	(Include file)
MENUS1	PAS	5994	1-29-90	10:33a	
MENUS2	PAS	5896	1-29-90	10:34a	
MENUS3	PAS	3910	1-29-90	10:34a	
MENUS4	PAS	24076	1-29-90	10:34a	
MENUSI	PAS	16589	1-29-90	10:34a	
MENUSM	PAS	28321	3-19-90	1:19p	
MGNPHS	PAS	4875	1-29-90	10:34a	
MKFINDD	PAS	5955	10-18-89	1:43p	(Include file)
MKUPDTT	PAS	4796	4-26-88	10:28a	(Include file)
MONTE	PAS	63151	3-19-90	10:34a	
NMCHK	PAS	877	12-14-89	9:32a	(Include file)
OPS	PAS	61528	1-29-90	10:35a	
OPTMG	PAS	10351	1-29-90	10:35a	
PLOTS	PAS	38697	2-02-90	3:30p	
POLARS	PAS	3249	1-29-90	10:35a	
POLRT	PAS	4558	1-29-90	10:35a	
PRTEGA	PAS	30608	7-27-90	12:40p	
QRPICL	PAS	2272	1-09-90	10:01a	(Include file)
RENMNETT	PAS	13586	6-16-88	5:19p	(Include file)
RLCNETT	PAS	5550	11-23-89	5:44p	(Include file)
RODENT	PAS	11511	2-01-90	11:45a	
SENSITS	PAS	8877	1-29-90	10:36a	
SHOWNDE	PAS	6172	11-27-89	9:53a	(Include file)
SLNNETV	PAS	21830	1-29-90	10:29a	
SMLGRAR	PAS	63740	9-11-90	10:32a	
SMLINC2	PAS	6395	3-05-90	9:54a	(Include file)
SMLREST	PAS	1967	1-29-90	10:37a	
SSWOP	PAS	1291	1-29-90	10:37a	
SVNET	PAS	4970	1-29-90	10:37a	(Include file)
TOPOLGYY	PAS	7003	4-26-88	10:28a	(Include file)
VNCALC	PAS	5004	1-29-90	10:38a	
VNODE	PAS	1816	1-29-90	10:38a	

What follows is a summary of all the modules (Pascal Units) that comprises Elecsim. The procedures and functions of that unit, and the dependent units are listed.

Pascal Unit	Dependent Units		Procedures and Functions	
Anglevi	Crt	Elecv	Angle (F)	
Checkop	Crt		Check_Val	
Chinit	Crt	Elecv	Ch_Set	
Dispnet	Crt Elecv	Smlgrar	Disp_Comp	
Drawnet	Crt Elecv Graph Menusi	Ops Printer Smlgrar		
Elecsim	Crt Dos Elecv Hex	Graph Menusm Overlay	Main program	
Elecv	Crt Smlgrar		Declaration of constants, variables and types that are used by the other modules.	
Findmgn	Crt	Elecv	Find_Mgn	
Fouriers	Anglevi Crt Elecv Findmgn Getfrq Graph	Ops Printer Slnnetv Smlgrar Vnode	Fixscale Fourier Getnodes Label_X Label_Y X_Axis	Y_Axis Zeroline
Getcmp	Crt Elecv	Smlgrar	Get_Comp	
Getfrq	Checkop Crt Elecv	Graph Ops Smlgrar	Get_Frequency	
Hexx	Crt		Hex	Hexbyte
Lbrnet	Anglevi Crt Elecv	Graph Smlgrar Ops	List_Brn	
Listnet	Anglevi Crt Dispnet Elecv	Lbrnet Ops Smlgrar	List_Net	

Pascal Unit	Dependent Units		Procedures and Functions
Ments	Anglevi	Graph	Menu_Tune
	Checkop	Ops	Optim_Anal
	Crt	Optmc	Optimising
	Dispnet	Slnr	Tune
	Elecv	Smlgrar	
	Findmgn	Sswop	
	Getcmp	Vnode	
	Getfrq		
Menus1	Crt	Smlgrar	Menu_1
	Elecv	Svnet	
	Ops		
Menus2	Chinit	Menusi	Menu_2
	Crt	Mgnphs	
	Drawnet	Ops	
	Elecv	Smlgrar	
	Getcmp	Svnet	
	Listnet		
Menus3	Crt	Ops	Menu_3
	Elecv	Smlgrar	
	Getfrq		
Menus4	Anglevi	Optmc	Curnt_Calc
	Checkop	Plots	Menu_4
	Crt	Polars	Power_Calc
	Elecv	Polrt	Set_Ccalc_Nodes
	Findmgn	Sensits	
	Fouriers	Slnnetv	
	Getfrq	Smlgrar	
	Ments	Vncalc	
	Monte	Vnode	
	Ops		
Menusi	Checkop	Getcmp	Insert_Comp
	Crt	Getfrq	Menu_I
	Dispnet	Ops	
	Elecv	Smlgrar	
Menusm	Chinit	Menus3	Getmsimage
	Crt	Menus4	Main_Menu
	Dos	Ops	Showdatfiles
	Elecv	Overlay	
	Graph	Printer	
	Menus1	Prtega	
	Menus2	Smlgrar	
	Menus3	Svnet	
Mgnphs	Anglevi	Getfrq	Change
	Crt	Ops	
	Elecv	Smlgrar	

Pascal Unit	Dependent Units		Procedures and Functions	
Monte	Anglevi	Graph	Clear_Monte	Ran
	Checkop	Ops	Fixscale	Recovmonte
	Crt	Printer	Gauss	Savemonte
	Dispnet	Slnnetv	Getnodes	Setup
	Dos	Smlgrar	Getxfreq	Testran
	Elecv	Sswop	Label_X	X_Axis
	Findmgn	Vnode	Label_Y	Y_Axis
	Getcmp		Lbl_Logx	Zeroline
	Getfrq		Menu_Monte	
Ops	Anglevi		Bim_Rim	Move_P
	Crt		Calc_Mut	Name_Check
	Dos		Conv_Admit	Null_Rec
	Elecv		Conv_Rlc	Readfreq
	Graph		Countcomp	Read_Net
	Smlgrar		Del_Net_Rec	Recov_Net
	Sswop		Find_Brnch	Renm_Net
			Find_Gm	Set_P
			Find_MK	Show_Nodes
			Find_Nmax	ShowNtlfiles
			Initfreq	Titles
			Initialise	Topol_Chk
			Initrecov	Update_MK
			Kill_Node	Writefreq
			List_Brn	Write_Net
Optmc	Anglevi	Elecv	Optm_Change	
	Crt	Ops		
	Dispnet	Smlgrar		
Plots	Anglevi	Slnnetv	Fixscale	Para_Lin_Plot
	Checkop	Smlgrar	Getnodes	Para_Lin_Plot3
	Crt	Vnode	Getvertinc	Para_Log_Plot
	Elecv		Getxcomp	Plotauto
	Findmgn		Getxfreq	Plotpnt
	Getcmp		Label_X	Plott
	Getfrq		Label_Y	X_Axis
	Graph		Lbl_Logx	Y_Axis
	Ops		Log_Plot	Zeroline
	Printer		Norm_Plot	
Polars	Anglevi	Getfrq	Polaring	
	Crt	Ops		
	Elecv	Slnnetv		
	Findmgn	Smlgrar		
	Graph	Vnode		
Polrt	Anglevi	Graph	Getnodes	
	Crt	Ops	Polart	
	Elecv	Slnnetv		
	Findmgn	Smlgrar		
	Getfrq	Vnode		

Pascal Unit	Dependent Units		Procedures and Functions	
Prtega			Choosegreypalette Egadump Epsonprint Getcolourbyte Getepsonline Getepson8dot Getlj2line Getlj2dotpair Invertgreypalette Lj2print Lj2setup	Lj2shutdown New_Layer New_Line Printega Quant16 Send_Double_Density Send_Quadruple_Density Send_Single_Density Set_Home_Head Sort
Rodent	Dos		M2speed Mcondition Mconditionup Mgraphicshape Mhide Mkbd Mmotion Moveratio Mpage	Mpos Mput Mshow Mstatus Mtexthape Musersub MvlimitX MvlimitY
Sensits	Crt Elecv Findmgn Getfrq	Ops Smlgrar Slnnetv Vnode	Sensitivity	
Slnnetv	Crt Elecs Ops	Smlgrar Svnet	Ac_Sln_Net Find_G_B	Gb_Mat_Store Solve

Pascal Unit	Dependent Units		Procedures and Functions	
Smlgrar	Crt Dos Drivers Font Graph Printer Rodent Smlrest		Clr_Eol Clr_Frq_Line Clr_Line Clr_Ms_Line Clr_Scr Dn_Scr File_Check Home Kbd_Get Left_Margin Lt_Scr Mandi Mandil Mandr Mands Menu_Clear Ms_Clear Ms_Menu Ms_Options Ms_Picload Ms_Picsave Ms_Qpicload Ms_Qpicsave Ms_Qrpicload Ms_Qrpicsave Ms_Stringload	Ms_Window Next_Line Path_Check Ph_Window Ps_Clear Ps_Wait Ps_Wfi Ps_Wfil Ps_Wfr Ps_Wfsy Ps_Write Read_Int Read_Real Read_String Rt_Scr Scr_Format Scr_Restore Set_Cp Ss_Clear Ss_Mandi Ss_Mandil Ss_Mandr Ss_Write Unitss Up_Scr
Smlrest	Crt		Bleat Blink_On Bold_On	Restore Rvid_On
Sswop	Crt		Swop	
Svnet	Crt Dos Elecv	Ops Smlgrar	Save_Net	
Vncalc	Anglevi Crt Elecv Findmgn Getfrq	Ops Slnnetv Smlgrar Vnode	V_Calc_Node	
Vnode	Crt	Elecv	V_Node	

ELECTOR ADA SOURCE CODE

APPENDIX B

Ada_Extensions

```
package Ada_Extensions is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This is the base or foundation object on which all the other
--           objects are build. It contains several data types, as well as
--           the dynamic string and maths objects.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   RADIAN : constant := 57.295_779_513;

   subtype INTEGER_TYPE   is INTEGER;
   subtype REAL_TYPE      is FLOAT;
   subtype COLUMN_TYPE    is INTEGER      range 1..80;
   subtype NODE_TYPE      is INTEGER      range 0..250;
   subtype OPERATION_TYPE is INTEGER;
   subtype POW_TYPE       is INTEGER      range 1..15;

   type FOREGROUND_TYPE is (
      BLACK,            BLUE,
      GREEN,            CYAN,
      RED,              PURPLE,
      BROWN,            LIGHT_GRAY,
      DARK_GRAY,        LIGHT_BLUE,
      LIGHT_GREEN,      LIGHT_CYAN,
      LIGHT_RED,        LIGHT_PURPLE,
      YELLOW,           WHITE);

   subtype BACKGROUND_TYPE is FOREGROUND_TYPE range BLACK..LIGHT_GRAY;

   type FIELD (VALUE_LENGTH      : POSITIVE := 10;
               TEXT_LENGTH       : POSITIVE := 10;
               IS_INTEGER_TYPE   : BOOLEAN  := TRUE) is
      record
         ACTUAL_LENGTH  : NATURAL;
         ACCEPTING      : BOOLEAN;
         ROW            : ROW_TYPE;
         TEXT_AGAINST   : INTEGER_TYPE;
         VALUE          : STRING (1..VALUE_LENGTH);
         TEXT           : STRING (1..TEXT_LENGTH);
         case IS_INTEGER_TYPE is
            when TRUE =>
               MAX_INTEGER,
               MIN_INTEGER,
               INTEGER_VALUE : INTEGER_TYPE;
            when others =>
               MAX_REAL,
               MIN_REAL,
               REAL_VALUE : REAL_TYPE;
         end case;
      end record;

   type FIELDS_TYPE is array (POSITIVE range <>) of FIELD;

   type LIST_LINE_TYPE (TEXT_LENGTH : POSITIVE := 30) is
      record
         TEXT  : STRING (1, TEXT_LENGTH);
         STATE : BOOLEAN := FALSE;
      end record;

   type LIST_ARRAY_TYPE is array (POSITIVE range <>) of LIST_LINE_TYPE;

   package Maths is

      function Angle (X_COORDINATE, Y_COORDINATE : in REAL_TYPE)
         return REAL_TYPE;

      function Find_Magnitude (REAL_VALUE, IMAG_VALUE : in REAL_TYPE)
         return REAL_TYPE;

      function Log (X : in REAL_TYPE) return REAL_TYPE;

      procedure Check_Value (MINIMUM, MAXIMUM : in REAL_TYPE;
         VALUE : in out REAL_TYPE);

      procedure Check_Value (MINIMUM, MAXIMUM : in INTEGER_TYPE;
         VALUE : in out INTEGER_TYPE);

      procedure Swop_Node (NODE1, NODE2 : in out NODE_TYPE);

   end Maths;

   package Texts is

      type TEXT_TYPE (TEXT_LENGTH : POSITIVE := 70) is
         record
            TEXT       : STRING (1..TEXT_LENGTH);
            TEXT_INDEX : NATURAL := 0;
         end record;

      procedure Combine (ITEM : in STRING; TEXT_LINE : in out TEXT_TYPE);

      procedure Combine (ITEM : in INTEGER_TYPE;
         ITEM_WIDTH : in INTEGER := INTEGER_TYPE'Width;
         TEXT_LINE  : in out TEXT_TYPE);

      procedure Combine (ITEM : in REAL_TYPE;
         WHOLE      : in INTEGER := 2;
         FRACTION   : in INTEGER := REAL_TYPE'digits - 1;
         EXPONENT   : in INTEGER := 3;
         ENG_NOTATION : in BOOLEAN := FALSE;
         TEXT_LINE  : in out TEXT_TYPE);

      procedure Convert_To_Integer (FROM : in STRING;
         ITEM : out INTEGER_TYPE);

      procedure Convert_To_Real (FROM : in STRING;
         ITEM : out REAL_TYPE);

      procedure Reset (TEXT_LINE : in out TEXT_TYPE);

   end Texts;

end Ada_Extensions;
```

```
with  Math_Lib,
      Text_IO;

package body Ada_Extensions is

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

package body Maths is

   use Math_Lib;

-- PURPOSE : Calculates  the angle  in  degrees  from  X_COORDINATE  and
--           Y_COORDINATE.

function Angle (X_COORDINATE, Y_COORDINATE : in REAL_TYPE)
   return REAL_TYPE is

   TEMP_ANGLE : REAL_TYPE;

begin
   if (X_COORDINATE = Y_COORDINATE) = 0.0 then
      if (X_COORDINATE >= 0.0) and (Y_COORDINATE = 0.0) then
         return 0.0;                    -- point on + X axis or origin
      elsif (X_COORDINATE < 0.0) then
         return 180.0;                  -- point on - X axis
      elsif (Y_COORDINATE < 0.0) then
         return -90.0;                  -- point on - Y axis
      elsif (Y_COORDINATE > 0.0) then
         return 90.0;                   -- point on + Y axis
      end if;
   else
      TEMP_ANGLE := RADIAN * Atan (Y_COORDINATE / X_COORDINATE);
      if (X_COORDINATE < 0.0) then
         if (Y_COORDINATE > 0.0) then
            TEMP_ANGLE := TEMP_ANGLE + 180.0;  -- angle in 2nd quadrant
         else
            TEMP_ANGLE := TEMP_ANGLE - 180.0;  -- angle in 3rd quadrant
         end if;
      end if;
   end if;
   return TEMP_ANGLE;
end Angle;

-- PURPOSE : Calculates the logarithm to base 10 of a  number  by  finding
--           the natural logarithm and then converting it to the new base.

function Find_Magnitude (REAL_VALUE, IMAG_VALUE : in REAL_TYPE)
   return REAL_TYPE is

begin
      return Sqrt (REAL_VALUE * REAL_VALUE + IMAG_VALUE * IMAG_VALUE);
end Find_Magnitude;

-- PURPOSE : Calculates the logarithm to base 10 of a  number  by  finding
```

```
--            the natural logarithm and then converting it to the new base.
-------------------------------------------------------------------------

function Log (X : in REAL_TYPE) return REAL_TYPE is

   LOGARITHM_10_TO_BASE_E : constant := 2.302_585_092_994_05;

begin
   return Ln (X) / LOGARITHM_10_TO_BASE_E;
end Log;

-------------------------------------------------------------------------
-- PURPOSE : VALUE is checked against the minimum and is assigned it if it
--           is smaller.  Otherwise VALUE is then checked against the
--           maximum and is assigned it if it is larger.
-------------------------------------------------------------------------

procedure Check_Value (MINIMUM, MAXIMUM : in INTEGER_TYPE;
   VALUE : in out INTEGER_TYPE) is

begin
   if (abs (VALUE) < MINIMUM) then
      VALUE := MINIMUM;
   elsif (abs (VALUE) > MAXIMUM) then
      VALUE := MAXIMUM;
   end if;
end Check_Value;

-------------------------------------------------------------------------
-- PURPOSE : VALUE is checked against the minimum and is assigned it if it
--           is smaller.  Otherwise VALUE is then checked against the
--           maximum and is assigned it if it is larger.
-------------------------------------------------------------------------

procedure Check_Value (MINIMUM, MAXIMUM : in REAL_TYPE;
   VALUE : in out REAL_TYPE) is

begin
   if (abs (VALUE) < MINIMUM) then
      VALUE := MINIMUM;
   elsif (abs (VALUE) > MAXIMUM) then
      VALUE := MAXIMUM;
   end if;
end Check_Value;

-------------------------------------------------------------------------
-- PURPOSE : The value of NODE1 is swopped with the value of NODE2.
-------------------------------------------------------------------------

procedure Swop_Node (NODE1, NODE2 : in out NODE_TYPE) is

   TEMP_NODE : NODE_TYPE;

begin
   TEMP_NODE := NODE1;
   NODE1 := NODE2;
   NODE2 := TEMP_NODE;
end Swop_Node;

-------------------------------------------------------------------------

end Maths;

-------------------------------------------------------------------------
-------------------------------------------------------------------------

package body Texts is

   -----------------------------------------------------------------------

   package Real_In_Out    is new Text_IO.Float_IO   (REAL_TYPE);
   package Integer_In_Out is new Text_IO.Integer_IO (INTEGER_TYPE);

   -----------------------------------------------------------------------
   -- PURPOSE  : Combines the ITEM string to the text held by TEXT_LINE.
   -----------------------------------------------------------------------

   procedure Combine (ITEM : in STRING;
      TEXT_LINE : in out TEXT_TYPE) is

      TEMP_INDEX1 : INTEGER := ITEM'last;
      TEMP_INDEX2 : INTEGER := TEXT_LINE.TEXT_INDEX + TEMP_INDEX1;

   begin
      if TEXT_LINE.TEXT_INDEX = TEXT_LINE.TEXT_LENGTH then
         return;
      elsif TEMP_INDEX2 > TEXT_LINE.TEXT_LENGTH then
         TEMP_INDEX1
            := TEMP_INDEX1 - (TEMP_INDEX2 - TEXT_LINE.TEXT_LENGTH);
         TEMP_INDEX2 := TEXT_LINE.TEXT_LENGTH;
      end if;
      TEXT_LINE.TEXT ((TEXT_LINE.TEXT_INDEX + 1)..TEMP_INDEX2)
         := ITEM (1..TEMP_INDEX1);
      TEXT_LINE.TEXT_INDEX := TEMP_INDEX2;
   end Combine;

   -----------------------------------------------------------------------
   -- PURPOSE : Converts an Integer number before adding it to the text that
   --           is held by TEXT_LINE.
   -----------------------------------------------------------------------

   procedure Combine (ITEM : in INTEGER_TYPE;
      ITEM_WIDTH : in INTEGER := INTEGER_TYPE'width;
      TEXT_LINE  : in out TEXT_TYPE) is

      TEMP_STRING : STRING (1..ITEM_WIDTH);

   begin
      Integer_In_Out.Put (TO => TEMP_STRING,
         ITEM => ITEM);
      Combine (ITEM => TEMP_STRING,
         TEXT_LINE => TEXT_LINE);
   end Combine;

   -----------------------------------------------------------------------
   -- PURPOSE : Converts a real number before adding it to the text that is
   --           held by TEXT_LINE.  The engineering constant for that number
   --           can be determined if there is one, and the number adjusted
   --           accordingly.
   -----------------------------------------------------------------------

   procedure Combine (ITEM : in REAL_TYPE;
      WHOLE       : in INTEGER := 2;
      FRACTION    : in INTEGER := REAL_TYPE'digits - 1;
      EXPONENT    : in INTEGER := 3;
      ENG_NOTATION : in BOOLEAN := FALSE;
      TEXT_LINE   : in out TEXT_TYPE) is

      ENG_STRING           : constant STRING := "YZAFPNUMKHDCSDHKMGTPEZY";
      ENGINEERING_CONSTANT : constant := 1.0e-3;
      LEAST_VALUE          : constant := 1.0e-18;
      EXTRA_MAX_EXPONENT   : constant := 1.0e+100;
      EXTRA_MIN_EXPONENT   : constant := 1.0e-100;

      VALUE                : REAL_TYPE := ITEM;
      ENG_VALUE            : REAL_TYPE := 0.999999;
      ENG_EXPONENT         : REAL_TYPE := 1.0e+15;
                             -- := (1.0 - (0.5 * (0.1 ** FRACTION))) * 1.0e+15;
      EXPONENTS            : INTEGER   := EXPONENT;
      PADDING              : INTEGER   := 1;
      COUNTER              : INTEGER   := 0;
      ENG_CHARACTER        : CHARACTER := ' ';
      FOUND                : BOOLEAN   := FALSE;

   begin
      if ENG_NOTATION then
         if (abs (VALUE) < ENG_EXPONENT) and (abs(VALUE) >= LEAST_VALUE) then
            while (abs (VALUE) < ENG_VALUE * ENG_EXPONENT) loop
               COUNTER := COUNTER + 1;
               ENG_EXPONENT := ENG_EXPONENT * ENGINEERING_CONSTANT;
            end loop;
            VALUE := VALUE / ENG_EXPONENT;
            ENG_CHARACTER := ENG_STRING (COUNTER);
            EXPONENTS := 0;
            PADDING   := 2;
            FOUND     := TRUE;
         elsif VALUE = 0.0 then
            EXPONENTS := 0;
            PADDING   := 2;
            FOUND     := TRUE;
         end if;
      end if;

      if (abs (VALUE) >= EXTRA_MAX_EXPONENT) or
         ((abs (VALUE) <= EXTRA_MIN_EXPONENT) and (VALUE /= 0.0)) then
         PADDING := PADDING + 1;
      end if;

      declare
         TEMP_STRING : STRING (1..(WHOLE+1+FRACTION + PADDING + EXPONENTS));
      begin
         Real_In_Out.Put (TO => TEMP_STRING,
            ITEM => VALUE,
            AFT  => FRACTION,
            EXP  => EXPONENTS);
         Combine (ITEM => TEMP_STRING,
            TEXT_LINE => TEXT_LINE);
      end;
      if ENG_NOTATION and FOUND then
         Combine (ITEM => " " & ENG_CHARACTER,
            TEXT_LINE => TEXT_LINE);
      end if;
   end Combine;

   -----------------------------------------------------------------------
   -- PURPOSE : Converts a string of characters representing an integer
   --           number to an integer.
   -----------------------------------------------------------------------

   procedure Convert_To_Integer (FROM : in STRING;
      ITEM : out INTEGER_TYPE) is

      LAST_POSITION : POSITIVE;

   begin
      Integer_In_Out.Get (FROM => FROM,
         ITEM => ITEM,
         LAST => LAST_POSITION);
   end Convert_To_Integer;

   -----------------------------------------------------------------------
   -- PURPOSE : Converts a string of characters representing a real number
   --           to a real.
   -----------------------------------------------------------------------

   procedure Convert_To_Real (FROM : in STRING;
      ITEM : out REAL_TYPE) is

      LAST_POSITION : POSITIVE;

   begin
      Real_In_Out.Get (FROM => FROM,
         ITEM => ITEM,
         LAST => LAST_POSITION);
   end Convert_To_Real;

   -----------------------------------------------------------------------
   -- PURPOSE : Resets the TEXT_LINE so that there are effectively no
   --           characters.
   -----------------------------------------------------------------------

   procedure Reset (TEXT_LINE : in out TEXT_TYPE) is

   begin
      TEXT_LINE.TEXT_INDEX := 0;
   end Reset;

   -----------------------------------------------------------------------

end Texts;

-------------------------------------------------------------------------
-------------------------------------------------------------------------

end Ada_Extensions;
```

Analysers

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Analysers is

-- Author  : Warren Gachls
-- Date    : June 1992
-- Purpose : This is a collection of analysers which calculate the current
--           power and sensitivity of the circuit. The nodes to be used in
--           a calculation can also be set.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure All_Calculation_Nodes (NODE_MAX : in NODE_TYPE);

   procedure Calculate_Current (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Calculate_Power (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Calculate_Voltage (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Sensitivity (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Set_Calculation_Nodes (NODE_MAX : in NODE_TYPE);

   procedure Show_Nodes;

   package Plotter is

      procedure Circuit_Response (NODE_MAX : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Fourier (NODE_MAX : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Phasors (NODE_MAX : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Polar (NODE_MAX : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

   end Plotter;

end Analysers;
```

```
with  Control_Solver,
      Math_Lib,
      Netlist_Supporter,
      Screens;

use   Screens;

package body Analysers is

   use Screens.Main;

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_POWER          : constant := 180;
   HP_SET_CALC_NODES : constant := 181;

   DECIBEL_FACTOR    : constant := 8.685_889_638_066; -- 20 / Ln (10)

   type NODE_ARRAY is array (1..MAX_CALCULATIONS) of NODE_TYPE;

   HIGHER_NODES,  LOWER_NODES    : NODE_ARRAY;

   TITLE1 : constant STRING := "    Nodes     Real Part    Imaginary Part" &
      "  Magnitude       Phase        dB          ";

-- PURPOSE : Sets which circuit nodes will be analysed.

procedure All_Calculation_Nodes (NODE_MAX : in NODE_TYPE) is

   COUNTER  : INTEGER;

begin
   HIGHER_NODES := (1..MAX_CALCULATIONS => 0);
   LOWER_NODES  := (1..MAX_CALCULATIONS => 0);
   for COUNTER in 1..MAX_CALCULATIONS loop
      if (COUNTER <= NODE_MAX) then
         HIGHER_NODES (COUNTER) := COUNTER;
      else
         exit;
      end if;
   end loop;
end All_Calculation_Nodes;

-- PURPOSE : Calculates  the  current  in the branches  and  displays the
--           results of the calculations on the main screen.

procedure Calculate_Current (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   COUNTER                          : INTEGER := 1;
   HIGH_NODE,          LOW_NODE     : NODE_TYPE;
   SCREEN_ROW                       : POSITIVE := 3;
   CAPACITORS,         CURRENT_FREQUENCY,
   DECIBELS,           I_MAGNITUDE,
   I_PHASE,            IN_CRNT,
   IN_VOLT,            IMAG_CRNTS,
   IMAG_VOLTS,         INDUCTORS,
   L_SUSEPTANCE,       MAGNITUDE,
   OMEGA,              REAL_CRNTS,
   REAL_VOLTS,         RESISTORS,
   RL_CRNT,            RL_VOLT,
   V_MAGNITUDE,        V_PHASE,
   VALUE,              Z_MAGNITUDE,
   Z_PHASE                          : REAL_TYPE;

begin
   Process_Title (TITLE => "CURRENT");
   Getter.Get_Frequency (FREQUENCY => CURRENT_FREQUENCY,
      NETLIST_OBJECT => NETLIST_OBJECT);
   OMEGA := 2.0 * Math_Lib.PI * CURRENT_FREQUENCY;
   Control_Solver.Solve_AC_Network (NODE => RLC,
      AC_FREQUENCY   => CURRENT_FREQUENCY,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   while (COUNTER <= MAX_CALCULATIONS) and
      (HIGHER_NODES (COUNTER) in 1..NODE_MAX) loop
      if SCREEN_ROW = 3 then
         Clear;
         Write (ITEM => TITLE1,
            FORM => UNDERLINED);
         Cursor_Next_Line;
         Cursor_Set (ROW => 3, COLUMN => 1);
      end if;
      Control_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (COUNTER),
         HIGH_NODE => HIGHER_NODES (COUNTER),
         REAL_VOLT => REAL_VOLTS,
         IMAG_VOLT => IMAG_VOLTS);
      V_MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS,
         IMAG_VALUE => IMAG_VOLTS);
      V_PHASE := Maths.Angle (X_COORDINATE => REAL_VOLTS,
         Y_COORDINATE => IMAG_VOLTS);
      begin
         Inner.Find_Branch (COMPONENT => RUT_INDCT,
            HIGH_NODE      => HIGHER_NODES (COUNTER),
            LOW_NODE       => LOWER_NODES (COUNTER),
            NETLIST_OBJECT => NETLIST_OBJECT);
         HIGH_NODE := 0;
         LOW_NODE  := 0;
         Inner.Read_Netlist (COMPONENT => GR_ADMIT,
            VALUE          => RESISTORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => INDUCT,
            VALUE          => INDUCTORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => CAPS,
            VALUE          => CAPACITORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => REAL_CRNT,
            VALUE          => RL_CRNT,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => IMAG_CRNT,
            VALUE          => IN_CRNT,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
```

```
      Inner.Read_Netlist (COMPONENT => REAL_VOLT,
         VALUE          => RL_VOLT,
         HIGH_NODE1     => HIGHER_NODES (COUNTER),
         LOW_NODE1      => LOWER_NODES (COUNTER),
         HIGH_NODE2     => HIGH_NODE,
         LOW_NODE2      => LOW_NODE,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Inner.Read_Netlist (COMPONENT => IMAG_VOLT,
         VALUE          => IM_VOLT,
         HIGH_NODE1     => HIGHER_NODES (COUNTER),
         LOW_NODE1      => LOWER_NODES (COUNTER),
         HIGH_NODE2     => HIGH_NODE,
         LOW_NODE2      => LOW_NODE,
         NETLIST_OBJECT => NETLIST_OBJECT);
      if (abs (RL_VOLT) + abs (IM_VOLT)) > MINIMUM_VALUE then
         V_MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS
            - RL_VOLT, IMAG_VALUE => IMAG_VOLTS - IM_VOLT);
         V_PHASE := Maths.Angle (X_COORDINATE => REAL_VOLTS - RL_VOLT,
            Y_COORDINATE => IMAG_VOLTS - IM_VOLT);
      end if;
      if INDUCTORS = 0.0 then
         L_SUSEPTANCE := 0.0;
      else
         L_SUSEPTANCE := 1.0 / OMEGA * INDUCTORS;
      end if;
      if (abs (RESISTORS) + abs (CAPACITORS) + L_SUSEPTANCE)
         > MINIMUM_VALUE then
         Z_MAGNITUDE := 1.0 / (Math_Lib.Sqrt (RESISTORS * RESISTORS +
            (OMEGA * CAPACITORS - L_SUSEPTANCE) ** 2));
      else
         Z_MAGNITUDE := 0.0;
      end if;
      Z_PHASE := -Maths.Angle (X_COORDINATE => RESISTORS,
         Y_COORDINATE => OMEGA * CAPACITORS - L_SUSEPTANCE);

      if Z_MAGNITUDE < 0.0 then
         IMAG_CENTS := IM_CENT;
         REAL_CENTS := RL_CENT;
         I_PHASE := Maths.Angle (X_COORDINATE => RL_CENT,
            Y_COORDINATE => IM_CENT);
         I_MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_CENTS,
            IMAG_VALUE => IMAG_CENTS);
      else
         I_MAGNITUDE := V_MAGNITUDE / Z_MAGNITUDE;
         I_PHASE := V_PHASE - Z_PHASE;
         REAL_CENTS := I_MAGNITUDE * Math_Lib.Cos (I_PHASE / RADIAN);
         IMAG_CENTS := I_MAGNITUDE * Math_Lib.Sin (I_PHASE / RADIAN);
      end if;

      if I_MAGNITUDE > MINIMUM_VALUE then
         DECIBELS := Math_Lib.Ln (I_MAGNITUDE) * DECIBEL_FACTOR;
      else
         DECIBELS := MAX_NEGATIVE_DECIBEL;
      end if;
      Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
      Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
      Write (ITEM => REAL_CENTS,
         WHOLE       => 5,
         FRACTION    => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => IMAG_CENTS,
         WHOLE       => 4,
         FRACTION    => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => I_MAGNITUDE,
         WHOLE       => 4,
         FRACTION    => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => I_PHASE,
         WHOLE    => 5,
         FRACTION => 5,
         EXPONENT => 0);
      Write (ITEM => DECIBELS,
         WHOLE    => 5,
         FRACTION => 5,
         EXPONENT => 0);
   exception
      when NODE_PAIR_DOES_NOT_EXIST =>
         Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
         Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
         Write (ITEM => "    No Branch exists");
   end;
   Cursor_Next_Line;
   SCREEN_ROW := SCREEN_ROW + 1;
   if SCREEN_ROW > 24 then
      SCREEN_ROW := 3;
      Status.Wait;
   end if;
   COUNTER := COUNTER + 1;
 end loop;
end Calculate_Current;

-- ------------------------------------------------------------------
-- PURPOSE : Calculates the  power used in the components and displays the
--           results of the calculations on the Main screen.
-- ------------------------------------------------------------------

procedure Calculate_Power (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   COUNTER                          : INTEGER := 1;
   HIGH_NODE,          LOW_NODE     : NODE_TYPE;
   SCREEN_ROW                       : POSITIVE := 3;
   APP_POWER,          CAPACITIVE_POWER,
   CAPACITORS,         IM_VOLT,
   INDUCTIVE_POWER,    INDUCTORS,
   L_SUSEPTANCE,       MAGNITUDE,
   OMEGA,              PHASE,
   POWER_FACTOR,       POWER_FREQUENCY,
   REACTIVE_POWER,     REAL_POWER,
   RESISTORS,          RL_VOLT,
   VALUE                            : REAL_TYPE;

   POWER_LIST : LIST_ARRAY_TYPE (1..2) := (
   (TEXT_LENGTH => 4, TEXT => "Peak", STATE => TRUE),
   (TEXT_LENGTH => 3, TEXT => "RMS",  STATE => FALSE));

begin
   Process_Title (TITLE => "POWER");
   Outer.Get_Frequency (FREQUENCY => POWER_FREQUENCY,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Prompts.Read (LIST => POWER_LIST,
      HELP_INFO => HP_POWER);
   Central_Solver.Solve_AC_Network (MODE => RLC,
      AC_FREQUENCY    => POWER_FREQUENCY,
      CALCULATE       => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   while (COUNTER <= MAX_CALCULATIONS) and
      (HIGHER_NODES (COUNTER) in 1..NODE_MAX) loop
      Central_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (COUNTER),
         HIGH_NODE => HIGHER_NODES (COUNTER),
         REAL_VOLT => RL_VOLT,
         IMAG_VOLT => IM_VOLT);
      if SCREEN_ROW = 3 then
         Clear;
         Write (ITEM => "    Nodes      Real       Reactive      Capacitive"
            & "      Inductive      Apparent    ", FORM => UNDERLINED);
         Cursor_Set (ROW => 14, COLUMN => 1);
         Write (ITEM => "    Nodes     Impedance      Phase      Power " &
            "Factor                ", FORM => UNDERLINED);
         Cursor_Next_Line;
      end if;
      begin
         Inner.Find_Branch (COMPONENT => NOT_INDUCT,
            HIGH_NODE      => HIGHER_NODES (COUNTER),
            LOW_NODE       => LOWER_NODES (COUNTER),
            NETLIST_OBJECT => NETLIST_OBJECT);
         HIGH_NODE := 0;
         LOW_NODE := 0;
         Inner.Read_Netlist (COMPONENT => REAL_VOLT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         RL_VOLT := VALUE - RL_VOLT;
         Inner.Read_Netlist (COMPONENT => IMAG_VOLT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         IM_VOLT := VALUE - IM_VOLT;
         MAGNITUDE := RL_VOLT * RL_VOLT + IM_VOLT * IM_VOLT;
         if POWER_LIST (2).STATE then
            MAGNITUDE := MAGNITUDE / 2.0;
         end if;
         Inner.Read_Netlist (COMPONENT => OR_ADMIT,
            VALUE          => RESISTORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => INDUCT,
            VALUE          => INDUCTORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => CAPS,
            VALUE          => CAPACITORS,
            HIGH_NODE1     => HIGHER_NODES (COUNTER),
            LOW_NODE1      => LOWER_NODES (COUNTER),
            HIGH_NODE2     => HIGH_NODE,
            LOW_NODE2      => LOW_NODE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         REAL_POWER := RESISTORS * MAGNITUDE;
         OMEGA := 2.0 * Math_Lib.PI * POWER_FREQUENCY;
         if INDUCTORS = 0.0 then
            L_SUSEPTANCE := 0.0;
         else
            L_SUSEPTANCE := 1.0 / OMEGA * INDUCTORS;
         end if;
         INDUCTIVE_POWER  := -L_SUSEPTANCE * MAGNITUDE;
         CAPACITIVE_POWER := OMEGA * CAPACITORS * MAGNITUDE;
         REACTIVE_POWER   := INDUCTIVE_POWER + CAPACITIVE_POWER;
         if REAL_POWER = 0.0 then
            APP_POWER := REACTIVE_POWER;
         elsif REACTIVE_POWER = 0.0 then
            APP_POWER := REAL_POWER;
         else
            APP_POWER := Math_Lib.Sqrt (REAL_POWER * REAL_POWER +
               REACTIVE_POWER * REACTIVE_POWER);
         end if;
         if APP_POWER = 0.0 then
            POWER_FACTOR := 0.0;
         else
            POWER_FACTOR := REAL_POWER / APP_POWER;
         end if;
         if (abs (RESISTORS) + abs (CAPACITORS) + L_SUSEPTANCE)
            > MINIMUM_VALUE then
            MAGNITUDE := 1.0 / (Math_Lib.Sqrt (RESISTORS * RESISTORS +
               (OMEGA * CAPACITORS - L_SUSEPTANCE) ** 2));
         else
            MAGNITUDE := 0.0;
         end if;
         PHASE := -Maths.Angle (X_COORDINATE => RESISTORS,
            Y_COORDINATE => OMEGA * CAPACITORS - L_SUSEPTANCE);
         Cursor_Set (ROW => SCREEN_ROW, COLUMN => 1);
         Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
         Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
         Write (ITEM => REAL_POWER,
            WHOLE       => 4,
            FRACTION    => 5,
            ENG_NOTATION => TRUE);
         Write (ITEM => REACTIVE_POWER,
            WHOLE       => 4,
            FRACTION    => 5,
            ENG_NOTATION => TRUE);
         Write (ITEM => CAPACITIVE_POWER,
            WHOLE       => 4,
            FRACTION    => 5,
            ENG_NOTATION => TRUE);
         Write (ITEM => INDUCTIVE_POWER,
            WHOLE       => 4,
            FRACTION    => 5,
```

```
            ENG_NOTATION => TRUE);
         Write (ITEM => ADJ_POWER,
            WHOLE        =>"4,
            FRACTION     => 6,
            ENG_NOTATION => TRUE);
         Cursor_Set (ROW => SCREEN_ROW + 12, COLUMN => 1);
         Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
         Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
         Write (ITEM => MAGNITUDE,
            WHOLE        => 4,
            FRACTION     => 5,
            ENG_NOTATION => TRUE);
         Write (ITEM => PHASE,
            WHOLE    => 5,
            FRACTION => 0,
            EXPONENT => 0);
         Write (ITEM => POWER_FACTOR,
            WHOLE        => 6,
            FRACTION     => 5,
            ENG_NOTATION => TRUE);
      exception
         when NODE_PAIR_DOES_NOT_EXIST =>
            Cursor_Set (ROW => SCREEN_ROW, COLUMN => 1);
            Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
            Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
            Write (ITEM => "    Branch does not exist");
            Cursor_Set (ROW => SCREEN_ROW + 12, COLUMN => 1);
            Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
            Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
            Write (ITEM => "    Branch does not exist");
      end;
      Cursor_Next_Line;
      SCREEN_ROW := SCREEN_ROW + 1;
      if SCREEN_ROW > 12 then
         SCREEN_ROW := 3;
         Status.Wait;
      end if;
      COUNTER := COUNTER + 1;
   end loop;
end Calculate_Power;

-- ------------------------------------------------------------------------
-- PURPOSE : Calculates the voltage of the nodes and displays the results
--           of the calculations on the Main screen.
-- ------------------------------------------------------------------------

procedure Calculate_Voltage (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   COUNTER                          : POSITIVE := 1;
   SCREEN_ROW                       : POSITIVE := 3;
   DECIBEL,              FREQUENCY,
   IMAG_VOLTS,           MAGNITUDE,
   PHASE,                REAL_VOLTS : REAL_TYPE;
   MAGNITUDE_RANGE_ERROR,
   DECIBEL_RANGE_ERROR              : BOOLEAN := FALSE;

begin
   Process_Title (TITLE => "NODE VOLTAGES");
   Enter.Get_Frequency (FREQUENCY => FREQUENCY,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Central_Solver.Solve_AC_Network (NODE => NAX,
      AC_FREQUENCY   => FREQUENCY,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   while (COUNTER < MAX_CALCULATIONS) and
      (HIGHER_NODES (COUNTER) in 1..NODE_MAX) loop
      Central_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (COUNTER),
         HIGH_NODE => HIGHER_NODES (COUNTER),
         REAL_VOLT => REAL_VOLTS,
         IMAG_VOLT => IMAG_VOLTS);
      MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS,
         IMAG_VALUE => IMAG_VOLTS);
      if (MAGNITUDE <= MIN_VOLT_VALUE * MIN_VOLT_VALUE) or
         (MAGNITUDE >= MAX_VOLT_VALUE * MAX_VOLT_VALUE) then
         MAGNITUDE_RANGE_ERROR := TRUE;
      end if;
      if MAGNITUDE > MINIMUM_VALUE then
         DECIBEL := Math_Lib.Ln (X => MAGNITUDE) * DECIBEL_FACTOR;
      else
         DECIBEL_RANGE_ERROR := TRUE;
         DECIBEL := MAX_NEGATIVE_DECIBEL;
      end if;
      PHASE := Maths.Angle (X_COORDINATE => REAL_VOLTS,
         Y_COORDINATE => IMAG_VOLTS);

      if SCREEN_ROW = 3 then
         Clear;
         Write (ITEM => TITLE1,
            FORM => UNDERLINED);
         Cursor_Next_Line;
         Cursor_Set (ROW => 3, COLUMN => 1);
      end if;

      Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
      Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
      Write (ITEM => REAL_VOLTS,
         WHOLE        => 5,
         FRACTION     => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => IMAG_VOLTS,
         WHOLE        => 4,
         FRACTION     => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => MAGNITUDE,
         WHOLE        => 4,
         FRACTION     => 5,
         ENG_NOTATION => TRUE);
      Write (ITEM => PHASE,
         WHOLE    => 5,
         FRACTION => 5,
         EXPONENT => 0);
      Write (ITEM => DECIBEL,
         WHOLE    => 5,
         FRACTION => 5,
         EXPONENT => 0);
      Cursor_Next_Line;
      SCREEN_ROW := SCREEN_ROW + 1;
      if SCREEN_ROW > 24 then
         SCREEN_ROW := 3;
         Status.Wait;
      end if;
      COUNTER := COUNTER + 1;
   end loop;
   if MAGNITUDE_RANGE_ERROR or DECIBEL_RANGE_ERROR then
      Errors.Write (MESSAGE => "The magnitude of the Real and/or " &
         "Imaginary Voltage was outside the range 1E-18 <-> 1E18.");
   end if;
end Calculate_Voltage;

-- ------------------------------------------------------------------------
-- PURPOSE : Calculates  the sensitivity of the  node voltages of  defined
--           node pairs to perturbations in component values.
-- ------------------------------------------------------------------------

procedure Sensitivity (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   type MAGNITUDE_TYPE is array (1..MAX_CALCULATIONS) of REAL_TYPE;

   NORMAL_MAGNITUDE,     NEW_MAGNITUDE     : MAGNITUDE_TYPE;
   COMPONENT                               : ELEMENTS;
   SCREEN_ROW                              : POSITIVE := 3;
   HIGH_NODE1,           LOW_NODE1,
   HIGH_NODE2,           LOW_NODE2         : NODE_TYPE;
   TEMP_VALUE,           VALUE,
   RL_VOLT,              IM_VOLT,
   SENSIT_FREQUENCY,     MAGNITUDE,
   SENSITIVE                               : REAL_TYPE;
   COUNTER                                 : INTEGER_TYPE := 1;

begin
   Process_Title (TITLE => "SENSITIVITY");
   Enter.Get_Frequency (FREQUENCY => SENSIT_FREQUENC...
      NETLIST_OBJECT => NETLIST_OBJECT);
   Central_Solver.Solve_AC_Network (NODE => NAX,
      AC_FREQUENCY   => SENSIT_FREQUENCY,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Prompts.Read (MESSAGE => "Enter the Sensitivity Perturbation",
      ITEM => SENSITIVE);
   Write (ITEM => "  For node pairs :      ");
   while (HIGHER_NODES (COUNTER) in 1..NODE_MAX) and (COUNTER <= 4) loop
      Write (ITEM => HIGHER_NODES (COUNTER), ITEM_WIDTH => 5);
      Write (ITEM => " -");
      Write (ITEM => LOWER_NODES (COUNTER), ITEM_WIDTH => 3);
      Central_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (COUNTER),
         HIGH_NODE => HIGHER_NODES (COUNTER),
         REAL_VOLT => RL_VOLT,
         IMAG_VOLT => IM_VOLT);
      NORMAL_MAGNITUDE (COUNTER) := Math_Lib.Sqrt (RL_VOLT * RL_VOLT +
         IM_VOLT * IM_VOLT);
      COUNTER := COUNTER + 1;
   end loop;
   Cursor_Next_Line;

   COMPONENT_VALUE (GR_ADMIT).DISPLAY := "R   ";
   for COMPONENT in CAPS..COUPLING loop
      case COMPONENT is
         when CAPS | GR_ADMIT | ACT_INDCT | CM_ADMIT | POLE_FREQ |
              COUPLING =>
            Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
            loop
               Inner.Read_Netlist (COMPONENT => COMPONENT,
                  VALUE        => VALUE,
                  HIGH_NODE1   => HIGH_NODE1,
                  LOW_NODE1    => LOW_NODE1,
                  HIGH_NODE2   => HIGH_NODE2,
                  LOW_NODE2    => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               if abs (VALUE) > MINIMUM_VALUE then
                  Cursor_Set (ROW => SCREEN_ROW, COLUMN => 3);
                  Write (ITEM => COMPONENT_VALUE (COMPONENT).DISPLAY);
                  Write (ITEM => HIGH_NODE1, ITEM_WIDTH => 4);
                  Write (ITEM => "-");
                  Write (ITEM => LOW_NODE1, ITEM_WIDTH => 3);
                  if COMPONENT >= GR_ADMIT then
                     Cursor_Next_Line;
                     SCREEN_ROW := SCREEN_ROW + 1;
                     Write (ITEM => HIGH_NODE2, ITEM_WIDTH => 9);
                     Write (ITEM => "-");
                     Write (ITEM => LOW_NODE2, ITEM_WIDTH => 3);
                  end if;
                  if COMPONENT = GR_ADMIT then
                     Write (ITEM => 1.0 / VALUE,
                        WHOLE        => 3,
                        FRACTION     => 5,
                        ENG_NOTATION => TRUE);
                     TEMP_VALUE := VALUE / (1.0 + SENSITIVE / 100.0);
                  else
                     Write (ITEM => VALUE,
                        WHOLE        => 3,
                        FRACTION     => 5,
                        ENG_NOTATION => TRUE);
                     TEMP_VALUE := VALUE * (1.0 + SENSITIVE / 100.0);
                  end if;
                  Inner.Write_Netlist (COMPONENT => COMPONENT,
                     VALUE        => TEMP_VALUE,
                     HIGH_NODE1   => HIGH_NODE1,
                     LOW_NODE1    => LOW_NODE1,
                     HIGH_NODE2   => HIGH_NODE2,
                     LOW_NODE2    => LOW_NODE2,
                     NETLIST_OBJECT => NETLIST_OBJECT);
                  if (COMPONENT = ACT_INDCT) or (COMPONENT = COUPLING) then
                     Inner.Update_Mutuals_and_Coupling (VALUE=>TEMP_VALUE,
                        HIGH_NODE1   => HIGH_NODE1,
                        LOW_NODE1    => LOW_NODE1,
                        HIGH_NODE2   => HIGH_NODE2,
                        LOW_NODE2    => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     Inner.BR_Inductance_Matrix (NETLIST_OBJECT =>
                        NETLIST_OBJECT);
                  end if;
                  Central_Solver.Solve_AC_Network (NODE => NAX,
                     AC_FREQUENCY   => SENSIT_FREQUENCY,
                     CALCULATE      => TRUE,
                     NETLIST_OBJECT => NETLIST_OBJECT);
                  COUNTER := 1;
                  while (HIGHER_NODES (COUNTER) in 1..NODE_MAX)
                     and (COUNTER <= 4) loop
```

```
                    Control_Solver.Get_Voltages (
                       LOW_NODE  => LOWER_NODES (COUNTER),
                       HIGH_NODE => HIGHER_NODES (COUNTER),
                       REAL_VOLT => RL_VOLT,
                       IMAG_VOLT => IM_VOLT);
                    NEW_MAGNITUDE (COUNTER) := Maths.Find_Magnitude (
                       REAL_VALUE => RL_VOLT,
                       IMAG_VALUE => IM_VOLT);
                    MAGNITUDE := ((NEW_MAGNITUDE (COUNTER) -
                       NORMAL_MAGNITUDE (COUNTER)) /
                       NORMAL_MAGNITUDE (COUNTER)) / (SENSITIVE / 100.0);
                    Cursor_Set (ROW => SCREEN_ROW,
                       COLUMN => 29 + (COUNTER - 1) * 13);
                    Write (ITEM => MAGNITUDE,
                       FRACTION     => 5,
                       ENG_NOTATION => TRUE);
                    COUNTER := COUNTER + 1;
                 end loop;

                 Inner.Write_Netlist (COMPONENT => COMPONENT,
                    VALUE       => VALUE,
                    HIGH_NODE1  => HIGH_NODE1,
                    LOW_NODE1   => LOW_NODE1,
                    HIGH_NODE2  => HIGH_NODE2,
                    LOW_NODE2   => LOW_NODE2,
                    NETLIST_OBJECT => NETLIST_OBJECT);
                 if (COMPONENT = ACT_INDCT) or (COMPONENT = COUPLING) then
                    Inner.Update_Matrix_and_Coupling (VALUE => VALUE,
                       HIGH_NODE1  => HIGH_NODE1,
                       LOW_NODE1   => LOW_NODE1,
                       HIGH_NODE2  => HIGH_NODE2,
                       LOW_NODE2   => LOW_NODE2,
                       NETLIST_OBJECT => NETLIST_OBJECT);
                    Inner.DE_Inductance_Matrix (NETLIST_OBJECT =>
                       NETLIST_OBJECT);
                 end if;
                 SCREEN_ROW := SCREEN_ROW + 1;
                 Cursor_Set (ROW => SCREEN_ROW, COLUMN => 2);
                 if SCREEN_ROW >= 24 then
                    SCREEN_ROW := 3;
                    Status.Wait;
                    Clear;
                    Write (ITEM => "  For more pairs :      ");
                    COUNTER := 1;
                    while (HIGHER_N                 <= NODE_MAX)
                       and (COUNT                   UNTER),
                       Write (IT                    ),
                          ITEM =
                       Write (                      ),
                       Write (                      NODE,
                          ITEM =                    ),
                       COUNTER                      + 1;
                    end loop;
                    Cursor_Set (and);
                 end if;
              end if;
              begin
                 Inner.Next_E_Node (NETLIST_OBJECT => NETLIST_OBJECT);
              exception
                 when NO_MORE_CHANGES => exit;
              end;
           end loop;
        when others => null;
     end case;
  end loop;
  Cursor_Next_Line;
  Cursor_Next_Line;
  Cursor_Set (ROW => SCREEN_ROW + 1     LUMN => 2);
  Write (ITEM => "Perturbation ste    size =");
  Write (ITEM => SENSITIVE,
     FRACTION => 3,
     EXPONENT => 0);
  Write (ITEM => " %");
end Sensitivity;

-- PURPOSE : Sets which circuit nodes will be analysed.

procedure Set_Calculation_Nodes (NODE_MAX : in NODE_TYPE) is

   type CALCULATION_TYPE is array (1..MAX_CALCULATIONS) of INTEGER_TYPE;

   CALCULATION_NODES                    : CALCULATION_TYPE;
   COUNTER                              : INTEGER;
   ERROR                                : BOOLEAN;
   NODE                                 : NODE_TYPE := 0;

begin
   Process_Title ("SET NODES");
   loop
      COUNTER := 0;
      CALCULATION_NODES := (1..MAX_CALCULATIONS => 0);
      loop
         COUNTER := COUNTER + 1;
         Prompts.Read (MESSAGE => "Enter Node (1st node > 0), exit = 0",
            HELP_INFO => HP_SET_CALC_NODES,
            ITEM      => CALCULATION_NODES (COUNTER));
         exit when (COUNTER >= MAX_CALCULATIONS - 1) or
            (CALCULATION_NODES (COUNTER) = 0);
      end loop;
      if CALCULATION_NODES (1) > 0 then
         COUNTER := 0;
         ERROR := FALSE;
         loop
            COUNTER := COUNTER + 1;
            if (abs (CALCULATION_NODES (COUNTER)) > NODE_MAX) or
               ((CALCULATION_NODES (COUNTER) < 0) and
               (CALCULATION_NODES (COUNTER + 1) < 0)) then
               Errors.Write (MESSAGE=>"Node Specification is Incorrect.");
               ERROR := TRUE;
            end if;
            exit when (COUNTER >= MAX_CALCULATIONS - 1) or ERROR;
         end loop;
      else
         Errors.Write (MESSAGE => "The 1st Node is less than 1.");
         ERROR := TRUE;
      end if;
      exit when not ERROR;
   end loop;
   HIGHER_NODES := (1..MAX_CALCULATIONS => 0);
   LOWER_NODES  := (1..MAX_CALCULATIONS => 0);
   COUNTER := 0;
   loop
      COUNTER := COUNTER + 1;
      if CALCULATION_NODES (COUNTER) < 0 then
         LOWER_NODES (NODE) := -CALCULATION_NODES (COUNTER);
      elsif CALCULATION_NODES (COUNTER) > 0 then
         NODE := NODE + 1;
         HIGHER_NODES (NODE) := CALCULATION_NODES (COUNTER);
         LOWER_NODES (NODE) := 0;
      end if;
      exit when (CALCULATION_NODES (COUNTER) = 0) or
         (COUNTER >= MAX_CALCULATIONS - 1);
   end loop;
end Set_Calculation_Nodes;

-- PURPOSE : Displays the circuit nodes which will be analysed.

procedure Show_Nodes is

   COUNTER : INTEGER_TYPE := 1;

begin
   Process_Title (TITLE => "NODES to be ANALYSED");
   Cursor_Next_Line;
   while (COUNTER < MAX_CALCULATIONS) and (HIGHER_NODES (COUNTER) > 0)
loop
      Write (ITEM => HIGHER_NODES (COUNTER),
         ITEM_WIDTH => 3);
      Write (ITEM => " -");
      Write (ITEM => LOWER_NODES (COUNTER),
         ITEM_WIDTH => 3);
      COUNTER := COUNTER + 1;
      if (COUNTER rem 7) = 1 then
         Cursor_Next_Line;
      end if;
   end loop;
end Show_Nodes;

package body Plotter is separate;

end Analysers;

with  Graphs,
      Unchecked_Deallocation;

use   Graphs;

separate (Analysers)

package body Plotter is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This holds all the analysers that display their results on
--           the graphic screen.  This they are seperated from the other
--           analysers.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_FOURIER             : constant := 190;
   HP_FREQUENCY_RESPONSE  : constant := 191;
   HP_POLAR               : constant := 192;

-- Types and variables used for Fourier analysis procedure.

   ABS_MAX_HARMONICS : constant := 1000;

   type HARMONIC_ARRAY_TYPE is array (0..ABS_MAX_HARMONICS) of REAL_TYPE;
   type HARMONIC_TYPE is
      record
         MAGNIL,
         MAGNOP,
         PHASEL,
         PHASEOP,
         Y_VALUES : HARMONIC_ARRAY_TYPE;
      end record;

   type HARMONIC_ACCESS_TYPE is access HARMONIC_TYPE;

   HARMONICS : HARMONIC_ACCESS_TYPE;

   FOURIER_LIST : LIST_ARRAY_TYPE (1..7) := (
   (TEXT_LENGTH => 11, TEXT => "Square Wave",          STATE => TRUE),
   (TEXT_LENGTH => 15, TEXT => "Triangular Wave",      STATE => FALSE),
   (TEXT_LENGTH => 9,  TEXT => "Sawtooth",             STATE => FALSE),
   (TEXT_LENGTH => 18, TEXT => "1/2 Rectified Sine",   STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "Full Rectified Sine",  STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "User Specified Wave",  STATE => FALSE),
   (TEXT_LENGTH => 7,  TEXT => "Impulse",              STATE => FALSE));

   FOURIER_FORM : FIELDS_TYPE (1..7) :=
```

```
   (1 =>
      (VALUE_LENGTH   => 4,
       TEXT_LENGTH    => 20,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 1,
       ROW            => 4,
       VALUE          => "1   ",
       TEXT           => "Harmonics (1 - 1000)",
       MAX_INTEGER    => 1000,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 1),
    2 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 3,
       ACTUAL_LENGTH  => 1,
       ROW            => 5,
       VALUE          => "1 ",
       TEXT           => "Higher Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 1),
    3 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 2,
       ACTUAL_LENGTH  => 1,
       ROW            => 6,
       VALUE          => "0 ",
       TEXT           => "Lower  Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 0,
       INTEGER_VALUE  => 0),
    4 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 10,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 6,
       ACTUAL_LENGTH  => 6,
       ROW            => 7,
       VALUE          => "1.0e-3          ",
       TEXT           => "Start Time",
       MAX_REAL       => 1.0e25,
       MIN_REAL       => 0.0,
       REAL_VALUE     => 1.0e-1),
    5 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 8,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 6,
       ROW            => 8,
       VALUE          => "1.1e-3          ",
       TEXT           => "End Time",
       MAX_REAL       => 1.0e25,
       MIN_REAL       => 0.0,
       REAL_VALUE     => 1.1e-1),
    6 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 14,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 2,
       ROW            => 9,
       VALUE          => "10",
       TEXT           => "Plot Increment",
       MAX_INTEGER    => 80,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 10),
    7 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 18,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 6,
       ROW            => 10,
       VALUE          => "1.0e3           ",
       TEXT           => "Impedance Frequency",
       MAX_REAL       => 1.0e25,
       MIN_REAL       => MINIMUM_VALUE,
       REAL_VALUE     => 1000.0));

-----------------------------------------------------------------
-- Types and variables used for Circuit_Response procedure.
-----------------------------------------------------------------

type Y_VALUE_TYPE  is array (0..MX_PIXELS, 1..5) of REAL_TYPE;
type Y_ACCESS_TYPE is access Y_VALUE_TYPE;

Y_VALUES : Y_ACCESS_TYPE;

MT_X_FORM : FIELDS_TYPE (1..7) :=
   (1 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 3,
       ROW            => 4,
       VALUE          => "1.0             ",
       TEXT           => "Minimum Value",
       MAX_REAL       => 1.0E40,
       MIN_REAL       => 0.0,
       REAL_VALUE     => 1.0),
    2 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 4,
       ROW            => 5,
       VALUE          => "10.0            ",
       TEXT           => "Maximum Value",
       MAX_REAL       => 1.0E40,
       MIN_REAL       => 0.0,
       REAL_VALUE     => 10.0),
    3 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 4,
       ACTUAL_LENGTH  => 1,
       ROW            => 6,
       VALUE          => "1 ",
       TEXT           => "Higher Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 1),
    4 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 3,
       ACTUAL_LENGTH  => 1,
       ROW            => 7,
       VALUE          => "0 ",
       TEXT           => "Lower  Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 0,
       INTEGER_VALUE  => 0),
    5 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => FALSE,
       TEST_AGAINST   => 6,
       ACTUAL_LENGTH  => 1,
       ROW            => 8,
       VALUE          => "1 ",
       TEXT           => "Higher Node 2",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 1),
    6 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => FALSE,
       TEST_AGAINST   => 5,
       ACTUAL_LENGTH  => 1,
       ROW            => 9,
       VALUE          => "0 ",
       TEXT           => "Lower  Node 2",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 0,
       INTEGER_VALUE  => 0),
    7 =>
      (VALUE_LENGTH   => 15,
       TEXT_LENGTH    => 19,
       IS_INTEGER_TYPE => FALSE,
       ACCEPTING      => FALSE,
       TEST_AGAINST   => 0,
       ACTUAL_LENGTH  => 5,
       ROW            => 8,
       VALUE          => "1.0e3           ",
       TEXT           => "Impedance Frequency",
       MAX_REAL       => 1.0E40,
       MIN_REAL       => 0.0,
       REAL_VALUE     => 1.0));

-----------------------------------------------------------------
-- Types and Variables used for Polar analysis procedure.
-----------------------------------------------------------------

MAX_POINTS : constant := 2500;

type POLAR_ARRAY  is array (1..MAX_POINTS) of REAL_TYPE;
type POLAR_TYPE is
   record
      REAL,
      IMAG : POLAR_ARRAY;
   end record;

type POLAR_ACCESS is access POLAR_TYPE;

POLAR_VOLT : POLAR_ACCESS;

POLAR_FORM : FIELDS_TYPE (1..5) :=
   (1 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 2,
       ACTUAL_LENGTH  => 1,
       ROW            => 4,
       VALUE          => "1 ",
       TEXT           => "Higher Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 1,
       INTEGER_VALUE  => 1),
    2 =>
      (VALUE_LENGTH   => 2,
       TEXT_LENGTH    => 13,
       IS_INTEGER_TYPE => TRUE,
       ACCEPTING      => TRUE,
       TEST_AGAINST   => 1,
       ACTUAL_LENGTH  => 1,
       ROW            => 5,
       VALUE          => "0 ",
       TEXT           => "Lower  Node 1",
       MAX_INTEGER    => MAX_NODES,
       MIN_INTEGER    => 0,
       INTEGER_VALUE  => 0),
    3 =>
      (VALUE_LENGTH   => 15,
```

```
            TEXT_LENGTH     => 15,
            IS_INTEGER_TYPE => FALSE,
            ACCEPTING       => TRUE,
            TEST_AGAINST    => 0,
            ACTUAL_LENGTH   => 5,
            ROW             => 6,
            VALUE           => "1.0e-3          ",
            TEXT            => "Start Frequency",
            MAX_REAL        => 1.0e25,
            MIN_REAL        => MINIMUM_VALUE,
            REAL_VALUE      => 1.0e-3),
         4 =>
            (VALUE_LENGTH   => 15,
            TEXT_LENGTH     => 15,
            IS_INTEGER_TYPE => FALSE,
            ACCEPTING       => TRUE,
            TEST_AGAINST    => 0,
            ACTUAL_LENGTH   => 5,
            ROW             => 7,
            VALUE           => "1.1e-3          ",
            TEXT            => "End   Frequency",
            MAX_REAL        => 1.0e25,
            MIN_REAL        => MINIMUM_VALUE,
            REAL_VALUE      => 1.1e-3),
         5 =>
            (VALUE_LENGTH   => 4,
            TEXT_LENGTH     => 16,
            IS_INTEGER_TYPE => TRUE,
            ACCEPTING       => TRUE,
            TEST_AGAINST    => 0,
            ACTUAL_LENGTH   => 2,
            ROW             => 8,
            VALUE           => "10  ",
            TEXT            => "Number of points",
            MAX_INTEGER     => MAX_POINTS,
            MIN_INTEGER     => 1,
            INTEGER_VALUE   => 10));

-- ------------------------------------------------------------------
-- PURPOSE : Plots  a steady state,  time domain  graph  of  node voltages
--           against time for a specified node pair.
-- ------------------------------------------------------------------

procedure Fourier5 (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   HIGH_NODE_1,              LOW_NODE_1,
   HIGH_NODE1,               LOW_NODE1,
   HIGH_NODE2,               LOW_NODE2          : NODE_TYPE;
   DISPLAY_LINE                                 : Texts.TEXT_TYPE (80);
   COUNT_FREQ,               COUNTER,
   OPTION,                   OPTION1,
   MAX_COUNT,                MAX_HARMONICS,
   X_INC,                    Y,
   Y_MEM,                    Y_PREV             : INTEGER_TYPE;
   END_TIME,                 FREQUENCY,
   IMAG_VOLTS,               MAGNITUDE,
   MAX_MAGNITUDE,            MIN_MAGNITUDE,
   OMEGA,                    REAL_VOLTS,
   START_TIME,               TIME,
   TIME_STEP,                Y_SCALE            : REAL_TYPE;

   procedure Dispose is new
      Unchecked_Deallocation (HARMONIC_TYPE, HARMONIC_ACCESS_TYPE);

begin
   Process_Title (TITLE => "FOURIER");
   HARMONICS := new HARMONIC_TYPE;
   begin
      HIGH_NODE1 := 1;
      LOW_NODE1  := 0;
      HIGH_NODE2 := 0;
      LOW_NODE2  := 0;
      Inner.Find_Branch (COMPONENT => REAL_VOLT,
         HIGH_NODE       => HIGH_NODE1,
         LOW_NODE        => LOW_NODE1,
         NETLIST_OBJECT => NETLIST_OBJECT);
      loop
         Prompts.Read (FORM => FOURIER_FORM,
            LIST      => FOURIER_LIST,
            EXCLUSIVE => TRUE,
            HELP_INFO => HP_FOURIER);
         MAX_HARMONICS := FOURIER_FORM (1).INTEGER_VALUE;
         HIGH_NODE_1   := FOURIER_FORM (2).INTEGER_VALUE;
         LOW_NODE_1    := FOURIER_FORM (3).INTEGER_VALUE;
         START_TIME    := FOURIER_FORM (4).REAL_VALUE;
         END_TIME      := FOURIER_FORM (5).REAL_VALUE;
         X_INC         := FOURIER_FORM (6).INTEGER_VALUE;
         FREQUENCY     := FOURIER_FORM (7).REAL_VALUE;
         if START_TIME >= END_TIME then
            Errors.Write (MESSAGE => "The start time must be less " &
               "than the End Time.");
         else
            exit;
         end if;
      end loop;
      HARMONICS.MAGNIL (0..MAX_HARMONICS) := (0..MAX_HARMONICS => 0.0);
      HARMONICS.PHASEL (0..MAX_HARMONICS) := (0..MAX_HARMONICS => 0.0);
      for OPTION in 1..7 loop
         if FOURIER_LIST (OPTION).STATE then
            case OPTION is
               when 1 => -- For square wave p-p of 2v 1st pulse centered
                  -- on y-axis
                  for COUNTER in 0..((MAX_HARMONICS / 2) - 1) loop
                     HARMONICS.MAGNIL (2 * COUNTER + 1) := 4.0 /
                        (REAL_TYPE (2 * COUNTER + 1) * Math_Lib.Pi);
                  end loop;
                  for COUNTER in 0..((MAX_HARMONICS / 4) - 1) loop
                     HARMONICS.PHASEL (4 * COUNTER + 3) := -180.0;
                  end loop;

               when 2 => -- For triangle wave 2v p-p 0 at origin
                  -- and at middle of positive slope
                  for COUNTER in 1..(MAX_HARMONICS / 2) loop
                     HARMONICS.MAGNIL (2 * COUNTER - 1) := 8.0 /
                        (REAL_TYPE (2 * COUNTER - 1) * Math_Lib.Pi)**2;
                  end loop;
                  if ((MAX_HARMONICS / 4) = 0) then
                     MAX_COUNT := MAX_HARMONICS / 4;
                  else
                     MAX_COUNT := MAX_HARMONICS / 4 + 1;
                  end if;
                  for COUNTER in 1..MAX_COUNT loop
                     HARMONICS.PHASEL (4 * COUNTER - 3) := 90.0;
                  end loop;
                  COUNTER := 1;
                  while ((4 * COUNTER - 1) <= MAX_HARMONICS) loop
                     HARMONICS.PHASEL (4 * COUNTER - 1) := -90.0;
                     COUNTER := COUNTER + 1;
                  end loop;

               when 3 => -- Sawtooth 1v p-p always positive op=0 at origin
                  HARMONICS.MAGNIL (0) := 0.5;
                  for COUNTER in 1..(MAX_HARMONICS / 2) loop
                     HARMONICS.MAGNIL (2 * COUNTER) :=
                        1.0 / (REAL_TYPE (COUNTER) * Math_Lib.Pi);
                     HARMONICS.PHASEL (2 * COUNTER) := 90.0;
                  end loop;

               when 4 => -- Half wave rectified sine 1v p-p
                  -- always +ve op=0 at origin
                  HARMONICS.MAGNIL (0) := 1.0 / Math_Lib.Pi;
                  HARMONICS.MAGNIL (1) := 0.5;
                  HARMONICS.PHASEL (1) := 90.0;
                  for COUNTER in 1..(MAX_HARMONICS / 2) loop
                     HARMONICS.MAGNIL (2 * COUNTER) := 2.0 /
                        (REAL_TYPE (4*COUNTER*COUNTER-1) * Math_Lib.Pi);
                     HARMONICS.PHASEL (2 * COUNTER) := 180.0;
                  end loop;

               when 5 => -- Full wave sine
                  HARMONICS.MAGNIL (0) := 2.0 / Math_Lib.Pi;
                  for COUNTER in 1..MAX_HARMONICS loop
                     HARMONICS.MAGNIL (COUNTER) := 4.0 /
                        (REAL_TYPE (4*COUNTER*COUNTER-1) * Math_Lib.Pi);
                     HARMONICS.PHASEL (2 * COUNTER) := 180.0;
                  end loop;

               when 6 =>
                  for COUNTER in 0..MAX_HARMONICS loop
                     Texts.Reset (TEXT_LINE => DISPLAY_LINE);
                     Texts.Combine(
                        ITEM      => "Enter the magnitude of Harmonic :",
                        TEXT_LINE => DISPLAY_LINE);
                     Texts.Combine (ITEM => COUNTER,
                        ITEM_WIDTH => 4,
                        TEXT_LINE  => DISPLAY_LINE);
                     Prompts.Read (MESSAGE =>
                        DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
                        ITEM => HARMONICS.MAGNIL (COUNTER));
                     Prompts.Read (MESSAGE => "Enter Phase.",
                        ITEM => HARMONICS.PHASEL (COUNTER));
                  end loop;

               when 7 =>
                  HARMONICS.MAGNIL (0..MAX_HARMONICS)
                     := (0..MAX_HARMONICS => 1.0);
                  HARMONICS.PHASEL (0..MAX_HARMONICS)
                     := (0..MAX_HARMONICS => 0.0);
               end case;
               OPTION1 := OPTION;
               exit;
            end if;
         end loop;

         Main.Cursor_Set (ROW => 8, COLUMN => 33);
         Main.Write (ITEM => "Harmonic calculated.");
         begin
            for COUNTER in 0..MAX_HARMONICS loop
               Main.Cursor_Set (ROW => 8, COLUMN => 26);
               Main.Write (ITEM => COUNTER);
               REAL_VOLTS := HARMONICS.MAGNIL (COUNTER) *
                  Math_Lib.Cos (HARMONICS.PHASEL (COUNTER) / RADIAN);
               IMAG_VOLTS := HARMONICS.MAGNIL (COUNTER) *
                  Math_Lib.Sin (HARMONICS.PHASEL (COUNTER) / RADIAN);
               Inner.Write_Netlist (COMPONENT => REAL_VOLT,
                  VALUE          => REAL_VOLTS,
                  HIGH_NODE1     => HIGH_NODE1,
                  LOW_NODE1      => LOW_NODE1,
                  HIGH_NODE2     => HIGH_NODE2,
                  LOW_NODE2      => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Inner.Write_Netlist (COMPONENT => IMAG_VOLT,
                  VALUE          => IMAG_VOLTS,
                  HIGH_NODE1     => HIGH_NODE1,
                  LOW_NODE1      => LOW_NODE1,
                  HIGH_NODE2     => HIGH_NODE2,
                  LOW_NODE2      => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Control_Solver.Solve_AC_Network (NODE => RLC,
                  AC_FREQUENCY   => REAL_TYPE (COUNTER) * FREQUENCY,
                  CALCULATE      => TRUE,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Control_Solver.Get_Voltages (LOW_NODE => LOW_NODE_1,
                  HIGH_NODE => HIGH_NODE_1,
                  REAL_VOLT => REAL_VOLTS,
                  IMAG_VOLT => IMAG_VOLTS);
               HARMONICS.MAGNOP (COUNTER) := Maths.Find_Magnitude (
                  REAL_VALUE => REAL_VOLTS,
                  IMAG_VALUE => IMAG_VOLTS);
               HARMONICS.PHASEOP (COUNTER) := Maths.Angle (
                  X_COORDINATE => REAL_VOLTS,
                  Y_COORDINATE => IMAG_VOLTS);
            end loop;
         exception
            when STORAGE_ERROR =>
               Dispose (HARMONICS);
               raise STORAGE_ERROR;
         end;

         Process_Title (TITLE => "FOURIER");
         Y_MIN := MAX_Y_COORD - 3 * Y_AXIS / 2;
         OMEGA := 2.0 * Math_Lib.Pi * FREQUENCY;
         TIME_STEP := (END_TIME - START_TIME) / REAL_TYPE (MX_PIXELS);
         COUNT_FREQ := 0;
         TIME := START_TIME;
         Main.Cursor_Set (ROW => 8, COLUMN => 27);
         Main.Write (ITEM => "Time calculated.");
         while TIME < END_TIME loop
            Main.Cursor_Set (ROW => 8, COLUMN => 24);
```

```
      Main.Write (ITEM => TIME,
         FRACTION => 5);
      MAGNITUDE := 0.0;
      for COUNTER in reverse 0..MAX_HARMONICS loop
         if HARMONICS.MAGNOP (COUNTER) /= 0.0 then
            MAGNITUDE := MAGNITUDE + HARMONICS.MAGNOP (COUNTER) *
               Math_Lib.Cos (REAL_TYPE (COUNTER) * OMEGA * TIME +
               HARMONICS.PHASEOP (COUNTER) / RADIAN);
         end if;
      end loop;
      HARMONICS.Y_VALUES (COUNT_FREQ) := MAGNITUDE;
      if COUNT_FREQ = 0 then
         MAX_MAGNITUDE := MAGNITUDE;
         MIN_MAGNITUDE := MAGNITUDE;
      else
         if MAGNITUDE > MAX_MAGNITUDE then
            MAX_MAGNITUDE := MAGNITUDE;
         elsif MAGNITUDE < MIN_MAGNITUDE then
            MIN_MAGNITUDE := MAGNITUDE;
         end if;
      end if;
      COUNT_FREQ := COUNT_FREQ + X_INC;
      TIME := TIME + TIME_STEP * REAL_TYPE (X_INC);
   end loop;
   if MIN_MAGNITUDE = MAX_MAGNITUDE then
      MAX_MAGNITUDE := abs (1.5 * MAX_MAGNITUDE);
   end if;
      Change_Screen (MODE_OF_SCREEN => GRAPHIC);
   Draw_X_Axis;
   Draw_Y_Axis;
   Label_Linear_X (X_MINIMUM => START_TIME,
      X_MAXIMUM => END_TIME);
   String_To_Graphic (MESSAGE => "Sec",
      ROW     => (Y_MIN + Y_AXIS) / ROW_FACTOR + 1,
      COLUMN  => (X_MAX + X_AXIS) / COLUMN_FACTOR + 7,
      COLOURS => WHITE);
   String_To_Graphic (MESSAGE => "Volt",
      ROW     => 1,
      COLUMN  => 3,
      COLOURS => WHITE);
   String_To_Graphic (
      MESSAGE =>"Fourier " & FOURIER_LIST (OPTION1).TEXT & " Response",
      ROW     => 1,
      COLUMN  => 15,
      COLOURS => WHITE);
   Texts.Reset (TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => "Using",
      TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => MAX_HARMONICS,
      ITEM_WIDTH => 5,
      TEXT_LINE  => DISPLAY_LINE);
   Texts.Combine (ITEM => " harmonics for",
      TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => HIGH_NODE_1,
      ITEM_WIDTH => 3,
      TEXT_LINE  => DISPLAY_LINE);
   Texts.Combine (ITEM => " -",
      TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => LOW_NODE_1,
      ITEM_WIDTH => 3,
      TEXT_LINE  => DISPLAY_LINE);
   String_To_Graphic (
      MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
      ROW     => 25,
      COLUMN  => 20,
      COLOURS => WHITE);
   Plx_Scale (Y_MINIMUM => MIN_MAGNITUDE,
      Y_MAXIMUM => MAX_MAGNITUDE);
   Y_SCALE := REAL_TYPE (NY_PIXELS) / (MAX_MAGNITUDE - MIN_MAGNITUDE);
   Label_Y (Y_MINIMUM => MIN_MAGNITUDE,
      Y_MAXIMUM => MAX_MAGNITUDE);
   Zero_Line (Y_MINIMUM => MIN_MAGNITUDE,
      Y_MAXIMUM => MAX_MAGNITUDE,
      Y_SCALE   => Y_SCALE);

   COUNT_FREQ := 0;
   TIME := START_TIME;
   Set_Plot_Colour (COLOUR => WHITE);
   while (TIME < END_TIME) loop
      MAGNITUDE := HARMONICS.Y_VALUES (COUNT_FREQ);
      if MIN_MAGNITUDE < 0.0 then
         Y := Y_MIN - Y_AXIS - INTEGER_TYPE (Y_SCALE * MAGNITUDE);
      else
         Y := Y_MIN - Y_AXIS -
            INTEGER_TYPE (Y_SCALE * (MAGNITUDE - MIN_MAGNITUDE));
      end if;
      Y_NEW := Y;
      if COUNT_FREQ > 0 then
         Line (X1 => X_MIN + X_AXIS + COUNT_FREQ - X_INC,
            Y1 => Y_PREV,
            X2 => X_MIN + X_AXIS + COUNT_FREQ,
            Y2 => Y_NEW);
      end if;
      COUNT_FREQ := COUNT_FREQ + X_INC;
      TIME := TIME + TIME_STEP * REAL_TYPE (X_INC);
      Y_PREV := Y_NEW;
   end loop;
   Continue_Processing;
   Change_Screen (MODE_OF_SCREEN => TEXT);
   Clear;

   -- Restore original V values
   REAL_VOLTS := 1.0;
   IMAG_VOLTS := 0.0;
   Inner.Write_Netlist (COMPONENT => REAL_VOLT,
      VALUE       => REAL_VOLTS,
      HIGH_NODE1  => HIGH_NODE1,
      LOW_NODE1   => LOW_NODE1,
      HIGH_NODE2  => HIGH_NODE2,
      LOW_NODE2   => LOW_NODE2,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Inner.Write_Netlist (COMPONENT => IMAG_VOLT,
      VALUE       => IMAG_VOLTS,
      HIGH_NODE1  => HIGH_NODE1,
      LOW_NODE1   => LOW_NODE1,
      HIGH_NODE2  => HIGH_NODE2,
      LOW_NODE2   => LOW_NODE2,
      NETLIST_OBJECT => NETLIST_OBJECT);
exception
   when NODE_PAIR_DOES_NOT_EXIST =>
      Errors.Write (MESSAGE => "Cannot find the independant voltage " &
         "source.");
   when USER_ESCAPE => null;
end;
Dispose (HARMONICS);
end Fouriers;
```

```
-- PURPOSE : Determines and plots the circuit response of the circuit.
--           There are five different options :
--             (1) Plotting frequency response on a linear axis.
--             (2) Plotting frequency response on a logarithmic axis.
--             (3) Plotting parametric response on a linear axis when a
--                 component value is varied.
--             (4) Plotting parametric response on a logarithmic axis when
--                 a component value is varied.
--             (5) Plotting node response on a linear axis when frequency
--                 and the component value is varied.
--           The nodes that are displayed are assumed to be already
--           defined.
-----------------------------------------------------------------------------

procedure Circuit_Response (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   type PLOT_COLOUR_TYPE is array (1..5) of FOREGROUND_TYPE;
   type Y_ARRAY_TYPE     is array (1..5) of INTEGER_TYPE;

   PLOT_CHARACTER   : constant STRING := "*o+#@";
   PLOT_COLOUR      : constant PLOT_COLOUR_TYPE := (LIGHT_RED, GREEN,
      LIGHT_PURPLE, LIGHT_GREEN, LIGHT_GRAY);
   CONTINUE_MESSAGE : constant STRING :=
      "Press any key to abort the graph being plotted.";

   NX_20 : constant := NX_PIXELS / 20;

   Y_MAX_AUTO,          Y_MIN_AUTO          : REAL_TYPE;
   CLEAR_COLUMN,        CLEAR_ROW,
   OPTION                                   : INTEGER_TYPE;

   PLOT_LIST : LIST_ARRAY_TYPE (1..5) := (
   (TEXT_LENGTH => 16, TEXT => "Linear Frequency",    STATE => TRUE),
   (TEXT_LENGTH => 13, TEXT => "Log Frequency",       STATE => FALSE),
   (TEXT_LENGTH => 16, TEXT => "Linear Component",    STATE => FALSE),
   (TEXT_LENGTH => 13, TEXT => "Log Component",       STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "Linear 3D Component", STATE => FALSE));

   procedure Dispose is new
      Unchecked_Deallocation (Y_VALUE_TYPE, Y_ACCESS_TYPE);

-----------------------------------------------------------------------------
-- PURPOSE : Gets a range of values by which the chosen component will be
--           varied for use in the parametric graphs.
-----------------------------------------------------------------------------

procedure Get_X_Component (FREQUENCY, X_MINIMUM, X_MAXIMUM :out REAL_TYPE)
   COMPONENT : out ELEMENT_TYPE;
   VALUE     : out REAL_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : out NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   MAX_DECADES : constant := 1.0E9;

   COMPONENT2,          COMPONENT3          : ELEMENT_TYPE;
   VALUE2                                   : REAL_TYPE;
   HIGH_NODE_1,         LOW_NODE_1,
   HIGH_NODE_2,         LOW_NODE_2          : NODE_TYPE;
   X_MAXIMUM2,          X_MINIMUM2          : REAL_TYPE;
   NUM_DECADES                              : REAL_TYPE;
   FOUND,               CORRECT_VALUES      : BOOLEAN;

begin
   loop
      CORRECT_VALUES := TRUE;
      Screen_Supporter.Get_Component (COMPONENT => COMPONENT3);
      case COMPONENT3 is
         when RESISTOR    => COMPONENT2 := CR_ADMIT;
         when CURR_SOURCE => COMPONENT2 := REAL_CURT;
         when VOLT_SOURCE => COMPONENT2 := REAL_VOLT;
         when others      => COMPONENT2 := COMPONENT3;
      end case;
      GET_X_FORM (1).MAX_REAL := COMPONENT_VALUE (COMPONENT3).MAX;
      GET_X_FORM (1).MIN_REAL := COMPONENT_VALUE (COMPONENT3).MIN;
      GET_X_FORM (2).MAX_REAL := COMPONENT_VALUE (COMPONENT3).MAX;
      GET_X_FORM (2).MIN_REAL := COMPONENT_VALUE (COMPONENT3).MIN;
      if COMPONENT3 >= CR_ADMIT then
         GET_X_FORM (5).ACCEPTING := TRUE;
         GET_X_FORM (6).ACCEPTING := TRUE;
         GET_X_FORM (7).ROW := 10;
      else
         GET_X_FORM (5).ACCEPTING := FALSE;
         GET_X_FORM (6).ACCEPTING := FALSE;
         GET_X_FORM (7).ROW := 8;
      end if;
      Prompts.Read (FORM => GET_X_FORM);
      X_MINIMUM2  := GET_X_FORM (1).REAL_VALUE;
      X_MAXIMUM2  := GET_X_FORM (2).REAL_VALUE;
      HIGH_NODE_1 := GET_X_FORM (3).INTEGER_VALUE;
      LOW_NODE_1  := GET_X_FORM (4).INTEGER_VALUE;
      HIGH_NODE_2 := GET_X_FORM (5).INTEGER_VALUE;
      LOW_NODE_2  := GET_X_FORM (6).INTEGER_VALUE;
      FREQUENCY   := GET_X_FORM (7).REAL_VALUE;

      if COMPONENT2 in CR_ADMIT..POLE_FREQ then
         Inner.Find_Dependent_Source (HIGH_NODE1 => HIGH_NODE_1,
            LOW_NODE1   => LOW_NODE_1,
            HIGH_NODE2  => HIGH_NODE_2,
            LOW_NODE2   => LOW_NODE_2,
            CREATE      => FALSE,
            FOUND_SOURCE => FOUND,
            NETLIST_OBJECT => NETLIST_OBJECT);
      else
         begin
            Inner.Find_Branch (COMPONENT => NOT_INDCT,
               HIGH_NODE   => HIGH_NODE_1,
               LOW_NODE    => LOW_NODE_1,
```

```
            NETLIST_OBJECT => NETLIST_OBJECT);
         FOUND := TRUE;
      exception
         when NODE_PAIR_DOES_NOT_EXIST => FOUND := FALSE;
      end;
   end if;
   if FOUND then
      Inner.Read_Netlist (COMPONENT => COMPONENT2,
         VALUE       => VALUE2,
         HIGH_NODE1  => HIGH_NODE_1,
         LOW_NODE1   => LOW_NODE_1,
         HIGH_NODE2  => HIGH_NODE_2,
         LOW_NODE2   => LOW_NODE_2,
         NETLIST_OBJECT => NETLIST_OBJECT);
   end if;
   if (VALUE2 < MINIMUM_VALUE) or not FOUND then
      Errors.Write (MESSAGE => "Such a component does not exist.");
      CORRECT_VALUES := FALSE;
   end if;
   if (abs (X_MAXIMUM2 - X_MINIMUM2) <= MINIMUM_VALUE) or
      (X_MAXIMUM2 <= X_MINIMUM2) then
      Errors.Write (MESSAGE => "The minimum and maximum values for the"
         & " horizontal axis are not acceptable.");
      CORRECT_VALUES := FALSE;
   end if;
   NUM_DECADES := X_MAXIMUM2 / X_MINIMUM2;
   if NUM_DECADES > MAX_DECADES then
      Errors.Write (MESSAGE => "The component range is too wide.");
      CORRECT_VALUES := FALSE;
   end if;
   exit when CORRECT_VALUES;
end loop;
if COMPONENT3 = RESISTOR then
   COMPONENT2 := RESISTOR;
end if;
COMPONENT  := COMPONENT2;
VALUE      := VALUE2;
HIGH_NODE1 := HIGH_NODE_1;
LOW_NODE1  := LOW_NODE_1;
HIGH_NODE2 := HIGH_NODE_2;
LOW_NODE2  := LOW_NODE_2;
X_MAXIMUM  := X_MAXIMUM2;
X_MINIMUM  := X_MINIMUM2;
end Get_X_Component;
```

```
-- PURPOSE : Draws the X and Y axis, displays the modes that are involved,
--           and labels the graph.
------------------------------------------------------------------------

procedure Graph_Set_Up (OPTION : in INTEGER_TYPE) is

   DISPLAY_LINE  : Texts.TEXT_TYPE;
   MODE_COUNT2   : INTEGER_TYPE := 1;

begin
   Change_Screen (MODE_OF_SCREEN => GRAPHIC);
   Draw_X_Axis;
   Draw_Y_Axis;
   Set_Plot_Colour (COLOUR => WHITE);
   String_To_Graphic (MESSAGE => "Mode Pair Key :",
      ROW    => 25,
      COLUMN => 10,
      COLOURS => YELLOW);
   if PLOT_DIST (5).STATE then
      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => HIGHER_MODES (1),
         ITEM_WIDTH => 3,
         TEXT_LINE  => DISPLAY_LINE);
      Texts.Combine (ITEM => "-",
         TEXT_LINE  => DISPLAY_LINE);
      Texts.Combine (ITEM => LOWER_MODES (1),
         ITEM_WIDTH => 3,
         TEXT_LINE  => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => 25,
         COLUMN  => 20 + 9 * MODE_COUNT2,
         COLOURS => PLOT_COLOUR (MODE_COUNT2));
   else
      while (MODE_COUNT2 <= 5) and
         (HIGHER_MODES (MODE_COUNT2) in 1..MODE_MAX) loop
         Texts.Reset (TEXT_LINE => DISPLAY_LINE);
         Texts.Combine (ITEM => HIGHER_MODES (MODE_COUNT2),
            ITEM_WIDTH => 2,
            TEXT_LINE  => DISPLAY_LINE);
         Texts.Combine (ITEM => "-",
            TEXT_LINE  => DISPLAY_LINE);
         Texts.Combine (ITEM => LOWER_MODES (MODE_COUNT2),
            ITEM_WIDTH => 3,
            TEXT_LINE  => DISPLAY_LINE);
         String_To_Graphic (
            MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
            ROW     => 25,
            COLUMN  => 20 + 9 * MODE_COUNT2,
            COLOURS => PLOT_COLOUR (MODE_COUNT2));
         MODE_COUNT2 := MODE_COUNT2 + 1;
      end loop;
   end if;
   case OPTION is
      when 1 =>
         String_To_Graphic (MESSAGE => "DB",
            ROW     => 1,
            COLUMN  => 4,
            COLOURS => WHITE);
         String_To_Graphic (MESSAGE => "MODE PLOT",
            ROW     => 1,
            COLUMN  => 37,
            COLOURS => WHITE);
      when 2 =>
         String_To_Graphic (MESSAGE => "Deg",
            ROW     => 1,
            COLUMN  => 4,
            COLOURS => WHITE);
         String_To_Graphic (MESSAGE => "PHASE PLOT",
            ROW     => 1,
            COLUMN  => 37,
            COLOURS => WHITE);
      when 3 =>
         String_To_Graphic (MESSAGE => "Volt",
            ROW     => 1,
            COLUMN  => 4,
            COLOURS => WHITE);
         String_To_Graphic (MESSAGE => "MAGNITUDE PLOT",
            ROW     => 1,
            COLUMN  => 37,
            COLOURS => WHITE);
      when others => null;
   end case;
end Graph_Set_Up;
```

```
-- PURPOSE : Solves the AC network at a particular frequency and then
--           stores up to five points.
------------------------------------------------------------------------

procedure Generate_Point (FREQUENCY : in REAL_TYPE;
   OPTION, COUNTER : in INTEGER_TYPE) is

   NODE_COUNT                          : INTEGER_TYPE := 1;
   REAL_VOLTS,          IMAG_VOLTS,
   MAGNITUDE                           : REAL_TYPE;

begin
   Central_Solver.Solve_AC_Network (NODE => NLC,
      AC_FREQUENCY   => FREQUENCY,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   while (NODE_COUNT <= 5) and
      (HIGHER_MODES (NODE_COUNT) in 1..NODE_MAX) loop
      Central_Solver.Get_Voltages (LOW_NODE => LOWER_MODES (NODE_COUNT),
         HIGH_NODE => HIGHER_MODES (NODE_COUNT),
         REAL_VOLT => REAL_VOLTS,
         IMAG_VOLT => IMAG_VOLTS);
      case OPTION is
         when 1 =>
            MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS,
               IMAG_VALUE => IMAG_VOLTS);
            if MAGNITUDE >= MINIMUM_VALUE then
               MAGNITUDE := LN_20 * Math_Lib.Ln (MAGNITUDE);
            else
               MAGNITUDE := -999.0;
            end if;
         when 2 =>
            MAGNITUDE := Maths.Angle (X_COORDINATE => REAL_VOLTS,
               Y_COORDINATE => IMAG_VOLTS);
         when 3 =>
            MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS,
               IMAG_VALUE => IMAG_VOLTS);
         when others => null;
      end case;
      Y_VALUES (COUNTER, MODE_COUNT) := MAGNITUDE;
      if (COUNTER = 0) and (MODE_COUNT = 1) then
         Y_MIN_AUTO := MAGNITUDE;
         Y_MAX_AUTO := Y_MIN_AUTO;
      end if;
      if (MAGNITUDE < Y_MIN_AUTO) then
         Y_MIN_AUTO := MAGNITUDE;
      elsif (MAGNITUDE > Y_MAX_AUTO) then
         Y_MAX_AUTO := MAGNITUDE;
      end if;
      MODE_COUNT := MODE_COUNT + 1;
   end loop;
end Generate_Point;
```

```
-- PURPOSE : Plots up to five points as well as different symbols at
--           certain intervals.
------------------------------------------------------------------------

procedure Plot_Point (Y_SCALE, Y_MINIMUM : in REAL_TYPE;
   COUNTER, X_INC : in INTEGER_TYPE;
   Y_NEW, Y_PREV  : in out Y_ARRAY_TYPE) is

   MODE_COUNT                          : INTEGER_TYPE := 1;
   PLOT_COUNT,          Y              : INTEGER_TYPE;
   MAGNITUDE                           : REAL_TYPE;

begin
   while (MODE_COUNT <= 5) and (HIGHER_MODES (MODE_COUNT) in 1..MODE_MAX)
loop
      MAGNITUDE := Y_VALUES (COUNTER, MODE_COUNT);
      if Y_MINIMUM < 0.0 then
         Y := Y_MIN - Y_AXIS - INTEGER_TYPE (Y_SCALE * MAGNITUDE);
      else
         Y := Y_MIN - Y_AXIS -
            INTEGER_TYPE (Y_SCALE * (MAGNITUDE - Y_MINIMUM));
      end if;
      Y_NEW (MODE_COUNT) := Y;
      if COUNTER > 0 then
         Set_Plot_Colour (COLOUR => PLOT_COLOUR (MODE_COUNT));
         Line (X1 => X_MIN + X_AXIS + COUNTER - X_INC,
            Y1 => Y_PREV (MODE_COUNT),
            X2 => X_MIN + X_AXIS + COUNTER,
            Y2 => Y_NEW (MODE_COUNT));
      end if;
      for CHAR_COUNT in COUNTER..(COUNTER + X_INC) loop
         if ((CHAR_COUNT rem MX_10) = 0) and (CHAR_COUNT /= 0) and
            (COUNTER <= MX_PIXELS) then
            PLOT_COUNT := (CHAR_COUNT / MX_10) rem 5 + 1;
            if (PLOT_COUNT = MODE_COUNT) and
               (Y in Y_AXIS..(MAX_Y_COORD - 5 * Y_AXIS / 2)) then
               Character_To_Graphic (MESSAGE =>PLOT_CHARACTER(PLOT_COUNT),
                  ROW     => Y / ROW_FACTOR + 1,
                  COLUMN  => (X_MIN + X_AXIS + CHAR_COUNT)/COLUMN_FACTOR,
                  COLOURS => PLOT_COLOUR (PLOT_COUNT));
            end if;
         end if;
      end loop;
      MODE_COUNT := MODE_COUNT + 1;
   end loop;
end Plot_Point;
```

```
-- PURPOSE : Plots the frequency response on a linear or logarithmic axis.
------------------------------------------------------------------------

procedure Plot (PARAMETRIC, LINEAR : in BOOLEAN) is

   DELTAS : constant := 1.0E-15;
```

```
   HIGH_NODE1,              LOW_NODE1,
   HIGH_NODE2,              LOW_NODE2          : NODE_TYPE;
   COMPONENT,               COMPONENT2         : ELEMENT_TYPE;
   X_LABEL                                     : STRING (1..4);
   COUNTER,                 NODE_COUNT,
   Y                                           : INTEGER_TYPE;
   COMPONENT_SAVE,          FREQ_POINT,
   POINTS,                  POINT_INC,
   VALUE,                   X_MAX_POINT,
   X_MIN_POINT,             Y_MAX_POINT,
   Y_MIN_POINT,             Y_SCALE            : REAL_TYPE;
   X_INC,                   OPTION             : INTEGER_TYPE;
   Y_NEW,                   Y_PREVIOUS         : Y_ARRAY_TYPE;
   HALTING                                     : BOOLEAN;

begin
   if PARAMETRIC then
      GET_X_FORM (7).ACCEPTING := TRUE;
      Get_X_Component (FREQUENCY => FREQ_POINT,
         X_MINIMUM      => X_MIN_POINT,
         X_MAXIMUM      => X_MAX_POINT,
         COMPONENT      => COMPONENT,
         VALUE          => COMPONENT_SAVE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      if COMPONENT = RESISTOR then
         COMPONENT2 := GR_ADMIT;
         VALUE := 1.0 / X_MIN_POINT;
      else
         COMPONENT2 := COMPONENT;
         VALUE := X_MIN_POINT;
      end if;
      Inner.Write_Netlist (COMPONENT => COMPONENT2,
         VALUE          => VALUE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      X_LABEL := COMPONENT_VALUE (COMPONENT).DISPLAY;
   else
      Get_X_Frequency (MIN_FREQUENCY => X_MIN_POINT,
         MAX_FREQUENCY => X_MAX_POINT);
      FREQ_POINT := X_MIN_POINT;
      X_LABEL := " Hz ";
   end if;
   Get_Vertical_Inc (VERTICAL_MIN => Y_MIN_POINT,
      VERTICAL_MAX => Y_MAX_POINT,
      Y_INCR       => Y_INC,
      OPTION       => OPTION);
   if LINEAR then
      POINT_INC := (X_MAX_POINT - X_MIN_POINT) / REAL_TYPE (NX_PIXELS);
   else
      POINT_INC := (Math_Lib.Ln (X_MAX_POINT) - Math_Lib.Ln (X_MIN_POINT))
         / REAL_TYPE (NX_PIXELS);
   end if;
   COUNTER := 0;
   POINTS := X_MIN_POINT;
   Cursor_Set (ROW => 8, COLUMN => 25);
   Write (ITEM => "Start Value   : ");
   Write (ITEM => X_MIN_POINT,
      FRACTION => 5);
   Cursor_Next_Line;
   Cursor_Set (ROW => 9, COLUMN => 25);
   Write (ITEM => "End Value     : ");
   Write (ITEM => X_MAX_POINT,
      FRACTION => 5);
   Cursor_Next_Line;
   Cursor_Set (ROW => 11, COLUMN => 25);
   Write (ITEM => "Present Value :");
   Cursor_Next_Line;
   Cursor_Set (ROW => 15, COLUMN => 15);
   Write (ITEM => CONTINUE_MESSAGE);
   loop
      Cursor_Set (ROW => 11, COLUMN => 41);
      Write (ITEM => POINTS,
         FRACTION => 5);
      Cursor_Next_Line;
      Generate_Point (FREQUENCY => FREQ_POINT,
         OPTION  => OPTION,
         COUNTER => COUNTER);
      COUNTER := COUNTER + X_INC;
      if LINEAR then
         POINTS := POINTS + POINT_INC * REAL_TYPE (X_INC);
      else
         POINTS := X_MIN_POINT * Math_Lib.Exp (POINT_INC
            * REAL_TYPE (COUNTER));
      end if;
      if PARAMETRIC then
         if COMPONENT = RESISTOR then
            VALUE := 1.0 / POINTS;
         else
            VALUE := POINTS;
         end if;
         Inner.Write_Netlist (COMPONENT => COMPONENT2,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
      else
         FREQ_POINT := POINTS;
      end if;
      Y_PREVIOUS := Y_NEW;
      HALTING := Halt_Processing;
      exit when (POINTS > X_MAX_POINT) or HALTING;
   end loop;
   if AUTO_SCALE then
      Y_MIN_POINT := Y_MIN_AUTO;
      Y_MAX_POINT := Y_MAX_AUTO;
      Fit_Scale (Y_MINIMUM => Y_MIN_POINT, Y_MAXIMUM => Y_MAX_POINT);
   end if;
   if abs (Y_MAX_POINT - Y_MIN_POINT) > DELTAS then
      Y_SCALE := REAL_TYPE (NY_PIXELS) / (Y_MAX_POINT - Y_MIN_POINT);
   else
      Y_SCALE := DELTAS;
   end if;
   if (abs (Y_SCALE * Y_MAX_POINT) > MAX_INTEGER) then
      Y_SCALE := abs (MAX_INTEGER / Y_MAX_POINT);
      Y_MAX_POINT := REAL_TYPE (NY_PIXELS) / Y_SCALE + Y_MIN_POINT;
   end if;
   Graph_Set_Up (OPTION => OPTION);
   String_To_Graphic (MESSAGE => X_LABEL (1..4),
      ROW    => (Y_MIN - Y_AXIS) / ROW_FACTOR + 1,
      COLUMN => (X_MAX - X_AXIS / 2) / COLUMN_FACTOR + 2,
      COLOURS => WHITE);
   if LINEAR then
      Label_Linear_X (X_MINIMUM => X_MIN_POINT,
         X_MAXIMUM => X_MAX_POINT);
   else
      Label_Log_X (X_MINIMUM => X_MIN_POINT,
         X_MAXIMUM => X_MAX_POINT);
   end if;
   Label_Y (Y_MINIMUM => Y_MIN_POINT,
      Y_MAXIMUM => Y_MAX_POINT);
   Zero_Line (Y_MINIMUM => Y_MIN_POINT,
      Y_MAXIMUM => Y_MAX_POINT,
      Y_SCALE   => Y_SCALE);
   Set_Plot_Window (X1 => X_MIN + X_AXIS,
      Y1 => Y_AXIS,
      X2 => MAX_X_COORD,
      Y2 => MAX_Y_COORD - 8 * Y_AXIS / 2);
   COUNTER := 0;
   loop
      Plot_Point (Y_SCALE => Y_SCALE,
         Y_MINIMUM => Y_MIN_POINT,
         COUNTER   => COUNTER,
         X_INC     => X_INC,
         Y_NEW     => Y_NEW,
         Y_PREV    => Y_PREVIOUS);
      COUNTER := COUNTER + X_INC;
      Y_PREVIOUS := Y_NEW;
      exit when COUNTER >= NX_PIXELS;
   end loop;
   if PARAMETRIC then
      Inner.Write_Netlist (COMPONENT => COMPONENT2,
         VALUE          => COMPONENT_SAVE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
   end if;
end Plot;
```

```
-- PURPOSE : Plots a node response on a linear axis when the frequency and
--           the component value is varied.
```

```
procedure Parametric_Linear_Plot2 is

   HIGH_NODE1,              LOW_NODE1,
   HIGH_NODE2,              LOW_NODE2          : NODE_TYPE;
   COLOUR,                  X_INC,
   X,                       Y                  : INTEGER_TYPE;
   COMP_INC,                COMP_POINT,
   COMPONENT_SAVE,          FREQ_INC,
   FREQUENCY,               FREQ_POINT,
   IMAG_VOLTS,              MAGNITUDE,
   MAX_MAGNITUDE,           MIN_MAGNITUDE,
   REAL_VOLTS,              SCALE_MAGNITUDE,
   VALUE,                   X_MAX_POINT,
   X_MIN_POINT,             X_SCALE,
   Y_MAX_POINT,             Y_MIN_POINT,
   Y_SCALE                                     : REAL_TYPE;
   HALTING                                     : BOOLEAN;
   COMPONENT,               COMPONENT2         : ELEMENT_TYPE;

   PARAMETRIC_FORM : FIELDS_TYPE (1..2) :=
      (1 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 26,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 0,
         ROW            => 4,
         VALUE          => "1.0e-1          ",
         TEXT           => "Minimum node voltage",
         MAX_REAL       => 1.0e15,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1.0e-3),
      2 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 26,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 6,
         ROW            => 5,
         VALUE          => "1.1e-3          ",
         TEXT           => "Maximum node voltage",
         MAX_REAL       => 1.0e15,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1.1e-3));

begin
   GET_X_FORM (7).ACCEPTING := FALSE;
   Get_X_Component (FREQUENCY => FREQUENCY,
      X_MINIMUM      => X_MIN_POINT,
      X_MAXIMUM      => X_MAX_POINT,
      COMPONENT      => COMPONENT,
      VALUE          => COMPONENT_SAVE,
      HIGH_NODE1     => HIGH_NODE1,
      LOW_NODE1      => LOW_NODE1,
      HIGH_NODE2     => HIGH_NODE2,
      LOW_NODE2      => LOW_NODE2,
      NETLIST_OBJECT => NETLIST_OBJECT);
   if COMPONENT = RESISTOR then
      COMPONENT2 := GR_ADMIT;
   else
      COMPONENT2 := COMPONENT;
   end if;
   Get_X_Frequency (MIN_FREQUENCY => X_MIN_POINT,
      MAX_FREQUENCY => X_MAX_POINT);
```

```
   loop
      Prompts.Read (FORM => PARAMETRIC_FORM);
      MIN_MAGNITUDE := PARAMETRIC_FORM (1).REAL_VALUE;
      MAX_MAGNITUDE := PARAMETRIC_FORM (2).REAL_VALUE;
      if MAX_MAGNITUDE <= MIN_MAGNITUDE then
         Errors.Write (MESSAGE => "The maximum cannot be equal or less" &
            " than the minimum.");
      else
         exit;
      end if;
   end loop;
   SCALE_MAGNITUDE := (MAX_MAGNITUDE - MIN_MAGNITUDE) / 16.0;
   Y_SCALE := REAL_TYPE (NY_PIXELS) / (Y_MAX_POINT - Y_MIN_POINT);
   X_SCALE := REAL_TYPE (NX_PIXELS) / (X_MAX_POINT - X_MIN_POINT);
   FREQ_INC := (X_MAX_POINT - X_MIN_POINT) / REAL_TYPE (NX_PIXELS);
   COMP_INC := (Y_MAX_POINT - Y_MIN_POINT) / REAL_TYPE (NY_PIXELS);
   Status.Write (MESSAGE => CONTINUE_MESSAGE);
   Graph_Set_Up (OPTION => 4);
   Label_Linear_X (X_MINIMUM => X_MIN_POINT,
      X_MAXIMUM => X_MAX_POINT);
   Label_Y (Y_MINIMUM => Y_MIN_POINT,
      Y_MAXIMUM => Y_MAX_POINT);
   Set_Plot_Window (X1 => X_MIN + X_AXIS,
      Y1 => Y_AXIS,
      X2 => MAX_X_COORD,
      Y2 => MAX_Y_COORD - 3 * Y_AXIS / 2);
   FREQ_POINT := X_MIN_POINT;
   loop
      COMP_POINT := Y_MIN_POINT;
      loop
         if COMPONENT = RESISTOR then
            VALUE := 1.0 / COMP_POINT;
         else
            VALUE := COMP_POINT;
         end if;
         Inner.Write_Netlist (COMPONENT => COMPONENT2,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Central_Solver.Solve_AC_Network (NODE => NLC,
            AC_FREQUENCY   => FREQ_POINT,
            CALCULATE      => TRUE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Central_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (1),
            HIGH_NODE => HIGHER_NODES (1),
            REAL_VOLT => REAL_VOLTS,
            IMAG_VOLT => IMAG_VOLTS);
         MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS,
            IMAG_VALUE => IMAG_VOLTS);
         Y := Y_MIN - Y_AXIS -
            INTEGER_TYPE (Y_SCALE * (COMP_POINT - Y_MIN_POINT));
         X := X_MIN + X_AXIS +
            INTEGER_TYPE (X_SCALE * (FREQ_POINT - X_MIN_POINT));
         if (MIN_MAGNITUDE < 0.0) then
            COLOUR := abs(INTEGER_TYPE (SCALE_MAGNITUDE*MAGNITUDE));remis;
         else
            COLOUR := abs (INTEGER_TYPE (SCALE_MAGNITUDE *
               (MAGNITUDE - MIN_MAGNITUDE))) rem 16;
         end if;
         Set_Plot_Colour (COLOUR => FOREGROUND_TYPE'val (COLOUR));
         Plot (X => X,
               Y => Y);
         COMP_POINT := COMP_POINT + COMP_INC;
         HALTING := Halt_Processing;
         exit when (COMP_POINT > Y_MAX_POINT) or HALTING;
      end loop;
      FREQ_POINT := FREQ_POINT + FREQ_INC;
      exit when (FREQ_POINT > X_MAX_POINT) or HALTING;
   end loop;
   Inner.Write_Netlist (COMPONENT => COMPONENT2,
      VALUE          => COMPONENT_SAVE,
      HIGH_NODE1     => HIGH_NODE1,
      LOW_NODE1      => LOW_NODE1,
      HIGH_NODE2     => HIGH_NODE2,
      LOW_NODE2      => LOW_NODE2,
      NETLIST_OBJECT => NETLIST_OBJECT);
end Parametric_Linear_Plot2;

-------------------------------------------------------------------------

begin  -- Circuit_Response;
   Process_Title (TITLE => "CIRCUIT RESPONSE");
   Y_VALUES := new Y_VALUE_TYPE;
   for CLEAR_ROW in 0..NX_PIXELS loop
      for CLEAR_COLUMN in 1..5 loop
         Y_VALUES (CLEAR_ROW, CLEAR_COLUMN) := 0.0;
      end loop;
   end loop;
   loop
      begin
         Process_Title (TITLE => "CIRCUIT RESPONSE");
         Y_MIN := MAX_Y_COORD - 3 * Y_AXIS / 2;
         Prompts.Read (LIST => PLOT_LIST,
            HELP_INFO => RF_FREQUENCY_RESPONSE);
         for COUNTER in 1..5 loop
            if PLOT_LIST (COUNTER).STATE then
               case COUNTER is
                  when 1 => Plot (PARAMETRIC => FALSE, LINEAR => TRUE);
                  when 2 => Plot (PARAMETRIC => FALSE, LINEAR => FALSE);
                  when 3 => Plot (PARAMETRIC => TRUE,  LINEAR => TRUE);
                  when 4 => Plot (PARAMETRIC => TRUE,  LINEAR => FALSE);
                  when 5 => Parametric_Linear_Plot;
               end case;
               exit;
            end if;
         end loop;
         Set_Plot_Window (X1 => 1,
            Y1 => 1,
            X2 => MAX_X_COORD,
            Y2 => MAX_Y_COORD);
      exception
         when STORAGE_ERROR =>
            Change_Screen (MODE_OF_SCREEN => TEXT);
            Dispose (Y_VALUES);
            Set_Plot_Window (X1 => 1,
               Y1 => 1,
               X2 => MAX_X_COORD,
               Y2 => MAX_Y_COORD);
            raise STORAGE_ERROR;
         when USER_ESCAPE => exit;
      end;
      Continue_Processing;
      Change_Screen (MODE_OF_SCREEN => TEXT);
      Clear;
   end loop;
   Dispose (Y_VALUES);
   Change_Screen (MODE_OF_SCREEN => TEXT);
end Circuit_Response;

-------------------------------------------------------------------------
-- PURPOSE : Plots a phasor diagram for the defined node pairs.
-------------------------------------------------------------------------

procedure Phasors (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   MAX_PHASOR : constant                := 8;
   X_CENTRE   : constant                := MAX_X_COORD / 2;
   Y_CENTRE   : constant INTEGER_TYPE := MAX_Y_COORD / 2;
   Y_SCALE    : constant INTEGER_TYPE := MAX_Y_COORD / 3;

   type VOLTAGE_ARRAY      is array (1..MAX_PHASOR) of REAL_TYPE;
   type PHASOR_COLOUR_TYPE is array (1..MAX_PHASOR) of FOREGROUND_TYPE;

   PHASOR_CHARACTER : constant STRING := "+*@$=%";
   PHASOR_COLOUR    : constant PHASOR_COLOUR_TYPE := (WHITE,
      YELLOW,       LIGHT_PURPLE,
      LIGHT_RED,    LIGHT_CYAN,
      LIGHT_GREEN,  LIGHT_BLUE,
      LIGHT_GRAY);

   DISPLAY_LINE                        : Texts.TEXT_TYPE;
   REAL_VOLT,        IMAG_VOLT         : VOLTAGE_ARRAY;
   COUNTER                             : INTEGER_TYPE := 1;
   MAGNITUDE,        MAXIMUM,
   PHASOR_FREQUENCY, RATIO             : REAL_TYPE;
   X_SCALE                             : INTEGER_TYPE := INTEGER_TYPE
                                        (REAL_TYPE(Y_SCALE) *
ASPECT_RATIO);

begin
   Process_Title (TITLE => "PHASOR PLOT");
   Outer.Get_Frequency (FREQUENCY => PHASOR_FREQUENCY,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Central_Solver.Solve_AC_Network (NODE => NLC,
      AC_FREQUENCY   => PHASOR_FREQUENCY,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Change_Screen (MODE_OF_SCREEN => GRAPHIC);
   String_To_Graphic (MESSAGE => "PHASOR PLOT",
      ROW    => 1,
      COLUMN => 34,
      C[illegible]RS => WHITE);
   while (COUNTER <= MAX_PHASOR) and
      (HIGHER_NODES (COUNTER) in 1..NODE_MAX) loop
      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => PHASOR_CHARACTER (COUNTER) & " " &
         TEXT_LINE  => DISPLAY_LINE);
      Texts.Combine (ITEM => HIGHER_NODES (COUNTER),
         ITEM_WIDTH => 2,
         TEXT_LINE  => DISPLAY_LINE);
      Texts.Combine (IT[illegible]
         TEXT_LINE  =>[illegible]NE);
      Texts.Combine (IT[illegible]R_NODES (COUNTER),
         ITEM_WIDTH => 2
         TEXT_LINE  => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => COUNTER,
         COLUMN  => 2,
         COLOURS => PHASOR_COLOUR (COUNTER));
      Central_Solver.Get_Voltages (LOW_NODE => LOWER_NODES (COUNTER),
         HIGH_NODE => HIGHER_NODES (COUNTER),
         REAL_VOLT => REAL_VOLT (COUNTER),
         IMAG_VOLT => IMAG_VOLT (COUNTER));
      MAGNITUDE := Maths.Find_Magnitude (
         REAL_VALUE => REAL_VOLT (COUNTER),
         IMAG_VALUE => IMAG_VOLT (COUNTER));
      if (MAGNITUDE > MAXIMUM) or (COUNTER = 1) then
         MAXIMUM := MAGNITUDE;
      end if;
      COUNTER := COUNTER + 1;
   end loop;
   Set_Plot_Colour (COLOUR => WHITE);
   RATIO := REAL_TYPE (Y_CENTRE) / MAXIMUM;
   Dot_Horizontal_Line (DOT_SPACING => 5,
      X1 => X_CENTRE - X_SCALE,
      X2 => X_CENTRE + X_SCALE,
      Y  => Y_CENTRE);
   Dot_Vertical_Line (DOT_SPACING => 5,
      Y1 => Y_CENTRE - Y_SCALE,
      Y2 => Y_CENTRE + Y_SCALE,
      X  => X_CENTRE);
   COUNTER := 1;
   while (COUNTER <= MAX_PHASOR) and
      (HIGHER_NODES (COUNTER) in 1..NODE_MAX) loop
      Set_Plot_Colour (COLOUR => PHASOR_COLOUR (COUNTER));
      Line (X1 => X_CENTRE,
         Y1 => Y_CENTRE,
         X2 => X_CENTRE + INTEGER_TYPE (REAL_VOLT (COUNTER) * RATIO),
         Y2 => Y_CENTRE - INTEGER_TYPE (IMAG_VOLT (COUNTER) * RATIO
            / ASPECT_RATIO));
      Character_To_Graphic (MESSAGE => PHASOR_CHARACTER (COUNTER),
         ROW    => (Y_CENTRE - INTEGER_TYPE (IMAG_VOLT (COUNTER) * RATIO
            / ASPECT_RATIO)) / ROW_FACTOR,
         COLUMN => (X_CENTRE + INTEGER_TYPE (REAL_VOLT (COUNTER)*RATIO))
            / COLUMN_FACTOR,
         COLOURS => PHASOR_COLOUR (COUNTER));
      COUNTER := COUNTER + 1;
   end loop;
   Continue_Processing;
   Change_Screen (MODE_OF_SCREEN => TEXT);
   Clear;
end Phasors;

-------------------------------------------------------------------------
-- PURPOSE : Plots a polar diagram f.r the defined node pair.
```

```
procedure Polar (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   X_CENTRE : constant                := MAX_X_COORD / 2;
   Y_CENTRE : constant INTEGER_TYPE := MAX_Y_COORD / 2;
   Y_SCALE  : constant INTEGER_TYPE := MAX_Y_COORD / 3;

   COUNTER,                PLOT_COUNTER       : NATURAL := 1;
   HIGH_NODE1,             LOW_NODE1          : NODE_TYPE;
   X,                      X_PREVIOUS,
   Y,                      Y_PREVIOUS,
   N_POINTS                                   : INTEGER_TYPE;
   END_FREQUENCY,          FREQUENCY_INCR,
   MAGNITUDE,              MAXIMUM,
   PLOT_FREQUENCY,         RATIO,
   START_FREQUENCY                            : REAL_TYPE;
   X_SCALE                                    : INTEGER_TYPE := INTEGER_TYPE
                                               (REAL_TYPE(Y_SCALE)*ASPECT_RATIO);

   POLAR_LIST : LIST_ARRAY_TYPE (1..2) := (
      (TEXT_LENGTH => 17, TEXT => "Linear Increments", STATE => TRUE),
      (TEXT_LENGTH => 14, TEXT => "Log Increments",    STATE => FALSE));

   procedure Dispose is new
      Unchecked_Deallocation (POLAR_TYPE, POLAR_ACCESS);

   use Main;

begin
   Process_Title (TITLE => "POLAR PLOT");
   loop
      Prompts.Read (FORM => POLAR_FORM,
         LIST      => POLAR_LIST,
         HELP_INFO => HP_POLAR);
      HIGH_NODE1      := POLAR_FORM (1).INTEGER_VALUE;
      LOW_NODE1       := POLAR_FORM (2).INTEGER_VALUE;
      START_FREQUENCY := POLAR_FORM (3).REAL_VALUE;
      END_FREQUENCY   := POLAR_FORM (4).REAL_VALUE;
      N_POINTS        := POLAR_FORM (5).INTEGER_VALUE;
      if START_FREQUENCY >= END_FREQUENCY then
         Errors.Write (MESSAGE => "The Start Frequency must be less " &
            "than the End Frequency.");
      else
         exit;
      end if;
   end loop;
   if POLAR_LIST (1).STATE then
      FREQUENCY_INCR := (END_FREQUENCY - START_FREQUENCY) /
         REAL_TYPE (N_POINTS);
   else
      FREQUENCY_INCR := (Math_Lib.Ln (END_FREQUENCY) -
         Math_Lib.Ln (START_FREQUENCY)) / REAL_TYPE (N_POINTS);
   end if;
   Cursor_Set (ROW => 6, COLUMN => 17);
   Write (ITEM => "Calculating the       point of the Polar Plot.");
   PLOT_FREQUENCY := START_FREQUENCY;
   POLAR_VOLT := new POLAR_TYPE;
   begin
      while (PLOT_FREQUENCY < END_FREQUENCY  and
         (COUNTER < MAX_POINTS) loop
         Control_Solver.Solve_AC_Network (MODE => RLC,
            AC_FREQUENCY   => PLOT_FREQUENCY,
            CALCULATE      => TRUE,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Control_Solver.Get_Voltages (LOW_NODE => LOW_NODE1,
            HIGH_NODE => HIGH_NODE1,
            REAL_VOLT => POLAR_VOLT.REAL (COUNTER),
            IMAG_VOLT => POLAR_VOLT.IMAG (COUNTER));
         MAGNITUDE := Maths.Find_Magnitude (
            REAL_VALUE => POLAR_VOLT.REAL (COUNTER),
            IMAG_VALUE => POLAR_VOLT.IMAG (COUNTER));
         if (MAGNITUDE > MAXIMUM) or (COUNTER = 1) then
            MAXIMUM := MAGNITUDE;
         end if;
         Cursor_Set (ROW => 6, COLUMN => 33);
         Write (ITEM => COUNTER,
            ITEM_WIDTH => 4);
         COUNTER := COUNTER + 1;
         if POLAR_LIST (1).STATE then
            PLOT_FREQUENCY := PLOT_FREQUENCY + FREQUENCY_INCR;
         else
            PLOT_FREQUENCY := START_FREQUENCY *
               Math_Lib.Exp (FREQUENCY_INCR * REAL_TYPE (COUNTER));
         end if;
      end loop;
   exception
      when STORAGE_ERROR =>
         Dispose (POLAR_VOLT);
         raise STORAGE_ERROR;
   end;
   MAXIMUM := MAXIMUM * 1.2;
   Change_Screen (MODE_OF_SCREEN => GRAPH);
   String_To_Graphic (MESSAGE => "POLAR PLOT",
      ROW     => 1,
      COLUMN  => 34,
      COLOURS => WHITE);
   Set_Plot_Colour (COLOUR => WHITE);
   RATIO := REAL_TYPE (Y_CENTRE) / 1.25 / MAXIMUM;
   Dot_Horizontal_Line (DOT_SPACING => 5,
      X1 => X_CENTRE - X_SCALE,
      X2 => X_CENTRE + X_SCALE,
      Y  => Y_CENTRE);
   Dot_Vertical_Line (DOT_SPACING => 5,
      Y1 => Y_CENTRE - Y_SCALE,
      Y2 => Y_CENTRE + Y_SCALE,
      X  => X_CENTRE);
   Set_Plot_Colour (COLOUR => YELLOW);
   while PLOT_COUNTER < COUNTER loop
      X := X_CENTRE + INTEGER_TYPE (POLAR_VOLT.REAL (PLOT_COUNTER)*RATIO);
      Y := Y_CENTRE - INTEGER_TYPE (POLAR_VOLT.IMAG (PLOT_COUNTER)*RATIO /
         ASPECT_RATIO);
      if PLOT_COUNTER > 1 then
         Line (X1 => X_PREVIOUS,
            Y1 => Y_PREVIOUS,
            X2 => X,
            Y2 => Y);
      end if;
      X_PREVIOUS := X;
      Y_PREVIOUS := Y;
      PLOT_COUNTER := PLOT_COUNTER + 1;
   end loop;
   Continue_Processing;
   Dispose (POLAR_VOLT);
   Change_Screen (MODE_OF_SCREEN => TEXT);
   Clear;
end Polar;
```

```
end Plotter;
```

Central_Solver

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Central_Solver is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This does the "solving" of the circuit and is used as the
--           core process in the analysis.  It solves the circuit by
--           determining the voltages for the nodes of the circuit.  There
--           is a large data structure which consists of a dynamic access
--           object to a record.  This record contains two arrays - one for
--           the real voltage component values and one for the imaginary
--           voltage component values.  These arrays store the voltage
--           values for all the nodes in the circuit.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Get_Voltages (LOW_NODE, HIGH_NODE : in NODE_TYPE;
      REAL_VOLT, IMAG_VOLT : out REAL_TYPE);

   procedure Solve_AC_Network (NODE : in CIRCUIT_NODE_TYPE;
      AC_FREQUENCY   : in     REAL_TYPE;
      CALCULATE      : in     BOOLEAN;
      NETLIST_OBJECT : in out NETLIST_TYPE);

end Central_Solver;
```

```
with  Math_Lib,
      Netlist_Supportar,
      Screens,
      Unchecked_Deallocation;

use   Screens;

package body Central_Solver is

   use Screens.Main;

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

      -- Variables used for storing the node voltages.

      MAX_NODES_1 : constant := MAX_NODES + 1;

      type VOLTAGE_ARRAY_TYPE is array (1..MAX_NODES_1) of REAL_TYPE;

      type VOLTAGE_TYPE is
         record
            VOLTAGE_REAL : VOLTAGE_ARRAY_TYPE;
            VOLTAGE_IMAG : VOLTAGE_ARRAY_TYPE;
         end record;

      NODE : VOLTAGE_TYPE;

      -- Used to hold the conductance and susceptance values.

      type MATRIX_GB_TYPE is array (0..MAX_NODES_1, 0..MAX_NODES)
         of REAL_TYPE;

      type MATRIX_GB_ACCESS is access MATRIX_GB_TYPE;

      B_MATRIX,              G_MATRIX            : MATRIX_GB_ACCESS;

      procedure Dispose is new
         Unchecked_Deallocation (MATRIX_GB_TYPE, MATRIX_GB_ACCESS);

-- PURPOSE : Gets the node voltage and adjusts it if the  LOW_NODE  is not
--           the reference node.
```

```
procedure Get_Voltages (LOW_NODE, HIGH_NODE : in NODE_TYPE;
   REAL_VOLT, IMAG_VOLT : out REAL_TYPE) is

begin
   if LOW_NODE = 0 then
      REAL_VOLT := NODE.VOLTAGE_REAL (HIGH_NODE);
      IMAG_VOLT := NODE.VOLTAGE_IMAG (HIGH_NODE);
   else
      REAL_VOLT := NODE.VOLTAGE_REAL (HIGH_NODE)
         - NODE.VOLTAGE_REAL (LOW_NODE);
      IMAG_VOLT := NODE.VOLTAGE_IMAG (HIGH_NODE)
         - NODE.VOLTAGE_IMAG (LOW_NODE);
   end if;
end Get_Voltages;

-- PURPOSE : Calculates the AC solution of the network by using a
--           conductance matrix (G) and a susceptance matrix (B).  The
--           results are displayed on the Main screen.

procedure Solve_AC_Network (NODE : in CIRCUIT_NODE_TYPE;
   AC_FREQUENCY   : in     REAL_TYPE;
   CALCULATE      : in     BOOLEAN;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   NETLIST2 : constant NETL[illegible] TYPE := NETLIST_OBJECT;

   FIRST_DATA                          : BOOLEAN := FALSE;
   NODE_MAX,                           : NODE_TYPE;

   procedure Store_GB_Matrices is

      COLUMN,              ROW                 : INTEGER;
      SCREEN_ROW                               : INTEGER := 3;

   begin
      for ROW in 1..NODE_MAX loop
         for COLUMN in 1..NODE_MAX loop
            if SCREEN_ROW = 3 then
               Process_Title (TITLE => "ADMITTANCE MATRIX");
               Write (ITEM => "    Row   Column       G (row, column)
                  & "       B (row, column)
                  FORM => UNDERLINED);
               Cursor_Next_Line;
               Cursor_Set (ROW => 3,
                           COLUMN => 1);
            end if;
            Write (ITEM => ROW, ITEM_WIDTH => 6);
            Write (ITEM => COLUMN, ITEM_WIDTH => 7);
            Write (ITEM => G_MATRIX (ROW, COLUMN),
               WHOLE          => 13,
               FRACTION       => 5,
               ENG_NOTATION => TRUE);
            Write (ITEM => B_MATRIX (ROW, COLUMN),
               WHOLE          => 15,
               FRACTION       => 5,
               ENG_NOTATION => TRUE);
            Cursor_Next_Line;
            SCREEN_ROW := SCREEN_ROW + 1;
            if SCREEN_ROW > 24 then
               Status.Wait;
               SCREEN_ROW := 3;
            end if;
         end loop;
      end loop;
   end Store_GB_Matrices;

   -- Variables RC = Row-Column

   procedure Solve_Linear_Equations is

      RC1,                 RC2,
      RC3,                 RC4,
      RC5                                      : INTEGER;
      T_AR,                T_AI,
      T_AAR,               T_AAI,
      T_BR,                T_BI,
      T_BBR,               T_BBI,
      TMAG,                RL_VOLT,
      IM_VOLT                                  : REAL_TYPE;

   begin
      for RC1 in 1..NODE_MAX loop
         RC3 := RC1 - 1;
         for RC2 in RC1..NODE_MAX loop
            RC4 := RC2 + 1;
            T_AAR := G_MATRIX (RC4, RC1);
            T_AAI := B_MATRIX (RC4, RC1);
            if RC3 > 0 then
               T_BBR := G_MATRIX (RC1, RC2);
               T_BBI := B_MATRIX (RC1, RC2);
               for RC5 in 1..RC3 loop
                  T_AR := G_MATRIX (RC5, RC2);
                  T_AI := B_MATRIX (RC5, RC2);
                  T_BR := G_MATRIX (RC1, RC5);
                  T_BI := B_MATRIX (RC1, RC5);
                  T_BBR := T_BBR - T_AR * T_BR + T_AI * T_BI;
                  T_BBI := T_BBI - T_AR * T_BI - T_AI * T_BR;
                  T_AR := G_MATRIX (RC5, RC1);
                  T_AI := B_MATRIX (RC5, RC1);
                  T_BR := G_MATRIX (RC4, RC5);
                  T_BI := B_MATRIX (RC4, RC5);
                  T_AAR := T_AAR - T_AR * T_BR + T_AI * T_BI;
                  T_AAI := T_AAI - T_AR * T_BI - T_AI * T_BR;
               end loop;
               G_MATRIX (RC1, RC2) := T_BBR;
               B_MATRIX (RC1, RC2) := T_BBI;
            end if;
            T_AR := G_MATRIX (RC1, RC1);
            T_AI := B_MATRIX (RC1, RC1);
            TMAG := T_AR * T_AR + T_AI * T_AI;
            if TMAG > MINIMUM_VALUE then
               G_MATRIX (RC4, RC1) := (T_AAR * T_AR + T_AAI * T_AI)/TMAG;
```

```
            B_MATRIX (RC4, RC1) := (T_AAI * T_AR - T_AAR * T_AI)/TMAG;
         else
            G_MATRIX (RC4, RC1) := 0.0;
            B_MATRIX (RC4, RC1) := 0.0;
         end if;
      end loop;
   end loop;

   for RC1 in 2..NODE_MAX loop
      RC2 := NODE_PLUS1 - RC1;
      RL_VOLT := G_MATRIX (NODE_PLUS1, RC2);
      IM_VOLT := B_MATRIX (NODE_PLUS1, RC2);
      RC4 := RC2 + 1;
      for RC5 in RC4..NODE_MAX loop
         T_AR := G_MATRIX (RC5, RC2);
         T_AI := B_MATRIX (RC5, RC2);
         T_BR := G_MATRIX (NODE_PLUS1, RC5);
         T_BI := B_MATRIX (NODE_PLUS1, RC5);
         RL_VOLT := RL_VOLT - T_AR * T_BR + T_AI * T_BI;
         IM_VOLT := IM_VOLT - T_AR * T_BI - T_AI * T_BR;
      end loop;
      G_MATRIX (NODE_PLUS1, RC2) := RL_VOLT;
      B_MATRIX (NODE_PLUS1, RC2) := IM_VOLT;
      NODE.VOLTAGE_REAL (RC2) := RL_VOLT;
      NODE.VOLTAGE_IMAG (RC2) := IM_VOLT;
   end loop;
   NODE.VOLTAGE_REAL (NODE_MAX) := G_MATRIX (NODE_PLUS1, NODE_MAX);
   NODE.VOLTAGE_IMAG (NODE_MAX) := B_MATRIX (NODE_PLUS1, NODE_MAX);
end Solve_Linear_Equations;

-------------------------------------------------------------------------

procedure Find_GB_Matrices is

   use Math_Lib;

   COMPONENT                  : ELEMENTS;
   COLUMN,                    ROW                : INTEGER;
   HIGH_NODE,                 LOW_NODE,
   HIGH_NODE1,                LOW_NODE1,
   HIGH_NODE2,                LOW_NODE2          : NODE_TYPE;
   R_VOLTS,                   I_VOLTS,
   R_ADMITS,                  I_ADMITS,
   CR_ADMITS,                 CI_ADMITS,
   RATIO,                     VALUE              : REAL_TYPE;
   OMEGA                      : REAL_TYPE := 2.0 * PI * AC_FREQUENCY;

begin
   for ROW in 0..NODE_PLUS1 loop
      for COLUMN in 0..NODE_MAX loop
         B_MATRIX (ROW, COLUMN) := 0.0;
         G_MATRIX (ROW, COLUMN) := 0.0;
      end loop;
   end loop;
   Netlists.Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
   loop
      for COMPONENT in CAPS..POLE_FREQ loop
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE,
            LOW_NODE1      => LOW_NODE,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         case COMPONENT is
            when CAPS =>
               I_ADMITS := OMEGA * VALUE;
            when INDUCT =>
               if (abs (OMEGA * VALUE) > MINIMUM_VALUE) and
                  (MODE = RLC) then
                  I_ADMITS := I_ADMITS - 1.0 / (OMEGA * VALUE);
               end if;
            when GR_ADMIT =>
               R_ADMITS := VALUE;
            when GI_ADMIT =>
               if MODE = ADMITTANCE then
                  I_ADMITS := VALUE;
               end if;
            when REAL_VOLT =>
               R_VOLTS := VALUE;
            when IMAG_VOLT =>
               I_VOLTS := R_VOLTS * I_ADMITS + VALUE * R_ADMITS;
               R_VOLTS := R_VOLTS * R_ADMITS - VALUE * I_ADMITS;
            when REAL_CRNT =>
               R_VOLTS := R_VOLTS - VALUE;
            when IMAG_CRNT =>
               I_VOLTS := I_VOLTS - VALUE;
               HIGH_NODE1 := HIGH_NODE;
               LOW_NODE1  := LOW_NODE;
            when CR_ADMIT =>
               CR_ADMITS := VALUE;
            when CI_ADMIT =>
               CI_ADMITS := VALUE;
            when CR_ADMIT =>
               if MODE = RLC then
                  CR_ADMITS := VALUE;
               end if;
            when POLE_FREQ =>
               if MODE = RLC then
                  if abs (VALUE) > MINIMUM_VALUE then
                     RATIO := AC_FREQUENCY / VALUE;
                     if abs (RATIO) < MAX_VOLT_VALUE then
                        CR_ADMITS := CR_ADMITS / (1.0 + RATIO * RATIO);
                     else
                        CR_ADMITS := 0.0;
                     end if;
                     CI_ADMITS := -RATIO * CR_ADMITS;
                  else
                     CI_ADMITS := 0.0;
                  end if;
               end if;
            when others =>
               null;
         end case;
      end loop;

      G_MATRIX (HIGH_NODE1, HIGH_NODE1)
         := G_MATRIX (HIGH_NODE1, HIGH_NODE1) + R_ADMITS;
      B_MATRIX (HIGH_NODE1, HIGH_NODE1)
         := B_MATRIX (HIGH_NODE1, HIGH_NODE1) + I_ADMITS;
      G_MATRIX (NODE_PLUS1, HIGH_NODE1)
         := G_MATRIX (NODE_PLUS1, HIGH_NODE1) + R_VOLTS;
      B_MATRIX (NODE_PLUS1, HIGH_NODE1)
         := B_MATRIX (NODE_PLUS1, HIGH_NODE1) + I_VOLTS;

      if LOW_NODE1 > 0 then
         G_MATRIX (LOW_NODE1, LOW_NODE1)
            := G_MATRIX (LOW_NODE1, LOW_NODE1) + R_ADMITS;
         B_MATRIX (LOW_NODE1, LOW_NODE1)
            := B_MATRIX (LOW_NODE1, LOW_NODE1) + I_ADMITS;
         G_MATRIX (HIGH_NODE1, LOW_NODE1)
            := G_MATRIX (HIGH_NODE1, LOW_NODE1) - R_ADMITS;
         B_MATRIX (HIGH_NODE1, LOW_NODE1)
            := B_MATRIX (HIGH_NODE1, LOW_NODE1) - I_ADMITS;
         G_MATRIX (LOW_NODE1, HIGH_NODE1)
            := G_MATRIX (LOW_NODE1, HIGH_NODE1) - R_ADMITS;
         B_MATRIX (LOW_NODE1, HIGH_NODE1)
            := B_MATRIX (LOW_NODE1, HIGH_NODE1) - I_ADMITS;
         G_MATRIX (NODE_PLUS1, LOW_NODE1)
            := G_MATRIX (NODE_PLUS1, LOW_NODE1) - R_VOLTS;
         B_MATRIX (NODE_PLUS1, LOW_NODE1)
            := B_MATRIX (NODE_PLUS1, LOW_NODE1) - I_VOLTS;
      end if;
      if (abs (CR_ADMITS) + abs (CI_ADMITS)) > MINIMUM_VALUE then
         HIGH_NODE1 := HIGH_NODE;
         LOW_NODE1  := LOW_NODE;
         G_MATRIX (HIGH_NODE2, HIGH_NODE2)
            := G_MATRIX (HIGH_NODE2, HIGH_NODE2) + CR_ADMITS;
         B_MATRIX (HIGH_NODE2, HIGH_NODE2)
            := B_MATRIX (HIGH_NODE2, HIGH_NODE2) + CI_ADMITS;
         if LOW_NODE2 > 0 then
            G_MATRIX (LOW_NODE2, HIGH_NODE2)
               := G_MATRIX (LOW_NODE2, HIGH_NODE2) - CR_ADMITS;
            B_MATRIX (LOW_NODE2, HIGH_NODE2)
               := B_MATRIX (LOW_NODE2, HIGH_NODE2) - CI_ADMITS;
         end if;
         if LOW_NODE1 > 0 then
            G_MATRIX (HIGH_NODE2, LOW_NODE1)
               := G_MATRIX (HIGH_NODE2, LOW_NODE1) - CR_ADMITS;
            B_MATRIX (HIGH_NODE2, LOW_NODE1)
               := B_MATRIX (HIGH_NODE2, LOW_NODE1) - CI_ADMITS;
         end if;
         if LOW_NODE2 > 0 then
            G_MATRIX (LOW_NODE2, LOW_NODE1)
               := G_MATRIX (LOW_NODE2, LOW_NODE1) + CR_ADMITS;
            B_MATRIX (LOW_NODE2, LOW_NODE1)
               := B_MATRIX (LOW_NODE2, LOW_NODE1) + CI_ADMITS;
         end if;
      end if;
      begin
         Netlists.Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
end Find_GB_Matrices;

-------------------------------------------------------------------------

begin  -- Solve_AC_Network
   Inner.Find_Node_Maximum (NODE_MAXIMUM => NODE_MAX,
      NETLIST_OBJECT => NETLIST_OBJECT);
   NODE_PLUS1 := NODE_MAX + 1;
   B_MATRIX   := new MATRIX_GB_TYPE;
   FIRST_DATA := TRUE;
   G_MATRIX   := new MATRIX_GB_TYPE;
   Find_GB_Matrices;
   if CALCULATE then
      Solve_Linear_Equations;
   else
      Store_GB_Matrices;
   end if;
   Dispose (G_MATRIX);
   Dispose (B_MATRIX);
   NETLIST_OBJECT := NETLIST2;
exception
   when STORAGE_ERROR =>
      if FIRST_DATA then
         Dispose (B_MATRIX);
      end if;
      raise STORAGE_ERROR;
end Solve_AC_Network;

-------------------------------------------------------------------------

end Control_Solver;
```

Dos_Environment

```
with  Ada_Extensions;
use   Ada_Extensions;

package Dos_Environment is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This allows services provided  by the underlying computer and
--           operating system  to be made  available.  These services  are
--           divided into two groups :
--              (1) BIOS  services  which are  concerned  with  handling a
--                  function call.
--              (2) Buffer commands which are used for the subscreens.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary checking code.  Checks can be  introduced  Again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   subtype ASCII_CHARACTERS is INTEGER_TYPE range 0..255;
   type FRAME_TYPE is array (1..10) of ASCII_CHARACTERS;
      pragma PACK (FRAME_TYPE);

   type SCREEN_COLOUR_TYPE is
      record
         FOREGROUND : FOREGROUND_TYPE := LIGHT_GRAY;
         BACKGROUND : BACKGROUND_TYPE := BLACK;
         BLINK      : BOOLEAN         := FALSE;
      end record;

   for SCREEN_COLOUR_TYPE use
      record
         FOREGROUND at 0 * 1 range 00..03;
         BACKGROUND at 0 * 1 range 04..06;
         BLINK      at 0 * 1 range 07..07;
      end record;

   ------------------------------  Cursor Functions  ------------------------------

   subtype LINE_TYPE is INTEGER_TYPE range 0..15;

   procedure Change_Cursor_Shape (START_LINE,
      ENDING_LINE : in LINE_TYPE;
      DISPLAY     : in BOOLEAN);

      pragma interface (assembly, Change_Cursor_Shape);


   procedure Hide_Cursor;

      pragma interface (assembly, Hide_Cursor);


   procedure Set_Cursor (ROW    : in ROW_TYPE    := 1;
                         COLUMN : in COLUMN_TYPE := 1);

      pragma interface (assembly, Set_Cursor);


   procedure Show_Cursor;

      pragma interface (assembly, Show_Cursor);

   ------------------------------  Mouse Functions  -------------------------------

   type MOUSE_BUTTONS_TYPE is (
      NO_BUTTON,              LEFT_BUTTON,
      RIGHT_BUTTON,           LEFT_RIGHT_BUTTONS,
      MIDDLE_BUTTON,          LEFT_MIDDLE_BUTTONS,
      MIDDLE_RIGHT_BUTTONS,   ALL_BUTTONS);


   procedure Check_For_Mouse (MOUSE_PRESENT : out BOOLEAN;
      NUMBER_OF_BUTTONS : out POSITIVE);

      pragma interface (assembly, Check_For_Mouse);


   procedure Hide_Text_Mouse;

      pragma interface (assembly, Hide_Text_Mouse);


   procedure Read_Text_Mouse (MOUSE_ROW : out ROW_TYPE;
      MOUSE_COLUMN  : out COLUMN_TYPE;
      MOUSE_BUTTONS : out MOUSE_BUTTONS_TYPE);

      pragma interface (assembly, Read_Text_Mouse);


   procedure Show_Text_Mouse;

      pragma interface (assembly, Show_Text_Mouse);

   ------------------------------  Keyboard Functions  ----------------------------

   subtype KEY_VALUES_TYPE is INTEGER_TYPE;

   KB_0          : constant := 16#0030#;  KB_9          : constant := 16#0039#;
   KB_UPPER_A    : constant := 16#0041#;  KB_UPPER_Z    : constant := 16#005A#;
   KB_LOWER_A    : constant := 16#0061#;  KB_LOWER_Z    : constant := 16#0080#;
   KB_ALT_C      : constant := 16#2E00#;  KB_ALT_F      : constant := 16#2100#;
   KB_ALT_H      : constant := 16#2300#;  KB_ALT_M      : constant := 16#3200#;
   KB_ALT_N      : constant := 16#3100#;  KB_ALT_O      : constant := 16#1800#;
   KB_ALT_Y      : constant := 16#1500#;

   KB_F1         : constant := 16#3B00#;  KB_F2         : constant := 16#3C00#;
   KB_F3         : constant := 16#3D00#;  KB_F4         : constant := 16#3E00#;
   KB_F5         : constant := 16#3F00#;  KB_F6         : constant := 16#4000#;
   KB_F7         : constant := 16#4100#;  KB_F8         : constant := 16#4200#;
   KB_F9         : constant := 16#4300#;  KB_F10        : constant := 16#4400#;
   KB_ALT_F10    : constant := 16#7100#;  KB_SHIFT_F1   : constant := 16#5400#;

   KB_BK_SLASH   : constant := 16#005C#;  KB_FD_SLASH   : constant := 16#002F#;
   KB_COLON      : constant := 16#003A#;  KB_PERIOD     : constant := 16#002E#;
   KB_PLUS       : constant := 16#002B#;  KB_MINUS      : constant := 16#002D#;
   KB_SPACE      : constant := 16#0020#;  KB_TAB        : constant := 16#0009#;
   KB_ENTER      : constant := 16#000D#;  KB_ESC        : constant := 16#001B#;
   KB_UP         : constant := 16#4800#;  KB_LEFT       : constant := 16#4B00#;
   KB_DOWN       : constant := 16#5000#;  KB_RIGHT      : constant := 16#4D00#;
   KB_PGUP       : constant := 16#4900#;  KB_PGDN       : constant := 16#5100#;
   KB_HOME       : constant := 16#4700#;  KB_END        : constant := 16#4F00#;
   KB_INST       : constant := 16#5200#;  KB_DEL        : constant := 16#5300#;
   KB_BACKSPACE  : constant := 16#0008#;  KB_NO_KEY     : constant := 16#0000#;
   KB_SHIFT_TAB  : constant := 16#0F00#;


   procedure Get_Keyboard_Character (KEY : out KEY_VALUES_TYPE);

      pragma interface (assembly, Get_Keyboard_Character);


   function Keyboard_Character_Ready return BOOLEAN;

      pragma interface (assembly, Keyboard_Character_Ready);

   ------------------  Screen and Window Buffer Initialisation Function  ----------

   procedure Screen_Setup (Y_RESOLUTION : out INTEGER_TYPE);

      pragma interface (assembly, Screen_Setup);


   ------------------------------  Main Buffer Functions  -------------------------

   procedure Clear_Main_Buffer (MAIN_COLOURS : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Clear_Main_Buffer);


   procedure Main_Buffer_Line (LINE_COLOURS : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Main_Buffer_Line);


   procedure Main_Buffer_To_Screen;

      pragma interface (assembly, Main_Buffer_To_Screen);


   procedure String_To_Main_Buffer (MESSAGE : in STRING;
      ROW      : in ROW_TYPE;
      COLUMN   : in COLUMN_TYPE;
      COLOURS1,
      COLOURS2 : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, String_To_Main_Buffer);


   ------------------------------  Window Buffer Functions  -----------------------

   procedure Character_To_Window_Buffer (
      CHARACTER_CODE : in ASCII_CHARACTERS;
      COUNT          : in INTEGER_TYPE;
      ROW            : in ROW_TYPE;
      COLUMN         : in COLUMN_TYPE;
      COLOURS        : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Character_To_Window_Buffer);


   procedure Clear_Window_Buffer (COLOURS : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Clear_Window_Buffer);


   procedure Frame_Window_Buffer (BORDERS : in FRAME_TYPE;
      COLOURS : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Frame_Window_Buffer);


   procedure Make_Line_Bar (ROW : in ROW_TYPE;
      COLUMN,
      BAR_LENGTH : in COLUMN_TYPE;
      COLOURS    : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, Make_Line_Bar);


   procedure Remove_Line_Bar;

      pragma interface (assembly, Remove_Line_Bar);


   procedure Now_Window_Location (TOP_ROW : in ROW_TYPE;
```

```
      LEFT_SIDE : in COLUMN_TYPE);

      pragma interface (assembly, New_Window_Location);

   procedure Make_Shadow;

      pragma interface (assembly, Make_Shadow);

   procedure String_To_Window_Buffer (MESSAGE : in STRING;
      ROW      : in ROW_TYPE;
      COLUMN   : in COLUMN_TYPE;
      COLOUR1,
      COLOUR2 : in SCREEN_COLOUR_TYPE);

      pragma interface (assembly, String_To_Window_Buffer);

   procedure Window_Buffer_Setup (NUMBER_OF_ROW : in ROW_TYPE;
      NUMBER_OF_COLUMNS : in COLUMN_TYPE);

      pragma interface (assembly, Window_Buffer_Setup);

   procedure Window_Buffer_Title (TITLE_NAME : in STRING;
      COLOUR : in SCREEN_COLOUR_TYPE);

      pragma interface (Assembly, Window_Buffer_Title);

   procedure Window_Buffer_To_Screen;

      pragma interface (assembly, Window_Buffer_To_Screen);

   ------------------- Direct Screen Writing Function -------------------

   procedure Main_Title;

      pragma interface (assembly, Main_Title);

   ------------------- Disk functions using DOS interrupts -------------------

   subtype PATH_STRING_TYPE is STRING (1..64);
   subtype FILENAME_TYPE    is STRING (1..12);

   procedure Change_Drive (PATH_STRING : out PATH_STRING_TYPE;
      DRIVE : in CHARACTER);

      pragma interface (assembly, Change_Drive);

   procedure Change_Directory (PATH_STRING : in out PATH_STRING_TYPE);

      pragma interface (assembly, Change_Directory);

   procedure Find_First_File (ERROR : out INTEGER;
      FILENAME : out FILENAME_TYPE;
      MODE     : in MODULEM);

      pragma interface (assembly, Find_First_File);

   procedure Find_Next_File (ERROR : out INTEGER;
      FILENAME : out FILENAME_TYPE);

      pragma interface (assembly, Find_Next_File);

   procedure Get_Drive_Directory (PATH_STRING : in out PATH_STRING_TYPE);

      pragma interface (assembly, Get_Drive_Directory);

   ------------------------------------------------------------------------

end Dos_Environment;
```

Drawer

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Drawer is

   ------------------------------------------------------------------------
   -- Author  : Warren Buckle
   -- Date    : June 1992
   -- Purpose : This draws a schematic of the circuit on the main screen. The
   --           circuit can be scrolled and zoomed. It draws the circuit on a
   --           graphic screen instead of the text screen that is used by the
   --           other processes.
   ------------------------------------------------------------------------

   ------------------------------------------------------------------------
   -- Suppression of these checks are done because the assignment is known to
   -- be correct and it will not cause an error.  Thus these checks will only
   -- add  unnecessary checking code.  Checks  can be  introduced  again by
   -- omitting the SUPPRESS pragma for that check.
   ------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   ------------------------------------------------------------------------

   procedure Draw_Circuit (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   ------------------------------------------------------------------------

end Drawer;
```

```
with  Dos_Environment,
      Graphs,
      Screens,
      Inputter,
      Unchecked_Deallocation;

use   Dos_Environment,
      Graphs,
      Screens;

package body Drawer is

   ------------------------------------------------------------------------
   -- Suppression of these checks are done because the assignment is known to
   -- be correct and it will not cause an error.  Thus these checks will only
   -- add  unnecessary checking code.  Checks  can be  introduced  again by
   -- omitting the SUPPRESS pragma for that check.
   ------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);

   ------------------------------------------------------------------------

   HP_DRAW_THE_NETLIST : constant := 150;
   HP_DRAW_CIRCUIT     : constant := 151;
   HP_DRAW_CIRCUIT2    : constant := 152;

   ------------------------------------------------------------------------
   -- Constants  used  for  drawing  circuit.
   ------------------------------------------------------------------------

   DEF_SOURCE_COLOUR : constant FOREGROUND_TYPE := LIGHT_GREEN;
   MAX_HIGH          : constant                 := MAX_NODES;
   MAX_LOW           : constant                 := MAX_HIGH / 2;
   MAXIMUM_COUNT     : constant                 := 3600;
   NODE_LINE_COLOUR  : constant FOREGROUND_TYPE := WHITE;
   PASSIVE_COLOUR    : constant FOREGROUND_TYPE := LIGHT_CYAN;
   PLUS_SIGN         : constant STRING          := " + ";
   SOURCE_COLOUR     : constant FOREGROUND_TYPE := LIGHT_MAGENTA;
   ZOOMING_FACTOR    : constant                 := 1.25;

   type FORM_TYPE is (FULL, TOP, LEFT);
   type ZOOM_TYPE is (RESET, UP, DOWN);

   type DRAWING_COMPONENT is
      record
         NUMBER    : INTEGER_TYPE := 0;
         HIGH_NODE,
         LOW_NODE  : NODE_TYPE := 0;
      end record;

   type DRAWING_NODE_TYPE is array (1..MAX_NODES) of INTEGER_TYPE;
   type GROUP_TYPE is array (1..MAX_LOW, 1..MAX_HIGH) of INTEGER_TYPE;
   type NUMBER_NODES_TYPE is array (1..MAXIMUM_COUNT) of DRAWING_COMPONENT;

   type DRAW_CIRCUIT_TYPE is
      record
         DRAWING_NODES  : DRAWING_NODE_TYPE;
         NUMBER_NODES   : NUMBER_NODES_TYPE;
         GROUP          : GROUP_TYPE;
      end record;

   type DRAW_ACCESS is access DRAW_CIRCUIT_TYPE;
```

```
   DRAW_CKT : DRAW_ACCESS;

      DRAW_LIST : REST_ARRAY_TYPE (1..2) := (
      (TEXT_LENGTH => 12, TEXT => "Default mode placement", STATE =>TRUE),
      (TEXT_LENGTH => 17, TEXT => "Fast drawing mode",     STATE => TRUE));

   procedure Dispose is new
      Unchecked_Deallocation (DRAW_CIRCUIT_TYPE, DRAW_ACCESS);

-- ------------------------------------------------------------------------
-- PURPOSE : Draws an  ellipse where XC and YC are the centre coordinates,
--           and  AR  and  BR  are the radii  of the  X  axis and  Y  axis
--           respectively. (Adapted from Meridian Graphics Library)
-- ------------------------------------------------------------------------

procedure Ellipse (XC, YC, AR, BR : in INTEGER_TYPE;
   FORM : in FORM_TYPE) is

   X           : INTEGER_TYPE := 0;
   Y           : INTEGER_TYPE := BR;
   A           : LONG_INTEGER := LONG_INTEGER(REAL_TYPE(AR)* ASPECT_RATIO);
   B           : LONG_INTEGER := LONG_INTEGER (BR);
   A_SQUARED   : LONG_INTEGER := A * A;
   A2_SQUARED  : LONG_INTEGER := 2 * A_SQUARED;
   B_SQUARED   : LONG_INTEGER := B * B;
   B2_SQUARED  : LONG_INTEGER := 2 * B_SQUARED;
   D           : LONG_INTEGER := B_SQUARED - A_SQUARED * B + A_SQUARED / 4;
   DX          : LONG_INTEGER := 0;                  -- Initial delta X
   DY          : LONG_INTEGER := A2_SQUARED * B;  -- Initial delta Y

Begin
   while DX < DY loop
      if FORM = FULL then
         Plot_Four_Pixels (X_CENTRE => XC,
            Y_CENTRE => YC,
            X_COORD  => X,
            Y_COORD  => Y);
      else
         Plot (Y => YC - Y,
            X => XC + X);
         if FORM = TOP then
            Plot (Y => YC - Y,
               X => XC - X);
         else
            Plot (Y => YC + Y,
               X => XC + X);
         end if;
      end if;
      if D > 0 then
         Y  := Y - 1;
         DY := DY - A2_SQUARED;
         D  := D - DY;
      end if;
      X  := X + 1;
      DX := DX + B2_SQUARED;
      D  := D + B_SQUARED + DX;
   end loop;

   -- Adjust the new midpoint value
   D := D + (3 * (A_SQUARED - B_SQUARED) / 2 - (DX + DY)) / 2;

   while Y >= 0 loop
      if FORM = FULL then
         Plot_Four_Pixels (X_CENTRE => XC,
            Y_CENTRE => YC,
            X_COORD  => X,
            Y_COORD  => Y);
      else
         Plot (Y => YC - Y,
            X => XC + X);
         if FORM = TOP then
            Plot (Y => YC - Y,
               X => XC - X);
         else
            Plot (Y => YC + Y,
               X => XC + X);
         end if;
      end if;
      if D < 0 then
         X  := X + 1;
         DX := DX + B2_SQUARED;
         D  := D + DX;
      end if;
      Y  := Y - 1;
      DY := DY - A2_SQUARED;
      D  := D + A_SQUARED - DY;
   end loop;
end Ellipse;

-- ------------------------------------------------------------------------
-- PURPOSE : Draws a circle where  XC  and  YC are the centre coordinates,
--           and R is the radius.
-- ------------------------------------------------------------------------

procedure Circle (XC, YC, R : in INTEGER_TYPE;
   FORM : in FORM_TYPE) is

Begin
   Ellipse (XC => XC,
      YC   => YC,
      AR   => R,
      BR   => R,
      FORM => FORM);
end Circle;

-- ------------------------------------------------------------------------
-- PURPOSE : Draws all the components of the circuit.
-- ------------------------------------------------------------------------

procedure Draw_Circuit (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   ARROW1,                 ARROW2,
   ARROW3,                 ARROW4,
   COUNTER,                DELTA_HIGH_NODE,
   DELTA_LEVEL,            DELTA_LOW_NODE,
   DX_ELC_2A,              DY_ELC_2A,
   DX_ELC_2B,              DY_ELC_2B,
   FIRST_LEVEL,            GROUND_LINE,
   H,                      HEIGHT
   HT_PRIM,                HT_PREV,
   L,                      LEAD_COMP,
   LEAD_X,                 MAX,
   MAX_COUNT,              MAX_H,
   MAX_L,                  MAX_LEVEL,
   N,                      SIZE,
   SIZE_C,                 SUB_C,
   SUB_CC,                 V_INC,
   WIDTH,                  WIDTHE,
   X,                      X_SCROLL,
   XS,                     XTI,
   Y,                      Y_SCROLL,
   YS,                     YTI                  : INTEGER_TYPE;
   VALUE,                  ZOOM_FACTOR          : REAL_TYPE;
   CORRECT,                RESPONSE,
   SEARCH_NETLIST,         SLOW_DRAW,
   REDRAW,                 VERTICAL             : BOOLEAN;
   COLOUR                                       : FOREGROUND_TYPE;
   NODE_1@                                      : NODE_TYPE;
   INPUTS                                       : Inputter.INPUT_TYPE;

-- ------------------------------------------------------------------------
-- PURPOSE : Draws a voltage source given certain parameters.
-- ------------------------------------------------------------------------

procedure Draw_Voltage (X, Y, SIZE : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

begin
   Set_Plot_Colour (COLOUR => SOURCE_COLOUR);
   if VERTICAL then
      Circle (XC => X,
         YC   => Y - SIZE,
         R    => SIZE,
         FORM => FULL);
      String_To_Graphic (MESSAGE => "V",
         ROW     => (Y + YTI - 2) / ROW_FACTOR,
         COLUMN  => (X + XTI - 4) / COLUMN_FACTOR,
         COLOURS => SOURCE_COLOUR);
      String_To_Graphic (MESSAGE => PLUS_SIGN,
         ROW     => (Y - XTI) / ROW_FACTOR,
         COLUMN  => (X + XTI - 4) / COLUMN_FACTOR,
         COLOURS => SOURCE_COLOUR);
   else
      Circle (XC => X + SIZE,
         YC   => Y,
         R    => SIZE,
         FORM => FULL);
      String_To_Graphic (MESSAGE => "V",
         ROW     => (Y - YTI) / ROW_FACTOR,
         COLUMN  => (X - 1) / COLUMN_FACTOR,
         COLOURS => SOURCE_COLOUR);
      String_To_Graphic (MESSAGE => PLUS_SIGN,
         ROW     => (Y - YTI) / ROW_FACTOR,
         COLUMN  => (X + SIZE) / COLUMN_FACTOR,
         COLOURS => SOURCE_COLOUR);
   end if;
   Set_Plot_Colour (COLOUR => COLOUR);
end Draw_Voltage;

-- ------------------------------------------------------------------------
-- PURPOSE : Draws  a  voltage  controlled  current  source  given certain
--           parameters.
-- ------------------------------------------------------------------------

procedure Draw_Dependant (X, Y, SIZE, SUB : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

   SD2 : INTEGER_TYPE := SIZE / 2;
   SD3 : INTEGER_TYPE := SIZE / 3;
   X2,                  Y2                   : INTEGER_TYPE;

begin
   Set_Plot_Colour (COLOUR => DEP_SOURCE_COLOUR);
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - LEAD_COMP + SUB);
      Y2 := Y - LEAD_COMP + SUB;
      Line (X1 => X,
         Y1 => Y2 - ARROW1,
         X2 => X,
         Y2 => Y2 - ARROW2);
      Line (X1 => X - ARROW1,
         Y1 => Y2 - ARROW3,
         X2 => X,
         Y2 => Y2 - ARROW1);
      Line (X1 => X + ARROW1,
         Y1 => Y2 - ARROW3,
         X2 => X,
         Y2 => Y2 - ARROW1);
      Line (X1 => X,
         Y1 => Y2,
         X2 => X + SD2,
         Y2 => Y2 - SD2);
      Line (X1 => X + SD2,
         Y1 => Y2 - SD2,
         X2 => X,
         Y2 => Y2 - SIZE);
      Line (X1 => X,
         Y1 => Y2 - SIZE,
         X1 => X - SD2,
         Y2 => Y2 - SD2);
      Line (X1 => X - SD2,
         Y1 => Y2 - SD2,
         X2 => X,
         Y2 => Y2);
      Line (X1 => X,
         Y1 => Y2 - SIZE,
         X2 => X,
         Y2 => Y2 - SIZE - LEAD_COMP + SUB);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_COMP - SUB,
         Y2 => Y);
      X2 := X + LEAD_COMP - SUB;
      Line (X1 => X2 + ARROW1,
         Y1 => Y,
```

```
            X2 => X2 + ARROW2,
            Y2 => Y);
         Line (X1 => X2 + ARROW3,
            Y1 => Y - ARROW4,
            X2 => X2 + ARROW1,
            Y2 => Y);
         Line (X1 => X2 + ARROW3,
            Y1 => Y + ARROW4,
            X2 => X2 + ARROW1,
            Y2 => Y);
         Line (X1 => X2,
            Y1 => Y,
            X2 => X2 + SD2,
            Y2 => Y + SD3);
         Line (X1 => X2 + SD2,
            Y1 => Y + SD3,
            X2 => X2 + SIZE,
            Y2 => Y);
         Line (X1 => X2 + SIZE,
            Y1 => Y,
            X2 => X2 + SD2,
            Y2 => Y - SD3);
         Line (X1 => X2 + SD2,
            Y1 => Y - SD3,
            X2 => X2,
            Y2 => Y);
         Line (X1 => X1 + SIZE,
            Y1 => Y,
            X1 => X2 + SIZE + LEAD_COMP + SD2 - SUB,
            Y2 => Y);
      end if;
      Set_Plot_Colour (COLOUR => COLOUR);
   end Draw_Dependent;

   -- ------------------------------------------------------------------
   -- PURPOSE : Draws a current source given certain parameters.
   -- ------------------------------------------------------------------

   procedure Draw_Current (X, Y, SIZE, SUB : in INTEGER_TYPE;
      VERTICAL : in BOOLEAN) is

      X2,                        Y2                : INTEGER_TYPE;

   begin
      Set_Plot_Colour (COLOUR => SOURCE_COLOUR);
      if VERTICAL then
         Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - LEAD_COMP + SUB - ARROW4);
         Y2 := Y - LEAD_COMP + SUB;
         Circle (XC => X,
            YC   => Y2 - SIZE,
            R    => SIZE,
            FORM => FULL);
         Line (X1 => X,
            Y1 => Y2 - 2 * SIZE + ARROW4,
            X2 => X,
            Y2 => Y2 - 2 * SIZE - LEAD_COMP + SUB - ARROW4);
         Line (X1 => X,
            Y1 => Y2 - ARROW1,
            X2 => X,
            Y2 => Y2 - ARROW2);
         Line (X1 => X - ARROW1,
            Y1 => Y2 - ARROW3,
            X2 => X,
            Y2 => Y2 - ARROW1);
         Line (X1 => X,
            Y1 => Y,
            X2 => X + LEAD_COMP - SUB,
            Y2 => Y);
         X2 := X + LEAD_COMP - SUB;
         Circle (XC => X2 + SIZE,
            YC   => Y,
            R    => SIZE,
            FORM => FULL);
         Line (X1 => X2 + 2 * SIZE,
            Y1 => Y,
            X2 => X2 + 2 * SIZE + LEAD_COMP + SIZE - SUB,
            Y2 => Y);
         Line (X1 => X2 + ARROW1,
            Y1 => Y,
            X2 => X2 + ARROW2,
            Y2 => Y);
         Line (X1 => X2 + ARROW3,
            Y1 => Y - ARROW1,
            X2 => X2 + ARROW1,
            Y2 => Y);
         Line (X1 => X2 + ARROW3,
            Y1 => Y + ARROW1,
            X2 => X2 + ARROW1,
            Y2 => Y);
      end if;
      Set_Plot_Colour (COLOUR => COLOUR);
   end Draw_Current;

   -- ------------------------------------------------------------------
   -- PURPOSE : Draws a passive component  given certain parameters.  Passive
   --           components are capacitors, inductors and resistors.
   -- ------------------------------------------------------------------

   procedure Draw_Passive (X, Y, WIDTH, HEIGHT : in INTEGER_TYPE;
      SYMBOL   : in CHARACTER;
      VERTICAL : in BOOLEAN) is

      WD2                                 : INTEGER_TYPE := WIDTH / 2;
      XMW2,                  XPW2,
      YMW2,                  YPW2,
      HEIGHT2                             : INTEGER_TYPE;

   begin
      Set_Plot_Colour (COLOUR => PASSIVE_COLOUR);
      if VERTICAL then
         XMW2 := X - WD2;
         XPW2 := X + WD2;
      else
         YMW2 := Y - WD2;
         YPW2 := Y + WD2;
      end if;
      case SYMBOL is
         when 'C' =>
            HEIGHT2 := HEIGHT / 10;
            if VERTICAL then
               Line (X1 => X,
                  Y1 => Y,
                  X2 => X,
                  Y2 => Y - 3 * HEIGHT2);
               Line (X1 => XMW2,
                  Y1 => Y - 3 * HEIGHT2,
                  X2 => XPW2,
                  Y2 => Y - 3 * HEIGHT2);
               Line (X1 => XMW2,
                  Y1 => Y - 7 * HEIGHT2,
                  X2 => XPW2,
                  Y2 => Y - 7 * HEIGHT2);
               Line (X1 => X,
                  Y1 => Y - 7 * HEIGHT2,
                  X2 => X,
                  Y2 => Y - HEIGHT);
            else
               Line (X1 => X,
                  Y1 => Y,
                  X1 => X + 3 * HEIGHT2,
                  Y2 => Y);
               Line (X1 => X + 3 * HEIGHT2,
                  Y1 => YMW2,
                  X2 => X + 3 * HEIGHT2,
                  Y2 => YPW2);
               Line (X1 => X + 7 * HEIGHT2,
                  Y1 => YMW2,
                  X2 => X + 7 * HEIGHT2,
                  Y2 => YPW2);
               Line (X1 => X + 7 * HEIGHT2,
                  Y1 => Y,
                  X2 => X + HEIGHT,
                  Y2 => Y);
            end if;
         when 'R' =>
            HEIGHT2 := HEIGHT / 7;
            if VERTICAL then
               Line (X1 => X,
                  Y1 => Y,
                  X2 => XMW2,
                  Y2 => Y - HEIGHT2);
               Line (X1 => XMW2,
                  Y1 => Y - HEIGHT2,
                  X2 => XPW2,
                  Y2 => Y - 2 * HEIGHT2);
               Line (X1 => XPW2,
                  Y1 => Y - 2 * HEIGHT2,
                  X2 => XMW2,
                  Y2 => Y - 3 * HEIGHT2);
               Line (X1 => XMW2,
                  Y1 => Y - 3 * HEIGHT2,
                  X2 => XPW2,
                  Y2 => Y - 4 * HEIGHT2);
               Line (X1 => XPW2,
                  Y1 => Y - 4 * HEIGHT2,
                  X2 => XMW2,
                  Y2 => Y - 5 * HEIGHT2);
               Line (X1 => XMW2,
                  Y1 => Y - 5 * HEIGHT2,
                  X2 => XPW2,
                  Y2 => Y - 6 * HEIGHT2);
               Line (X1 => XPW2,
                  Y1 => Y - 6 * HEIGHT2,
                  X2 => X,
                  Y2 => Y - HEIGHT);
            else
               Line (X1 => X,
                  Y1 => Y,
                  X2 => X + HEIGHT2,
                  Y2 => YMW2);
               Line (X1 => X + HEIGHT2,
                  Y1 => YMW2,
                  X2 => X + 1 * HEIGHT2,
                  Y2 => YPW2);
               Line (X1 => X + 2 * HEIGHT2,
                  Y1 => YPW2,
                  X2 => X + 3 * HEIGHT2,
                  Y2 => YMW2);
               Line (X1 => X + 3 * HEIGHT2,
                  Y1 => YMW2,
                  X2 => X + 4 * HEIGHT2,
                  Y2 => YPW2);
               Line (X1 => X + 4 * HEIGHT2,
                  Y1 => YPW2,
                  X2 => X + 5 * HEIGHT2,
                  Y2 => YMW2);
               Line (X1 => X + 5 * HEIGHT2,
                  Y1 => YMW2,
                  X2 => X + 6 * HEIGHT2,
                  Y2 => YPW2);
               Line (X1 => X + 6 * HEIGHT2,
                  Y1 => YPW2,
                  X2 => X + HEIGHT,
                  Y2 => Y);
            end if;
         when 'L' =>
            HEIGHT2 := HEIGHT / 8;
            if VERTICAL then
               Circle (XC => X,
                  YC   => Y - HEIGHT2,
                  R    => HEIGHT2 + 1,
                  FORM => LEFT);
               Circle (XC => X,
                  YC   => Y - HEIGHT / 2,
                  R    => HEIGHT2 + 1,
                  FORM => LEFT);
               Circle (XC => X,
                  YC   => Y - HEIGHT + HEIGHT2,
                  R    => HEIGHT2 + 1,
                  FORM => LEFT);
            else
               Circle (XC => X + HEIGHT2,
```

```
            YC   => Y,
            R    => HEIGHT2 - 1,
            FORM => TOP);
         Circle (XC => X + HEIGHT / 2,
            YC   => Y,
            R    => HEIGHT2 - 1,
            FORM => TOP);
         Circle (XC => X + HEIGHT - HEIGHT2,
            YC   => Y,
            R    => HEIGHT2 - 1,
            FORM => TOP);
      end if;
   when others => null;
end case;
Set_Plot_Colour (COLOUR => COLOUR);
end Draw_Passive;

-- PURPOSE : Draws a passive component for a branch in the network given
--           certain parameters.

procedure Draw_1_RLC (VOLT_EXISTS, VERTICAL : in BOOLEAN;
   X, Y, SUB : in INTEGER_TYPE;
   SYMBOL    : in CHARACTER) is

   LEAD_PASSIVE   : INTEGER_TYPE := LEAD_COMP - SUB;
   Y2             : INTEGER_TYPE := Y;

begin
   WIDTH := WIDTH2;
   if VERTICAL then
      HEIGHT := HZ_PELV)
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - LEAD_PASSIVE);
   else
      HEIGHT := HZ_PELV * 2;
      WIDTH  := WIDTH / 2;
      Circle (XC => X,
         YC   => Y,
         R    => 1,
         FORM => FULL);
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_PASSIVE,
         Y2 => Y);
   end if;
   if VOLT_EXISTS and (SYMBOL = 'L') then
      Y2 := Y + 1;
   end if;
   if VERTICAL then
      Draw_Passive (X => X,
         Y      => Y2 - LEAD_PASSIVE,
         WIDTH  => WIDTH,
         HEIGHT => HEIGHT,
         SYMBOL => SYMBOL,
         VERTICAL => VERTICAL);
      Line (X1 => X,
         Y1 => Y2 - LEAD_PASSIVE - HEIGHT,
         X2 => X,
         Y2 => Y2 - 2 * LEAD_PASSIVE - HEIGHT + 1);
   else
      Draw_Passive (X => X + LEAD_PASSIVE,
         Y      => Y2,
         WIDTH  => WIDTH,
         HEIGHT => HEIGHT,
         SYMBOL => SYMBOL,
         VERTICAL => VERTICAL);
      Line (X1 => X + LEAD_PASSIVE + HEIGHT,
         Y1 => Y2,
         X2 => X + 2 * LEAD_PASSIVE + HEIGHT,
         Y2 => Y2);
   end if;
end Draw_1_RLC;

-- PURPOSE : Draws two different passive components for a branch in the
--           network given certain parameters.

procedure Draw_2_RLC (VOLT_EXISTS, VERTICAL : in BOOLEAN;
   X, Y, SUB      : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER) is

   X2 : INTEGER_TYPE := X;
   Y2 : INTEGER_TYPE := Y;

begin
   if VERTICAL then
      X2 := X2 - DX_RLC_2A;
   end if;
   Draw_1_RLC (VOLT_EXISTS => VOLT_EXISTS,
      VERTICAL => VERTICAL,
      X        => X2,
      Y        => Y2,
      SUB      => SUB,
      SYMBOL   => SYMBOL1);
   if VERTICAL then
      Line (X1 => X2,
         Y1 => Y2 - DELTA_LOW_NODE + 2 * SUB,
         X2 => X2 + DX_RLC_2B,
         Y2 => Y2 - DELTA_LOW_NODE + 2 * SUB);
      X2 := X2 + DX_RLC_2B;
   else
      Line (X1 => X2,
         Y1 => Y2,
         X2 => X2,
         Y2 => Y2 - DY_RLC_2A);
      if VOLT_EXISTS then
         X2 := X2 - DX_RLC_2B;
      end if;
      Y2 := Y2 - DY_RLC_2A;
      Line (X1 => X2 + DELTA_HIGH_NODE - SUB,
         Y1 => Y2,
         X2 => X2 + DELTA_HIGH_NODE - SUB,
         Y2 => Y2 + DY_RLC_2A);
      if VOLT_EXISTS then
         X2 := X2 + DX_RLC_2B;
      end if;
   end if;
   Draw_1_RLC (VOLT_EXISTS => VOLT_EXISTS,
      VERTICAL => VERTICAL,
      X        => X2,
      Y        => Y2,
      SUB      => SUB,
      SYMBOL   => SYMBOL2);
end Draw_2_RLC;

-- PURPOSE : Draws three different passive components for a branch in the
--           network given certain parameters.

procedure Draw_3_RLC (VOLT_EXISTS, VERTICAL : in BOOLEAN;
   X, Y, SUB : in INTEGER_TYPE) is

   X2 : INTEGER_TYPE := X;
   Y2 : INTEGER_TYPE := Y;

begin
   Draw_2_RLC (VOLT_EXISTS => VOLT_EXISTS,
      VERTICAL => VERTICAL,
      X        => X2,
      Y        => Y2,
      SUB      => SUB,
      SYMBOL1  => 'R',
      SYMBOL2  => 'L');
   if VERTICAL then
      X2 := X2 - DX_RLC_2A;
      Line (X1 => X2,
         Y1 => Y2 - DELTA_LOW_NODE + 2 * SUB,
         X2 => X2 + 2 * DX_RLC_2B,
         Y2 => Y2 - DELTA_LOW_NODE + 2 * SUB);
      X2 := X2 + 2 * DX_RLC_2B;
      Y2 := Y2 + DY_RLC_2A;
   else
      Line (X1 => X2,
         Y1 => Y2,
         X2 => X2,
         Y2 => Y2 - DY_RLC_2B);
      if VOLT_EXISTS then
         Line (X1 => X2 + DELTA_HIGH_NODE - DX_RLC_2B - SUB,
            Y1 => Y2,
            X1 => X2 + DELTA_HIGH_NODE - DX_RLC_2B - SUB,
            Y2 => Y2 - DY_RLC_2B);
      else
         Line (X1 => X2 + DELTA_HIGH_NODE,
            Y1 => Y2,
            X1 => X2 + DELTA_HIGH_NODE,
            Y2 => Y2 - DY_RLC_2B);
      end if;
      Y2 := Y2 - DY_RLC_2B + DY_RLC_2A;
   end if;
   Draw_1_RLC (VOLT_EXISTS => VOLT_EXISTS,
      VERTICAL => VERTICAL,
      X        => X2,
      Y        => Y2 - DY_RLC_2A,
      SUB      => SUB,
      SYMBOL   => 'C');
end Draw_3_RLC;

-- PURPOSE : Draws a voltage controlled current source and a passive
--           component given certain parameters.

procedure Draw_Dependent_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL   : in CHARACTER;
   VERTICAL : in BOOLEAN) is

   DX_DEPENDENT : INTEGER_TYPE := DX_RLC_2B / 2;
   DY_DEPENDENT : INTEGER_TYPE := DY_RLC_2A / 2;

begin
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - LEAD_E);
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X,
         Y2 => Y - DELTA_LOW_NODE);
      Line (X1 => X - DX_DEPENDENT,
         Y1 => Y - LEAD_E,
         X2 => X + DX_RLC_2B,
         Y2 => Y - LEAD_E);
      Line (X1 => X - DX_DEPENDENT,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X + DX_RLC_2B,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Dependent (X => X - DX_DEPENDENT,
         Y      => Y - LEAD_E,
         SIZE   => SIZE_C,
         SUB    => SUB_C,
         VERTICAL => TRUE);
      Draw_1_RLC (VOLT_EXISTS => TRUE,
         VERTICAL => VERTICAL,
         X        => X + DX_RLC_2B,
         Y        => Y - LEAD_E,
         SUB      => SUB_C,
         SYMBOL   => SYMBOL);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
         X1 => X + DELTA_HIGH_NODE - LEAD_E,
```

```
      Y2 => Y);
   Draw_Dependent (X => X + LEAD_E,
      Y       => Y,
      SIZE    => SIZE_C,
      SUB     => SUB_C,
      VERTICAL => FALSE);
   Draw_1_RLC (VOLT_EXISTS => TRUE,
      VERTICAL => VERTICAL,
      X       => X + LEAD_E,
      Y       => Y - DY_RLC_2A - DY_DEPENDENT,
      SUB     => SUB_C,
      SYMBOL  => SYMBOL);
  end if;
end Draw_Dependent_1_RLC;

-- PURPOSE : Draws a  voltage controlled current source  and two different
--           passive components given certain parameters.

procedure Draw_Dependent_2_RLC (X, Y : in INTEGER_TYPE;
  SYMBOL1, SYMBOL2 : in CHARACTER;
  VERTICAL         : in BOOLEAN) is

  DY_DEPENDENT : INTEGER_TYPE := DY_RLC_2A / 2;
  DX_DEPENDENT : INTEGER_TYPE := DX_RLC_2B / 2;
  X2           : INTEGER_TYPE;

begin
  if VERTICAL then
    Line (X1 => X,
      Y1 => Y - DELTA_LOW_NODE,
      X2 => X - DX_DEPENDENT,
      Y2 => Y - DELTA_LOW_NODE);
    X2 := X - DX_DEPENDENT;
    Line (X1 => X2,
      Y1 => Y,
      X2 => X2,
      Y2 => Y - LEAD_E);
    Line (X1 => X,
      Y1 => Y - DELTA_LOW_NODE + LEAD_E,
      X2 => X2,
      Y2 => Y - DELTA_LOW_NODE);
    Line (X1 => X2 - DX_DEPENDENT,
      Y1 => Y - LEAD_E,
      X2 => X2 + DX_RLC_2B,
      Y2 => Y - LEAD_E);
    Line (X1 => X2 - DX_DEPENDENT,
      Y1 => Y - DELTA_LOW_NODE + LEAD_E,
      X2 => X2 + DX_RLC_2B,
      Y2 => Y - DELTA_LOW_NODE + LEAD_E);
    Draw_Dependent (X => X2 - DX_DEPENDENT,
      Y       => Y - LEAD_E,
      SIZE    => SIZE_C,
      SUB     => SUB_C,
      VERTICAL => TRUE);
    Draw_1_RLC (VOLT_EXISTS => TRUE,
      VERTICAL => VERTICAL,
      X       => X2 + DX_RLC_2B + DX_DEPENDENT,
      Y       => Y - LEAD_E,
      SUB     => SUB_C,
      SYMBOL1 => SYMBOL1,
      SYMBOL2 => SYMBOL2);
    Line (X1 => X2,
      Y1 => Y - LEAD_E,
      X2 => X2 + DX_RLC_2A + DX_RLC_2B,
      Y2 => Y - LEAD_E);
  else
    Line (X1 => X,
      Y1 => Y,
      X2 => X + LEAD_E,
      Y2 => Y);
    Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
      Y1 => Y,
      X2 => X + DELTA_HIGH_NODE,
      Y2 => Y);
    Line (X1 => X + LEAD_E,
      Y1 => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
      X2 => X + LEAD_E,
      Y2 => Y);
    Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
      Y1 => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
      X2 => X + DELTA_HIGH_NODE - LEAD_E,
      Y2 => Y);
    Draw_Dependent (X => X + LEAD_E,
      Y       => Y,
      SIZE    => SIZE_C,
      SUB     => SUB_C,
      VERTICAL => FALSE);
    Draw_1_RLC (VOLT_EXISTS => FALSE,
      VERTICAL => VERTICAL,
      X       => X + LEAD_E,
      Y       => Y - DY_RLC_2A - DY_DEPENDENT,
      SUB     => SUB_C,
      SYMBOL  => 'R');
    Draw_1_RLC (VOLT_EXISTS => FALSE,
      VERTICAL => VERTICAL,
      X       => X + LEAD_E,
      Y       => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
      SUB     => SUB_C,
      SYMBOL  => 'L');
    Draw_1_RLC (VOLT_EXISTS => FALSE,
      VERTICAL => VERTICAL,
      X       => X + LEAD_E,
      Y       => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
      SUB     => SUB_C,
      SYMBOL  => 'C');
  end if;
end Draw_Dependent_3_RLC;

-- PURPOSE : Draws a current source  and a passive component given certain
--           parameters.

procedure Draw_Current_1_RLC (X, Y : in INTEGER_TYPE;
  SYMBOL   : in CHARACTER;
  VERTICAL : in BOOLEAN) is

  DY_DEPENDENT : INTEGER_TYPE := DY_RLC_2A / 2;
  DX_DEPENDENT : INTEGER_TYPE := DX_RLC_2B / 2;

begin
  if VERTICAL then
    Line (X1 => X,
      Y1 => Y,
      X2 => X,
      Y2 => Y - LEAD_E);
    Line (X1 => X,
      Y1 => Y - DELTA_LOW_NODE + LEAD_E,
      X2 => X,
      Y2 => Y - DELTA_LOW_NODE);
    Line (X1 => X - DX_DEPENDENT,
      Y1 => Y - LEAD_E,
      X2 => X + DX_RLC_2B,
      Y2 => Y - LEAD_E);
    Line (X1 => X - DX_DEPENDENT,
      Y1 => Y - DELTA_LOW_NODE + LEAD_E,
      X2 => X + DX_RLC_2B,
      Y2 => Y - DELTA_LOW_NODE + LEAD_E);
    Draw_Current (X => X - DX_DEPENDENT,
      Y       => Y - LEAD_E,
      SIZE    => SIZE_C / 2,
      SUB     => DY_RLC_2A,
      VERTICAL => TRUE);
    Draw_1_RLC (VOLT_EXISTS => TRUE,
      VERTICAL => VERTICAL,
      X       => X + DX_RLC_2B,
      Y       => Y - LEAD_E,
      SUB     => SUB_C,
      SYMBOL  => SYMBOL);
  else
    Line (X1 => X,
      Y1 => Y,
      X2 => X + LEAD_E,
```

```
      Y2 => Y);
   Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
      Y1 => Y,
      X2 => X + DELTA_HIGH_NODE,
      Y2 => Y);
   Line (X1 => X + LEAD_E,
      Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
      X2 => X + LEAD_E,
      Y2 => Y);
   Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
      Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
      X2 => X + DELTA_HIGH_NODE - LEAD_E,
      Y2 => Y);
   Draw_Current (X => X + LEAD_E,
      Y       => Y,
      SIZE    => SIZE_C / 2,
      SUB     => DY_RLC_2A,
      VERTICAL => FALSE);
   Draw_1_RLC (VOLT_EXISTS => TRUE,
      VERTICAL => VERTICAL,
      X       => X + LEAD_E,
      Y       => Y - DY_RLC_2A - DY_DEPENDENT,
      SUB     => SUB_C,
      SYMBOL  => SYMBOL);
 end if;
end Draw_Current_1_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a  current source  and two different passive components
--           given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL         : in BOOLEAN) is

   DY_DEPENDENT  : INTEGER_TYPE := DY_RLC_2A / 2;
   DX_DEPENDENT  : INTEGER_TYPE := DX_RLC_2B / 2;
   X2            : INTEGER_TYPE;

begin
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE,
         X2 => X - DX_DEPENDENT,
         Y2 => Y - DELTA_LOW_NODE);
      X2 := X - DX_DEPENDENT;
      Line (X1 => X2,
         Y1 => Y,
         X2 => X2,
         Y2 => Y - LEAD_E);
      Line (X1 => X2,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X2,
         Y2 => Y - DELTA_LOW_NODE);
      Line (X1 => X2 - DX_RLC_2B,
         Y1 => Y - LEAD_E,
         X2 => X2 + DX_RLC_2B,
         Y2 => Y - LEAD_E);
      Line (X1 => X2 - DX_DEPENDENT,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X2 + DX_RLC_2B,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Current (X => X2 - DX_DEPENDENT,
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C / 2,
         SUB     => DY_RLC_2A,
         VERTICAL => TRUE);
      Draw_2_RLC (VOLT_EXISTS => TRUE,
         VERTICAL => VERTICAL,
         X       => X2 + DX_RLC_2B + DX_DEPENDENT,
         Y       => Y - LEAD_E,
         SUB     => SUB_C,
         SYMBOL1  => SYMBOL1,
         SYMBOL2  => SYMBOL2);
      Line (X1 => X2,
         Y1 => Y - LEAD_E,
         X2 => X2 + DX_RLC_2A + DX_RLC_2B,
         Y2 => Y - LEAD_E);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y);
      Draw_Current (X => X + LEAD_E,
         Y       => Y,
         SIZE    => SIZE_C / 2,
         SUB     => DY_RLC_2A,
         VERTICAL => FALSE);
      Draw_1_RLC (VOLT_EXISTS => FALSE,
         VERTICAL => VERTICAL,
         X       => X + LEAD_E,
         Y       => Y - DY_RLC_2A - DY_DEPENDENT,
         SUB     => SUB_C,
         SYMBOL   => SYMBOL1);
      Draw_1_RLC (VOLT_EXISTS => FALSE,
         VERTICAL => VERTICAL,
         X       => X + LEAD_E,
         Y       => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
         SUB     => SUB_C,
         SYMBOL   => SYMBOL2);
   end if;
end Draw_Current_2_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a current source and three different passive components
--           given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

   DY_DEPENDENT  : INTEGER_TYPE := DY_RLC_2A / 2;
   X2            : INTEGER_TYPE;

begin
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE,
         X2 => X - DX_RLC_2A,
         Y2 => Y - DELTA_LOW_NODE);
      X2 := X - DX_RLC_2A;
      Line (X1 => X2,
         Y1 => Y,
         X2 => X2,
         Y2 => Y - LEAD_E);
      Line (X1 => X2,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X2,
         Y2 => Y - DELTA_LOW_NODE);
      Line (X1 => X2 - DX_RLC_2B,
         Y1 => Y - LEAD_E,
         X2 => X2 + DX_RLC_2B / 2,
         Y2 => Y - LEAD_E);
      Line (X1 => X2 - DX_RLC_2B,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X2 + DX_RLC_2B / 2,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Current (X => X2 - DX_RLC_2B,
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C / 2,
         SUB     => DY_RLC_2A,
         VERTICAL => TRUE);
      Draw_3_RLC (VOLT_EXISTS => TRUE,
         VERTICAL => VERTICAL,
         X       => X2 + DX_RLC_2B,
         Y       => Y - LEAD_E,
         SUB     => SUB_C);
      Line (X1 => X2,
         Y1 => Y - LEAD_E,
         X2 => X2 + 2 * DX_RLC_2B + DX_RLC_2B / 4,
         Y2 => Y - LEAD_E);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y);
      Draw_Current (X => X + LEAD_E,
         Y       => Y,
         SIZE    => SIZE_C / 2,
         SUB     => DY_RLC_2A,
         VERTICAL => FALSE);
      Draw_1_RLC (VOLT_EXISTS => FALSE,
         VERTICAL => VERTICAL,
         X       => X + LEAD_E,
         Y       => Y - DY_RLC_2A - DY_DEPENDENT,
         SUB     => SUB_C,
         SYMBOL   => 'R');
      Draw_1_RLC (VOLT_EXISTS => FALSE,
         VERTICAL => VERTICAL,
         X       => X + LEAD_E,
         Y       => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
         SUB     => SUB_C,
         SYMBOL   => 'L');
      Draw_1_RLC (VOLT_EXISTS => FALSE,
         VERTICAL => VERTICAL,
         X       => X + LEAD_E,
         Y       => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
         SUB     => SUB_C,
         SYMBOL   => 'C');
   end if;
end Draw_Current_3_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a  voltage source and a passive component given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Voltage_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL   : in CHARACTER;
   VERTICAL : in BOOLEAN) is

   LEAD_2,                SIZE2,
   X2                                  : INTEGER_TYPE;

begin
   if VERTICAL then
      LEAD_2 := LEAD_E / 2;
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - LEAD_2 - ARROW4);
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X,
         Y2 => Y - DELTA_LOW_NODE);
      Draw_Voltage (X => X,
         Y       => Y - LEAD_2,
         SIZE    => SIZE_C / 2,
         VERTICAL => VERTICAL);
      X2 := X - DX_RLC_2B / 4;
      Draw_1_RLC (VOLT_EXISTS => TRUE,
         VERTICAL => VERTICAL,
         X       => X2 + DX_RLC_2B / 4,
         Y       => Y - SIZE_C - LEAD_2 + ARROW4,
         SUB     => DX_RLC_2A,
         SYMBOL   => SYMBOL);
```

```
   else
      SIZE2 := INTEGER_TYPE (REAL_TYPE (SIZE_C) * 0.4);
      Line (X1 => X,
            Y1 => Y,
            X2 => X + LEAD_E,
            Y2 => Y);
      Draw_Voltage (X => X + LEAD_E,
                    Y        => Y,
                    SIZE     => SIZE2 / 2,
                    VERTICAL => VERTICAL);
      Draw_1_RLC (VOLT_EXISTS => TRUE,
                  VERTICAL => VERTICAL,
                  X        => X + SIZE2 + LEAD_E,
                  Y        => Y,
                  SUB      => NT_PRLV,
                  SYMBOL   => SYMBOL);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E,
            Y2 => Y);
   end if;
end Draw_Voltage_1_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage source and two different passive components
--           given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Voltage_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL         : in BOOLEAN) is

   LEAD_2,                 SIZE2                : INTEGER_TYPE;

begin
   if VERTICAL then
      LEAD_2 := LEAD_E / 2;
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - LEAD_2 - ARROW4);
      Draw_Voltage (X => X,
                    Y        => Y - LEAD_2,
                    SIZE     => SIZE_C / 2,
                    VERTICAL => VERTICAL);
      Line (X1 => X,
            Y1 => Y - LEAD_2 - SIZE_C + ARROW4,
            X2 => X,
            Y2 => Y - LEAD_2 - SIZE_C - ARROW4);
      Line (X1 => X - DX_RLC_2B / 2,
            Y1 => Y - LEAD_2 - SIZE_C - ARROW4,
            X2 => X + DX_RLC_2B / 2,
            Y2 => Y - LEAD_2 - SIZE_C - ARROW4);
      Draw_2_RLC (VOLT_EXISTS => TRUE,
                  VERTICAL => VERTICAL,
                  X        => X + DX_RLC_2B / 4,
                  Y        => Y - SIZE_C - LEAD_2 - ARROW4,
                  SUB      => 2 * SUB_C,
                  SYMBOL1  => SYMBOL1,
                  SYMBOL2  => SYMBOL2);
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE - ARROW4 + LEAD_E,
            X2 => X,
            Y2 => Y - DELTA_LOW_NODE);
   else
      SIZE2 := INTEGER_TYPE (REAL_TYPE (SIZE_C) * 0.8);
      Line (X1 => X,
            Y1 => Y,
            X2 => X + LEAD_E,
            Y2 => Y);
      Draw_Voltage (X => X + LEAD_E,
                    Y        => Y,
                    SIZE     => SIZE2 / 2,
                    VERTICAL => VERTICAL);
      Line (X1 => X + SIZE2 + LEAD_E,
            Y1 => Y,
            X2 => X + SIZE2 + LEAD_E + ARROW4,
            Y2 => Y);
      Line (X1 => X + SIZE2 + LEAD_E + ARROW4,
            Y1 => Y,
            X2 => X + SIZE2 + LEAD_E + ARROW4,
            Y2 => Y - DX_RLC_2B / 2);
      Draw_2_RLC (VOLT_EXISTS => TRUE,
                  VERTICAL => VERTICAL,
                  X        => X + SIZE2 + LEAD_E + ARROW4,
                  Y        => Y,
                  SUB      => NT_PRLV,
                  SYMBOL1  => SYMBOL1,
                  SYMBOL2  => SYMBOL2);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y);
   end if;
end Draw_Voltage_2_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage source and three different passive components
--           given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Voltage_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

   LEAD_2,                 SIZE2                : INTEGER_TYPE;

begin
   if VERTICAL then
      LEAD_2 := LEAD_E / 2;
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - LEAD_2 - ARROW4);
      Draw_Voltage (X => X,
                    Y        => Y - LEAD_2,
                    SIZE     => SIZE_C / 2,
                    VERTICAL => VERTICAL);
      Line (X1 => X,
            Y1 => Y - LEAD_2 - SIZE_C + ARROW4,
            X2 => X,
            Y2 => Y - LEAD_2 - SIZE_C - ARROW4);
      Line (X1 => X - DX_RLC_2B,
            Y1 => Y - LEAD_2 - SIZE_C - ARROW4,
            X2 => X + DX_RLC_2B,
            Y2 => Y - LEAD_2 - SIZE_C - ARROW4);
      Draw_3_RLC (VOLT_EXISTS => TRUE,
                  VERTICAL => VERTICAL,
                  X        => X - DX_RLC_2B / 4,
                  Y        => Y - SIZE_C - LEAD_2 - ARROW4,
                  SUB      => 2 * SUB_C);
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E,
            X2 => X,
            Y2 => Y - DELTA_LOW_NODE);
   else
      SIZE2 := INTEGER_TYPE (REAL_TYPE (SIZE_C) * 0.8);
      Line (X1 => X,
            Y1 => Y,
            X2 => X + LEAD_E,
            Y2 => Y);
      Draw_Voltage (X => X + LEAD_E,
                    Y        => Y,
                    SIZE     => SIZE2 / 2,
                    VERTICAL => VERTICAL);
      Line (X1 => X + SIZE2 + LEAD_E,
            Y1 => Y,
            X2 => X + SIZE2 + LEAD_E + ARROW4,
            Y2 => Y);
      Draw_3_RLC (VOLT_EXISTS => TRUE,
                  VERTICAL => VERTICAL,
                  X        => X + SIZE2 + LEAD_E + ARROW4,
                  Y        => Y,
                  SUB      => NT_PRLV);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y);
   end if;
end Draw_Voltage_3_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a voltage source,
--           and a passive component given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Dependent_Voltage_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL   : in CHARACTER;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Voltage_1_RLC (X => X,
                       Y        => Y,
                       SYMBOL   => SYMBOL,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
                      Y        => Y,
                      SIZE     => SIZE_C,
                      SUB      => SUB_C / 2,
                      VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 2 * DY_RLC_2A,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E,
            Y2 => Y - 2 * DY_RLC_2A);
      Draw_Dependent (X => X,
                      Y        => Y - 2 * DY_RLC_2A,
                      SIZE     => SIZE_C,
                      SUB      => SUB_C,
                      VERTICAL => FALSE);
   end if;
end Draw_Dependent_Voltage_1_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a voltage source,
--           and two different passive components given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Dependent_Voltage_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL         : in BOOLEAN) is

begin
   Draw_Voltage_2_RLC (X => X,
                       Y        => Y,
                       SYMBOL1  => SYMBOL1,
                       SYMBOL2  => SYMBOL2,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E + V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X + 2 * DX_RLC_2B,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
                      Y        => Y,
                      SIZE     => SIZE_C,
                      SUB      => SUB_C / 2,
                      VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 2 * DY_RLC_2A);
```

```
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 2 * DY_RLC_2A,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E,
            Y2 => Y - 2 * DY_RLC_2A);
      Draw_Dependent (X => X,
                      Y => Y - 2 * DY_RLC_2A,
                      SIZE => SIZE_C,
                      SUB => SUB_C,
                      VERTICAL => FALSE);
   end if;
end Draw_Dependent_Voltage_2_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a voltage source,
--           and three different passive components given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Dependent_Voltage_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Voltage_3_RLC (X => X,
                       Y => Y,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X + 2 * DX_RLC_2B,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
                      Y => Y,
                      SIZE => SIZE_C,
                      SUB => SUB_C / 2,
                      VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 3 * DY_RLC_3A - ARROW4,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Draw_Dependent (X => X,
                      Y => Y - 3 * DY_RLC_3A - ARROW4,
                      SIZE => SIZE_C,
                      SUB => SUB_C,
                      VERTICAL => FALSE);
   end if;
end Draw_Dependent_Voltage_3_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a current source, a voltage source, and a passive
--           component given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Voltage_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL  : in CHARACTER;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Voltage_1_RLC (X => X,
                       Y => Y,
                       SYMBOL => SYMBOL,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Line (X1 => X + 2 * DX_RLC_2B,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Current (X => X + 2 * DX_RLC_2B,
                    Y => Y,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 2 * DY_RLC_2A,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E - ARROW4,
            Y2 => Y - 2 * DY_RLC_2A);
      Draw_Current (X => X,
                    Y => Y - 2 * DY_RLC_2A,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => FALSE);
   end if;
end Draw_Current_Voltage_1_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a current source, a voltage source, and two different
--           passive components given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Voltage_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL         : in BOOLEAN) is

begin
   Draw_Voltage_2_RLC (X => X,
                       Y => Y,
                       SYMBOL1 => SYMBOL1,
                       SYMBOL2 => SYMBOL2,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X + 2 * DX_RLC_2B,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Current (X => X + 2 * DX_RLC_2B,
                    Y => Y,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 2 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 2 * DY_RLC_2A,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E - ARROW4,
            Y2 => Y - 2 * DY_RLC_2A);
      Draw_Current (X => X,
                    Y => Y - 2 * DY_RLC_2A,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => FALSE);
   end if;
end Draw_Current_Voltage_2_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a current source, a voltage source, and three different
--           passive components given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Voltage_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Voltage_3_RLC (X => X,
                       Y => Y,
                       VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X + 2 * DX_RLC_2B,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
            X2 => X + 2 * DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Current (X => X + 2 * DX_RLC_2B,
                    Y => Y,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => TRUE);
   else
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y,
            X2 => X + DELTA_HIGH_NODE,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Line (X1 => X + DELTA_HIGH_NODE,
            Y1 => Y - 3 * DY_RLC_3A - ARROW4,
            X2 => X + DELTA_HIGH_NODE - 2 * LEAD_E - ARROW4,
            Y2 => Y - 3 * DY_RLC_3A - ARROW4);
      Draw_Current (X => X,
                    Y => Y - 3 * DY_RLC_3A - ARROW4,
                    SIZE => SIZE_C / 2,
                    SUB => DY_RLC_2A,
                    VERTICAL => FALSE);
   end if;
end Draw_Current_Voltage_3_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source and a current
--           source given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Dependent (X, Y, SIZE, SUB : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

   DX_DEPENDENT : INTEGER_TYPE := INTEGER_TYPE (DX_RLC_2A / 2);
   DX_DEPENDENT : INTEGER_TYPE := INTEGER_TYPE (DX_RLC_2B / 2);

begin
   if VERTICAL then
      Line (X1 => X,
            Y1 => Y,
            X2 => X,
            Y2 => Y - LEAD_E);
      Line (X1 => X,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E,
            X2 => X,
            Y2 => Y - DELTA_LOW_NODE);
      Line (X1 => X - DX_DEPENDENT,
            Y1 => Y - LEAD_E,
            X2 => X + DX_RLC_2B,
            Y2 => Y - LEAD_E);
      Line (X1 => X - DX_DEPENDENT,
            Y1 => Y - DELTA_LOW_NODE + LEAD_E,
            X2 => X + DX_RLC_2B,
            Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Current (X => X - DX_DEPENDENT,
```

```
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C / 2,
         SUB     => DY_RLC_2A,
         VERTICAL => TRUE);
      Draw_Dependent (X => X + DX_RLC_2B,
         Y       => Y - LEAD_E,
         SIZE    => SIZE,
         SUB     => SUB,
         VERTICAL => TRUE);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E - ARROW4,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
         X2 => X + LEAD_E,
         Y2 => Y);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y - DY_RLC_2A - DY_DEPENDENT,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y);
      Draw_Current (X => X + LEAD_E,
         Y       => Y,
         SIZE    => INTEGER_TYPE (SIZE_C / 2),
         SUB     => DY_RLC_2A,
         VERTICAL => FALSE);
      Draw_Dependent (X => X + LEAD_E,
         Y       => Y - DY_RLC_2A - DY_DEPENDENT,
         SIZE    => SIZE,
         SUB     => SUB,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependent;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a current source,
--           and a passive component given certain parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Dependent_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL   : in CHARACTER;
   VERTICAL : in BOOLEAN) is

   DY_DEPENDENT : INTEGER_TYPE := INTEGER_TYPE (DY_RLC_2A / 2);

begin
   Draw_Current_1_RLC (X => X,
      Y       => Y,
      SYMBOL  => SYMBOL,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - LEAD_E,
         X2 => X + 2 * DX_RLC_2B,
         Y2 => Y - LEAD_E);
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X + 1 * DX_RLC_2B,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E - ARROW4,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y - 2 * DY_RLC_2A - DY_DEPENDENT);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y - 2 * DY_RLC_2A - DY_DEPENDENT);
      Draw_Dependent (X => X + LEAD_E,
         Y       => Y - 2 * DY_RLC_2A - DY_DEPENDENT,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependent_1_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a current source,
--           and two different passive components given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Dependent_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL         : in BOOLEAN) is

   DY_DEPENDENT : INTEGER_TYPE := INTEGER_TYPE (DY_RLC_2A / 2);

begin
   Draw_Current_2_RLC (X => X,
      Y       => Y,
      SYMBOL1 => SYMBOL1,
      SYMBOL2 => SYMBOL2,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - LEAD_E,
         X2 => X + 1 * DX_RLC_2B,
         Y2 => Y - LEAD_E);
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X + 2 * DX_RLC_2B,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E - ARROW4,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y - 3 * DY_RLC_2A - DY_DEPENDENT);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y - 3 * DY_RLC_2A - DY_DEPENDENT);
      Draw_Dependent (X => X + LEAD_E,
         Y       => Y - 3 * DY_RLC_2A - DY_DEPENDENT,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependent_2_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a current source,
--           and three different passive components given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Dependent_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

   DY_DEPENDENT : INTEGER_TYPE := INTEGER_TYPE (DY_RLC_2A / 2);

begin
   Draw_Current_3_RLC (X => X,
      Y       => Y,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - LEAD_E,
         X2 => X + 2 * DX_RLC_2B,
         Y2 => Y - LEAD_E);
      Line (X1 => X + DX_RLC_2B,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X + 2 * DX_RLC_2B,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Draw_Dependent (X => X + 2 * DX_RLC_2B,
         Y       => Y - LEAD_E,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E - ARROW4,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y);
      Line (X1 => X + LEAD_E,
         Y1 => Y,
         X2 => X + LEAD_E,
         Y2 => Y - 4 * DY_RLC_2A - DY_DEPENDENT);
      Line (X1 => X + DELTA_HIGH_NODE - LEAD_E,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE - LEAD_E,
         Y2 => Y - 4 * DY_RLC_2A - DY_DEPENDENT);
      Draw_Dependent (X => X + LEAD_E,
         Y       => Y - 4 * DY_RLC_2A - DY_DEPENDENT,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependent_3_RLC;

-- ------------------------------------------------------------------
-- PURPOSE : Draws a voltage controlled current source, a current source,
--           a voltage source, and a passive component given certain
--           parameters.
-- ------------------------------------------------------------------

procedure Draw_Current_Dependent_Voltage_1_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL   : in CHARACTER;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Current_Voltage_1_RLC (X => X,
      Y       => Y,
      SYMBOL  => SYMBOL,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E);
      Line (X1 => X - LEAD_COMP,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E + V_INC,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Dependent (X => X - LEAD_COMP,
         Y       => Y,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - 4 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y - 4 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y - 4 * DY_RLC_2A,
         X2 => X + DELTA_HIGH_NODE - 4 * LEAD_E - ARROW4,
         Y2 => Y - 4 * DY_RLC_2A);
      Draw_Dependent (X => X,
         Y       => Y - 4 * DY_RLC_2A,
         SIZE    => SIZE_C,
         SUB     => SUB_C,
         VERTICAL => FALSE);
```

```
   end if;
end Draw_Current_Dependant_Voltage_1_RLC;

-- PURPOSE : Draws a voltage controlled current source,  A current source,
--           a voltage source,  and two different passive components given
--           certain parameters.

procedure Draw_Current_Dependant_Voltage_2_RLC (X, Y : in INTEGER_TYPE;
   SYMBOL1, SYMBOL2 : in CHARACTER;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Current_Voltage_2_RLC (X => X,
      Y        => Y,
      SYMBOL1  => SYMBOL1,
      SYMBOL2  => SYMBOL2,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X - LEAD_COMP,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Dependant (X => X - LEAD_COMP,
         Y        => Y,
         SIZE     => SIZE_C,
         SUB      => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - 4 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y - 4 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y - 4 * DY_RLC_2A,
         X2 => X + DELTA_HIGH_NODE - 4 * LEAD_E - ARROW4,
         Y2 => Y - 4 * DY_RLC_2A);
      Draw_Dependant (X => X,
         Y        => Y - 4 * DY_RLC_2A,
         SIZE     => SIZE_C,
         SUB      => SUB_C,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependant_Voltage_2_RLC;

-- PURPOSE : Draws a voltage controlled current source,  A current source,
--           a voltage source,  and three different passive components
--           given certain parameters.

procedure Draw_Current_Dependant_Voltage_3_RLC (X, Y : in INTEGER_TYPE;
   VERTICAL : in BOOLEAN) is

begin
   Draw_Current_Voltage_3_RLC (X => X,
      Y        => Y,
      VERTICAL => VERTICAL);
   if VERTICAL then
      Line (X1 => X,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + LEAD_E - V_INC);
      Line (X1 => X - LEAD_COMP,
         Y1 => Y - DELTA_LOW_NODE + LEAD_E - V_INC,
         X2 => X - LEAD_COMP,
         Y2 => Y - DELTA_LOW_NODE + 2 * LEAD_E);
      Draw_Dependant (X => X - LEAD_COMP,
         Y        => Y,
         SIZE     => SIZE_C,
         SUB      => SUB_C,
         VERTICAL => TRUE);
   else
      Line (X1 => X,
         Y1 => Y,
         X2 => X,
         Y2 => Y - 5 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y,
         X2 => X + DELTA_HIGH_NODE,
         Y2 => Y - 5 * DY_RLC_2A);
      Line (X1 => X + DELTA_HIGH_NODE,
         Y1 => Y - 5 * DY_RLC_2A,
         X2 => X + DELTA_HIGH_NODE - 5 * LEAD_E - ARROW4,
         Y2 => Y - 5 * DY_RLC_2A);
      Draw_Dependant (X => X,
         Y        => Y - 5 * DY_RLC_2A,
         SIZE     => SIZE_C,
         SUB      => SUB_C,
         VERTICAL => FALSE);
   end if;
end Draw_Current_Dependant_Voltage_3_RLC;

-- PURPOSE : Finds if the specified node is in the list of drawing nodes.

function Col (NODE_I : in NODE_TYPE) return INTEGER_TYPE is

   COUNTER : INTEGER_TYPE := 1;

begin
   while (COUNTER <= MAX_NODES) and then
      (DRAW_CIR.GRAPH_NODE (COUNTER) /= NODE_I, temp
      COUNTER := COUNTER + 1;
   end loop;
   return COUNTER;
end Col;

-- PURPOSE : Works out if the branch is vertical or horizontal, and then
--           determines which components are in the branch.

procedure Draw_Branch (NUMBER : in     NODE_TYPE;
   HIGH_NODE1, LOW_NODE1 : in     NODE_TYPE;
   ERROR                 : in out BOOLEAN;
   COLOUR                : in     FOREGROUND_TYPE) is

   X_OK                             : BOOLEAN := FALSE;
   FOUND                            : BOOLEAN;
   COL_HIGH1,           COL_LOW1,
   X_PLUS                           : INTEGER_TYPE;

begin
   COL_HIGH1 := Col (NODE_I => HIGH_NODE1);
   COL_LOW1  := Col (NODE_I => LOW_NODE1);
   if (LOW_NODE1 = 0) then
      VERTICAL := TRUE;
      Y := GROUND_LINE + YS;
      X := NODE_10 + (COL_HIGH1 - 1 + X_SCROLL) * DELTA_HIGH_NODE;
      if (X in 1..MAX_X_COORD) and (Y in 1..(MAX_Y_COORD)) then
         X_OK := TRUE;
      end if;
   else
      VERTICAL := FALSE;
      if (COL_LOW1 < COL_HIGH1) then
         X := NODE_10 + (COL_LOW1 - 1 + X_SCROLL) * DELTA_HIGH_NODE;
      else
         X := NODE_10 + (COL_HIGH1 - 1 + X_SCROLL) * DELTA_HIGH_NODE;
      end if;
      if SLOW_DRAW or (X in 1..MAX_X_COORD) then
         X_OK := TRUE;
         if (abs (COL_HIGH1 - COL_LOW1) = 1) then
            Y := FIRST_LEVEL + YS;
         else
            FOUND := FALSE;
            L := 1;
            loop
               N := 2 * L;
               loop
                  M := 1;
                  loop
                     if (COL_LOW1 = (L + (M - 1) * (N - L))) and
                        (COL_HIGH1 = (L + M * (N - L))) then
                        if (DRAW_CIR.GROUP (L, N) = 1) then
                           MAX_LEVEL := MAX_LEVEL + 1;
                           DRAW_CIR.GROUP (L, N) := MAX_LEVEL;
                        end if;
                        Y := FIRST_LEVEL -
                           (DRAW_CIR.GROUP (L, N) - 1 - Y_SCROLL)
                           * DELTA_LEVEL;
                        FOUND := TRUE;
                     end if;
                     M := M + 1;
                     exit when FOUND or (M > MAX);
                  end loop;
                  N := N + 1;
                  exit when FOUND or (N > MAX_N);
               end loop;
               L := L + 1;
               exit when FOUND or (L > MAX_L);
            end loop;
            if (N > MAX) and (N > MAX_N) and (L > MAX_L) then
               ERROR := TRUE;
            else
               ERROR := FALSE;
            end if;
            X_PLUS := (COL_HIGH1 - COL_LOW1) * DELTA_HIGH_NODE;
            Line (X1 => X + X_PLUS,
               Y1 => FIRST_LEVEL + YS,
               X2 => X + X_PLUS,
               Y2 => Y);
            Line (X1 => X,
               Y1 => FIRST_LEVEL + YS,
               X2 => X,
               Y2 => Y);
            Line (X1 => X + DELTA_HIGH_NODE,
               Y1 => Y,
               X2 => X + X_PLUS,
               Y2 => Y);
            Circle (XC => X,
               YC   => Y,
               R    => 2,
               FORM => FULL);
            Circle (XC => X + X_PLUS,
               YC   => Y,
               R    => 2,
               FORM => FULL);
         end if;
      end if;
   end if;

   if SLOW_DRAW or (X_OK and (Y in 1..(MAX_Y_COORD))) then
      if NUMBER in 1..7 then
         if VERTICAL then
            Set_Plot_Colour (COLOUR => NODE_LINE_COLOUR);
            SUB_CC := SUB_C;
            Line (X1 => X,
               Y1 => Y - DELTA_LOW_NODE + DY_RLC_2B,
               X2 => X,
               Y2 => Y - DELTA_LOW_NODE);
         else
            SUB_CC := 0;
         end if;
      end if;

      case NUMBER is
         when 1 =>
            Draw_1_RLC (VOLT_EXISTS => FALSE,
               VERTICAL => VERTICAL,
               X        => X,
               Y        => Y,
               SUB      => SUB_CC,
               SYMBOL   => 'R');
         when 2 =>
            Draw_1_RLC (VOLT_EXISTS => FALSE,
               VERTICAL => VERTICAL,
               X        => X,
               Y        => Y,
               SUB      => SUB_CC,
```

```
            SYMBOL   => 'L');
      when 4 =>
         Draw_1_RLC (VOLT_EXISTS => FALSE,
            VERTICAL => VERTICAL,
            X        => X,
            Y        => Y,
            SUB      => SUB_CC,
            SYMBOL   => 'C');
      when 5 =>
         Draw_2_RLC (VOLT_EXISTS => FALSE,
            VERTICAL => VERTICAL,
            X        => X,
            Y        => Y,
            SUB      => SUB_CC,
            SYMBOL1  => 'R',
            SYMBOL2  => 'L');
      when 6 =>
         Draw_2_RLC (VOLT_EXISTS => FALSE,
            VERTICAL => VERTICAL,
            X        => X,
            Y        => Y,
            SUB      => SUB_CC,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C');
      when 7 =>
         Draw_2_RLC (VOLT_EXISTS => FALSE,
            VERTICAL => VERTICAL,
            X        => X,
            Y        => Y,
            SUB      => SUB_CC,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C');
      when 8 =>
         Draw_3_RLC (VOLT_EXISTS => FALSE,
            VERTICAL => VERTICAL,
            X        => X,
            Y        => Y,
            SUB      => SUB_CC);
      when 9 =>
         Draw_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'R',
            VERTICAL => VERTICAL);
      when 10 =>
         Draw_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'L',
            VERTICAL => VERTICAL);
      when 12 =>
         Draw_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'C',
            VERTICAL => VERTICAL);
      when 11 =>
         Draw_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'L',
            VERTICAL => VERTICAL);
      when 13 =>
         Draw_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 14 =>
         Draw_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 15 =>
         Draw_Voltage_3_RLC (X => X,
            Y        => Y,
            VERTICAL => VERTICAL);
      when 16 =>
         Draw_Current (X => X,
            Y        => Y,
            SIZE     => SIZE_C / 2,
            SUB      => 0,
            VERTICAL => VERTICAL);
      when 17 =>
         Draw_Current_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'R',
            VERTICAL => VERTICAL);
      when 18 =>
         Draw_Current_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'L',
            VERTICAL => VERTICAL);
      when 20 =>
         Draw_Current_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'C',
            VERTICAL => VERTICAL);
      when 19 =>
         Draw_Current_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'L',
            VERTICAL => VERTICAL);
      when 21 =>
         Draw_Current_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 22 =>
         Draw_Current_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 23 =>
         Draw_Current_3_RLC (X => X,
            Y        => Y,
            VERTICAL => VERTICAL);
      when 32 =>
         Draw_Dependent (X => X,
            Y        => Y,
            SIZE     => SIZE_C,
            SUB      => 0,
            VERTICAL => VERTICAL);
      when 33 =>
         Draw_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'R',
            VERTICAL => VERTICAL);
      when 34 =>
         Draw_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'L',
            VERTICAL => VERTICAL);
      when 36 =>
         Draw_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'C',
            VERTICAL => VERTICAL);
      when 35 =>
         Draw_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'L',
            VERTICAL => VERTICAL);
      when 37 =>
         Draw_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 38 =>
         Draw_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 39 =>
         Draw_Dependent_3_RLC (X => X,
            Y        => Y,
            VERTICAL => VERTICAL);
      when 25 =>
         Draw_Current_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'R',
            VERTICAL => VERTICAL);
      when 26 =>
         Draw_Current_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'L',
            VERTICAL => VERTICAL);
      when 24 =>
         Draw_Current_Voltage_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'C',
            VERTICAL => VERTICAL);
      when 27 =>
         Draw_Current_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'R',
            VERTICAL => VERTICAL);
      when 29 =>
         Draw_Current_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 30 =>
         Draw_Current_Voltage_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 31 =>
         Draw_Current_Voltage_3_RLC (X => X,
            Y        => Y,
            VERTICAL => VERTICAL);
      when 48 =>
         Draw_Current_Dependent (X => X,
            Y        => Y,
            SIZE     => SIZE_C,
            SUB      => SUB_C,
            VERTICAL => VERTICAL);
      when 49 =>
         Draw_Current_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'R',
            VERTICAL => VERTICAL);
      when 50 =>
         Draw_Current_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'L',
            VERTICAL => VERTICAL);
      when 52 =>
         Draw_Current_Dependent_1_RLC (X => X,
            Y        => Y,
            SYMBOL   => 'C',
            VERTICAL => VERTICAL);
      when 51 =>
         Draw_Current_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'R',
            VERTICAL => VERTICAL);
      when 53 =>
         Draw_Current_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'R',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 54 =>
         Draw_Current_Dependent_2_RLC (X => X,
            Y        => Y,
            SYMBOL1  => 'L',
            SYMBOL2  => 'C',
            VERTICAL => VERTICAL);
      when 55 =>
         Draw_Current_Dependent_3_RLC (X => X,
            Y        => Y,
```

```
                VERTICAL => VERTICAL);
            when 41 =>
               Draw_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'R',
                  VERTICAL => VERTICAL);
            when 42 =>
               Draw_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'L',
                  VERTICAL => VERTICAL);
            when 44 =>
               Draw_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'C',
                  VERTICAL => VERTICAL);
            when 43 =>
               Draw_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'L',
                  SYMBOL2  => 'R',
                  VERTICAL => VERTICAL);
            when 45 =>
               Draw_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'R',
                  SYMBOL2  => 'C',
                  VERTICAL => VERTICAL);
            when 46 =>
               Draw_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'L',
                  SYMBOL2  => 'C',
                  VERTICAL => VERTICAL);
            when 47 =>
               Draw_Dependant_Voltage_3_RLC (X => X,
                  Y        => Y,
                  VERTICAL => VERTICAL);
            when 57 =>
               Draw_Current_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'R',
                  VERTICAL => VERTICAL);
            when 58 =>
               Draw_Current_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'L',
                  VERTICAL => VERTICAL);
            when 60 =>
               Draw_Current_Dependant_Voltage_1_RLC (X => X,
                  Y        => Y,
                  SYMBOL   => 'C',
                  VERTICAL => VERTICAL);
            when 59 =>
               Draw_Current_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'L',
                  SYMBOL2  => 'R',
                  VERTICAL => VERTICAL);
            when 61 =>
               Draw_Current_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'R',
                  SYMBOL2  => 'C',
                  VERTICAL => VERTICAL);
            when 62 =>
               Draw_Current_Dependant_Voltage_2_RLC (X => X,
                  Y        => Y,
                  SYMBOL1  => 'L',
                  SYMBOL2  => 'C',
                  VERTICAL => VERTICAL);
            when 63 =>
               Draw_Current_Dependant_Voltage_3_RLC (X => X,
                  Y        => Y,
                  VERTICAL => VERTICAL);
            when others => null;
         end case;
      end if;
   end Draw_Branch;

   -----------------------------------------------------------------------
   -- PURPOSE : Determines which components are in a branch.
   -----------------------------------------------------------------------

   procedure Find_Branch_Components (NUMBER,
      NUMBER_DEPENDENT : in out INTEGER_TYPE;
      HIGH_DEPENDENT2, LOW_DEPENDENT2, HIGH_F1, LOW_F1 : in out NODE_TYPE;
      NET_LIST_OBJECT : in out NETLIST_TYPE) is

      COMPONENT                              : ELEMENTS;
      DEPENDENT_ALONE,        DONE_CURRENT,
      DONE_DEPENDENT,         DONE_VOLTAGE   : BOOLEAN := FALSE;
      HIGH_F1,                LOW_F1,
      HIGH_NODE1,             LOW_NODE1,
      HIGH_NODE2,             LOW_NODE2,
      HIGH_DEPENDENT,         LOW_DEPENDENT  : NODE_TYPE;
      NETLIST2                               : NETLIST_TYPE :=
   NETLIST_OBJECT;

   begin
      NUMBER_DEPENDENT := 0;
      NUMBER := 0;
      for COMPONENT in CAPS..COUPLING loop
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         if COMPONENT = CAPS then
            HIGH_F1 := HIGH_NODE1;
            LOW_F1  := LOW_NODE1;
         end if;
         if abs (VALUE) > MINIMUM_VALUE then
            case COMPONENT is
               when CAPS      => NUMBER := NUMBER + 4;
               when GR_ADMIT  => NUMBER := NUMBER + 1;
               when ACT_INDCT => NUMBER := NUMBER + 2;
               when REAL_VOLT | IMAG_VOLT =>
                  if not DONE_VOLTAGE then
                     DONE_VOLTAGE := TRUE;
                     NUMBER := NUMBER + 8;
                  end if;
               when REAL_CRNT | IMAG_CRNT =>
                  if not DONE_CURRENT then
                     DONE_CURRENT := TRUE;
                     NUMBER := NUMBER + 16;
                  end if;
               when CR_ADMIT..CM_ADMIT =>
                  if not DONE_DEPENDENT then
                     HIGH_DEPENDENT := HIGH_NODE1;
                     LOW_DEPENDENT  := LOW_NODE1;
                     DEPENDENT_ALONE := TRUE;
                     if (HIGH_DEPENDENT /= HIGH_F1) or
                        (LOW_DEPENDENT /= LOW_F1) then
                        Inner.First_Branch (NETLIST_OBJECT => NETLIST2);
                        while DEPENDENT_ALONE loop
                           Inner.Read_Netlist (COMPONENT => CAPS,
                              VALUE          => VALUE,
                              HIGH_NODE1     => HIGH_F1,
                              LOW_NODE1      => LOW_F1,
                              HIGH_NODE2     => HIGH_NODE2,
                              LOW_NODE2      => LOW_NODE2,
                              NETLIST_OBJECT => NETLIST2);
                           if (HIGH_DEPENDENT = HIGH_F1) and
                              (LOW_DEPENDENT = LOW_F1) then
                              DEPENDENT_ALONE := FALSE;
                           else
                              begin
                                 Inner.Next_Branch(NETLIST_OBJECT=>NETLIST2);
                              exception
                                 when NO_MORE_BRANCHES => exit;
                              end;
                           end if;
                        end loop;
                        if DEPENDENT_ALONE then
                           NUMBER_DEPENDENT := 32;
                           HIGH_DEPENDENT2 := HIGH_DEPENDENT;
                           LOW_DEPENDENT2  := LOW_DEPENDENT;
                        end if;
                     end if;
                     DONE_DEPENDENT := TRUE;
                  end if;
               when others => null;
            end case;
         end if;
      end loop;
      Inner.First_Branch (NETLIST_OBJECT => NETLIST2);
      while NUMBER < 32 loop
         Inner.Read_Netlist (COMPONENT => CM_ADMIT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_DEPENDENT,
            LOW_NODE1      => LOW_DEPENDENT,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST2);
         if (HIGH_F1 = HIGH_DEPENDENT) and (LOW_F1 = LOW_DEPENDENT) and
            (abs (VALUE) > MINIMUM_VALUE) then
            NUMBER := NUMBER + 32;
         else
            begin
               Inner.Next_Branch (NETLIST_OBJECT => NETLIST2);
            exception
               when NO_MORE_BRANCHES => exit;
            end;
         end if;
      end loop;
   end Find_Branch_Components;

   -----------------------------------------------------------------------
   -- PURPOSE : Draws the individual branches of the circuit.
   -----------------------------------------------------------------------

   procedure Draw_Netlist (DRAWING_SUCCESS : out BOOLEAN) is

      COUNTER                                : INTEGER_TYPE := 1;
      NUMBER,                 NUMBER_DEPENDENT  : INTEGER_TYPE;
      DEPENDENT_ALONE,        DONE_CURRENT,
      DONE_VOLTAGE,           DONE_DEPENDENT    : BOOLEAN;
      NOT_OK                                    : BOOLEAN := FALSE;
      HIGH_NODE1,             LOW_NODE1,
      HIGH_NODE2,             LOW_NODE2,
      HIGH_DEPENDENT1,        LOW_DEPENDENT1,
      HIGH_DEPENDENT2,        LOW_DEPENDENT2    : NODE_TYPE;

   begin
      if SEARCH_NETLIST then
         Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
         while (COUNTER <= MAXIMUM_COUNT) and (not NOT_OK) loop
            Find_Branch_Components (NUMBER => NUMBER,
               NUMBER_DEPENDENT => NUMBER_DEPENDENT,
               HIGH_DEPENDENT2  => HIGH_DEPENDENT2,
               LOW_DEPENDENT2   => LOW_DEPENDENT2,
               HIGH_F1          => HIGH_NODE1,
               LOW_F1           => LOW_NODE1,
               NETLIST_OBJECT   => NETLIST_OBJECT);
            if (NUMBER /= 0) then
               Draw_Branch (NUMBER => NUMBER,
                  HIGH_NODE1 => HIGH_NODE1,
                  LOW_NODE1  => LOW_NODE1,
                  ERROR      => NOT_OK,
                  COLOUR     => NODE_LINE_COLOUR);
            end if;
            DRAW_CIR.NUMBER_NODES (COUNTER) := (NUMBER => NUMBER,
               HIGH_NODE => HIGH_NODE1,
               LOW_NODE  => LOW_F1);
            COUNTER := COUNTER + 1;
            if (NUMBER_DEPENDENT /= 0) then
               DRAW_CIR.NUMBER_NODES (COUNTER) := (NUMBER =>NUMBER_DEPENDENT,
                  HIGH_NODE => HIGH_DEPENDENT2,
                  LOW_NODE  => LOW_DEPENDENT2);
               COUNTER := COUNTER + 1;
               Draw_Branch (NUMBER => NUMBER_DEPENDENT,
                  HIGH_NODE1 => HIGH_DEPENDENT2,
                  LOW_NODE1  => LOW_DEPENDENT2,
                  ERROR      => NOT_OK,
                  COLOUR     => NODE_LINE_COLOUR);
            end if;
            begin
```

```
            Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
         exception
            when NO_MORE_BRANCHES => exit;
         end;
      end loop;
      MAX_COUNT := COUNTER;
   else
      while (COUNTER <= MAX_COUNT) and (not NOT_OK) loop
         Draw_Branch (NUMBER => DRAW_CIR.NUMBER_NODES (COUNTER).NUMBER,
            HIGH_NODE1 => DRAW_CIR.NUMBER_NODES (COUNTER).HIGH_NODE,
            LOW_NODE1  => DRAW_CIR.NUMBER_NODES (COUNTER).LOW_NODE,
            ERROR      => NOT_OK,
            COLOUR     => NODE_LINE_COLOUR);
         COUNTER := COUNTER + 1;
      end loop;
   end if;
   DRAWING_SUCCESS := not NOT_OK;
end Draw_Netlist;

-- PURPOSE : Labels the nodes of the circuit.

procedure Label_Nodes is

   XL,                  YL,
   COUNTER1,            COUNTER2          : INTEGER_TYPE;
   DISPLAY_LINE                           : Texts.TEXT_TYPE (15);

begin
   Set_Plot_Colour (COLOUR => NODE_LINE_COLOUR);
   YL := GROUND_LINE + YS + ROW_FACTOR;
   XL := NODE_10 + X_SCROLL * DELTA_HIGH_NODE +
      (NODE_MAX * DELTA_HIGH_NODE) / 2;
   if (XL in 1..MAX_X_COORD) and (YL in 1..MAX_Y_COORD) then
      String_To_Graphic (MESSAGE => "0",
         ROW     => YL / ROW_FACTOR,
         COLUMN  => XL / COLUMN_FACTOR,
         COLOURS => NODE_LINE_COLOUR);
   end if;
   YL := FIRST_LEVEL + YS;
   for COUNTER1 in 1..NODE_MAX loop
      if X_SCROLL > 3 then
         XL :=  NODE_10 + (X_SCROLL + COUNTER1 - 1) * DELTA_HIGH_NODE - 6;
         if (XL in 1..MAX_X_COORD) and (YL in 1..MAX_Y_COORD) then
            Texts.Reset (TEXT_LINE => DISPLAY_LINE);
            Texts.Combine (ITEM => DRAW_CIR.DRAWING_NODES (COUNTER1),
               ITEM_WIDTH => 2,
               TEXT_LINE  => DISPLAY_LINE);
            String_To_Graphic (
               MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
               ROW     => YL / ROW_FACTOR,
               COLUMN  => XL / COLUMN_FACTOR,
               COLOURS => NODE_LINE_COLOUR);
            Circle (XC => XL + 6,
               YC   => YL,
               R    => 2,
               FORM => FULL);
         end if;
      else
         XL :=  NODE_10 + (COUNTER1 - 1) * DELTA_HIGH_NODE - 6;
         if (XL in 1..MAX_X_COORD) and (YL in 1..MAX_Y_COORD) then
            COUNTER2 := COUNTER1 - X_SCROLL;
            if COUNTER2 in 1..MAX_NODES then
               Texts.Reset (TEXT_LINE => DISPLAY_LINE);
               Texts.Combine (ITEM => DRAW_CIR.DRAWING_NODES (COUNTER2),
                  ITEM_WIDTH => 2,
                  TEXT_LINE  => DISPLAY_LINE);
               String_To_Graphic (
                  MESSAGE =>DISPLAY_LINE.TEXT(1..DISPLAY_LINE.TEXT_INDEX),
                  ROW     => YL / ROW_FACTOR,
                  COLUMN  => XL / COLUMN_FACTOR,
                  COLOURS => NODE_LINE_COLOUR);
               Circle (XC => XL + 6,
                  YC   => YL,
                  R    => 2,
                  FORM => FULL);
            end if;
         end if;
      end if;
   end loop;
end Label_Nodes;

-- PURPOSE : Sets and changes the values of several variables used in
--           drawing the circuit. This allows the drawing to be expanded
--           or condensed.

procedure Zoom (TYPE_OF_ZOOM : in ZOOM_TYPE) is

   CHANGE_VALUES : BOOLEAN := FALSE;

begin
   if TYPE_OF_ZOOM = RESET then
      if MAX_Y_COORD = 350 then
         ZOOM_FACTOR := 1.0;
      else
         ZOOM_FACTOR := 0.64;
      end if;
      X_SCROLL := 0;
      Y_SCROLL := 0;
      CHANGE_VALUES := TRUE;
   elsif TYPE_OF_ZOOM = UP then
      if ZOOM_FACTOR < 3.814 then
         ZOOM_FACTOR := ZOOM_FACTOR * ZOOMING_FACTOR;
         CHANGE_VALUES := TRUE;
      end if;
   else
      if ZOOM_FACTOR > 0.31768 then
         ZOOM_FACTOR := ZOOM_FACTOR / ZOOMING_FACTOR;
         CHANGE_VALUES := TRUE;
      end if;
   end if;

   if CHANGE_VALUES then
      ARROW1          := INTEGER_TYPE (5.0  * ZOOM_FACTOR);
      ARROW2          := INTEGER_TYPE (15.0 * ZOOM_FACTOR);
      ARROW3          := INTEGER_TYPE (10.0 * ZOOM_FACTOR);
      ARROW4          := INTEGER_TYPE (3.0  * ZOOM_FACTOR);
      V_INC           := INTEGER_TYPE (2.0  * ZOOM_FACTOR);
      XT1             := INTEGER_TYPE (16.0*ZOOM_FACTOR);--2*COLUMN_FACTOR
      YT1             := INTEGER_TYPE (8.0  * ZOOM_FACTOR);--ROW_FACTOR=8
      WIDTHP          := INTEGER_TYPE (15.0 * ZOOM_FACTOR);
      HT_PRLV         := INTEGER_TYPE (12.0 * ZOOM_FACTOR);
      HT_PRLH         := INTEGER_TYPE (18.0 * ZOOM_FACTOR);
      DX_RLC_2A       := INTEGER_TYPE (15.0 * ZOOM_FACTOR);
      DX_RLC_2B       := INTEGER_TYPE (30.0 * ZOOM_FACTOR);
      DY_RLC_2A       := INTEGER_TYPE (12.0 * ZOOM_FACTOR);
      DY_RLC_2B       := INTEGER_TYPE (25.0 * ZOOM_FACTOR);
      SIZE_C          := INTEGER_TYPE (24.0 * ZOOM_FACTOR);
      SUB_C           := INTEGER_TYPE (10.0 * ZOOM_FACTOR);
      LEAD_CONP       := INTEGER_TYPE (30.0 * ZOOM_FACTOR);
      LEAD_E          := INTEGER_TYPE (10.0 * ZOOM_FACTOR);
      DELTA_HIGH_NODE := INTEGER_TYPE (95.0 * ZOOM_FACTOR);
      DELTA_LOW_NODE  := INTEGER_TYPE (44.0 * ZOOM_FACTOR);
      FIRST_LEVEL     := GROUND_LINE - DELTA_LOW_NODE;
      DELTA_LEVEL     := INTEGER_TYPE (72.0 * ZOOM_FACTOR);
   end if;
end Zoom;

-- PURPOSE : Draws a the netlist and gives an error message if the drawing
--           node ordering is not acceptable.

procedure Draw_The_Netlist is

begin
   Set_Plot_Colour (COLOUR => NODE_LINE_COLOUR);
   Label_Nodes;
   Line (X1    => NODE_10 + XS,
         Y1    => GND_LINE + YS,
         X2    => NODE_MAX * DELTA_HIGH_NODE + XS + NODE_10,
         Y2    => GROUND_LINE + YS);
   Draw_Netlist (DRAWING_SUCCESS => CORRECT);
   if not CORRECT then
      Change_Screen (MODE_OF_SCREEN => TEXT);
      Errors.Write (MESSAGE => "The drawing node ordering is not "
         & "acceptable.",
         HELP_INFO => HP_DRAW_THE_NETLIST);
      raise USER_ESCAPE;
   end if;
end Draw_The_Netlist;

begin
   Main.Process_Title (TITLE => "DRAW CIRCUIT");
   SEARCH_NETLIST := TRUE;
   MAX := NODE_MAX;
   MAX_H := MAX;
   MAX_L := MAX_H / 2;
   NODE_10 := 40;
   GROUND_LINE := INTEGER_TYPE (REAL_TYPE (MAX_Y_COORD) * 0.917_943_806);
   Zoom (TYPE_OF_ZOOM => RESET);
   MAX_LEVEL := 1;
   DRAW_CIR := NEW DRAW_CIRCUIT_TYPE;
   for L in 1..MAX_LOW loop
      for H in 1..MAX_HIGH loop
         DRAW_CIR.GROUP (L, H) := 1;
      end loop;
   end loop;
   for L in 1..MAX_NODES loop
      DRAW_CIR.DRAWING_NODES (L) := L;
   end loop;
   begin
      COLOUR := NODE_LINE_COLOUR;
      Prompts.Read (LIST => DRAW_LIST,
         EXCLUSIVE => FALSE,
         HELP_INFO => HP_DRAW_CIRCUIT);
      SLOW_DRAW := not DRAW_LIST (2).START;
      if not DRAW_LIST (1).START then
         L := 1;
         loop
            Prompts.Read (MESSAGE => "Enter the drawing node order ( 0 " &
               "to Quit",
               ITEM => N);
            if (N /= 0) and (N in 1..MAX_NODES) then
               DRAW_CIR.DRAWING_NODES (L) := N;
            end if;
            L := L + 1;
            exit when (L > MAX_NODES) or (N = 0);
         end loop;
      end if;
      Status.Write (MESSAGE => "Use the CURSOR ARROW, PGUP, PGDN, HOME, "
         & "END and ESC keys to scroll or zoom the circuit and to exit.",
         HELP_INFO => HP_DRAW_CIRCUIT2);
      Change_Screen (MODE_OF_SCREEN => GRAPHIC);
      XS := 0;
      YS := 0;
      Draw_The_Netlist;
      SEARCH_NETLIST := FALSE;
      loop
         REDRAW := TRUE;
         begin
            Inputter.Get_Input (INPUTS => INPUTS);
            case INPUTS.KEY is
               when KB_LEFT  => X_SCROLL := X_SCROLL - 1;
               when KB_RIGHT => X_SCROLL := X_SCROLL + 1;
               when KB_UP    => Y_SCROLL := Y_SCROLL - 1;
               when KB_DOWN  => Y_SCROLL := Y_SCROLL + 1;
               when KB_HOME  => Zoom (TYPE_OF_ZOOM => RESET);
               when KB_END   => X_SCROLL := 0 - NODE_MAX;
               when KB_PGUP  => Zoom (TYPE_OF_ZOOM => UP);
               when KB_PGDN  => Zoom (TYPE_OF_ZOOM => DOWN);
               when KB_ALT_C | KB_ESC => exit;
               when others => REDRAW := FALSE;
            end case;
         exception
            when CONSTRAINT_ERROR =>
               Zoom (TYPE_OF_ZOOM => RESET);
         end;
         if REDRAW then
            Clear_Screen (COLOUR => BLUE);
            XS := X_SCROLL * DELTA_HIGH_NODE;
            YS := Y_SCROLL * DELTA_LEVEL;
            Draw_The_Netlist;
         end if;
      end loop;
```

```
   exception
      when USER_ESCAPE => null;
   end;
   Dispose (DRAW_CIR);
   Change_Screen (MODE_OF_SCREEN => TEXT);
end Draw_Circle;
```

```
end Drawer;
```

Elector

```
with  Dos_Environment,
      Scheduler;

procedure Elector is

-- ------------------------------------------------------------------------
-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This is the main subprogram  which is called by the operating
--           system.
-- ------------------------------------------------------------------------

-- ------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-- ------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-- ------------------------------------------------------------------------

begin
   Scheduler.Initialize;
   Scheduler.Scheduling;
   Dos_Environment.Show_Cursor;
end Elector;
```

Graphs

```
with  Ada_Extensions;
use   Ada_Extensions;

package Graphs is

-- Author  : Warren Buckl;
-- Date    : June 1992
-- Purpose : This is the foundation  for displaying plots  and circuits on
--           the  graphic screen.  It  contains subprograms  to change the
--           screen to graphics mode and back to text mode,  as well as to
--           plot points, draw lines  etc.  Some of these subprograms call
--           Assembly Language procedures which do the actual work.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   -- Constants used for drawing and plotting the graph

   MAX_X_COORD     : constant := 640;
   COLUMN_FACTOR   : constant := MAX_X_COORD / 80;
   MAX_INTEGER     : constant := 2147_0483_647;
   X_MIN           : constant := 1;
   X_MAX           : constant := 640;
   Y_MAX           : constant := 1;
   X_AXIS          : constant := 75;
   NX_PIXELS       : constant := X_MAX - X_AXIS - X_AXIS / 2;
   LN_10           : constant := 2.302_585_092_994_046; -- LOG (10)

   -- Variables used for plotting graph

   MAX_Y_COORD,            NY_PIXELS,
   ROW_FACTOR,             Y_AXIS,
   Y_MIN                                 : INTEGER_TYPE;
   ASPECT_RATIO                          : REAL_TYPE := 1.37;
   AUTO_SCALE                            : BOOLEAN;

   type SCREEN_MODE is (TEXT, GRAPHIC);

   procedure Change_Screen (MODE_OF_SCREEN : in SCREEN_MODE);

      pragma Interface (Assembly, Change_Screen);

   procedure Character_To_Graphic (MESSAGE : in CHARACTER;
      ROW, COLUMN  : in INTEGER_TYPE;
      COLOURS      : in FOREGROUND_TYPE);

      pragma Interface (Assembly, Character_To_Graphic);

   procedure Clear_Screen (COLOUR : in FOREGROUND_TYPE);

      pragma Interface (Assembly, Clear_Screen);

   procedure Dot_Horizontal_Line (DOT_SPACING : in NATURAL;
      X1, X2, Y : in INTEGER_TYPE);

      pragma Interface (Assembly, Dot_Horizontal_Line);

   procedure Dot_Vertical_Line (DOT_SPACING : in NATURAL;
      Y1, Y2, X : in INTEGER_TYPE);

      pragma Interface (Assembly, Dot_Vertical_Line);

   procedure Line (X1, Y1, X2, Y2 : in INTEGER_TYPE);

      pragma Interface (Assembly, Line);

   procedure Plot (X, Y : in INTEGER_TYPE);

      pragma Interface (Assembly, Plot);

   procedure Plot_Four_Pixels (X_CENTRE, Y_CENTRE, X_COORD,
      Y_COORD : in INTEGER_TYPE);

      pragma Interface (Assembly, Plot_Four_Pixels);

   procedure Set_Plot_Colour (COLOUR : in FOREGROUND_TYPE);

      pragma Interface (Assembly, Set_Plot_Colour);

   procedure Set_Plot_Window (X1, Y1, X2, Y2 : in INTEGER_TYPE);

      pragma Interface (Assembly, Set_Plot_Window);

   procedure String_To_Graphic (MESSAGE : in STRING;
      ROW, COLUMN  : in INTEGER_TYPE;
      COLOURS      : in FOREGROUND_TYPE);

      pragma Interface (Assembly, String_To_Graphic);

   procedure Setup;

   procedure Draw_X_Axis;

   procedure Draw_Y_Axis;

   procedure Fix_Scale (Y_MINIMUM, Y_MAXIMUM : in out REAL_TYPE);

   procedure Get_Vertical_Inc (VERTICAL_MIN, VERTICAL_MAX : out REAL_TYPE;
      X_INCR, OPTION : out INTEGER_TYPE);

   procedure Get_X_Frequency (MIN_FREQUENCY, MAX_FREQUENCY:out REAL_TYPE);

   procedure Label_Linear_X (X_MINIMUM, X_MAXIMUM : in REAL_TYPE);

   procedure Label_Log_X (X_MINIMUM, X_MAXIMUM : in REAL_TYPE);

   procedure Label_Y (Y_MINIMUM, Y_MAXIMUM : in REAL_TYPE);

   procedure Zero_Line (Y_MINIMUM, Y_MAXIMUM, Y_SCALE : in REAL_TYPE);

end Graphs;
```

```
with  Math_Lib,
      Screens;

use   Screens;

package body Graphs is

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_GET_X_FREQUENCY   : constant := 140;
   HP_GET_X_FREQUENCY2  : constant := 141;
   HP_GET_VERTICAL_INC  : constant := 142;
   HP_GET_VERTICAL_INC2 : constant := 143;
   HP_GET_VERTICAL_INC3 : constant := 144;

   -- Constants used for drawing and plotting the graph

   MINIMUM_VALUE : constant := 1.0E-30;
   LABEL_X       : constant := 5;
   NX_DIV        : constant := 5;
   NY_DIV        : constant := 10;
   RATIO_X       : constant := NX_PIXELS / NX_DIV;

   RATIO_Y : INTEGER_TYPE;

   -- Variables for the Get_Vertical_Inc procedure.

   VERTICAL_FORM : FIELDS_TYPE (1..3) :=
      (1 =>
         (VALUE_LENGTH  => 2,
         TEXT_LENGTH    => 14,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 1,
         ROW            => 4,
         VALUE          => "5 ",
         TEXT           => "Plot increment",
         MAX_INTEGER    => 50,
         MIN_INTEGER    => 1,
         INTEGER_VALUE  => 5),
       2 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 22,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => FALSE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 6,
         ROW            => 5,
         VALUE          => "1.0e-3          ",
         TEXT           => "Minimum vertical scale",
         MAX_REAL       => 1.0e25,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1.0e-3),
       3 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 22,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 6,
         ROW            => 6,
         VALUE          => "1.1e-3          ",
         TEXT           => "Maximum vertical scale",
         MAX_REAL       => 1.0e25,
         MIN_REAL       => MINIMUM_VALUE,
```

```
      REAL_VALUE     => 1.1e-3)};

   PLOT_TYPE_LIST : LIST_ARRAY_TYPE (1..3) := (
   (TEXT_LENGTH => 9,  TEXT => "Bode Plot",     STATE => TRUE),
   (TEXT_LENGTH => 10, TEXT => "Phase Plot",    STATE => FALSE),
   (TEXT_LENGTH => 14, TEXT => "Magnitude Plot", STATE => FALSE));

   -- Variables for the Get_X_Frequency procedure.

   GET_X_FREQUENCY_FORM : FIELDS_TYPE (1..2) :=
      (1 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 15,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 6,
         ROW            => 4,
         VALUE          => "1.0e-3          ",
         TEXT           => "Start Frequency",
         MAX_REAL       => 1.0e25,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1.0e-3),
      2 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 15,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 6,
         ROW            => 5,
         VALUE          => "1.1e-3          ",
         TEXT           => "End   Frequency",
         MAX_REAL       => 1.0e25,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1.1e-3)};

-- PURPOSE : Sets up the user interface as well as determining the values
--           of certain variables used for graphs.

procedure Setup is

begin
   Y_AXIS := 20;
   Initialise_User_Interface (Y_RESOLUTION => MAX_Y_COORD);
   if MAX_Y_COORD = 200 then
      ASPECT_RATIO := 2.4;
      Y_AXIS := 11;
   end if;
   ROW_FACTOR := MAX_Y_COORD / 25;
   NY_PIXELS  := MAX_Y_COORD - 3 * Y_AXIS - Y_AXIS / 2;
   RATIO_Y    := NY_PIXELS / NY_DIV;
end Setup;

-- PURPOSE : Determines the scale of the graph.

procedure Fix_Scale (Y_MINIMUM, Y_MAXIMUM : in out REAL_TYPE) is

   DECADE         : REAL_TYPE := 1.0E-37;
   MIN_SCALE_DIV  : REAL_TYPE := (Y_MAXIMUM-Y_MINIMUM)/REAL_TYPE(NY_DIV);
   NUMBER         : REAL_TYPE;
   COUNTER        : INTEGER_TYPE;
   FOUND          : BOOLEAN   := FALSE;

begin
   loop
      for COUNTER in 1..5 loop
         case COUNTER is
            when 1 => NUMBER := 1.0;
            when 2 => NUMBER := 2.0;
            when 3 => NUMBER := 4.0;
            when 4 => NUMBER := 5.0;
            when 5 => NUMBER := 8.0;
         end case;
         if (not FOUND) and (MIN_SCALE_DIV <= NUMBER * DECADE) then
            FOUND := TRUE;
            MIN_SCALE_DIV := NUMBER * DECADE;
         end if;
      end loop;
      DECADE := DECADE * 10.0;
      exit when (DECADE >= 1.0E37) or FOUND;
   end loop;
   if FOUND then
      if (MIN_SCALE_DIV > 1.0) then
         Y_MINIMUM := REAL_TYPE (INTEGER_TYPE (Y_MINIMUM - 0.5));
      end if;
      Y_MAXIMUM := Y_MINIMUM + REAL_TYPE (NY_DIV) * MIN_SCALE_DIV;
   end if;
end Fix_Scale;

-- PURPOSE : Draws the X axis.

procedure Draw_X_Axis is

begin
   Set_Plot_Colour (COLOUR => YELLOW);
   Line (X1 => X_MIN + X_AXIS,
      Y1 => Y_MIN - Y_AXIS,
      X2 => X_MAX - X_AXIS / 2,
      Y2 => Y_MIN - Y_AXIS);
end Draw_X_Axis;

-- PURPOSE : Draws the Y axis.

procedure Draw_Y_Axis is

begin
   Set_Plot_Colour (COLOUR => YELLOW);
   Line (X1 => X_MIN + X_AXIS,
      Y1 => Y_MIN - Y_AXIS,
      X2 => X_MIN + X_AXIS,
      Y2 => Y_MAX + Y_AXIS);
end Draw_Y_Axis;

-- PURPOSE : Labels a linear X axis.

procedure Label_Linear_X (X_MINIMUM, X_MAXIMUM : in REAL_TYPE) is

   OFFSET : constant := 43;

   DISPLAY_LINE  : Texts.TEXT_TYPE;
   COUNTER       : INTEGER_TYPE;
   X_INC         : REAL_TYPE := (X_MAXIMUM-X_MINIMUM)/REAL_TYPE (NX_DIV);
   VALUE         : REAL_TYPE;

begin
   Set_Plot_Colour (COLOUR => LIGHT_CYAN);
   for COUNTER in 0..NX_DIV loop
      Line (X1 => X_MIN + X_AXIS + RATIO_X * COUNTER,
         Y1 => Y_MIN - Y_AXIS + 5,
         X2 => X_MIN + X_AXIS + RATIO_X * COUNTER,
         Y2 => Y_MIN - Y_AXIS);
      Dot_Vertical_Line (DOT_SPACING => 7,
         Y1 => Y_MIN - Y_AXIS,
         Y2 => Y_MAX + Y_AXIS,
         X  => X_MIN + X_AXIS + RATIO_X * COUNTER);
      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => X_MINIMUM + X_INC * REAL_TYPE (COUNTER),
         FRACTION       => 2,
         ENG_NOTATION => TRUE,
         TEXT_LINE      => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => 23,
         COLUMN  => (X_MIN + X_AXIS + RATIO_X * COUNTER - OFFSET)
            / COLUMN_FACTOR,
         COLOURS => LIGHT_CYAN);
   end loop;
end Label_Linear_X;

-- PURPOSE : Labels a log X axis.

procedure Label_Log_X (X_MINIMUM, X_MAXIMUM : in REAL_TYPE) is

   OFFSET       : constant := 40;
   MAX_DECADES  : constant := 1.0E7;

   DISPLAY_LINE    : Texts.TEXT_TYPE;
   COUNTER         : INTEGER_TYPE;
   DECADE          : REAL_TYPE := 1.0;
   INDEX           : REAL_TYPE := 2.0;
   FREQ_INC        : REAL_TYPE;
   X_MINIMUM2      : REAL_TYPE := X_MINIMUM;
   SECOND          : BOOLEAN;
   COUNT_DEC       : INTEGER_TYPE := 2;

begin
   if (X_MAXIMUM / X_MINIMUM) > MAX_DECADES then
      SECOND := TRUE;
   else
      SECOND := FALSE;
   end if;
   Set_Plot_Colour (COLOUR => LIGHT_CYAN);
   FREQ_INC := (Math_Lib.Ln (X_MAXIMUM) - Math_Lib.Ln (X_MINIMUM))
      / REAL_TYPE (NX_PIXELS);
   Line (X1 => X_MIN + X_AXIS,
      Y1 => Y_MIN - Y_AXIS + 5,
      X2 => X_MIN + X_AXIS,
      Y2 => Y_MIN - Y_AXIS);
   Texts.Combine (ITEM => X_MINIMUM,
      WHOLE          => 3,
      FRACTION       => 2,
      ENG_NOTATION => TRUE,
      TEXT_LINE      => DISPLAY_LINE);
   String_To_Graphic (
      MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
      ROW     => 23,
      COLUMN  => (X_MIN + X_AXIS - OFFSET) / COLUMN_FACTOR,
      COLOURS => LIGHT_CYAN);
   Line (X1 => X_MIN + X_AXIS + NX_PIXELS,
      Y1 => Y_MIN - Y_AXIS + 5,
      X2 => X_MIN + X_AXIS + NX_PIXELS,
      Y2 => Y_MIN - Y_AXIS);
   Set_Plot_Colour (COLOUR => LIGHT_CYAN);
   for COUNTER in 0..NX_PIXELS loop
      if Math_Lib.Exp (FREQ_INC * REAL_TYPE (COUNTER)) >=
         (INDEX * DECADE) then
         Dot_Vertical_Line (DOT_SPACING => 7,
            Y1 => Y_MIN - Y_AXIS,
            Y2 => Y_MAX + Y_AXIS,
            X  => X_MIN + X_AXIS + COUNTER);
         if (INDEX = 10.0) and ((((COUNT_DEC rem 2 = 1) and SECOND) or
            not SECOND) then
            Line (X1 => X_MIN + X_AXIS + COUNTER,
               Y1 => Y_MIN - Y_AXIS + 5,
               X2 => X_MIN + X_AXIS + COUNTER,
               Y2 => Y_MIN - Y_AXIS);
            Texts.Reset (TEXT_LINE => DISPLAY_LINE);
            Texts.Combine (ITEM => X_MINIMUM2 * DECADE * 10.0,
               WHOLE          => 3,
               FRACTION       => 2,
               ENG_NOTATION => TRUE,
               TEXT_LINE      => DISPLAY_LINE);
            String_To_Graphic (
               MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
               ROW     => 23,
               COLUMN  => (X_MIN + X_AXIS + COUNTER - OFFSET)
                  / COLUMN_FACTOR,
               COLOURS => LIGHT_CYAN);
         end if;
         if INDEX < 10.0 then
            INDEX := INDEX + 1.0;
         else
            DECADE := DECADE * 10.0;
            INDEX := 2.0;
            COUNT_DEC := COUNT_DEC + 1;
```

```
         end if;
      end if;
   end loop;
end Label_Log_X;

-- PURPOSE : Labels the Y axis.

procedure Label_Y (Y_MINIMUM, Y_MAXIMUM : in REAL_TYPE) is

   SCALE_OFF    : constant := 3;

   COUNTER      : INTEGER_TYPE;
   Y_INC        : REAL_TYPE := (Y_MAXIMUM - Y_MINIMUM)/REAL_TYPE (NY_DIV);
   DISPLAY_LINE : Texts.TEXT_TYPE;

begin
   Set_Plot_Colour (COLOUR => LIGHT_CYAN);
   for COUNTER in 0..NY_DIV loop
      Line (X1 => X_MIN + X_AXIS - SCALE_OFF,
         Y1 => Y_MIN - Y_AXIS - RATIO_Y * COUNTER,
         X2 => X_MIN + X_AXIS - SCALE_OFF + 5,
         Y2 => Y_MIN - Y_AXIS - RATIO_Y * COUNTER);
      Dot_Horizontal_Line (DOT_SPACING => 7,
         X1 => X_MIN + X_AXIS,
         X2 => X_MAX - X_AXIS / 2,
         Y  => Y_MIN - Y_AXIS - RATIO_Y * COUNTER);
      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => Y_MINIMUM + Y_INC * REAL_TYPE (COUNTER),
         FRACTION     => 2,
         ENG_NOTATION => TRUE,
         TEXT_LINE    => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => 1 + (Y_MIN - Y_AXIS - RATIO_Y * COUNTER + SCALE_OFF)
            / ROW_FACTOR,
         COLUMN  => 1,
         COLOURS => LIGHT_CYAN);
   end loop;
end Label_Y;

-- PURPOSE : Get the starting and end frequency of the graph.

procedure Get_X_Frequency (MIN_FREQUENCY,MAX_FREQUENCY : out REAL_TYPE) is

   MAX_DECADES : constant := 1.0E9;

   MIN_FREQUENCY2,      MAX_FREQUENCY2,
   NUM_DECADES                          : REAL_TYPE;
   CORRECT_FREQUENCY                    : BOOLEAN;

begin
   loop
      CORRECT_FREQUENCY := TRUE;
      Prompts.Read (FORM => GET_X_FREQUENCY_FORM);
      MIN_FREQUENCY2 := GET_X_FREQUENCY_FORM (1).REAL_VALUE;
      MAX_FREQUENCY2 := GET_X_FREQUENCY_FORM (2).REAL_VALUE;
      if (abs (MAX_FREQUENCY2 - MIN_FREQUENCY2) < MINIMUM_VALUE) or
         (MAX_FREQUENCY2 < MIN_FREQUENCY2) then
         Errors.Write (MESSAGE => "The maximum and minimum values for the"
            & " horizontal axis are not acceptable.",
            HELP_INFO => HP_GET_X_FREQUENCY);
         CORRECT_FREQUENCY := FALSE;
      end if;
      NUM_DECADES := MAX_FREQUENCY2 / MIN_FREQUENCY2;
      if NUM_DECADES > MAX_DECADES then
         Errors.Write (MESSAGE => "The frequency range is too wide.",
            HELP_INFO => HP_GET_X_FREQUENCY2);
         CORRECT_FREQUENCY := FALSE;
      end if;
      exit when CORRECT_FREQUENCY;
   end loop;
   MIN_FREQUENCY := MIN_FREQUENCY2;
   MAX_FREQUENCY := MAX_FREQUENCY2;
end Get_X_Frequency;

-- PURPOSE : Gets the maximum and minimum values for the horizontal axis
--           and the the x increment.

procedure Get_Vertical_Inc (VERTICAL_MIN, VERTICAL_MAX : out REAL_TYPE;
   X_INCR, OPTION : out INTEGER_TYPE) is

   COUNTER                              : INTEGER_TYPE;
   VERTICAL_MIN2,       VERTICAL_MAX2   : REAL_TYPE := 0.0;

begin
   Prompts.Read (MESSAGE => "Do you want Auto-scaling ?",
      HELP_INFO => HP_GET_VERTICAL_INC,
      ITEM      => AUTO_SCALE);
   if not AUTO_SCALE then
      VERTICAL_FORM (2).ACCEPTING := TRUE;
      VERTICAL_FORM (3).ACCEPTING := TRUE;
   else
      VERTICAL_FORM (2).ACCEPTING := FALSE;
      VERTICAL_FORM (3).ACCEPTING := FALSE;
   end if;
   Prompts.Read (FORM => VERTICAL_FORM,
      LIST      => PLOT_TYPE_LIST,
      EXCLUSIVE => TRUE,
      HELP_INFO => HP_GET_VERTICAL_INC1);
   if not AUTO_SCALE then
      loop
         VERTICAL_MIN2 := VERTICAL_FORM (2).REAL_VALUE;
         VERTICAL_MAX2 := VERTICAL_FORM (3).REAL_VALUE;
         if (abs (VERTICAL_MAX2 - VERTICAL_MIN2) < MINIMUM_VALUE) or
            (VERTICAL_MAX2 < VERTICAL_MIN2) then
            Errors.Write (MESSAGE => "The maximum and minimum values for"
               & " the horizontal axis are not acceptable.",
               HELP_INFO => HP_GET_VERTICAL_INC2);
         else
            exit;
         end if;
         Prompts.Read (FORM => VERTICAL_FORM,
            LIST      => PLOT_TYPE_LIST,
            EXCLUSIVE => TRUE);
      end loop;
   end if;
   for COUNTER in 2..3 loop
      if PLOT_TYPE_LIST (COUNTER).STATE then
         OPTION := COUNTER;
      end if;
   end loop;
   VERTICAL_MIN := VERTICAL_MIN2;
   VERTICAL_MAX := VERTICAL_MAX2;
   X_INCR := VERTICAL_FORM (1).INTEGER_VALUE;
end Get_Vertical_Inc;

-- PURPOSE : Draws the zero line axis for Y = 0.

procedure Zero_Line (Y_MINIMUM, Y_MAXIMUM, Y_SCALE : in REAL_TYPE) is

begin
   if Y_MINIMUM                 <= 0
      Y_MIN := Y_Min + INTEGER_TYPE (Y_MINIMUM * Y_SCALE);
      Set_Plot_Colour (COLOUR => YELLOW);
      if Y_MAXIMUM > 0.0 then
         Line (X1 => X_MIN + X_AXIS,
            Y1 => Y_MIN - Y_AXIS,
            X2 => X_MIN + X_AXIS + NX_PIXELS,
            Y2 => Y_MIN - Y_AXIS);
      end if;
   end if;
end Zero_Line;

end Graphs;
```

Help

```
with  Ada_Extensions;
use   Ada_Extensions;

package Help is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This displays help text  on the physical screen relating to a
--           subscreen  or a  highlighted option  on a menu.  This package
--           instantiates  the  Sequential_IO generic package.  It uses the
--           disk device to read the  ELECTOR.HLP  file which contains the
--           help text.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Initialize_Help_Path;

   procedure Show (HELP_REQUIRED : in INTEGER_TYPE);

end Help;
```

```
with  Dos_Environment,
      Inputter,
      Screen_Objects,
      Sequential_IO;

use   Dos_Environment,
      Screen_Objects;

package body Help_I

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HELP_PATH   : Dos_Environment.PATH_STRING_TYPE;
   PATH_LENGTH : INTEGER_TYPE;

-- PURPOSE : Gets the initial default path.

procedure Initialize_Help_Path is

   COUNTER : INTEGER_TYPE;

begin
   Dos_Environment.Get_Drive_Directory (PATH_STRING => HELP_PATH);
   for COUNTER in 1..HELP_PATH'length loop
      if HELP_PATH (COUNTER) = CHARACTER'val (0) then
         PATH_LENGTH := COUNTER - 1;
         exit;
      end if;
   end loop;
   if HELP_PATH (PATH_LENGTH) = '\' then
      PATH_LENGTH := PATH_LENGTH - 1;
   end if;
end Initialize_Help_Path;

-- PURPOSE : Shows the  Help Item that is requested.  If the file with the
--           help text cannot be found, or it the requested item cannot be
--           found, the text in the help subscreen informs the user of the
--           error.

procedure Show (HELP_REQUIRED : in INTEGER_TYPE) is

   OF_FINISHED : constant := 30;

   type HELP_ITEM_TYPE is
      record
         INDEX         : INTEGER_TYPE;
         LINE1_LENGTH,
         LINE2_LENGTH  : NATURAL := 0;
         LINE1,
         LINE2         : STRING (1..250);
      end record;

   type HELP_SCREEN_TYPE is
      record
         SCR      : Windows.WINDOW_TYPE;
         BUTTON1  : Buttons.BUTTON_TYPE;
         BUTTON2  : Buttons.BUTTON_TYPE;
      end record;

   HELP_SCREEN : HELP_SCREEN_TYPE;
   HELP_ITEM   : HELP_ITEM_TYPE;
   INPUTS      : Inputter.INPUT_TYPE;

-- PURPOSE : Finds the Help Item from the disk file  ELECTOR.HLP  if it
--           exists. If not, an appropriate error message is set.

procedure Find_Help is

   package Help_IO is new Sequential_IO (HELP_ITEM_TYPE);

   HELP_FILE  : Help_IO.FILE_TYPE;
   FOUND_HELP : BOOLEAN := TRUE;

begin
   begin
      Help_IO.Open (FILE => HELP_FILE,
         MODE => Help_IO.IN_FILE,
         NAME => HELP_PATH(1..PATH_LENGTH) & "\ELECTOR.HLP");
   exception
      when Help_IO.NAME_ERROR =>
         FOUND_HELP := FALSE;
         HELP_ITEM.LINE1_LENGTH := 30;
         HELP_ITEM.LINE1 (1..HELP_ITEM.LINE1_LENGTH)
            := "The Help file cannot be found.";
         HELP_ITEM.LINE2_LENGTH := 0;
   end;
   if FOUND_HELP then
      FOUND_HELP := FALSE;
      while not Help_IO.End_Of_File (FILE => HELP_FILE) loop
         Help_IO.Read (FILE => HELP_FILE,
            ITEM => HELP_ITEM);
         if HELP_REQUIRED = HELP_ITEM.INDEX then
            FOUND_HELP := TRUE;
            exit;
         end if;
      end loop;
      if not FOUND_HELP then
         HELP_ITEM.LINE1_LENGTH := 41;
         HELP_ITEM.LINE1 (1..HELP_ITEM.LINE1_LENGTH)
            := "There is no Help available for this item.";
         HELP_ITEM.LINE2_LENGTH := 0;
      end if;
      Help_IO.Close (FILE => HELP_FILE);
   end if;
end Find_Help;

-- PURPOSE : Displays the Help Item on the screen by means of a  pop-up
--           subscreen.

procedure Display_Help is

   HELP_LENGTHS : constant := 13;
   HELP_SCREEN_COLOUR    : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => WHITE, BACKGROUND => CYAN, BLINK => FALSE);
   BUTTON_COLOUR         : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => RED, BACKGROUND => LIGHT_GRAY, BLINK => FALSE);
   BUTTON_SELECT_COLOUR : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND=>LIGHT_GREEN, BACKGROUND=>LIGHT_GRAY, BLINK=>FALSE);
   BUTTON_UNDER_COLOUR  : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => LIGHT_GRAY, BACKGROUND => BLACK, BLINK => FALSE);
   FRAME_COLOUR          : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => WHITE, BACKGROUND => CYAN, BLINK => FALSE);
   TITLE_COLOUR          : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => WHITE, BACKGROUND => CYAN, BLINK => FALSE);

begin
   Windows.Create (WINDOW => HELP_SCREEN.SCR,
      NAME          => " HELP ",
      KEYS_TEXT     => "Esc=Escape",
      TOP_ROW       => 4,
      LEFT_SIDE     => 11,
      LENGTHS       => HELP_LENGTHS,
      WIDTHS        => 58,
      TEXT_COLOUR   => HELP_SCREEN_COLOUR,
      FRAME_COLOUR  => FRAME_COLOUR,
      TITLE_COLOUR  => TITLE_COLOUR);
   Buttons.Create (BUTTON => HELP_SCREEN.BUTTON1,
      NAME          => " F-inished ",
      OPERATION     => OP_FINISHED,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 2,
      HOT_KEY       => KB_ALT_F);
   Buttons.Create (BUTTON => HELP_SCREEN.BUTTON2,
      NAME          => " M-ove ",
      OPERATION     => OP_MOVE,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 13,
      HOT_KEY       => KB_ALT_M);
   Windows.Draw (WINDOW => HELP_SCREEN.SCR,
      TEXT     => HELP_ITEM.LINE1 (1..HELP_ITEM.LINE1_LENGTH),
      TEXT_ROW => 3);
   if HELP_ITEM.LINE2_LENGTH /= 0 then
      Windows.Paragraphed (WINDOW => HELP_SCREEN.SCR,
         LINE => HELP_ITEM.LINE2 (1..HELP_ITEM.LINE2_LENGTH),
         ROW  => 5);
   end if;
   Buttons.Draw (BUTTON => HELP_SCREEN.BUTTON1);
   Buttons.Draw (BUTTON => HELP_SCREEN.BUTTON2);
   Window_Buffer_To_Screen;
end Display_Help;
```

```
begin
   Find_Help;
   Display_Help;
   Show_Text_Mouse;
   loop
      Inputter.Get_Input (INPUTS => INPUTS);
      if INPUTS.KEY = KB_ESC then
         INPUTS.KEY := KB_ALT_F;
      end if;
      Windows.Handle_Input (WINDOW => HELP_SCREEN.SCR,
         INPUTS => INPUTS);
      Buttons.Handle_Input (BUTTON => HELP_SCREEN.BUTTON1,
         INPUTS => INPUTS);
      Buttons.Handle_Input (BUTTON => HELP_SCREEN.BUTTON2,
         INPUTS => INPUTS);
      if INPUTS.OPERATION = OP_MOVE then
         Windows.Handle_Input (WINDOW => HELP_SCREEN.SCR,
            INPUTS => INPUTS);
      elsif (INPUTS.OPERATION = OP_FINISHED) then
         exit;
      end if;
   end loop;
   Windows.Close;
end Show;

-------------------------------------------------------------------------------

end Help;
```

Inputter

```
-------------------------------------------------------------------------------

with  Ada_Extensions,
      Dos_Environment;

use   Ada_Extensions,
      Dos_Environment;

package Inputter is

-------------------------------------------------------------------------------
-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This is the central subprogram  that collects input  from the
--           keyboard and the mouse.
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------------

   OP_DO : constant := 0;

   -----------------------------------------------------------------------

   type INPUT_TYPE is
      record
         GOT_MOUSE : BOOLEAN;
         BUTTON    : MOUSE_BUTTONS_TYPE;
         Y         : INTEGER_TYPE;
         X         : INTEGER_TYPE;
         KEY       : KEY_VALUES_TYPE;
         OPERATION : OPERATION_TYPE;
      end record;

   procedure Set_Up;

   procedure Get_Input (INPUTS : out INPUT_TYPE);

   -----------------------------------------------------------------------

end Inputter;

-------------------------------------------------------------------------------

package body Inputter is

-------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------------

   MOUSE_ACTIVE : BOOLEAN := FALSE;

-------------------------------------------------------------------------------
-- PURPOSE : Checks to see if a mouse is active.  If so, it sets the mouse
--           pointer to the centre of the screen,  and also  the number of
--           buttons on the mouse.
-------------------------------------------------------------------------------

procedure Set_Up is

   MOUSE_BUTTONS : INTEGER_TYPE;

begin
   Check_For_Mouse (MOUSE_PRESENT => MOUSE_ACTIVE,
      NUMBER_OF_BUTTONS => MOUSE_BUTTONS);
end Set_Up;

-------------------------------------------------------------------------------
-- PURPOSE : If the  mouse is active,  it sets  the fields  of the  INPUTS
--           record to the row and column of the mouse pointer, as well as
--           any buttons pressed. The keyboard is then checked.  If a key
--           was pressed, it is returned as well.
-------------------------------------------------------------------------------

procedure Get_Input (INPUTS : out Inputter.INPUT_TYPE) is

begin
   INPUTS.GOT_MOUSE := MOUSE_ACTIVE;
   if MOUSE_ACTIVE then
      Read_Text_Mouse (MOUSE_ROW => INPUTS.Y,
         MOUSE_COLUMN  => INPUTS.X,
         MOUSE_BUTTONS => INPUTS.BUTTON);
   else
      INPUTS.X := 1;
      INPUTS.Y := 1;
      INPUTS.BUTTON := NO_BUTTON;
   end if;
```

```
   if Keyboard_Character_Ready then
      Get_Keyboard_Character (KEY => INPUTS.KEY);
   else
      INPUTS.KEY := KB_NO_KEY;
   end if;
   INPUTS.OPERATION := OP_NO;
end Get_Input;

--------------------------------------------------------------------------------

end Inputter;
```

Menus

```
--------------------------------------------------------------------------------

with Ada_Extensions;
use  Ada_Extensions;

package Menus is

--------------------------------------------------------------------------------
-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This shows and handles input to the menus displayed on the
--           physical screen. These menus show which options are available
--           and can be selected, and which cannot be selected.
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error. Thus these checks will only
-- add unnecessary checking code. Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
--------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

--------------------------------------------------------------------------------

   NO_SUCH_OPERATION_IN_MENUS : exception;

   OP_ADMITTANCE              : constant := 20;
   OP_BRANCH_CURRENTS         : constant := 21;
   OP_CHANGE_NON_PU           : constant := 22;
   OP_CONVERT_TO_ADMITTANCE   : constant := 23;
   OP_CONVERT_TO_R_L_C        : constant := 24;
   OP_DEFINE                  : constant := 25;
   OP_DELETE_NODE             : constant := 26;
   OP_DRAW                    : constant := 27;
   OP_EXTRA_RESULTS           : constant := 28;
   OP_FOURIER                 : constant := 29;
   OP_INSERT_COMPONENT        : constant := 30;
   OP_LIST_COMPONENTS         : constant := 31;
   OP_LOAD                    : constant := 32;
   OP_NODE_CLEAR              : constant := 33;
   OP_NODE_INSERT             : constant := 34;
   OP_NODE_PERFORM            : constant := 35;
   OP_NODE_RECOVER            : constant := 36;
   OP_NODE_SAVE               : constant := 37;
   OP_NODE_SHOW               : constant := 38;
   OP_NODE_TEST               : constant := 39;
   OP_NODE_WORST              : constant := 40;
   OP_NODE_WORST1             : constant := 41;
   OP_NEW_NETWORK             : constant := 42;
   OP_NODE_VOLTAGES           : constant := 43;
   OP_OPTIM_BANDWIDTH         : constant := 44;
   OP_OPTIM_CLEAR             : constant := 45;
   OP_OPTIM_INSERT            : constant := 46;
   OP_OPTIM_MAXIMISE          : constant := 47;
   OP_OPTIM_MINIMISE          : constant := 48;
   OP_OPTIM_SET               : constant := 49;
   OP_OPTIM_SHOW              : constant := 50;
   OP_PHASOR                  : constant := 51;
   OP_PLOTS                   : constant := 52;
   OP_POLAR                   : constant := 53;
   OP_POWER                   : constant := 54;
   OP_QUIT                    : constant := 55;
   OP_RENUMBER_NODE           : constant := 56;
   OP_SAVE                    : constant := 57;
   OP_SENSITIVITY             : constant := 58;
   OP_SET_DIRECTORY           : constant := 59;
   OP_SET_DRIVE               : constant := 60;
   OP_SHOW_BRANCH             : constant := 61;
   OP_SHOW_COMPONENT          : constant := 62;
   OP_SHOW_NODES              : constant := 63;
   OP_TITLE                   : constant := 64;
   OP_TUTORIAL                : constant := 65;

   procedure Get_Menu_Operation (OPERATION : out OPERATION_TYPE);

   procedure Change_Menu_Option (OPERATION : IN OPERATION_TYPE;
      STATE : in BOOLEAN);

   --------------------------------------------------------------------------------

end Menus;

--------------------------------------------------------------------------------

with  Dos_Environment,
      Help,
      Inputter;

use   Dos_Environment,
      Inputter;

package body Menus is

--------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error. Thus these checks will only
-- add unnecessary checking code. Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
--------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
```

```
   pragma SUPPRESS (RANGE_CHECK);
-------------------------------------------------------------------------------
   SUB_MENU                : constant := 10;
   PARENT_MENU             : constant := 11;
   RIGHT_MENU              : constant := 12;
   LEFT_MENU               : constant := 13;
   NON_SELECTABLE          : constant := 14;
   OP_DONE                 : constant := 15;
   OP_HELPED               : constant := 16;
-------------------------------------------------------------------------------
   type TYPES_OF_MENU_TYPE is (STACKED_MENU, SPREAD_MENU);

   type MENU_TYPE (LENGTHS : ROW_TYPE);
   type MENU_LINK is access MENU_TYPE;

   type MENU_ITEM_TYPE (TEXT_WIDTH : COLUMN_TYPE := 10) is
      record
         TEXT          : STRING (1..TEXT_WIDTH);
         OPERATION     : OPERATION_TYPE;
         SELECTABLE    : BOOLEAN;
         HOT_KEY       : KEY_VALUES_TYPE;
         SHORTCUT_KEY  : CHARACTER;
         CHILD         : MENU_LINK;
      end record;

   type DATA_TYPE is array (POSITIVE range <>) of MENU_ITEM_TYPE;

   type MENU_TYPE (LENGTHS : ROW_TYPE) is
      record
         TYPE_OF_MENU  : TYPES_OF_MENU_TYPE;
         PARENT        : MENU_LINK;
         DATA          : DATA_TYPE (1..LENGTHS);
         CHOICE        : ROW_TYPE;
         TOP_ROW       : ROW_TYPE;
         LEFT_SIDE     : COLUMN_TYPE;
         WIDTHS        : COLUMN_TYPE;
      end record;

   type COLOUR_TYPE is
      record
         FRAME       : SCREEN_COLOUR_TYPE;
         TEXT        : SCREEN_COLOUR_TYPE;
         NON_SELECT  : SCREEN_COLOUR_TYPE;
         SHORTCUT    : SCREEN_COLOUR_TYPE;
         SELECT_BAR  : SCREEN_COLOUR_TYPE;
      end record;

   type MENU_BASE_TYPE is
      record
         FIRST       : MENU_LINK;
         CURRENT     : MENU_LINK;
         PREVIOUS    : MENU_LINK;
         COLOUR      : COLOUR_TYPE;
      end record;
-------------------------------------------------------------------------------
   BASE  : MENU_BASE_TYPE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package Spread is


   procedure Draw_Menu (MENU : in MENU_LINK; COLOUR : in COLOUR_TYPE);

   procedure Handle_Input (MENU : in MENU_LINK;
      COLOUR : in COLOUR_TYPE;
      INPUTS : in out INPUT_TYPE);


end Spread;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package body Spread is

-------------------------------------------------------------------------------
-- PURPOSE : Draws a spread menu on the physical screen as well as the
--           selection bar.
-------------------------------------------------------------------------------

procedure Draw_Menu (MENU : in MENU_LINK;
   COLOUR : in COLOUR_TYPE) is

   COUNTER : POSITIVE;

begin
   Main_Buffer_Line (LINE_COLOURS => (FOREGROUND =>COLOUR.TEXT.FOREGROUND,
      BACKGROUND => COLOUR.TEXT.BACKGROUND,
      BLINK      => COLOUR.TEXT.BLINK));
   for COUNTER in 1..MENU.LENGTHS loop
      String_To_Main_Buffer (MESSAGE => MENU.DATA (COUNTER).TEXT,
         ROW        => MENU.TOP_ROW,
         COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS * (COUNTER - 1) +
                       (MENU.WIDTHS + 2 - MENU.DATA(COUNTER).TEXT_WIDTH)/2,
         COLOURS1 => COLOUR.TEXT,
         COLOURS2 => COLOUR.SHORTCUT);
   end loop;
   String_To_Main_Buffer (MESSAGE => "~F1=Help",
      ROW      => MENU.TOP_ROW,
      COLUMN   => 72,
      COLOURS1 => COLOUR.TEXT,
      COLOURS2 => COLOUR.SHORTCUT);
   Main_Buffer_To_Screen;
   Make_Line_Bar (ROW => MENU.TOP_ROW,
      COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS * (MENU.CHOICE - 1),
      BAR_LENGTH => MENU.WIDTHS,
      COLOURS    => COLOUR.SELECT_BAR);
end Draw_Menu;

-------------------------------------------------------------------------------
-- PURPOSE : Handles input that concerns a spread menu, otherwise it
--           ignores it.
-------------------------------------------------------------------------------

procedure Handle_Input (MENU : in MENU_LINK;
   COLOUR : in COLOUR_TYPE;
   INPUTS : in out INPUT_TYPE) is

   COUNTER, CODE : INTEGER_TYPE;

begin
   if INPUTS.GOT_MOUSE and (INPUTS.BUTTON = LEFT_BUTTON) then
      if (INPUTS.Y = MENU.TOP_ROW) and (INPUTS.X in MENU.LEFT_SIDE..
         (MENU.LEFT_SIDE + MENU.WIDTHS * MENU.LENGTHS)) then
         MENU.CHOICE := (INPUTS.X - MENU.LEFT_SIDE) / MENU.WIDTHS + 1;
         if MENU.DATA (MENU.CHOICE).SELECTABLE then
            INPUTS.OPERATION := MENU.DATA (MENU.CHOICE).OPERATION;
         end if;
         Remove_Line_Bar;
         Make_Line_Bar (ROW => MENU.TOP_ROW,
            COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS * (MENU.CHOICE-1),
            BAR_LENGTH => MENU.WIDTHS,
            COLOURS    => COLOUR.SELECT_BAR);
      end if;
   else
      case INPUTS.KEY is
         when KB_ENTER =>
            if MENU.DATA (MENU.CHOICE).SELECTABLE then
               INPUTS.OPERATION := MENU.DATA (MENU.CHOICE).OPERATION;
            end if;
         when KB_LEFT =>
            Remove_Line_Bar;
            if MENU.CHOICE = 1 then
               MENU.CHOICE := MENU.LENGTHS;
            else
               MENU.CHOICE := MENU.CHOICE - 1;
            end if;
            Make_Line_Bar (ROW => MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS*(MENU.CHOICE-1),
               BAR_LENGTH => MENU.WIDTHS,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_RIGHT =>
            Remove_Line_Bar;
            if MENU.CHOICE = MENU.LENGTHS then
               MENU.CHOICE := 1;
            else
               MENU.CHOICE := MENU.CHOICE + 1;
            end if;
            Make_Line_Bar (ROW => MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS*(MENU.CHOICE-1),
               BAR_LENGTH => MENU.WIDTHS,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_HOME =>
            Remove_Line_Bar;
            MENU.CHOICE := 1;
            Make_Line_Bar (ROW => MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE,
               BAR_LENGTH => MENU.WIDTHS,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_END =>
            Remove_Line_Bar;
            MENU.CHOICE := MENU.LENGTHS;
            Make_Line_Bar (ROW => MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + MENU.WIDTHS*(MENU.CHOICE-1),
               BAR_LENGTH => MENU.WIDTHS,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_UPPER_A..KB_UPPER_Z | KB_LOWER_A..KB_LOWER_Z =>
            for COUNTER in 1..MENU.LENGTHS loop
               CODE := CHARACTER'pos (MENU.DATA (COUNTER).SHORTCUT_KEY);
               if (INPUTS.KEY = CODE) or (INPUTS.KEY = (CODE + 32)) then
                  if MENU.DATA (COUNTER).SELECTABLE then
                     INPUTS.OPERATION := MENU.DATA (COUNTER).OPERATION;
                     MENU.CHOICE := COUNTER;
                     Remove_Line_Bar;
                     Make_Line_Bar (ROW => MENU.TOP_ROW,
                        COLUMN     => MENU.LEFT_SIDE +
                                      MENU.WIDTHS * (MENU.CHOICE - 1),
                        BAR_LENGTH => MENU.WIDTHS,
                        COLOURS    => COLOUR.SELECT_BAR);
                  end if;
               end if;
            end loop;
         when KB_F1 =>
            Help.Show (HELP_REQUIRED => MENU.DATA(MENU.CHOICE).OPERATION);
            Draw_Menu (MENU => MENU,
               COLOUR => COLOUR);
            Show_Text_Mouse;
         when others => null;
      end case;
   end if;
end Handle_Input;

-------------------------------------------------------------------------------

end Spread;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package Stacked is

   procedure Draw_Menu (MENU : in MENU_LINK; COLOUR : in COLOUR_TYPE);

   procedure Handle_Input (MENU : in MENU_LINK; COLOUR : in COLOUR_TYPE;
      INPUTS : in out INPUT_TYPE);


end Stacked;
-------------------------------------------------------------------------------

package body Stacked is

   NEW_PRESS : BOOLEAN := TRUE;

-------------------------------------------------------------------------------
-- PURPOSE : Draws the stack menu on the physical screen as well as the
--           selection bar.
-------------------------------------------------------------------------------

procedure Draw_Menu (MENU : in MENU_LINK; COLOUR : in COLOUR_TYPE) is
```

```
   NORMAL_FRAME : constant FRAME_TYPE
      := (218, 196, 191, 179, 192, 196, 217, 179, 32, 179); -- ┌─┐│└─┘│ │

   COUNTER : POSITIVE;

begin
   Window_Buffer_Setup (NUMBER_OF_ROW => MENU.LENGTHS + 2,
      NUMBER_OF_COLUMNS => MENU.WIDTHS);
   Clear_Window_Buffer (COLOURS => COLOUR.TEXT);
   Frame_Window_Buffer (BORDERS => NORMAL_FRAME,
      COLOURS => COLOUR.FRAME);
   for COUNTER in 1..MENU.LENGTHS loop
      if MENU.DATA (COUNTER).SELECTABLE then
         String_To_Window_Buffer (MESSAGE => MENU.DATA (COUNTER).TEXT,
            ROW     => COUNTER,
            COLUMN  => 2,
            COLOURS1 => COLOUR.TEXT,
            COLOURS2 => COLOUR.SHORTCUT);
      else
         String_To_Window_Buffer (MESSAGE => MENU.DATA (COUNTER).TEXT,
            ROW     => COUNTER,
            COLUMN  => 2,
            COLOURS1 => COLOUR.NON_SELECT,
            COLOURS2 => COLOUR.NON_SELECT);
      end if;
   end loop;
   New_Window_Location (TOP_ROW => MENU.TOP_ROW,
      LEFT_SIDE => MENU.LEFT_SIDE);
   Window_Buffer_To_Screen;
   Make_Shadow;
   Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
      COLUMN     => MENU.LEFT_SIDE + 2,
      BAR_LENGTH => MENU.WIDTHS - 4,
      COLOURS    => COLOUR.SELECT_BAR);
end Draw_Menu;

-------------------------------------------------------------------------
-- PURPOSE : Handles input that concerns a stack menu, otherwise it
--           ignores it.
-------------------------------------------------------------------------

procedure Handle_Input (MENU : in MENU_LINK; COLOUR : in COLOUR_TYPE;
   INPUTS : in out INPUT_TYPE) is

   COUNTER, CODE : POSITIVE;

begin
   if INPUTS.GOT_MOUSE and (INPUTS.BUTTON=LEFT_BUTTON) and NEW_PRESS then
      if INPUTS.Y in (MENU.TOP_ROW + 1)..(MENU.TOP_ROW + MENU.LENGTHS) and
         INPUTS.X in (MENU.LEFT_SIDE + 2)..
         (MENU.LEFT_SIDE + MENU.WIDTHS - 2) then
         MENU.CHOICE := INPUTS.Y - MENU.TOP_ROW;
         if MENU.DATA (MENU.CHOICE).SELECTABLE then
            INPUTS.OPERATION := MENU.DATA (MENU.CHOICE).OPERATION;
         end if
      else
         INPUTS.OPERATION := PARENT_MENU;
      end if;
      Remove_Line_Bar;
      Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
         COLUMN     => MENU.LEFT_SIDE + 2,
         BAR_LENGTH => MENU.WIDTHS - 4,
         COLOURS    => COLOUR.SELECT_BAR);
      NEW_PRESS := FALSE;
   else
      case INPUTS.KEY is
         when KB_ENTER =>
            if MENU.DATA (MENU.CHOICE).SELECTABLE then
               INPUTS.OPERATION := MENU.DATA (MENU.CHOICE).OPERATION;
            end if;
         when KB_UP =>
            Remove_Line_Bar;
            if MENU.CHOICE = 1 then
               MENU.CHOICE := MENU.LENGTHS;
            else
               MENU.CHOICE := MENU.CHOICE - 1;
            end if;
            Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + 2,
               BAR_LENGTH => MENU.WIDTHS - 4,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_DOWN =>
            Remove_Line_Bar;
            if MENU.CHOICE = MENU.LENGTHS then
               MENU.CHOICE := 1;
            else
               MENU.CHOICE := MENU.CHOICE + 1;
            end if;
            Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + 2,
               BAR_LENGTH => MENU.WIDTHS - 4,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_HOME =>
            Remove_Line_Bar;
            MENU.CHOICE := 1;
            Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + 2,
               BAR_LENGTH => MENU.WIDTHS - 4,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_END =>
            Remove_Line_Bar;
            MENU.CHOICE := MENU.LENGTHS;
            Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
               COLUMN     => MENU.LEFT_SIDE + 2,
               BAR_LENGTH => MENU.WIDTHS - 4,
               COLOURS    => COLOUR.SELECT_BAR);
         when KB_RIGHT => INPUTS.OPERATION := RIGHT_MENU;
         when KB_LEFT  => INPUTS.OPERATION := LEFT_MENU;
         when KB_UPPER_A..KB_UPPER_Z | KB_LOWER_A..KB_LOWER_Z =>
            for COUNTER in 1..MENU.LENGTHS loop
               CODE := CHARACTER'pos (MENU.DATA (COUNTER).SHORTCUT_KEY);
               if (INPUTS.KEY = CODE) or (INPUTS.KEY = (CODE + 32)) then
                  INPUTS.OPERATION := MENU.DATA (COUNTER).OPERATION;
                  MENU.CHOICE := COUNTER;
                  Remove_Line_Bar;
                  Make_Line_Bar (ROW => MENU.CHOICE + MENU.TOP_ROW,
                     COLUMN     => MENU.LEFT_SIDE + 2,
                     BAR_LENGTH => MENU.WIDTHS - 4,
                     COLOURS    => COLOUR.SELECT_BAR);
               end if;
            end loop;
         when KB_F1 =>
            Main_Buffer_To_Screen;
            Help.Show (HELP_REQUIRED => MENU.DATA(MENU.CHOICE).OPERATION);
            INPUTS.OPERATION := OP_HELPED;
            Show_Text_Mouse;
         when KB_ESC =>
            INPUTS.OPERATION := PARENT_MENU;
         when others => null;
      end case;
      if INPUTS.BUTTON /= LEFT_BUTTON then
         NEW_PRESS := TRUE;
      end if;
   end if;
end Handle_Input;

-------------------------------------------------------------------------

end Stacked;

-------------------------------------------------------------------------
-- PURPOSE : Determines if the menu is a spread or stacked menu,  and then
--           calls the appropriate method to draw the menu.
-------------------------------------------------------------------------

procedure Draw_Menu (MENU : in MENU_LINK;
   COLOUR : in COLOUR_TYPE) is

begin
   if MENU.TYPE_OF_MENU = SPREAD_MENU then
      Spread.Draw_Menu (MENU => MENU,
         COLOUR => COLOUR);
   else
      Stacked.Draw_Menu (MENU => MENU,
         COLOUR => COLOUR);
   end if;
end Draw_Menu;

-------------------------------------------------------------------------
-- PURPOSE : This draws all the menus on the screen up to that  pointed to
--           by BASE.CURRENT
-------------------------------------------------------------------------

procedure Redraw is

   MENU_DRAW : MENU_LINK;

begin
   Main_Buffer_To_Screen;
   MENU_DRAW := BASE.FIRST;
   Draw_Menu (MENU => MENU_DRAW,
      COLOUR => BASE.COLOUR);
   while MENU_DRAW /= BASE.CURRENT loop
      MENU_DRAW := MENU_DRAW.DATA (MENU_DRAW.CHOICE).CHILD;
      Draw_Menu (MENU => MENU_DRAW,
         COLOUR => BASE.COLOUR);
   end loop;
end Redraw;

-------------------------------------------------------------------------
-- PURPOSE : Searches the menu data structure to find the  OPERATION  that
--           corresponds to a particular hotkey.
-------------------------------------------------------------------------

procedure Find_HotKey (MENU : in MENU_LINK;
   KEY       : in KEY_VALUES_TYPE;
   OPERATION : out OPERATION_TYPE) is

   COUNTER    : POSITIVE;
   OPERATIONS : OPERATION_TYPE := OP_DO;

begin
   for COUNTER in 1..MENU.DATA'last loop
      if MENU.DATA (COUNTER).OPERATION = SUB_MENU then
         Find_Hotkey (MENU => MENU.DATA (COUNTER).CHILD,
            KEY       => KEY,
            OPERATION => OPERATIONS);
         if OPERATIONS /= OP_DO then
            exit;
         end if;
      elsif MENU.DATA (COUNTER).HOT_KEY = KEY then
         if MENU.DATA (COUNTER).SELECTABLE then
            OPERATIONS := MENU.DATA (COUNTER).OPERATION;
         else
            OPERATIONS := NON_SELECTABLE;
         end if;
         exit;
      end if;
   end loop;
   OPERATION := OPERATIONS;
end Find_Hotkey;

-------------------------------------------------------------------------
-- PURPOSE : Determines if the menu is a spread or stacked menu,  and then
--           calls the appropriate method to handle input to that menu.
-------------------------------------------------------------------------

procedure Handle_Input (MENU : in MENU_LINK;
   COLOUR : in COLOUR_TYPE;
   INPUTS : in out INPUT_TYPE) is

begin
   if MENU.TYPE_OF_MENU = SPREAD_MENU then
      Spread.Handle_Input (MENU => MENU,
         COLOUR => COLOUR,
         INPUTS => INPUTS);
   else
      Stacked.Handle_Input (MENU => MENU,
         COLOUR => COLOUR,
         INPUTS => INPUTS);
   end if;
end Handle_Input;

-------------------------------------------------------------------------
-- PURPOSE : Procedure  that  recursively  calls itself to find  a certain
--           menu option.  If it is found,  the state of the option is set
--           so that it can be selected, or not selected.
-------------------------------------------------------------------------

procedure Find_Menu_Option (MENU : in MENU_LINK;
```

```
   STATE     : in BOOLEAN;
   OPERATION : in out OPERATION_TYPE) is

   COUNTER   : POSITIVE;

begin
   for COUNTER in 1..MENU.DATA'last loop
      if MENU.DATA (COUNTER).OPERATION = SUB_MENU then
         Find_Menu_Option (MENU => MENU.DATA (COUNTER).CHILD,
            OPERATION => OPERATION,
            STATE     => STATE);
         if OPERATION = OP_DONE then
            exit;
         end if;
      elsif MENU.DATA (COUNTER).OPERATION = OPERATION then
         MENU.DATA (COUNTER).SELECTABLE := STATE;
         OPERATION := OP_DONE;
         exit;
      end if;
   end loop;
end Find_Menu_Option;

-------------------------------------------------------------------------
-- PURPOSE : Makes an option in the menu structure non-selectable or
--           selectable.
-------------------------------------------------------------------------

procedure Change_Menu_Option (OPERATION : in OPERATION_TYPE;
   STATE : in BOOLEAN) is

   OPERATIONS : OPERATION_TYPE := OPERATION;

begin
   Find_Menu_Option (MENU => BASE.FIRST,
      OPERATION => OPERATIONS,
      STATE     => STATE);
   if OPERATIONS /= OP_DONE then
      raise NO_SUCH_OPERATION_IN_MENUS;
   end if;
end Change_Menu_Option;

-------------------------------------------------------------------------
-- PURPOSE : Gets an OPERATION  from the user which is determined from the
--           menu data structure.
-------------------------------------------------------------------------

procedure Get_Menu_Operation (OPERATION : out OPERATION_TYPE) is

   TEMP_OPERATION : OPERATION_TYPE;
   INPUTS         : INPUT_TYPE;
   COUNTER        : INTEGER_TYPE;

begin
   Show_Text_Mouse;
   Draw_Menu (MENU => BASE.FIRST,
      COLOUR => BASE.COLOUR);
   BASE.CURRENT := BASE.FIRST;
   loop
      Get_Input (INPUTS => INPUTS);
      case INPUTS.KEY is
         when KB_F2..KB_F10 | KB_SHIFT_F1..KB_ALT_F10 =>
            Find_Hotkey (MENU => BASE.FIRST,
               KEY       => INPUTS.KEY,
               OPERATION => INPUTS.OPERATION);
         when others =>
            Handle_Input (MENU => BASE.CURRENT,
               COLOUR => BASE.COLOUR,
               INPUTS => INPUTS);
      end case;
      case INPUTS.OPERATION is
         when OP_HELPED => Redraw;
         when LEFT_MENU =>
            loop
               if BASE.CURRENT.PARENT.CHOICE = 1 then
                  BASE.CURRENT.PARENT.CHOICE:=BASE.CURRENT.PARENT.LENGTHS;
               else
                  BASE.CURRENT.PARENT.CHOICE:=BASE.CURRENT.PARENT.CHOICE-1;
               end if;
               exit when BASE.CURRENT.PARENT.DATA
                  (BASE.CURRENT.PARENT.CHOICE).CHILD /= null;
            end loop;
            BASE.CURRENT := BASE.CURRENT.PARENT.DATA
               (BASE.CURRENT.PARENT.CHOICE).CHILD;
            Redraw;
         when RIGHT_MENU =>
            loop
               if BASE.CURRENT.PARENT.CHOICE =
                  BASE.CURRENT.PARENT.LENGTHS then
                  BASE.CURRENT.PARENT.CHOICE := 1;
               else
                  BASE.CURRENT.PARENT.CHOICE :=
                     BASE.CURRENT.PARENT.CHOICE + 1;
               end if;
               exit when BASE.CURRENT.PARENT.DATA
                  (BASE.CURRENT.PARENT.CHOICE).CHILD /= null;
            end loop;
            BASE.CURRENT := BASE.CURRENT.PARENT.DATA
               (BASE.CURRENT.PARENT.CHOICE).CHILD;
            Redraw;
         when SUB_MENU | PARENT_MENU =>
            BASE.PREVIOUS := BASE.CURRENT;
            if INPUTS.OPERATION = SUB_MENU then
               if BASE.CURRENT.DATA(BASE.CURRENT.CHOICE).CHILD/=null then
                  BASE.CURRENT :=
                     BASE.CURRENT.DATA (BASE.CURRENT.CHOICE).CHILD;
               end if;
            else
               if BASE.CURRENT.PARENT /= null then
                  BASE.CURRENT := BASE.CURRENT.PARENT;
               end if;
            end if;
            if BASE.CURRENT /= BASE.PREVIOUS then
               if INPUTS.OPERATION = SUB_MENU then
                  Draw_Menu (MENU => BASE.CURRENT,
                     COLOUR => BASE.COLOUR);
               else
                  Redraw;
               end if;
            end if;
         when others =>
            if INPUTS.OPERATION /= OP_DO then
               exit;
            end if;
      end case;
   end loop;
   OPERATION := INPUTS.OPERATION;
   Hide_Text_Mouse;
end Get_Menu_Operation;

-------------------------------------------------------------------------

begin
   -- Set up main (spread) menu
   BASE.COLOUR := (
    FRAME =>
      (FOREGROUND => BLACK,  BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
    TEXT  =>
      (FOREGROUND => BLACK, BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
    NON_SELECT =>
      (FOREGROUND => DARK_GRAY, BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
    SHORTCUT   =>
      (FOREGROUND => RED, BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
    SELECT_BAR =>
      (FOREGROUND => RED, BACKGROUND => GREEN, BLINK => FALSE));
   BASE.FIRST := new MENU_TYPE (LENGTHS => 5);

   BASE.FIRST.TYPE_OF_MENU := SPREAD_MENU;
   BASE.FIRST.PARENT       := null;
   BASE.FIRST.CHOICE       := 1;
   BASE.FIRST.TOP_ROW      := 1;
   BASE.FIRST.LEFT_SIDE    := 1;
   BASE.FIRST.WIDTHS       := 12;
   BASE.FIRST.DATA         := (
     (TEXT_WIDTH   => 6,
      TEXT         => "~F~ile",
      OPERATION    => SUB_MENU,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'F',
      CHILD        => null),

     (TEXT_WIDTH   => 6,
      TEXT         => "~E~dit",
      OPERATION    => SUB_MENU,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'E',
      CHILD        => null),

     (TEXT_WIDTH   => 7,
      TEXT         => "~M~odes",
      OPERATION    => SUB_MENU,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'M',
      CHILD        => null),

     (TEXT_WIDTH   => 9,
      TEXT         => "~A~nalyse",
      OPERATION    => SUB_MENU,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_K ,
      SHORTCUT_KEY => 'A',
      CHILD        => null),

     (TEXT_WIDTH   => 11,
      TEXT         => "~T~utorials",
      OPERATION    => OP_TUTORIAL,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'T',
      CHILD        => null));

   -- Second level menus are declared below

   BASE.FIRST.DATA (1).CHILD := new
      MENU_TYPE (LENGTHS => 6);
   BASE.FIRST.DATA (2).CHILD := new
      MENU_TYPE (LENGTHS => 8);
   BASE.FIRST.DATA (3).CHILD := new
      MENU_TYPE (LENGTHS => 4);
   BASE.FIRST.DATA (4).CHILD := new
      MENU_TYPE (LENGTHS => 13);

   BASE.CURRENT := BASE.FIRST.DATA (1).CHILD;
   BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
   BASE.CURRENT.PARENT       := BASE.FIRST;
   BASE.CURRENT.CHOICE       := 1;
   BASE.CURRENT.TOP_ROW      := 2;
   BASE.CURRENT.LEFT_SIDE    := 1;
   BASE.CURRENT.WIDTHS       := 16
   BASE.CURRENT.DATA         := (
     (TEXT_WIDTH   => 14,
      TEXT         => "~L~oad       F2",
      OPERATION    => OP_LOAD,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_F2,
      SHORTCUT_KEY => 'L',
      CHILD        => null),

     (TEXT_WIDTH   => 14,
      TEXT         => "~S~ave       F3",
      OPERATION    => OP_SAVE,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_F3,
      SHORTCUT_KEY => 'S',
      CHILD        => null),

     (TEXT_WIDTH   => 7,
      TEXT         => "~D~rive",
      OPERATION    => OP_SET_DRIVE,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'D',
      CHILD        => null),

     (TEXT_WIDTH   => 11,
      TEXT         => "D~i~rectory",
      OPERATION    => OP_SET_DIRECTORY,
      SELECTABLE   => TRUE,
```

```
    NOT_KEY      => KB_NO_KEY,
    SHORTCUT_KEY => 'I',
    CHILD        => null),

   (TEXT_WIDTH   => 9,
    TEXT         => "~R-esults",
    OPERATION    => OP_EXTRA_RESULTS,
    SELECTABLE   => TRUE,
    NOT_KEY      => KB_NO_KEY,
    SHORTCUT_KEY => 'R',
    CHILD        => null),

   (TEXT_WIDTH   => 14,
    TEXT         => "~Q-uit      F10",
    OPERATION    => OP_QUIT,
    SELECTABLE   => TRUE,
    NOT_KEY      => KB_F10,
    SHORTCUT_KEY => 'Q',
    CHILD        => null)));

BASE.CURRENT := BASE.FIRST.DATA (2).CHILD;
BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
BASE.CURRENT.PARENT       := BASE.FIRST;
BASE.CURRENT.CHOICE       := 1;
BASE.CURRENT.TOP_ROW      := 2;
BASE.CURRENT.LEFT_SIDE    := 13;
BASE.CURRENT.WIDTHH       := 24;
BASE.CURRENT.DATA         := (
  (TEXT_WIDTH   => 13,
   TEXT         => "~N-ew Circuit",
   OPERATION    => OP_NEW_NETWORK,
   SELECTABLE   => TRUE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'N',
   CHILD        => null),

  (TEXT_WIDTH   => 7,
   TEXT         => "~T-itle",
   OPERATION    => OP_TITLE,
   SELECTABLE   => TRUE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'T',
   CHILD        => null),

  (TEXT_WIDTH   => 20,
   TEXT         => "~I-nsert          F5",
   OPERATION    => OP_INSERT_COMPONENT,
   SELECTABLE   => TRUE,
   NOT_KEY      => KB_F5,
   SHORTCUT_KEY => 'I',
   CHILD        => null),

  (TEXT_WIDTH   => 19,
   TEXT         => "~C-hange Magn/Phase",
   OPERATION    => OP_CHANGE_MGN_PH,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'C',
   CHILD        => null),

  (TEXT_WIDTH   => 20,
   TEXT         => "~L-ist Components F6",
   OPERATION    => OP_LIST_COMPONENTS,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_F6,
   SHORTCUT_KEY => 'L',
   CHILD        => null),

  (TEXT_WIDTH   => 13,
   TEXT         => "Show ~B-ranch",
   OPERATION    => OP_SHOW_BRANCH,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'B',
   CHILD        => null),

  (TEXT_WIDTH   => 16,
   TEXT         => "~S-how Component",
   OPERATION    => OP_SHOW_COMPONENT,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'S',
   CHILD        => null),

  (TEXT_WIDTH   => 20,
   TEXT         => "~D-raw Circuit    F7",
   OPERATION    => OP_DRAW,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_F7,
   SHORTCUT_KEY => 'D',
   CHILD        => null)));

BASE.CURRENT := BASE.FIRST.DATA    .CHILD;
BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
BASE.CURRENT.PARENT       := BASE.FIRST;
BASE.CURRENT.CHOICE       := 1;
BASE.CURRENT.TOP_ROW      := 2;
BASE.CURRENT.LEFT_SIDE    := 25;
BASE.CURRENT.WIDTHH       := 27;
BASE.CURRENT.DATA         := (
  (TEXT_WIDTH   => 23,
   TEXT         => "Convert to ~A-dmittance",
   OPERATION    => OP_CONVERT_TO_ADMITTANCE,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'A',
   CHILD        => null),

  (TEXT_WIDTH   => 16,
   TEXT         => "~C-onvert to RLC",
   OPERATION    => OP_CONVERT_TO_RLC,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'C',
   CHILD        => null),

  (TEXT_WIDTH   => 23,
   TEXT         => "~R-enumber Node      F4",
   OPERATION    => OP_RENUMBER_NODE,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_F4,
   SHORTCUT_KEY => 'R',
   CHILD        => null),

  (TEXT_WIDTH   => 13,
   TEXT         => "~D-elete Node",
   OPERATION    => OP_DELETE_NODE,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'D',
   CHILD        => null)));

BASE.CURRENT := BASE.FIRST.DATA (4).CHILD;
BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
BASE.CURRENT.PARENT       := BASE.FIRST;
BASE.CURRENT.CHOICE       := 1;
BASE.CURRENT.TOP_ROW      := 2;
BASE.CURRENT.LEFT_SIDE    := 37;
BASE.CURRENT.WIDTHH       := 26;
BASE.CURRENT.DATA         := (
  (TEXT_WIDTH   => 22,
   TEXT         => "~D-efine Nodes       F8",
   OPERATION    => OP_DEFINE,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_F8,
   SHORTCUT_KEY => 'D',
   CHILD        => null),

  (TEXT_WIDTH   => 22,
   TEXT         => "Show ~N-odes         F9",
   OPERATION    => OP_SHOW_NODES,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_F9,
   SHORTCUT_KEY => 'N',
   CHILD        => null),

  (TEXT_WIDTH   => 22,
   TEXT         => "~C-alculate Admittance",
   OPERATION    => OP_ADMITTANCE,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'C',
   CHILD        => null),

  (TEXT_WIDTH   => 15,
   TEXT         => "Node ~V-oltages",
   OPERATION    => OP_NODE_VOLTAGES,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'V',
   CHILD        => null),

  (TEXT_WIDTH   => 17,
   TEXT         => "~B-ranch Currents",
   OPERATION    => OP_BRANCH_CURRENTS,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'B',
   CHILD        => null),

  (TEXT_WIDTH   => 7,
   TEXT         => "~P-ower",
   OPERATION    => OP_POWER,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'P',
   CHILD        => null),

  (TEXT_WIDTH   => 18,
   TEXT         => "Circuit ~R-esponse",
   OPERATION    => OP_PLOTS,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'R',
   CHILD        => null),

  (TEXT_WIDTH   => 9,
   TEXT         => "~F-ourier",
   OPERATION    => OP_FOURIER,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'F',
   CHILD        => null),

  (TEXT_WIDTH   => 8,
   TEXT         => "P-h~asor",
   OPERATION    => OP_PHASOR,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'H',
   CHILD        => null),

  (TEXT_WIDTH   => 7,
   TEXT         => "Po~l-ar",
   OPERATION    => OP_POLAR,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'L',
   CHILD        => null),

  (TEXT_WIDTH   => 13,
   TEXT         => "~S-ensitivity",
   OPERATION    => OP_SENSITIVITY,
   SELECTABLE   => FALSE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'S',
   CHILD        => null),

  (TEXT_WIDTH   => 22,
   TEXT         => "~M-onte Carlo       =>",
   OPERATION    => SUB_MENU,
   SELECTABLE   => TRUE,
   NOT_KEY      => KB_NO_KEY,
   SHORTCUT_KEY => 'M',
   CHILD        => null),

  (TEXT_WIDTH   => 22,
   TEXT         => "~O-ptimize          =>",
   OPERATION    => SUB_MENU,
   SELECTABLE   => TRUE,
```

```
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'O',
      CHILD        => null));

   -- Third level menus are declared below

   BASE.CURRENT := BASE.FIRST.DATA (4).CHILD;
   BASE.CURRENT.DATA (12).CHILD := new
      MENU_TYPE (LENGTH8 => 9);

   BASE.CURRENT := BASE.CURRENT.DATA (12).CHILD;
   BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
   BASE.CURRENT.PARENT       := BASE.FIRST.DATA (4).CHILD;
   BASE.CURRENT.CHOICE       := 1;
   BASE.CURRENT.TOP_ROW      := 10;
   BASE.CURRENT.LEFT_SIDE    := 50;
   BASE.CURRENT.WIDTH8       := 21;
   BASE.CURRENT.DATA         := (
     (TEXT_WIDTH   => 9,
      TEXT         => "~R~ecover",
      OPERATION    => OP_MONTE_RECOVER,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'R',
      CHILD        => null),

     (TEXT_WIDTH   => 12,
      TEXT         => "~S~ave Monte",
      OPERATION    => OP_MONTE_SAVE,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'S',
      CHILD        => null),

     (TEXT_WIDTH   => 13,
      TEXT         => "~C~lear Monte",
      OPERATION    => OP_MONTE_CLEAR,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'C',
      CHILD        => null),

     (TEXT_WIDTH   => 14,
      TEXT         => "~I~nsert Monte",
      OPERATION    => OP_MONTE_INSERT,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'I',
      CHILD        => null),

     (TEXT_WIDTH   => 17,
      TEXT         => "~S~h~ow Components",
      OPERATION    => OP_MONTE_SHOW,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'H',
      CHILD        => null),

     (TEXT_WIDTH   => 9,
      TEXT         => "~P~erform",
      OPERATION    => OP_MONTE_PERFORM,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'P',
      CHILD        => null),

      (TEXT_WIDTH  => 13,
      TEXT         => "~T~    Random",
      OPERATION    => OP_MO~ &_TEST,
      SELECTABLE   => TRUE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'T',
      CHILD        => null),

     (TEXT_WIDTH   => 12,
      TEXT         => "~W~orst Case",
      OPERATION    => OP_MONTE_WORST,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'W',
      CHILD        => null),

     (TEXT_WIDTH   => 14,
      TEXT         => "Honte C~h~ee 2",
      OPERATION    => OP_MONTE_WO~RT2,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'A',
      CHILD        => null));

   BASE.CURRENT := BASE.FIRST.DATA (4).CHILD;
   BASE.CURRENT.DATA (13).CHILD := new
      MENU_TYPE (LENGTH8 => 7);

   BASE.CURRENT := BASE.CURRENT.DATA (13).CHILD;
   BASE.CURRENT.TYPE_OF_MENU := STACKED_MENU;
   BASE.CURRENT.PARENT       := BASE.FIRST.DATA (4).CHILD;
   BASE.CURRENT.CHOICE       := 1;
   BASE.CURRENT.TOP_ROW      := 11;
   BASE.CURRENT.LEFT_SIDE    := 50;
   BASE.CURRENT.WIDTH8       := 21;
   BASE.CURRENT.DATA         := (
     (TEXT_WIDTH   => 17,
      TEXT         => "~C~lear Variables",
      OPERATION    => OP_OPTIM_CLEAR,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KE"
      SHORTCUT_KEY => 'C',
      CHILD        => null),

     (TEXT_WIDTH   => 12,
      TEXT         => "~I~nsert Variables",
      OPERATION    => OP_OPTIM_INSERT,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'I',
      CHILD        => null),

     (TEXT_WIDTH   => 17,
      TEXT         => "  I~H~ow Components",
      OPERATION    => OP_OPTIM_SHOW,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'H',
      CHILD        => null),

     (TEXT_WIDTH   => 11,
      TEXT         => "~B~andwidth",
      OPERATION    => OP_OPTIM_BANDWIDTH,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'B',
      CHILD        => null),

     (TEXT_WIDTH   => 10,
      TEXT         => "~M~axim1se",
      OPERATION    => OP_OPTIM_MAXIMISE,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'M',
      CHILD        => null),

     (TEXT_WIDTH   => 17,
      TEXT         => "Mi~n~imise",
      OPER         => OP_OPTIM_MINIMISE,
      SEL          => FALSE,
      HOT          => KB_NO_KEY,
      SHOR    KEY  => 'N',
      CHILD        => null),

     (TEXT_WIDTH   => 3,
      TEXT         => "~S~et",
      OPERATION    => OP_OPTIM_SET,
      SELECTABLE   => FALSE,
      HOT_KEY      => KB_NO_KEY,
      SHORTCUT_KEY => 'S',
      CHILD        => null));

   BASE.CURRENT := BASE.FIRST;

end Menus;
```

Monte

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Monte is

-- Author  : Warren Bockle
-- Date    : June 1992
-- Purpose : This performs statistical analysis using the Monte-Carlo
--           simulation methods for specified deviations in component
--           values. The graphs are drawn on the graphics screen while
--           text is output to a text screen.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Clear_Variables;

   procedure Destroy_Data_Structure;

   procedure Load (NETLIST1 : in out NETLIST_TYPE);

   procedure Insert (NODE_MAX : in NODE_TYPE;
      NETLIST1 : in out NETLIST_TYPE);

   procedure Perform (NODE_MAX : in NODE_TYPE;
      NETLIST1 : in out NETLIST_TYPE);

   procedure Save;

   procedure Show (NETLIST1 : in out NetLIST_TYPE);

   procedure Test_Random;

   procedure Worst (SINGLE : in BOOLEAN;
      NODE_MAX : in NODE_TYPE;
      NETLIST1 : in out NETLIST_TYPE);

end Monte;
```

```
with  Bit_Ops,
      Control_Solver,
      Graphs,
      Math_Lib,
      Netlist_Supporter,
      Screens,
      Unchecked_Deallocation;

use   Bit_Ops,
      Graphs,
      Screens;

package body Monte is

   use Screens.Main;

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_MONTE_INSERT : constant := 170;
   HP_PERFORM      : constant := 171;
   HP_PERFORM2     : constant := 172;
   HP_TEST_RANDOM  : constant := 173;
   HP_WORST        : constant := 174;

   MAXIMUM_SIMULATIONS : constant := 500;

   subtype LONG_INTEGER_TYPE is LONG_INTEGER;

   type VOLT_ARRAY_TYPE is array (1..MAXIMUM_SIMULATIONS) of REAL_TYPE;
   type Y_VALUES_ARRAY_TYPE is array (0..MX_PIXELS, 1..2) of REAL_TYPE;

   type MONTE_DATA_STRUCTURE_TYPE is
      record
         REAL_VOLTS,
         IMAG_VOLTS   : VOLT_ARRAY_TYPE;
         Y_VALUES     : Y_VALUES_ARRAY_TYPE;
         NETLIST2,
         NETLIST3     : NETLIST_TYPE;
      end record;

   type MONTE_ACCESS_TYPE is access MONTE_DATA_STRUCTURE_TYPE;

   MONT                                  : MONTE_ACCESS_TYPE;
   MONT_EXISTS                           : BOOLEAN := FALSE;
   SEED                                  : LONG_INTEGER_TYPE;
   X2_PREV,              X1_PREV,
   Y2_PREV,              DEPTH_PREV      : INTEGER_TYPE;

   MONTE_LIST1 : LIST_ARRAY_TYPE (1..3) := (
      (TEXT_LENGTH => 4,  TEXT => "Mode",      STATE => TRUE),
      (TEXT_LENGTH => 5,  TEXT => "Phase",     STATE => FALSE),
      (TEXT_LENGTH => 9,  TEXT => "Magnitude", STATE => FALSE));

   MONTE_LIST2 : LIST_ARRAY_TYPE (1..2) := (
   (TEXT_LENGTH => 11, TEXT => "Gaussian Distribution", STATE => TRUE),
   (TEXT_LENGTH => 10, TEXT => "Uniform Distribution",  STATE => FALSE));

   MONTE_FORM1 : FIELDS_TYPE (1..6) :=
      (1 =>
         (VALUE_LENGTH  => 2,
         TEXT_LENGTH    => 13,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 2,
         ACTUAL_LENGTH  => 1,
         ROW            => 4,
         VALUE          => "1 ",
         TEXT           => "Higher Node 1",
         MAX_INTEGER    => MAX_NODES,
         MIN_INTEGER    => 1,
         INTEGER_VALUE  => 1),
      2 =>
         (VALUE_LENGTH  => 2,
         TEXT_LENGTH    => 12,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 1,
         ACTUAL_LENGTH  => 1,
         ROW            => 5,
         VALUE          => "0 ",
         TEXT           => "Lower Node 1",
         MAX_INTEGER    => MAX_NODES,
         MIN_INTEGER    => 0,
         INTEGER_VALUE  => 0),
      3 =>
         (VALUE_LENGTH  => 3,
         TEXT_LENGTH    => 21,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 2,
         ROW            => 6,
         VALUE          => "10 ",
         TEXT           => "Number of Simulations",
         MAX_INTEGER    => MAXIMUM_SIMULATIONS,
         MIN_INTEGER    => 2,
         INTEGER_VALUE  => 10),
      4 =>
         (VALUE_LENGTH  => 3,
         TEXT_LENGTH    => 18,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 2,
         ROW            => 7,
         VALUE          => "10 ",
         TEXT           => "Random number seed",
         MAX_INTEGER    => 999,
         MIN_INTEGER    => 0,
         INTEGER_VALUE  => 10),
      5 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 19,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 5,
         ROW            => 8,
         VALUE          => "1.0E3          ",
         TEXT           => "Impedance Frequency",
         MAX_REAL       => 1.0E25,
         MIN_REAL       => MINIMUM_VALUE,
         REAL_VALUE     => 1000.0),
      6 =>
         (VALUE_LENGTH  => 2,
         TEXT_LENGTH    => 14,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING      => FALSE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 2,
         ROW            => 6,
         VALUE          => "10",
         TEXT           => "Plot Increment",
         MAX_INTEGER    => 50,
         MIN_INTEGER    => 1,
         INTEGER_VALUE  => 10));

   PERFORM_LIST : LIST_ARRAY_TYPE (1..2) := (
   (TEXT_LENGTH => 13, TEXT => "Interim results", STATE => TRUE),
   (TEXT_LENGTH =>  9, TEXT => "Bar graph",       STATE => FALSE));

   INSERT_FORM : FIELDS_TYPE (1..5) :=
      (1 =>
         (VALUE_LENGTH  => 15,
         TEXT_LENGTH    => 20,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING      => TRUE,
         TEST_AGAINST   => 0,
         ACTUAL_LENGTH  => 3,
         ROW            => 4,
         VALUE          => "1.0            ",
         TEXT           => "Standard deviation %",
         MAX_REAL       => 1.0E40,
```

```
            MIN_REAL        => 0.0,
            REAL_VALUE      => 1.0),
         2 =>
            (VALUE_LENGTH   => 2,
            TEXT_LENGTH     => 13,
            IS_INTEGER_TYPE => TRUE,
            ACCEPTING       => TRUE,
            TEST_AGAINST    => 3,
            ACTUAL_LENGTH   => 1,
            ROW             => 5,
            VALUE           => "1 ",
            TEXT            => "Higher Node 1",
            MAX_INTEGER     => MAX_NODES,
            MIN_INTEGER     => 1,
            INTEGER_VALUE   => 1),
         3 =>
            (VALUE_LENGTH   => 2,
            TEXT_LENGTH     => 13,
            IS_INTEGER_TYPE => TRUE,
            ACCEPTING       => TRUE,
            TEST_AGAINST    => 2,
            ACTUAL_LENGTH   => 1,
            ROW             => 6,
            VALUE           => "0 ",
            TEXT            => "Lower  Node 1",
            MAX_INTEGER     => MAX_NODES,
            MIN_INTEGER     => 0,
            INTEGER_VALUE   => 0),
         4 =>
            (VALUE_LENGTH   => 2,
            TEXT_LENGTH     => 13,
            IS_INTEGER_TYPE => TRUE,
            ACCEPTING       => FALSE,
            TEST_AGAINST    => 5,
            ACTUAL_LENGTH   => 1,
            ROW             => 7,
            VALUE           => "1 ",
            TEXT            => "Higher Node 2",
            MAX_INTEGER     => MAX_NODES,
            MIN_INTEGER     => 1,
            INTEGER_VALUE   => 1),
         5 =>
            (VALUE_LENGTH   => 2,
            TEXT_LENGTH     => 13,
            IS_INTEGER_TYPE => TRUE,
            ACCEPTING       => FALSE,
            TEST_AGAINST    => 4,
            ACTUAL_LENGTH   => 1,
            ROW             => 8,
            VALUE           => "0 ",
            TEXT            => "Lower  Node 2",
            MAX_INTEGER     => MAX_NODES,
            MIN_INTEGER     => 0,
            INTEGER_VALUE   => 0)));

   TEST_RANDOM_FORM : FIELDS_TYPE (1..3) :=
      (1 =>
         (VALUE_LENGTH   => 3,
         TEXT_LENGTH     => 21,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ACTUAL_LENGTH   => 2,
         ROW             => 4,
         VALUE           => "10 ",
         TEXT            => "Number of Simulations",
         MAX_INTEGER     => MAXIMUM_SIMULATIONS,
         MIN_INTEGER     => 2,
         INTEGER_VALUE   => 10),
      2 =>
         (VALUE_LENGTH   => 3,
         TEXT_LENGTH     => 18,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ACTUAL_LENGTH   => 2,
         ROW             => 5,
         VALUE           => "10 ",
         TEXT            => "Random number seed",
         MAX_INTEGER     => 999,
         MIN_INTEGER     => 0,
         INTEGER_VALUE   => 10),
      3 =>
         (VALUE_LENGTH   => 13,
         TEXT_LENGTH     => 20,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ACTUAL_LENGTH   => 3,
         ROW             => 6,
         VALUE           => "1.0          ",
         TEXT            => "Standard deviation 1",
         MAX_REAL        => 250.0,
         MIN_REAL        => 0.0,
         REAL_VALUE      => 1.0)));

   MES_ERROR_1 : constant STRING := "The statistical variation in "
      & "magnitude is too small to compute accurately.";

   MES_MONTE_1 : constant STRING := "    Component    Nodes      Value     "
      & "  Units    \ Standard Dvn.    Distribution     ";

-------------------------------------------------------------------------------
-- PURPOSE : Draws a 3D bar with a certain depth.
-------------------------------------------------------------------------------

procedure Bar_3D (X1, Y1, X2, Y2, DEPTH : in INTEGER_TYPE) is

   COUNTER : INTEGER_TYPE := 0;

begin
   -- Remove hidden lines from the previous bar
   Set_Plot_Colour (COLOUR => BLUE);
   Line (X1 => X2_PREV, -- bottom right diagonal
      Y1 => Y1_PREV,
      X2 => X2_PREV + DEPTH_PREV,
      Y2 => Y1_PREV - DEPTH_PREV);
   Line (X1 => X2_PREV + DEPTH_PREV, -- back right side
      Y1 => Y2 - DEPTH,
      X2 => X2_PREV + DEPTH_PREV,
      Y2 => Y1_PREV - DEPTH_PREV);
   if Y2 < Y2_PREV then
      Line (X1 => X2_PREV, -- top right diagonal
         Y1 => Y2_PREV,
         X2 => X2_PREV + DEPTH_PREV,
         Y2 => Y2_PREV - DEPTH_PREV);
      Line (X1 => X1, -- back top side
         Y1 => Y2_PREV - DEPTH_PREV,
         X2 => X2_PREV + DEPTH_PREV,
         Y2 => Y2_PREV - DEPTH_PREV);
   end if;

   Set_Plot_Colour (COLOUR => LIGHT_PURPLE);
   Line (X1 => X1, -- left front side
      Y1 => Y1,
      X2 => X1,
      Y2 => Y2);
   Line (X1 => X1, -- bottom front side
      Y1 => Y1,
      X2 => X2,
      Y2 => Y1);
   Line (X1 => X2, -- right front side
      Y1 => Y1,
      X2 => X2,
      Y2 => Y2);
   Line (X1 => X1, -- top front side
      Y1 => Y2,
      X2 => X2,
      Y2 => Y2);
   Line (X1 => X1, -- top left diagonal
      Y1 => Y2,
      X2 => X1 + DEPTH,
      Y2 => Y2 - DEPTH);
   Line (X1 => X2, -- top right diagonal
      Y1 => Y2,
      X2 => X2 + DEPTH,
      Y2 => Y2 - DEPTH);
   Line (X1 => X2, -- bottom right diagonal
      Y1 => Y1,
      X2 => X2 + DEPTH,
      Y2 => Y1 - DEPTH);
   Line (X1 => X2 + DEPTH, -- back right side
      Y1 => Y1 - DEPTH,
      X2 => X2 + DEPTH,
      Y2 => Y2 - DEPTH);
   Line (X1 => X1 + DEPTH, -- back top side
      Y1 => Y2 - DEPTH,
      X2 => X2 + DEPTH,
      Y2 => Y2 - DEPTH);

   Set_Plot_Colour (COLOUR => LIGHT_RED);
   loop
      Dot_Vertical_Line (DOT_SPACING => 2,
         Y1 => Y2,
         Y2 => Y1,
         X  => X1 + COUNTER);
      COUNTER := COUNTER + 3;
      exit when (X1 + COUNTER) > X2;
   end loop;
   DEPTH_PREV := DEPTH;
   X2_PREV    := X2;
   Y1_PREV    := Y1;
   Y2_PREV    := Y2;
end Bar_3D;

-------------------------------------------------------------------------------
-- PURPOSE : Clears  all the components  in the branches of  NETLIST2  and
--           NETLIST3.
-------------------------------------------------------------------------------

procedure Clear_Variables is

begin
   if MONT_EXISTS then
      Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
      Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST3);
      loop
         Inner.Clear_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.Clear_Branch (NETLIST_OBJECT => MONT.NETLIST3);
         begin
            Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
            Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST3);
         exception
            when NO_MORE_BRANCHES => exit;
         end;
      end loop;
   end if;
end Clear_Variables;

-------------------------------------------------------------------------------
-- PURPOSE : Destroys the dynamic data structure used for the  Monte-Carlo
--           calculations.  This  frees memory for other  memory intensive
--           operations. A flag is cleared showing that the data structure
--           no longer exists.
-------------------------------------------------------------------------------

procedure Destroy_Data_Structure is

   procedure Dispose is new
      Unchecked_Deallocation(MONTE_DATA_STRUCTURE_TYPE,MONTE_ACCESS_TYPE);

begin
   if MONT_EXISTS then
      Inner.Initialize (NETLIST_OBJECT => MONT.NETLIST2);
      Inner.Initialize (NETLIST_OBJECT => MONT.NETLIST3);
      Dispose (MONT);
      MONT_EXISTS := FALSE;
   end if;
end Destroy_Data_Structure;

-------------------------------------------------------------------------------
-- PURPOSE : Produces a "random" value using an Integer Seed.
-------------------------------------------------------------------------------

function Random return REAL_TYPE is

   TEMP1 : LONG_INTEGER_TYPE;

begin
   TEMP1 := SEED / 127_773;
```

```
   SEED := 16_807 * (SEED - TEMP1 * 127_773) - TEMP1 * 2_836;
   if SEED < 0 then
      SEED := SEED + 2_147_483_647;
   end if;
   return REAL_TYPE (SEED) * 4.656_612_8756-10;
end Random;

-- PURPOSE : Calculate a Gauss distribution.

procedure Gauss (STANDARD_DEVIATION, VALUE1 : in REAL_TYPE;
   VALUE2 : out REAL_TYPE) is

   COUNTER : POSITIVE;
   SUM     : REAL_TYPE := 0.0;

begin
   for COUNTER in 1..12 loop
      SUM := SUM + Random;
   end loop;
   VALUE2 := (SUM - 6.0) * STANDARD_DEVIATION + VALUE1 + VALUE1;
end Gauss;

-- PURPOSE : Creates NETLIST2 and NETLIST3 so that they have the same
--           number of branches as NETLIST2.

procedure Setup_Monte (NETLIST1 : in out NETLIST_TYPE) is

   MADE_BASE : BOOLEAN := FALSE;

begin
   Y_MIN := MAX_Y_COORD - 3 * Y_AXIS / 2;
   if not MONT_EXISTS then
      MONT := NEW MONTE_DATA_STRUCTURE_TYPE;
      MADE_BASE := TRUE;
      Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
      Inner.New_Branch   (NETLIST_OBJECT => MONT.NETLIST2);
      Inner.New_Branch   (NETLIST_OBJECT => MONT.NETLIST3);
      loop
         begin
            Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         exception
            when NO_MORE_BRANCHES => exit;
         end;
         Inner.New_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.New_Branch (NETLIST_OBJECT => MONT.NETLIST3);
      end loop;
      MONT_EXISTS := TRUE;
   end if;
exception
   when STORAGE_ERROR =>
      Errors.Write (MESSAGE =>  "There is not enough memory for " &
         "Monte Processes.");
      if MADE_BASE then
         Destroy_Data_Structure;
      end if;
      raise USER_ESCAPE;
end Setup_Monte;

-- PURPOSE : Insert data for a component for the Monte simulation.

procedure Insert (NODE_MAX : in NODE_TYPE;
   NETLIST1 : in out NETLIST_TYPE) is

   SCREEN_ROW                          : INTEGER_TYPE := 3;
   COMPONENT,           COMPONENT2     : ELEMENT_TYPE;
   GAUSSIAN                            : BOOLEAN;
   STANDARD_DVN,        VALUE          : REAL_TYPE;
   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2      : NODE_TYPE;
   NETLIST4                            : NETLIST_TYPE := NETLIST1;

begin
   Process_Title (TITLE => "MONTE COMPONENTS AND TOLERANCES");
   Setup_Monte (NETLIST1 => NETLIST1);
   loop
      begin
         Screen_Supporter.Get_Component (COMPONENT => COMPONENT);
         case COMPONENT is
            when RESISTOR    => COMPONENT2 := GR_ADMIT;
            when CENT_SOURCE => COMPONENT2 := REAL_CENT;
            when VOLT_SOURCE => COMPONENT2 := REAL_VOLT;
            when others      => COMPONENT2 := COMPONENT;
         end case;
         if COMPONENT2 >= GR_ADMIT then
            INSERT_FORM (4).ACCEPTING := TRUE;
            INSERT_FORM (5).ACCEPTING := TRUE;
         else
            INSERT_FORM (4).ACCEPTING := FALSE;
            INSERT_FORM (5).ACCEPTING := FALSE;
         end if;
         Prompts.Read (FORM => INSERT_FORM,
            LIST      => MONTE_LIST2,
            HELP_INFO => HP_MONTE_INSERT);
         STANDARD_DVN := INSERT_FORM (1).REAL_VALUE;
         HIGH_NODE1   := INSERT_FORM (2).INTEGER_VALUE;
         LOW_NODE1    := INSERT_FORM (3).INTEGER_VALUE;
         HIGH_NODE2   := INSERT_FORM (4).INTEGER_VALUE;
         LOW_NODE2    := INSERT_FORM (5).INTEGER_VALUE;
         Inner.Find_Branch (COMPONENT => MUT_INDEX,
            HIGH_NODE    => HIGH_NODE1,
            LOW_NODE     => LOW_NODE1,
            NETLIST_OBJECT => NETLIST4);
         Inner.Read_Netlist (COMPONENT => COMPONENT2,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST4);
         if abs (VALUE) > 0.0 and then VALUE /= UN_VALUE then
            exit;
         else
            raise MEANINGLESS_COMPONENT;
         end if;
      exception
         when MEANINGLESS_COMPONENT | NODE_PAIR_DOES_NOT_EXIST =>
            Errors.Write (MESSAGE => "Such a component does not exist");
      end;
   end loop;

   Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   while NETLIST1 /= NETLIST4 loop
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST3);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;

   Inner.Write_Netlist (COMPONENT => COMPONENT2,
      VALUE          => VALUE,
      HIGH_NODE1     => HIGH_NODE1,
      LOW_NODE1      => LOW_NODE1,
      HIGH_NODE2     => HIGH_NODE2,
      LOW_NODE2      => LOW_NODE2,
      NETLIST_OBJECT => MONT.NETLIST1);
   STANDARD_DVN := STANDARD_DVN / 100.0;
   if MONTE_LIST2 (2).STATE then
      STANDARD_DVN := -STANDARD_DVN;
   end if;
   Inner.Write_Netlist (COMPONENT => COMPONENT2,
      VALUE          => STANDARD_DVN,
      HIGH_NODE1     => HIGH_NODE1,
      LOW_NODE1      => LOW_NODE1,
      HIGH_NODE2     => HIGH_NODE2,
      LOW_NODE2      => LOW_NODE2,
      NETLIST_OBJECT => MONT.NETLIST2);
   Write (ITEM => RES_MONTE_),
      FORM => UNDERLINED);
   if (COMPONENT2 = GR_ADMIT) and (VALUE /= 0.0) then
      VALUE := 1.0 / VALUE;
   end if;
   Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
      HIGH_NODE1 => HIGH_NODE1,
      LOW_NODE1  => LOW_NODE1,
      HIGH_NODE2 => HIGH_NODE2,
      LOW_NODE2  => LOW_NODE2,
      VALUE      => VALUE,
      PHASE      => 0.0,
      ROW        => SCREEN_ROW);
   Write (ITEM => abs (STANDARD_DVN) * 100.0,
      WHOLE    => 7,
      FRACTION => 5);
   Cursor_Set (ROW => 3, COLUMN => 47);
   if STANDARD_DVN > 0.0 then
      Write (ITEM => "Gaussian");
   else
      Write (ITEM => "Uniform");
   end if;
end Insert;

-- PURPOSE : Load monte netlist data for the monte simulation.

procedure Load (NETLIST1 : in out NETLIST_TYPE) is

begin
   Setup_Monte (NETLIST1 => NETLIST1);
   Outer.Load_Monte_Netlist (NETLIST_OBJECT1 => MONT.NETLIST2,
      NETLIST_OBJECT2 => MONT.NETLIST3);
end Load;

-- PURPOSE : Perform the Monte-Carlo simulations.

procedure Perform (NODE_MAX : in NODE_TYPE;
   NETLIST1 : in out NETLIST_TYPE) is

   HIGH_NODE_1,              LOW_NODE_1,
   HIGH_NODE1,               LOW_NODE1,
   HIGH_NODE2,               LOW_NODE2          : NODE_TYPE;
   INCR,                     NUM_SIMULATIONS,
   NUM_SUM,                  OPTION,
   SCREEN_ROW,               SEED1,
   WIDTH,                    X,
   Y                                            : INTEGER_TYPE;
   DISPLAY_LINE                                 : Texts.TEXT_TYPE;
   BAR_GRAPH,                INTERIM,
   RESPONSE                                     : BOOLEAN;
   COUNTER,
   MINF,                     MAXF,
   IB1,                      IB2,
   IB3,                      IB4                : INTEGER_TYPE := 0;
   ANGLE_E,                  DECIBEL,
   FREQUENCY_MONT1,          IMAG_VOLTS1,
   INCR,                     MAGNITUDE,
   MAX,                      MAX_ANGLE_E,
   MAX_DECIBEL,              MAX_MAGNITUDE,
   MEAN,                     MEAN_ANGLE_E,
   MEAN_DECIBEL,             MIN,
   MIN_ANGLE_E,              MIN_DECIBEL,
   MIN_MAGNITUDE,            NOMINAL_MAGNITUDE,
   NORM,                     NUM_SIMULATIONS2,
   REAL_VOLTS1,              STANDARD_DVN,
   STANDARD_DVN_ANGLE_E, STANDARD_DVN_DECIBEL,
   STANDARD_DVN_MAGNITUDE, VALUE,
   VALUE2,                   VALUES,
   VALUE4,                   TEMP_VALUE         : REAL_TYPE;
   SUM_ANGLE_E,              SUM_ANGLE_E2,
   SUM_DECIBEL,              SUM_DECIBEL2,
   SUM_MAGNITUDE,            SUM_MAGNITUDE2     : REAL_TYPE := 0.0;

begin
   Process_Title (TITLE => "MONTE-CARLO SIMULATION");
   Setup_Monte (NETLIST1 => NETLIST1);
   MONTE_FORM1 (3).ACCEPTING := TRUE;
   MONTE_FORM1 (4).ACCEPTING := TRUE;
   MONTE_FORM1 (5).ACCEPTING := TRUE;
```

```
MONTE_FORM1 (6).ACCEPTING := FALSE;
Prompts.Read (FORM => MONTE_FORM1,
   LIST      => PERFORM_LIST,
   EXCLUSIVE => FALSE,
   HELP_INFO => HP_PERFORM);
HIGH_NODE_1     := MONTE_FORM1 (1).INTEGER_VALUE;
LOW_NODE_1      := MONTE_FORM1 (2).INTEGER_VALUE;
NUM_SIMULATIONS := MONTE_FORM1 (3).INTEGER_VALUE;
SEED1           := MONTE_FORM1 (4).INTEGER_VALUE;
FREQUENCY_MONTE := MONTE_FORM1 (5).REAL_VALUE;
NUM_SIMULATIONS2 := REAL_TYPE (NUM_SIMULATIONS);
SEED := 2 * (LONG_INTEGER_TYPE (abs (SEED1)) + 16_183) + 1;
if PERFORM_LIST (1).STATE then
   Cursor_Next_Line;
   Cursor_Set (ROW => 3, COLUMN => 3);
   Write (ITEM => "  Count     Real Voltage     Imag Voltage  ",
      FORM => UNDERLINED);
   Cursor_Next_Line;
end if;
loop
   COUNTER := COUNTER + 1;
   Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST3);
   loop
      if not Inner.Empty_Branch (NETLIST_OBJECT => MONT.NETLIST2) then
         for COMPONENT in CAPS..COUPLING loop
            begin
               Inner.Read_Netlist (COMPONENT => COMPONENT,
                  VALUE       => VALUE1,
                  HIGH_NODE1  => HIGH_NODE1,
                  LOW_NODE1   => LOW_NODE1,
                  HIGH_NODE2  => HIGH_NODE2,
                  LOW_NODE2   => LOW_NODE2,
                  NETLIST_OBJECT => MONT.NETLIST2);
               if abs (VALUE2) > MINIMUM_VALUE then
                  Inner.Read_Netlist (COMPONENT => COMPONENT,
                     VALUE       => STANDARD_DVM,
                     HIGH_NODE1  => HIGH_NODE1,
                     LOW_NODE1   => LOW_NODE1,
                     HIGH_NODE2  => HIGH_NODE2,
                     LOW_NODE2   => LOW_NODE2,
                     NETLIST_OBJECT => MONT.NETLIST3);
                  if STANDARD_DVM < 0.0 then
                     loop
                        TEMP_VALUE := Random;
                        VALUE := VALUE2 + 2.0*abs(STANDARD_DVM*VALUE2)
                           * (TEMP_VALUE - 0.5);
                        exit when (VALUE2 * VALUE) > 0.0;
                     end loop;
                     VALUE2 := VALUE;
                  else
                     loop
                        Gauss (STANDARD_DEVIATION => STANDARD_DVM,
                           VALUE1 => VALUE2,
                           VALUE2 => VALUE4);
                        exit when (VALUE2 * VALUE) > 0.0;
                     end loop;
                     VALUE2 := VALUE4;
                  end if;
                  Inner.Write_Netlist (COMPONENT => COMPONENT,
                     VALUE       => VALUE2,
                     HIGH_NODE1  => HIGH_NODE1,
                     LOW_NODE1   => LOW_NODE1,
                     HIGH_NODE2  => HIGH_NODE2,
                     LOW_NODE2   => LOW_NODE2,
                     NETLIST_OBJECT => NETLIST1);
               end if;
            exception
               when BRANCH_OUT_OF_RANGE => null;
            end;
         end loop;
      end if;
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST3);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
   Central_Solver.Solve_AC_Network (NODE => RLC,
      AC_FREQUENCY   => FREQUENCY_MONTE,
      CALCULATE      => TRUE,
      NETLIST_OBJECT => NETLIST1);
   Central_Solver.Get_Voltages (LOW_NODE => LOW_NODE_1,
      HIGH_NODE => HIGH_NODE_1,
      REAL_VOLT => MONT.REAL_VOLTS (COUNTER),
      IMAG_VOLT => MONT.IMAG_VOLTS (COUNTER));
   if PERFORM_LIST (1).STATE then
      Cursor_Next_Line;
      Cursor_Set (ROW => 4, COLUMN => 6);
      Write (ITEM => COUNTER,
         ITEM_WIDTH => 3);
      Write (ITEM => MONT.REAL_VOLTS (COUNTER),
         WHOLE    => 7,
         FRACTION => 5);
      Write (ITEM => MONT.IMAG_VOLTS (COUNTER),
         WHOLE    => 6,
         FRACTION => 5);
   end if;
   exit when (COUNTER >= NUM_SIMULATIONS) or Halt_Processing;
end loop;
MAX_ANGLE_E := Maths.Angle (X_COORDINATE => MONT.REAL_VOLTS (COUNTER),
   Y_COORDINATE => MONT.IMAG_VOLTS (COUNTER));
MIN_ANGLE_E := MAX_ANGLE_E;
MAX_MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE =>MONT.REAL_VOLTS(1),
   IMAG_VALUE => MONT.IMAG_VOLTS (1));
MIN_MAGNITUDE := MAX_MAGNITUDE;
MIN_DECIBEL := Math_Lib.Ln (MIN_MAGNITUDE) * LN_20;
Cursor_Next_Line;
Cursor_Set (ROW => 6, COLUMN => 3);
Write (ITEM => "    Volts        Decibels     Degrees  ",
   FORM => UNDERLINED);
SCREEN_ROW := 7;
for COUNTER in 1..NUM_SIMULATIONS loop
   if SCREEN_ROW > 14 then
      Status.Wait;
      SCREEN_ROW := 4;
      Process_Title (TITLE => "MONTE-CARLO SIMULATION");
      Cursor_Set (ROW => 3, COLUMN => 3);
      Write (ITEM => "    Volts        Decibels     Degrees  ",
         FORM => UNDERLINED);
   end if;
   MAGNITUDE := Maths.Find_Magnitude (
      REAL_VALUE => MONT.REAL_VOLTS (COUNTER),
      IMAG_VALUE => MONT.IMAG_VOLTS (COUNTER));
   DECIBEL := Math_Lib.Ln (MAGNITUDE) * LN_20;
   ANGLE_E := Maths.Angle (X_COORDINATE => MONT.REAL_VOLTS (COUNTER),
      Y_COORDINATE => MONT.IMAG_VOLTS (COUNTER));
   if ANGLE_E > MAX_ANGLE_E then
      MAX_ANGLE_E := ANGLE_E;
   end if;
   if ANGLE_E < MIN_ANGLE_E then
      MIN_ANGLE_E := ANGLE_E;
   end if;
   Cursor_Next_Line;
   Cursor_Set (ROW => SCREEN_ROW, COLUMN => 3);
   Write (ITEM => MAGNITUDE,
      FRACTION => 5);
   Write (ITEM => DECIBEL,
      WHOLE    => 4,
      FRACTION => 2);
   Write (ITEM => ANGLE_E,
      EXPONENT => 0,
      WHOLE    => 4,
      FRACTION => 5);
   SCREEN_ROW := SCREEN_ROW + 1;
end loop;
DECIBEL := 0.0;
for COUNTER in 1..NUM_SIMULATIONS loop
   if (MONT.REAL_VOLTS (COUNTER) > 0.0) and
      (MONT.IMAG_VOLTS (COUNTER) >= 0.0) then
      IB1 := 1;
   end if;
   if (MONT.REAL_VOLTS (COUNTER) <= 0.0) and
      (MONT.IMAG_VOLTS (COUNTER) >= 0.0) then
      IB2 := 1;
   end if;
   if (MONT.REAL_VOLTS (COUNTER) > 0.0) and
      (MONT.IMAG_VOLTS (COUNTER) < 0.0) then
      IB4 := 1;
   end if;
   if (MONT.REAL_VOLTS (COUNTER) <= 0.0) and
      (MONT.IMAG_VOLTS (COUNTER) < 0.0) then
      IB3 := 1;
   end if;
end loop;
NUM_SUM := IB1 * 8 + IB2 * 4 + IB3 * 2 + IB4;
Status.Wait;
Process_Title (TITLE => "MONTE-CARLO SIMULATION");
SCREEN_ROW := 4;
Cursor_Set (ROW => SCREEN_ROW, COLUMN => 2);
Write (ITEM => " Angles lie in Quadrants");
Write (ITEM => IB1, ITEM_WIDTH => 3);
Write (ITEM => ",");
Write (ITEM => IB2 * 2, ITEM_WIDTH => 3);
Write (ITEM => ",");
Write (ITEM => IB3 * 3, ITEM_WIDTH => 3);
Write (ITEM => ",");
Write (ITEM => IB4 * 4, ITEM_WIDTH => 3);
case NUM_SUM is
   when 9 | 11 | 13 =>
      NINF := 1;
   when 6 |  7 | 14 | 15 =>
      NEGF := 1;
   when 5 | 10 =>
      NEGF := 1;
      Errors.Write (MESSAGE => "Angles lie in 2 isolated quadrants.",
         HELP_INFO => HP_PERFORM2);
   when others => null;
end case;
for COUNTER in 1..NUM_SIMULATIONS loop
   MAGNITUDE := Maths.Find_Magnitude (
      REAL_VALUE => MONT.REAL_VOLTS (COUNTER),
      IMAG_VALUE => MONT.IMAG_VOLTS (COUNTER));
   if MAGNITUDE > MAX_MAGNITUDE then
      MAX_MAGNITUDE := MAGNITUDE;
   end if;
   if MAGNITUDE < MIN_MAGNITUDE then
      MIN_MAGNITUDE := MAGNITUDE;
   end if;
   SUM_MAGNITUDE := SUM_MAGNITUDE + MAGNITUDE;
   SUM_MAGNITUDE2 := SUM_MAGNITUDE2 + MAGNITUDE * MAGNITUDE;
   DECIBEL := Math_Lib.Ln (MAGNITUDE) * LN_20;
   ANGLE_E := Maths.Angle (X_COORDINATE => MONT.REAL_VOLTS (COUNTER),
      Y_COORDINATE => MONT.IMAG_VOLTS (COUNTER));
   if (NEGF = 1) and (ANGLE_E < 0.0) then
      ANGLE_E := ANGLE_E + 360.0;
   end if;
   if NINF = 1 then
      ANGLE_E := ANGLE_E + abs (MIN_ANGLE_E);
   end if;
   SUM_ANGLE_E  := SUM_ANGLE_E  + ANGLE_E;
   SUM_ANGLE_E2 := SUM_ANGLE_E2 + ANGLE_E * ANGLE_E;
   SUM_DECIBEL  := SUM_DECIBEL  + DECIBEL;
   SUM_DECIBEL2 := SUM_DECIBEL2 + DECIBEL * DECIBEL;
end loop;
STANDARD_DVM_MAGNITUDE := (SUM_MAGNITUDE2 - SUM_MAGNITUDE *
   SUM_MAGNITUDE / NUM_SIMULATIONS2) / (NUM_SIMULATIONS2 - 1.0);
if STANDARD_DVM_MAGNITUDE < 0.0 then
   Errors.Write (MESSAGE => RES_ERROR_1);
else
   STANDARD_DVM_MAGNITUDE := Math_Lib.Sqrt (STANDARD_DVM_MAGNITUDE);
end if;
STANDARD_DVM_ANGLE_E := (SUM_ANGLE_E2 - SUM_ANGLE_E * SUM_ANGLE_E
   / NUM_SIMULATIONS2) / (NUM_SIMULATIONS2 - 1.0);
if STANDARD_DVM_ANGLE_E < 0.0 then
   Errors.Write (MESSAGE => RES_ERROR_1);
else
   STANDARD_DVM_ANGLE_E := Math_Lib.Sqrt (STANDARD_DVM_ANGLE_E);
end if;
STANDARD_DVM_DECIBEL := (SUM_DECIBEL2 - SUM_DECIBEL * SUM_DECIBEL
   / NUM_SIMULATIONS2) / (NUM_SIMULATIONS2 - 1.0);
if STANDARD_DVM_DECIBEL < 0.0 then
   Errors.Write (MESSAGE => RES_ERROR_1);
else
   STANDARD_DVM_DECIBEL := Math_Lib.Sqrt (STANDARD_DVM_DECIBEL);
end if;
MEAN_DECIBEL := Math_Lib.Ln (SUM_MAGNITUDE / NUM_SIMULATIONS2) * LN_20;
```

```
   Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   loop
      for COMPONENT in CAPS..COUPLING loop
         begin
            Inner.Read_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE2,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => MONT.NETLIST2);
            if abs (VALUE2) > MINIMUM_VALUE then
               Inner.Write_Netlist (COMPONENT => COMPONENT,
                  VALUE          => VALUE2,
                  HIGH_NODE1     => HIGH_NODE1,
                  LOW_NODE1      => LOW_NODE1,
                  HIGH_NODE2     => HIGH_NODE2,
                  LOW_NODE2      => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST1);
            end if;
         exception
            when BRANCH_OUT_OF_RANGE => null;
         end;
      end loop;
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
   Central_Solver.Solve_AC_Network (MODE => RLC,
      AC_FREQUENCY    => FREQUENCY_MONT1,
      CALCULATE       => TRUE,
      NETLIST_OBJECT => NETLIST1);
   Central_Solver.Get_Voltages (LOW_NODE => LOW_NODE_1,
      HIGH_NODE => HIGH_NODE_1,
      REAL_VOLT => REAL_VOLTS2,
      IMAG_VOLT => IMAG_VOLTS2);
   ANGLE_E := Maths.Angle (X_COORDINATE => REAL_VOLTS2,
      Y_COORDINATE => IMAG_VOLTS2);
   NOMINAL_MAGNITUDE := Maths.Find_Magnitude (REAL_VALUE => REAL_VOLTS2,
      IMAG_VALUE => IMAG_VOLTS2);
   DECIBEL := Math_Lib.Ln (NOMINAL_MAGNITUDE) * LN_20;
   if NEGP = 1 then
      MEAN_ANGLE_E := SUM_ANGLE_E / NUM_SIMULATIONS2 - 360.0;
   end if;
   if MINP = 1 then
      MEAN_ANGLE_E := SUM_ANGLE_E / NUM_SIMULATIONS2 - abs (MIN_ANGLE_E);
   end if;
   if (NEGP = 0) and (MINP = 0) then
      MEAN_ANGLE_E := SUM_ANGLE_E / NUM_SIMULATIONS2;
   end if;
   MAX_DECIBEL := Math_Lib.Ln (MAX_MAGNITUDE) * LN_20;
   MIN_DECIBEL := Math_Lib.Ln (MIN_MAGNITUDE) * LN_20;
   SCREEN_ROW := 17;
   Cursor_Next_Line;
   Cursor_Set (ROW => SCREEN_ROW, COLUMN => 15);
   Write (ITEM => "          Volts       Decibel      Degrees     ",
      FORM => UNDERLINED);
   Cursor_Next_Line;
   Cursor_Set (ROW => 18, COLUMN => 17);
   Write (ITEM => "Nominal");
   Write (ITEM => NOMINAL_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 19, COLUMN => 17);
   Write (ITEM => "Mean");
   Write (ITEM => SUM_MAGNITUDE / NUM_SIMULATIONS2,
      WHOLE          => 7,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MEAN_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MEAN_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 20, COLUMN => 17);
   Write (ITEM => "Maximum");
   Write (ITEM => MAX_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MAX_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MAX_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 21, COLUMN => 17);
   Write (ITEM => "Minimum");
   Write (ITEM => MIN_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MIN_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => MIN_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 22, COLUMN => 17);
   Write (ITEM => "Std Dvn");
   Write (ITEM => STANDARD_DVN_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => STANDARD_DVN_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => STANDARD_DVN_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 23, COLUMN => 17);
   Write (ITEM => "Variance");
   Write (ITEM => STANDARD_DVN_MAGNITUDE * STANDARD_DVN_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => 2.0 * STANDARD_DVN_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => STANDARD_DVN_ANGLE_E * STANDARD_DVN_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Cursor_Next_Line;
   Cursor_Set (ROW => 24, COLUMN => 17);
   Write (ITEM => "2 Sigma");
   Write (ITEM => 3.0 * STANDARD_DVN_MAGNITUDE,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => 3.0 * STANDARD_DVN_DECIBEL,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   Write (ITEM => 3.0 * STANDARD_DVN_ANGLE_E,
      WHOLE          => 4,
      FRACTION       => 5,
      ENG_NOTATION => TRUE);
   if PERFORM_LIST (2).STATE then
      Prompts.Read (LIST => MONTE_LIST1);
      Prompts.Read (MESSAGE => "Do you accept the default line spacing ?",
         ITEM => RESPONSE);
      if RESPONSE then
         INCR := REAL_TYPE (MX_PIXELS) / NUM_SIMULATIONS2;
      else
         Prompts.Read (MESSAGE => "Enter Spacing.", ITEM => INCR);
         INCR := REAL_TYPE (INCR);
      end if;

      Change_Screen (MODE_OF_SCREEN => GRAPHIC);
      Draw_X_Axis;
      Draw_Y_Axis;

      for COUNTER in 1..3 loop
         if MONTE_LIST1 (COUNTER).STATE then
            case COUNTER is
               when 1 =>
                  String_To_Graphic (MESSAGE => "|DB|",
                     ROW     => 1,
                     COLUMN  => 2,
                     COLOURS => WHITE);
                  if (MAX_DECIBEL < 0.0) or (MIN_DECIBEL < 0.0) then
                     MIN := abs (MAX_DECIBEL);
                     MAX := abs (MIN_DECIBEL);
                  else
                     MIN := abs (MIN_DECIBEL);
                     MAX := abs (MAX_DECIBEL);
                  end if;
                  MEAN := abs (MEAN_DECIBEL);
               when 2 =>
                  String_To_Graphic (MESSAGE => "|Deg|",
                     ROW     => 1,
                     COLUMN  => 2,
                     COLOURS => WHITE);
                  if (MAX_ANGLE_E < 0.0) or (MIN_ANGLE_E < 0.0) then
                     MIN := abs (MAX_ANGLE_E);
                     MAX := abs (MIN_ANGLE_E);
                  else
                     MIN := abs (MIN_ANGLE_E);
                     MAX := abs (MAX_ANGLE_E);
                  end if;
                  MEAN := abs (MEAN_ANGLE_E);
               when 3 =>
                  String_To_Graphic (MESSAGE => "|Volt|",
                     ROW     => 1,
                     COLUMN  => 2,
                     COLOURS => WHITE);
                  MIN := MIN_MAGNITUDE;
                  MAX := MAX_MAGNITUDE;
                  MEAN := SUM_MAGNITUDE / NUM_SIMULATIONS2;
            end case;
            OPTION := COUNTER;
            exit;
         end if;
      end loop;

      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => "MONTE-CARLO Bar Plot for node pair",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => HIGH_NODE_1,
         ITEM_WIDTH => 3,
         TEXT_LINE  => DISPLAY_LINE);
      Texts.Combine (ITEM => "-",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => LOW_NODE_1,
         ITEM_WIDTH => 3,
         TEXT_LINE  => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => 1,
         COLUMN  => 20,
         COLOURS => WHITE);
```

```
      Label_Linear_X (X_MINIMUM => 1.0,
         X_MAXIMUM => NUM_SIMULATIONS2);
      Fix_Scale (Y_MINIMUM => MIN,
         Y_MAXIMUM => MAX);
      Label_Y (Y_MINIMUM => MIN,
         Y_MAXIMUM => MAX);
      if MAX /= MIN then
         WORK := REAL_TYPE (MY_PIXELS) / (MAX - MIN);
      end if;
      Set_Plot_Colour (COLOUR => WHITE);
      for COUNTER in 1..NUM_SIMULATIONS loop
         case OPTION is
            when 1 =>
               VALUE3 := Maths.Find_Magnitude (
                  REAL_VALUE => MONT.REAL_VOLTS (COUNTER),
                  IMAG_VALUE => MONT.IMAG_VOLTS (COUNTER));
               if VALUE3 > MINIMUM_VALUE then
                  VALUE3 := abs (Math_Lib.Ln (VALUE3) * LN_20) - MIN;
               else
                  VALUE3 := 0.0;
               end if;
            when 2 =>
               VALUE3 := abs (Maths.Angle (
                  X_COORDINATE => MONT.REAL_VOLTS (COUNTER),
                  Y_COORDINATE => MONT.IMAG_VOLTS (COUNTER)) - MIN;
            when 3 =>
               VALUE3 := Maths.Find_Magnitude (
                  REAL_VALUE => MONT.REAL_VOLTS (COUNTER),
                  IMAG_VALUE => MONT.IMAG_VOLTS (COUNTER))-MIN_MAGNITUDE;
            when others => null;
         end case;
         VALUE3 := VALUE3 * WORK;
         INCS := INTEGER_TYPE (REAL_TYPE (COUNTER) * INCR);
         WIDTH := INCS / COUNTER;
         Y := INTEGER_TYPE (VALUE3);
         X := INCS + Y_MIN + X_AXIS - WIDTH;
         if INCR > 10.0 then
            Bar_3D (X1 => X,
               Y1    => Y_MIN - Y_AXIS,
               X2    => X + WIDTH,
               Y2    => Y_MIN - Y_AXIS - Y,
               DEPTH => WIDTH / 4);
         else
            Line (X1 => X,
               Y1 => Y_MIN - Y_AXIS,
               X2 => X,
               Y2 => Y_MIN - Y_AXIS - Y);
         end if;
      end loop;
      Continue_Processing;
      Change_Screen (MODE_OF_SCREEN => TEXT);
   end if;
end Perform;

-------------------------------------------------------------------------
-- PURPOSE : Saves monte netlist data used in the monte simulations.
-------------------------------------------------------------------------

procedure Save is

begin
   Outer.Save_Monte_Netlist (NETLIST_OBJECT1 => MONT.NETLIST2,
      NETLIST_OBJECT2 => MONT.NETLIST3);
end Save;

-------------------------------------------------------------------------
-- PURPOSE : Shows the components, their tolerances and types of
--           distributions that are selected for the Monte - Carlo
--           simulation.
-------------------------------------------------------------------------

procedure Show (NETLIST1 : in out NETLIST_TYPE) is

   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2          : NODE_TYPE;
   PHASE,               VALUE              : REAL_TYPE;
   SCREEN_ROW                              : ROW_TYPE     := 3;

begin
   Process_Title (TITLE => "MONTE COMPONENTS AND TOLERANCES");
   Write (ITEM => MES_MONTE_1,
      FORM => UNDERLINED);
   Setup_Monte (NETLIST1 => NETLIST1);
   Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST3);

   loop
      for COMPONENT in CAPS..TUPLING loop
         begin
            Inner.Read_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => MONT.NETLIST2);
            if abs (VALUE) > MINIMUM_VALUE then
               if (COMPONENT = GR_ADMIT) and (VALUE /= 0.0) then
                  VALUE := 1.0 / VALUE;
               end if;
               Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
                  HIGH_NODE1 => HIGH_NODE1,
                  LOW_NODE1  => LOW_NODE1,
                  HIGH_NODE2 => HIGH_NODE2,
                  LOW_NODE2  => LOW_NODE2,
                  VALUE      => VALUE,
                  PHASE      => PHASE,
                  ROW        => SCREEN_ROW);
               Inner.Read_Netlist (COMPONENT => COMPONENT,
                  VALUE          => VALUE,
                  HIGH_NODE1     => HIGH_NODE1,
                  LOW_NODE1      => LOW_NODE1,
                  HIGH_NODE2     => HIGH_NODE2,
                  LOW_NODE2      => LOW_NODE2,
                  NETLIST_OBJECT => MONT.NETLIST3);
               Write (ITEM => abs (VALUE) * 100.0,
                  WHOLE    => 7,
                  FRACTION => 5);
               Cursor_Set (ROW => SCREEN_ROW - 1, COLUMN => 67);
               if VALUE > 0.0 then
                  Write (ITEM => "Gaussian");
               else
                  Write (ITEM => "Uniform");
               end if;
            end if;
         exception
            when BRANCH_OUT_OF_RANGE => null;
         end;
      end loop;
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST3);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
end Show;

-------------------------------------------------------------------------
-- PURPOSE : Test the random number generator used in the Monte analysis.
-------------------------------------------------------------------------

procedure Test_Random is

   type FREQUENCY_ARRAY_TYPE is array (0..9) of INTEGER_TYPE;

   FREQUENCY : FREQUENCY_ARRAY_TYPE := (0..9 => 0);

   CHI,                    CONF,
   MEAN                                : REAL_TYPE := 0.0;
   EXPECT,                 N,
   NUM_SIMULATIONS2,       REAL_MINIMUM,
   REAL_MAXIMUM,           STANDARD_DVN,
   VALUE1,                 VALUE2      : REAL_TYPE;
   COUNTER,                MAXIMUM,
   MINIMUM,                NORMAL,
   NUM_SIMULATIONS,        SEED1,
   X,                      Y           : INTEGER_TYPE;
   INCR                                : INTEGER_TYPE := MY_PIXELS/10;
   DISPLAY_LINE                        : Texts.TEXT_TYPE;

begin
   Process_Title (TITLE => "TEST RANDOM");
   Y_MIN := MAX_Y_COORD - 3 * Y_AXIS / 2;
   Prompts.Read (LIST => MONTE_LIST2,
      HELP_INFO => HP_TEST_RANDOM);
   if MONTE_LIST2 (1).STATE then
      TEST_RANDOM_FORM (3).ACCEPTING := TRUE;
      VALUE1 := 4.5;
   else
      TEST_RANDOM_FORM (3).ACCEPTING := FALSE;
   end if;
   Prompts.Read (FORM => TEST_RANDOM_FORM);
   NUM_SIMULATIONS  := TEST_RANDOM_FORM (1).INTEGER_VALUE;
   SEED1            := TEST_RANDOM_FORM (2).INTEGER_VALUE;
   STANDARD_DVN     := TEST_RANDOM_FORM (3).REAL_VALUE;
   NUM_SIMULATIONS2 := REAL_TYPE (NUM_SIMULATIONS);
   SEED := 2 * abs (LONG_INTEGER_TYPE (SEED1)) + 16_183) + 1;
   EXPECT := 0.1 * NUM_SIMULATIONS2;
   N := Math_Lib.Sqrt (2.0 * NUM_SIMULATIONS2 - 1.0);

   for COUNTER in 1..NUM_SIMULATIONS loop
      if MONTE_LIST2 (2).STATE then
         Y := INTEGER_TYPE (Random * 10.0);
      else
         loop
            Gauss (STANDARD_DEVIATION => STANDARD_DVN,
               VALUE1 => VALUE1,
               VALUE2 => VALUE2);
            exit when (VALUE1 * VALUE2 > 0.0);
         end loop;
         Y := INTEGER_TYPE (VALUE2);
      end if;
      if Y in 0..9 then
         FREQUENCY (Y) := FREQUENCY (Y) + 1;
      end if;
      MEAN := MEAN + REAL_TYPE (Y);
   end loop;

   MAXIMUM := FREQUENCY (0);
   MINIMUM := MAXIMUM;
   for COUNTER in 0..9 loop
      if FREQUENCY (COUNTER) > MAXIMUM then
         MAXIMUM := FREQUENCY (COUNTER);
      elsif FREQUENCY (COUNTER) < MINIMUM then
         MINIMUM := FREQUENCY (COUNTER);
      end if;
      CHI := CHI + ((REAL_TYPE (FREQUENCY (COUNTER)) - EXPECT) ** 2)
         / EXPECT;
   end loop;
   if CHI < 2.558 then
      CONF := 0.99;
   elsif CHI < 3.940 then
      CONF := 0.95;
   elsif CHI < 6.737 then
      CONF := 0.75;
   elsif CHI < 9.342 then
      CONF := 0.5;
   elsif CHI < 12.55 then
      CONF := 0.25;
   elsif CHI < 16.81 then
      CONF := 0.05;
   end if;

   Change_Screen (MODE_OF_SCREEN => GRAPHIC);
   Draw_X_Axis;
   Draw_Y_Axis;
   Label_Linear_X (X_MINIMUM => 1.0,
      X_MAXIMUM => 10.0);
   MEAN := MEAN / NUM_SIMULATIONS2;
   Texts.Combine (ITEM => "Mean =",
      TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => MEAN,
      FRACTION  => 3,
      TEXT_LINE => DISPLAY_LINE);
   String_To_Graphic (
      MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
      ROW     => 25,
```

```
      COLUMN  => 17,
      COLOURS => WHITE);

   if MONTE_LIST1 (1).STATE then
      Texts.Reset (TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => "Chi = ",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => CHI,
         FRACTION  => 3,
         EXPONENT  => 0,
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => "    Conf =",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => CONF,
         FRACTION  => 3,
         EXPONENT  => 0,
         TEXT_LINE => DISPLAY_LINE);
      String_To_Graphic (
         MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
         ROW     => 23,
         COLUMN  => 36,
         COLOURS => WHITE);
   end if;
   REAL_MINIMUM := REAL_TYPE (MINIMUM);
   REAL_MAXIMUM := REAL_TYPE (MAXIMUM);

   Pix_Scale (Y_MINIMUM => REAL_MINIMUM,
      Y_MAXIMUM => REAL_MAXIMUM);
   Label_Y (Y_MINIMUM => REAL_MINIMUM,
      Y_MAXIMUM => REAL_MAXIMUM);
   NORMAL := NY_PIXELS / INTEGER_TYPE (REAL_MAXIMUM - REAL_MINIMUM);

   for COUNTER in 0..9 loop
      X := COUNTER * INCR + X_MIN + X_AXIS;
      Y := (FREQUENCY (COUNTER) - MINIMUM) * NORMAL;
      Bar_3D (X1 =>
         Y1     => Y_MAX - Y_AXIS,
         X2     => X + INCR,
         Y2     => Y_MIN - Y_AXIS - Y,
         DEPTH  => (INCR + 1) / 4);
   end loop;
   Continue_Processing;
   Change_Screen (MODE_OF_SCREEN => TEXT);
end Test_Random;

-----------------------------------------------------------------------
-- PURPOSE : Determines the worst case monte simulation.
-----------------------------------------------------------------------

procedure Worst (SINGLE : in BOOLEAN;
   MODE_MAX : in MODE_TYPE;
   NETLIST1 : in out NETLIST_TYPE) is

   type Y_ARRAY_TYPE is array (1..2) of INTEGER_TYPE;

   Y_NEW,                  Y_PREV                : Y_ARRAY_TYPE;
   HIGH_NODE_1,            LOW_NODE_1,
   HIGH_NODE1,             LOW_NODE1,
   HIGH_NODE2,             LOW_NODE2             : NODE_TYPE;
   COMPONENT                                     : ELEMENTS;
   DISPLAY_LINE                                  : Texts.TEXT_TYPE;
   COUNTER1,               COUNTER2,
   NUM_SIMULATIONS,        OPTION,
   SEED1,                  TWO_COUNT_WORST,
   X_INC,                  Y,                    : INTEGER_TYPE;
   ANGLE_E,                FREQUENCY_INC,
   FREQUENCY_MONTE,        MAGNITUDE,
   MAX_ANGLE_E,            MAX_DECIBEL,
   MAX_FREQUENCY,          MAX_MAGNITUDE,
   MAX_VALUE,              MIN_ANGLE_E,
   MIN_FREQUENCY,          MIN_MAGNITUDE,
   MIN_DECIBEL,            MIN_VALUE,
   STANDARD_DVN,           TEMP_VALUE,
   VALUE,                  VALUE1,
   VALUE2,                 VALUE3,
   VALUE4,                 Y_MAX_POINT,
   Y_MIN_POINT,            Y_SCALE               : REAL_TYPE;

begin
   Process_Title (TITLE => "WORST MONTE");
   Setup_Monte (NETLIST1 => NETLIST1);
   MONTE_FORM1 (5).ACCEPTING := FALSE;
   MONTE_FORM1 (6).ACCEPTING := TRUE;
   if SINGLE then
      MONTE_FORM1 (3).ACCEPTING := TRUE;
      MONTE_FORM1 (4).ACCEPTING := TRUE;
      MONTE_FORM1 (6).ROW := 8;
   else
      MONTE_FORM1 (3).ACCEPTING := FALSE;
      MONTE_FORM1 (4).ACCEPTING := FALSE;
      MONTE_FORM1 (6).ROW := 6;
   end if;

   Prompts.Read (FORM => MONTE_FORM1,
      LIST    => MONTE_LIST1,
      EXCLUSIVE => TRUE,
      HELP_INFO => HP_WORST);
   HIGH_NODE_1     := MONTE_FORM1 (1).INTEGER_VALUE;
   LOW_NODE_1      := MONTE_FORM1 (2).INTEGER_VALUE;
   NUM_SIMULATIONS := MONTE_FORM1 (3).INTEGER_VALUE;
   SEED1           := MONTE_FORM1 (4).INTEGER_VALUE;
   X_INC           := MONTE_FORM1 (6).INTEGER_VALUE;
   SEED := 2 * (LONG_INTEGER_TYPE (abs (SEED1)) + 16_183) + 1;
   Get_X_Frequency (MIN_FREQUENCY => MIN_FREQUENCY,
      MAX_FREQUENCY => MAX_FREQUENCY);

   for COUNTER1 in 1..3 loop
      if MONTE_LIST1 (COUNTER1).STATE then
         OPTION := COUNTER1;
         exit;
      end if;
   end loop;
   FREQUENCY_MONTE := MIN_FREQUENCY;
   FREQUENCY_INC := (Math_Lib.Ln (MAX_FREQUENCY) -
      Math_Lib.Ln (MIN_FREQUENCY)) / REAL_TYPE (NX_PIXELS);
   COUNTER1 := 0;
   Cursor_Set (ROW => 9, COLUMN => 25);
   Write (ITEM => "Minimum Frequency : ");
   Write (ITEM => MIN_FREQUENCY,
      FRACTION => 5);
   Cursor_Next_Line;
   Cursor_Set (ROW => 9, COLUMN => 25);
   Write (ITEM => "Maximum Frequency : ");
   Write (ITEM => MAX_FREQUENCY,
      FRACTION => 5);
   Cursor_Next_Line;
   Cursor_Set (ROW => 11, COLUMN => 25);
   Write (ITEM => "Present Frequency (");
   loop
      Cursor_Set (ROW => 11, COLUMN => 45);
      Write (ITEM => FREQUENCY_MONTE,
         FRACTION => 5);
      Cursor_Next_Line;
      COUNTER2 := 0;
      loop
         if SINGLE then
            COUNTER2 := COUNTER2 + 1;
         else
            TWO_COUNT_WORST := 1;
         end if;
         Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
         Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST3);
         loop
            if not Inner.Empty_Branch (NETLIST_OBJECT=>MONT.NETLIST2) then
               for COMPONENT in CAPS..COUPLING loop
                  begin
                     Inner.Read_Netlist (COMPONENT => COMPONENT,
                        VALUE          => VALUE3,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        NETLIST_OBJECT => MONT.NETLIST2);
                     if abs (VALUE3) > MINIMUM_VALUE then
                        Inner.Read_Netlist (COMPONENT => COMPONENT,
                           VALUE          => STANDARD_DVN,
                           HIGH_NODE1     => HIGH_NODE1,
                           LOW_NODE1      => LOW_NODE1,
                           HIGH_NODE2     => HIGH_NODE2,
                           LOW_NODE2      => LOW_NODE2,
                           NETLIST_OBJECT => MONT.NETLIST3);
                        if STANDARD_DVN < 0.0 then
                           if SINGLE then
                              if NUM_SIMULATIONS > 2 then
                                 loop
                                    TEMP_VALUE := Random;
                                    VALUE := VALUE3 + 2.0 *
                                       abs (STANDARD_DVN * VALUE3) *
                                       (TEMP_VALUE - 0.5);
                                    exit when VALUE3 * VALUE > 0.0;
                                 end loop;
                                 VALUE3 := VALUE;
                              elsif COUNTER2 = 1 then
                                 VALUE3 := VALUE3 +
                                    abs (VALUE3 * STANDARD_DVN);
                              else
                                 VALUE3 := VALUE3 -
                                    abs (VALUE3 * STANDARD_DVN);
                              end if;
                           else
                              if (COUNTER2 and TWO_COUNT_WORST) =
                                 TWO_COUNT_WORST then
                                 VALUE3 := VALUE3 -
                                    abs (VALUE3 * STANDARD_DVN);
                              else
                                 VALUE3 := VALUE3 +
                                    abs (VALUE3 * STANDARD_DVN);
                              end if;
                           end if;
                        else
                           if SINGLE then
                              if NUM_SIMULATIONS > 2 then
                                 loop
                                    Gauss (
                                       STANDARD_DEVIATION => STANDARD_DVN,
                                       VALUE1             => VALUE3,
                                       VALUE2             => VALUE4);
                                    exit when VALUE3 * VALUE4 > 0.0;
                                 end loop;
                                 VALUE3 := VALUE4;
                              elsif COUNTER2 = 1 then
                                 VALUE3 := VALUE3 +
                                    3.0 * VALUE3 * STANDARD_DVN;
                              else
                                 VALUE3 := VALUE3 -
                                    3.0 * VALUE3 * STANDARD_DVN;
                              end if;
                           else
                              if (COUNTER2 and TWO_COUNT_WORST) =
                                 TWO_COUNT_WORST then
                                 VALUE3 := VALUE3 -
                                    3.0 * VALUE3 * STANDARD_DVN;
                              else
                                 VALUE3 := VALUE3 +
                                    3.0 * VALUE3 * STANDARD_DVN;
                              end if;
                           end if;
                        end if;
                        if not SINGLE then
                           TWO_COUNT_WORST := TWO_COUNT_WORST + 2;
                        end if;
                        Inner.Write_Netlist (COMPONENT => COMPONENT,
                           VALUE          => VALUE3,
                           HIGH_NODE1     => HIGH_NODE1,
                           LOW_NODE1      => LOW_NODE1,
                           HIGH_NODE2     => HIGH_NODE2,
                           LOW_NODE2      => LOW_NODE2,
                           NETLIST_OBJECT => NETLIST1);
                     end if;
                  exception
                     when BRANCH_OUT_OF_RANGE => null;
                  end;
               end loop;
            end if;
            begin
               Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
               Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
               Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST3);
            exception
```

```
            when NO_MORE_BRANCHES => exit;
         end;
      end loop;
      Control_Solver.Solve_AC_Network (MODE => RLC,
         AC_FREQUENCY   => FREQUENCY_MONTE,
         CALCULATE      => TRUE,
         NETLIST_OBJECT => NETLIST1);
      if not SINGLE then
         COUNTER1 := COUNTER2 + 1;
      end if;
      Control_Solver.Get_Voltages (LOW_NODE => LOW_NODE_1,
         HIGH_NODE => HIGH_NODE_1,
         REAL_VOLT => MONT.REAL_VOLTS (COUNTER2),
         IMAG_VOLT => MONT.IMAG_VOLTS (COUNTER2));
      if SINGLE then
         exit when (COUNTER2 >= NUM_SIMULATIONS) or Halt_Processing;
      else
         exit when (COUNTER2 >= TWO_COUNT_WORST) or Halt_Processing;
      end if;
   end loop;
   if not SINGLE then
      NUM_SIMULATIONS := TWO_COUNT_WORST;
   end if;
   MAX_ANGLE_E := Maths.Angle (X_COORDINATE => MONT.REAL_VOLTS (1),
      Y_COORDINATE => MONT.IMAG_VOLTS (1));
   MIN_ANGLE_E := MAX_ANGLE_E;
   MAX_MAGNITUDE := Maths.Find_Magnitude (
      REAL_VALUE => MONT.REAL_VOLTS (1),
      IMAG_VALUE => MONT.IMAG_VOLTS (1));
   MIN_MAGNITUDE := MAX_MAGNITUDE;
   MIN_DECIBEL := Math_Lib.Ln (MIN_MAGNITUDE) * LN_20;
   MAX_DECIBEL := MIN_DECIBEL;
   for COUNTER2 in 2..NUM_SIMULATIONS loop
      ANGLE_E := Maths.Angle (
         X_COORDINATE => MONT.REAL_VOLTS (COUNTER2),
         Y_COORDINATE => MONT.IMAG_VOLTS (COUNTER2));
      MAGNITUDE := Maths.Find_Magnitude (
         REAL_VALUE => MONT.REAL_VOLTS (COUNTER2),
         IMAG_VALUE => MONT.IMAG_VOLTS (COUNTER2));
      if ANGLE_E > MAX_ANGLE_E then
         MAX_ANGLE_E := ANGLE_E;
      elsif ANGLE_E < MIN_ANGLE_E then
         MIN_ANGLE_E := ANGLE_E;
      end if;
      if MAGNITUDE > MAX_MAGNITUDE then
         MAX_MAGNITUDE := MAGNITUDE;
      elsif MAGNITUDE < MIN_MAGNITUDE then
         MIN_MAGNITUDE := MAGNITUDE;
      end if;
   end loop;
   MAX_DECIBEL := Math_Lib.Ln (MAX_MAGNITUDE) * LN_20;
   MIN_DECIBEL := Math_Lib.Ln (MIN_MAGNITUDE) * LN_20;
   case OPTION is
      when 1 =>
         VALUE1 := MIN_DECIBEL;
         VALUE2 := MAX_DECIBEL;
      when 2 =>
         VALUE1 := MIN_ANGLE_E;
         VALUE2 := MAX_ANGLE_E;
      when 3 =>
         VALUE1 := MIN_MAGNITUDE;
         VALUE2 := MAX_MAGNITUDE;
      when others => null;
   end case;
   MONT.Y_VALUES (COUNTER1, 1) := VALUE1;
   MONT.Y_VALUES (COUNTER1, 2) := VALUE2;
   if COUNTER1 = 0 then
      MAX_VALUE := VALUE1;
      MIN_VALUE := MAX_VALUE;
   end if;
   if VALUE1 > MAX_VALUE then
      MAX_VALUE := VALUE1;
   elsif VALUE1 < MIN_VALUE then
      MIN_VALUE := VALUE1;
   end if;
   if VALUE2 > MAX_VALUE then
      MAX_VALUE := VALUE2;
   elsif VALUE2 < MIN_VALUE then
      MIN_VALUE := VALUE2;
   end if;
   COUNTER1 := COUNTER1 + X_INC;
   FREQUENCY_MONTE := MIN_FREQUENCY *
      Math_Lib.Exp (FREQUENCY_INC * REAL_TYPE (COUNTER1));
   Y_PREV := Y_NEW;
   exit when FREQUENCY_MONTE > MAX_FREQUENCY;
end loop;
Change_Screen (MODE_OF_SCREEN => GRAPHIC);
Draw_X_Axis;
Draw_Y_Axis;
case OPTION is
   when 1 =>
      String_To_Graphic (MESSAGE => "DB",
         ROW     => 1,
         COLUMN  => 2,
         COLOURS => WHITE);
   when 2 =>
      String_To_Graphic (MESSAGE => "Deg",
         ROW     => 1,
         COLUMN  => 2,
         COLOURS => WHITE);
   when 3 =>
      String_To_Graphic (MESSAGE => "Volt",
         ROW     => 1,
         COLUMN  => 2,
         COLOURS => WHITE);
   when others => null;
end case;
Texts.Reset (TEXT_LINE => DISPLAY_LINE);
Texts.Combine (ITEM => "MONTE-CARLO Worst Case Plot for node pair",
   TEXT_LINE => DISPLAY_LINE);
Texts.Combine (ITEM => HIGH_NODE_1,
   ITEM_WIDTH => 3,
   TEXT_LINE  => DISPLAY_LINE);
Texts.Combine (ITEM => "-",
   TEXT_LINE => DISPLAY_LINE);
Texts.Combine (ITEM => LOW_NODE_1,
   ITEM_WIDTH => 3,
   TEXT_LINE  => DISPLAY_LINE);
String_To_Graphic (
   MESSAGE => DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX),
   ROW     => 1,
   COLUMN  => 20,
   COLOURS => WHITE);
String_To_Graphic (MESSAGE => "Hz",
   ROW     => (Y_MIN - Y_AXIS) / ROW_FACTOR + 1,
   COLUMN  => (X_MAX - X_AXIS) / COLUMN_FACTOR + 7,
   COLOURS => WHITE);
Label_Log_X (X_MINIMUM => MIN_FREQUENCY,
   X_MAXIMUM => MAX_FREQUENCY);
Y_MIN_POINT := MIN_VALUE;
Y_MAX_POINT := MAX_VALUE;
Fix_Scale (Y_MINIMUM => Y_MIN_POINT,
   Y_MAXIMUM => Y_MAX_POINT);
Y_SCALE := REAL_TYPE (NY_PIXELS) / (Y_MAX_POINT - Y_MIN_POINT);
if abs (Y_SCALE * Y_MAX_POINT) > MAX_INTEGER then
   Y_SCALE := abs (MAX_INTEGER / Y_MAX_POINT);
   Y_MAX_POINT := REAL_TYPE (NY_PIXELS) / Y_SCALE + Y_MIN_POINT;
end if;
Label_Y (Y_MINIMUM => Y_MIN_POINT,
   Y_MAXIMUM => Y_MAX_POINT);
Zero_Line (Y_MINIMUM => Y_MIN_POINT,
   Y_MAXIMUM => Y_MAX_POINT,
   Y_SCALE   => Y_SCALE);
COUNTER1 := 0;
FREQUENCY_MONTE := MIN_FREQUENCY;
loop
   MAGNITUDE := MONT.Y_VALUES (COUNTER1, 1);
   if Y_MIN_POINT < 0.0 then
      Y := Y_MIN - Y_AXIS - INTEGER_TYPE (Y_SCALE * MAGNITUDE);
   else
      Y := Y_MIN - Y_AXIS -
         INTEGER_TYPE (Y_SCALE * (MAGNITUDE - Y_MIN_POINT));
   end if;
   Y_NEW (1) := Y;
   MAGNITUDE := MONT.Y_VALUES (COUNTER1, 2);
   if Y_MIN_POINT < 0.0 then
      Y := Y_MIN - Y_AXIS - INTEGER_TYPE (Y_SCALE * MAGNITUDE);
   else
      Y := Y_MIN - Y_AXIS -
         INTEGER_TYPE (Y_SCALE * (MAGNITUDE - Y_MIN_POINT));
   end if;
   Y_NEW (2) := Y;
   if COUNTER1 > 0 then
      Set_Plot_Colour (COLOUR => LIGHT_RED);
      Line (X1 => X_MIN + X_AXIS + COUNTER1 - X_INC,
         Y1 => Y_PREV (1),
         X2 => X_MIN + X_AXIS + COUNTER1,
         Y2 => Y_NEW (1));
      Set_Plot_Colour (COLOUR => LIGHT_GREEN);
      Line (X1 => X_MIN + X_AXIS + COUNTER1 - X_INC,
         Y1 => Y_PREV (2),
         X2 => X_MIN + X_AXIS + COUNTER1,
         Y2 => Y_NEW (2));
   end if;
   COUNTER1 := COUNTER1 + X_INC;
   FREQUENCY_MONTE := MIN_FREQUENCY *
      Math_Lib.Exp (FREQUENCY_INC * REAL_TYPE (COUNTER1));
   Y_PREV := Y_NEW;
   exit when FREQUENCY_MONTE > MAX_FREQUENCY;
end loop;
Line (X1 => X_MIN + X_AXIS + COUNTER1 - X_INC,
   Y1 => Y_NEW (1),
   X2 => X_MIN + X_AXIS + COUNTER1 - X_INC,
   Y2 => Y_NEW (2));
Y_MIN := MAX_Y_COORD - 3 * Y_AXIS / 2;
Inner.First_Branch (NETLIST_OBJECT => NETLIST1);
Inner.First_Branch (NETLIST_OBJECT => MONT.NETLIST2);
loop
   for COMPONENT in CAPS..COUPLING loop
      begin
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE3,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => MONT.NETLIST2);
         if abs (VALUE3) > MINIMUM_VALUE then
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE3,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST1);
         end if;
      exception
         when BRANCH_OUT_OF_RANGE => null;
      end;
   end loop;
   begin
      Inner.Next_Branch (NETLIST_OBJECT => NETLIST1);
         Inner.Next_Branch (NETLIST_OBJECT => MONT.NETLIST2);
   exception
      when NO_MORE_BRANCHES => exit;
   end;
end loop;
Continue_Processing;
Change_Screen (MODE_OF_SCREEN => TEXT);
clear;
end Worst;
```

```
end Monte;
```

Netlist_Supporter

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Netlist_Supporter is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This provides methods to be used with the Netlist abstract
--           data type.  This package contains methods which are separated
--           from those in the Netlists package because they do not need
--           to know the implementation details of the NETLIST_TYPE.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Change_Magnitude_Phase (NETLIST_OBJECT :in out NETLIST_TYPE);

   procedure Insert_Component (NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure List_Netlist (ALL_COMPONENTS : in BOOLEAN;
      NETLIST_OBJECT : in out NETLIST_TYPE);

end Netlist_Supporter;
```

```
with  Math_Lib,
      Screens;

use   Screens;

package body Netlist_Supporter is

   use Screens.Main;

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_CHANGE_MGN_PHASE   : constant := 130;
   HP_CHANGE_MGN_PHASE2  : constant := 131;
   HP_INSERT_COMPONENT   : constant := 132;
   HP_LIST_NETLIST       : constant := 133;

   INSERT_FORM : FIELDS_TYPE (1..7) :=
      (1 =>
         (VALUE_LENGTH   => 15,
         TEXT_LENGTH     => 17,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ROW             => 4,
         ACTUAL_LENGTH   => 4,
         VALUE           => "10.0                ",
         TEXT            => "Value (0.0 = del)",
         MAX_REAL        => 1.0E40,
         MIN_REAL        => 0.0,
         REAL_VALUE      => 10.0),
      2 =>
         (VALUE_LENGTH   => 15,
         TEXT_LENGTH     => 19,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ACTUAL_LENGTH   => 5,
         ROW             => 5,
         VALUE           => "1.0e3               ",
         TEXT            => "Impedance Frequency",
         MAX_REAL        => 1.0E25,
         MIN_REAL        => MINIMUM_VALUE,
         REAL_VALUE      => 1000.0),
      3 =>
         (VALUE_LENGTH   => 2,
         TEXT_LENGTH     => 13,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 4,
         ACTUAL_LENGTH   => 1,
         ROW             => 6,
         VALUE           => "1 ",
         TEXT            => "Higher Node 1",
         MAX_INTEGER     => MAX_NODES,
         MIN_INTEGER     => 1,
         INTEGER_VALUE   => 1),
      4 =>
         (VALUE_LENGTH   => 2,
         TEXT_LENGTH     => 13,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 3,
         ACTUAL_LENGTH   => 1,
         ROW             => 7,
         VALUE           => "0 ",
         TEXT            => "Lower Node 1",
         MAX_INTEGER     => MAX_NODES,
         MIN_INTEGER     => 0,
         INTEGER_VALUE   => 0),
      5 =>
         (VALUE_LENGTH   => 2,
         TEXT_LENGTH     => 13,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => FALSE,
         TEST_AGAINST    => 6,
         ACTUAL_LENGTH   => 1,
         ROW             => 8,
         VALUE           => "1 ",
         TEXT            => "Higher Node 2",
         MAX_INTEGER     => MAX_NODES,
         MIN_INTEGER     => 1,
         INTEGER_VALUE   => 1),
      6 =>
         (VALUE_LENGTH   => 2,
         TEXT_LENGTH     => 13,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING       => FALSE,
         TEST_AGAINST    => 5,
         ACTUAL_LENGTH   => 1,
         ROW             => 9,
         VALUE           => "0 ",
         TEXT            => "Lower Node 2",
         MAX_INTEGER     => MAX_NODES,
         MIN_INTEGER     => 0,
         INTEGER_VALUE   => 0),
      7 =>
         (VALUE_LENGTH   => 15,
         TEXT_LENGTH     => 15,
         IS_INTEGER_TYPE => FALSE,
         ACCEPTING       => TRUE,
         TEST_AGAINST    => 0,
         ACTUAL_LENGTH   => 3,
         ROW             => 8,
         VALUE           => "1.0             ",
         TEXT            => "Phase (degrees)",
         MAX_REAL        => 90.0,
         MIN_REAL        => MINIMUM_VALUE,
         REAL_VALUE      => 0.0));

-- PURPOSE : Calculates a netlists admittance properties and displays
--           appropriate error messages if an error occurs.

procedure Insert_Conversion (FREQUENCY : in REAL_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   Inser.Convert_Admittance_TO_RLC (FREQUENCY => FREQUENCY,
      NETLIST_OBJECT => NETLIST_OBJECT);
exception
   when CANNOT_CONVERT_WITH_MUTUAL_INDUCTANCES =>
      Errors.Write (MESSAGE => "The conversion to a RLC circuit cannot"
         & " be done because there are mutual inductances.");
   when others =>
      Errors.Write (MESSAGE => "There was an Error in the conversion"
         & " to a RLC circuit");
end Insert_Conversion;

-- PURPOSE : Changes the magnitude and phase of a component.

procedure Change_Magnitude_Phase (NETLIST_OBJECT : in out NETLIST_TYPE) is

   NO_EXISTING_COMPONENT : exception;

   COMPONENT                              : ELEMENTS;
   SCREEN_ROW                             : POSITIVE := 9;
   HIGH_NODES1,          LOW_NODES1,
   HIGH_NODE1,           LOW_NODE1,
   HIGH_NODE2,           LOW_NODE2        : NODE_TYPE;
   VALUE,                TEMP_VALUE,
   MAGNITUDE,            PHASES,
   FREQUENCY                              : REAL_TYPE;
   COUNTER,              OPTION           : INTEGER_TYPE := 1;

   LIST                                   : LIST_ARRAY_TYPE (1..6) := (
      (TEXT_LENGTH => 17, TEXT => "Voltage Magnitude",    STATE => TRUE),
      (TEXT_LENGTH => 17, TEXT => "Current Magnitude",    STATE => FALSE),
      (TEXT_LENGTH => 20, TEXT => "Admittance Magnitude", STATE => FALSE),
      (TEXT_LENGTH => 18, TEXT => "Voltage Phase Angle",  STATE => FALSE),
      (TEXT_LENGTH => 19, TEXT => "Current Phase Angle",  STATE => FALSE),
      (TEXT_LENGTH => 22, TEXT => "Admittance Phase Angle",STATE=>FALSE));

begin
   Process_Title (TITLE => "CHANGED COMPONENT");
   Prompts.Read (FORM => INSERT_FORM (3..4),
      LIST      => LIST,
      EXCLUSIVE => TRUE,
      HELP_INFO => HP_CHANGE_MGN_PHASE);
   HIGH_NODE1 := INSERT_FORM (3).INTEGER_VALUE;
   LOW_NODE1  := INSERT_FORM (4).INTEGER_VALUE;
   for COUNTER in 1..6 loop
      if LIST (COUNTER).STATE then
         case COUNTER is
            when 1 | 4 => COMPONENT := REAL_VOLT;
            when 2 | 5 => COMPONENT := REAL_CURT;
            when 3 | 6 => COMPONENT := GR_ADMIT;
         end case;
```

```
            OPTION := COUNTER;
            exit;
         end if;
      end loop;
      Inner.Find_Branch (COMPONENT => MUT_INDUCT,
         HIGH_NODE      => HIGH_NODES1,
         LOW_NODE       => LOW_NODES1,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Inner.Read_Netlist (COMPONENT => COMPONENT,
         VALUE          => TEMP_VALUE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      COMPONENT := ELEMENTS'succ (COMPONENT);
      Inner.Read_Netlist (COMPONENT => COMPONENT,
         VALUE          => VALUE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      if (abs (TEMP_VALUE) + abs (VALUE)) < MINIMUM_VALUE then
         raise NO_EXISTING_COMPONENT;
      end if;

      case OPTION is
         when 1..3 =>
            PHASES := Maths.Angle (X_COORDINATE => TEMP_VALUE,
               Y_COORDINATE => VALUE);
            Prompts.Read (MESSAGE => "Enter a new Magnitude, which must be"
               & " greater than  1.0E-30",
               HELP_INFO => HP_CHANGE_MGN_PHASE2,
               ITEM      => MAGNITUDE);
         when 4..6 =>
            MAGNITUDE := Math_Lib.Sqrt (TEMP_VALUE * TEMP_VALUE
               + VALUE * VALUE);
            Prompts.Read (MESSAGE => "Enter a new Phase, which must be"
               & " greater than  1.0E-30",
               HELP_INFO => HP_CHANGE_MGN_PHASE2,
               ITEM      => PHASES);
         when others => null;
      end case;
      if PHASES = 90.0 then
         VALUE := MAGNITUDE;
      elsif PHASES = -90.0 then
         VALUE := -MAGNITUDE;
      elsif (PHASES = 180.0) or (PHASES = -180.0) or
         (PHASES = 0.0) then
         VALUE := 0.0;
      else
         VALUE := MAGNITUDE * Math_Lib.Sin (PHASES / RADIAN);
      end if;
      Inner.Write_Netlist (COMPONENT => COMPONENT,
         VALUE          => VALUE,
         HIGH_NODE1     => HIGH_NODES1,
         LOW_NODE1      => LOW_NODES1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      if (PHASES = 90.0) or (PHASES = -90.0) then
         VALUE := 0.0;
      elsif (PHASES = 180.0) or (PHASES = -180.0) then
         VALUE := -MAGNITUDE;
      elsif PHASES = 0.0 then
         VALUE := MAGNITUDE;
      else
         VALUE := MAGNITUDE * Math_Lib.Cos (PHASES / RADIAN);
      end if;
      Inner.Write_Netlist (COMPONENT => ELEMENTS'succ (COMPONENT),
         VALUE          => VALUE,
         HIGH_NODE1     => HIGH_NODES1,
         LOW_NODE1      => LOW_NODES1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Outer.Get_Frequency (FREQUENCY => FREQUENCY,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Insert_Conversion (FREQUENCY => FREQUENCY,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Write (ITEM => screen_supporter.READING1,
         FORM => UNDERLINED);
      Outer.List_Branch (LIST_MODE => ADMITTANCE,
         FORM          => PROCEDURE_FORM,
         ROW           => SCREEN_ROW,
         NETLIST_OBJECT => NETLIST_OBJECT);
   exception
      when NODE_PAIR_DOES_NOT_EXIST | NO_EXISTING_COMPONENT =>
         Errors.Write (MESSAGE => "Such a component does not exist.");
   end Change_Magnitude_Phase;

   ----------------------------------------------------------------------
   -- PURPOSE : Inserts a  component into  the  netlist  calculating  all the
   --           necessary inductances etc if there are any.
   ----------------------------------------------------------------------

   procedure Insert_Component (NETLIST_OBJECT : in out NETLIST_TYPE) is

      use Netlists.Inner;

      HIGH_NODE1,           LOW_NODE1,
      HIGH_NODE2,           LOW_NODE2          : NODE_TYPE;
      SCREEN_ROW                               : POSITIVE := 3;
      COMPONENT                                : ELEMENT_TYPE;
      FREQUENCY,            PHASE,
      VALUE                                    : REAL_TYPE;
      MORE_COMPONENTS                          : BOOLEAN;

      ----------------------------------------------------------------------

      procedure Insertion is

         MUTUAL_VALUE      : REAL_TYPE;
         FOUND_BRANCH      : BOOLEAN := TRUE;
         TEMP_COMPONENT    : ELEMENTS;

      begin
         case COMPONENT is
            when CAPC..CUPLING => TEMP_COMPONENT := COMPONENT;
            when RESISTOR      => TEMP_COMPONENT := CR_ADMIT;
            when CRNT_SOURCE   => TEMP_COMPONENT := REAL_CRNT;
            when VOLT_SOURCE   => TEMP_COMPONENT := REAL_VOLT;
            when others        => null;
         end case;
         if TEMP_COMPONENT <= POLE_FREQ then
            begin
               Find_Branch (COMPONENT => TEMP_COMPONENT,
                  HIGH_NODE      => HIGH_NODE1,
                  LOW_NODE       => LOW_NODE1,
                  NETLIST_OBJECT => NETLIST_OBJECT);
            exception
               when others => FOUND_BRANCH := FALSE;
            end;
         end if;

         if FOUND_BRANCH or (COMPONENT in CR_ADMIT..CUPLING) then
            declare
               FOUND_SOURCE : BOOLEAN;
            begin
               case COMPONENT is
                  when INDUCT =>
                     Write_Netlist (COMPONENT => INDUCT,
                        VALUE          => VALUE,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     if VALUE > MINIMUM_VALUE then
                        Write_Netlist (COMPONENT => ACT_INDCT,
                           VALUE          => VALUE,
                           HIGH_NODE1     => HIGH_NODE1,
                           LOW_NODE1      => LOW_NODE1,
                           HIGH_NODE2     => HIGH_NODE2,
                           LOW_NODE2      => LOW_NODE2,
                           NETLIST_OBJECT => NETLIST_OBJECT);
                     else
                        Write_Netlist (COMPONENT => ACT_INDCT,
                           VALUE          => VALUE,
                           HIGH_NODE1     => 0,
                           LOW_NODE1      => 0,
                           HIGH_NODE2     => HIGH_NODE2,
                           LOW_NODE2      => LOW_NODE2,
                           NETLIST_OBJECT => NETLIST_OBJECT);
                     end if;
                     Update_Mutuals_and_Coupling (VALUE => VALUE,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     NR_Inductance_Matrix (NETLIST_OBJECT => NETLIST_OBJECT);

                  when CR_ADMIT..POLE_FREQ =>
                     Find_Branch (COMPONENT => COMPONENT,
                        HIGH_NODE      => HIGH_NODE2,
                        LOW_NODE       => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     Find_Branch (COMPONENT => COMPONENT,
                        HIGH_NODE      => HIGH_NODE1,
                        LOW_NODE       => LOW_NODE1,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     Find_Dependent_Source (
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        FOUND_SOURCE   => FOUND_SOURCE,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     if VALUE < MINIMUM_VALUE then
                        HIGH_NODE1 := 0;
                     end if;
                     Write_Netlist (COMPONENT => COMPONENT,
                        VALUE          => VALUE,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);

                  when MUT_INDCT..CUPLING =>
                     Calculate_Mutual_Inductance (
                        HIGH_NODE1           => HIGH_NODE1,
                        LOW_NODE1            => LOW_NODE1,
                        HIGH_NODE2           => HIGH_NODE2,
                        LOW_NODE2            => LOW_NODE2,
                        MUTUAL_INDUCT_VALUE  => MUTUAL_VALUE,
                        NETLIST_OBJECT       => NETLIST_OBJECT);
                     Find_Mutuals_and_Coupling (COMPONENT => COMPONENT,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        MUTUAL_VALUE   => MUTUAL_VALUE,
                        VALUE          => VALUE,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     NR_Inductance_Matrix (NETLIST_OBJECT => NETLIST_OBJECT);

                  when REAL_VOLT | REAL_CRNT =>
                     Write_Netlist (COMPONENT => COMPONENT,
                        VALUE          => VALUE,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE1,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     Write_Netlist (COMPONENT=>ELEMENT_TYPE'succ(COMPONENT),
                        VALUE          => 0.0,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
                        HIGH_NODE2     => HIGH_NODE2,
                        LOW_NODE2      => LOW_NODE2,
                        NETLIST_OBJECT => NETLIST_OBJECT);

                  when IMAG_VOLT | IMAG_CRNT =>
                     Write_Netlist (COMPONENT => COMPONENT,
                        VALUE          => VALUE,
                        HIGH_NODE1     => HIGH_NODE1,
                        LOW_NODE1      => LOW_NODE1,
```

```
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Write_Netlist (COMPONENT=>ELEMENT_TYPE'pred(COMPONENT),
                  VALUE           => 0.0,
                  HIGH_NODE1      => HIGH_NODE1,
                  LOW_NODE1       => LOW_NODE1,
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);

            when VOLT_SOURCE | CRNT_SOURCE =>
               Write_Netlist (COMPONENT => TEMP_COMPONENT,
                  VALUE           => VALUE * Math_Lib.Cos(PHASE/RADIAN),
                  HIGH_NODE1      => HIGH_NODE1,
                  LOW_NODE1       => LOW_NODE1,
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Write_Netlist (
                  COMPONENT       => ELEMENT_TYPE'succ (TEMP_COMPONENT),
                  VALUE           => VALUE * Math_Lib.Sin(PHASE/RADIAN),
                  HIGH_NODE1      => HIGH_NODE1,
                  LOW_NODE1       => LOW_NODE1,
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);

            when CAPS | GR_ADMIT | GI_ADMIT =>
               Write_Netlist (COMPONENT => COMPONENT,
                  VALUE           => VALUE,
                  HIGH_NODE1      => HIGH_NODE1,
                  LOW_NODE1       => LOW_NODE1,
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);

            when RESISTOR =>
               if (VALUE /= 0.0) then
                  VALUE := 1.0 / VALUE;
               end if;
               COMPONENT := GR_ADMIT;
               Write_Netlist (COMPONENT => GR_ADMIT,
                  VALUE           => VALUE,
                  HIGH_NODE1      => HIGH_NODE1,
                  LOW_NODE1       => LOW_NODE1,
                  HIGH_NODE2      => HIGH_NODE2,
                  LOW_NODE2       => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);

            when others => null;
         end case;
      exception
         when STORAGE_ERROR =>
            Errors.Write ("Insufficient memory to insert component.",
               HELP_INFO => HP_INSERT_COMPONENT);
         when TOO_MANY_BRANCHES =>
            Errors.Write ("Too many branches.",
               HELP_INFO => HP_INSERT_COMPONENT);
         when others =>
            Errors.Write (MESSAGE => "The component could not be "
                  & "inserted.");
      end;

      if ((COMPONENT in CAPS..GR_ADMIT) or
         (COMPONENT in GR_ADMIT..COUPLING)) and (CIRCUIT_MODE=RLC) then
         Convert_RLC_To_Admittance (FREQUENCY => FREQUENCY,
            NETLIST_OBJECT => NETLIST_OBJECT);
      elsif COMPONENT = GI_ADMIT then
         Insert_Conversion (FREQUENCY => FREQUENCY,
            NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
   end if;
end Insertion;

-------------------------------------------------------------------------

begin  -- Insert_Component
   Process_Title (TITLE => "INSERT COMPONENT");
   loop
      Screen_Supporter.Get_Component (COMPONENT => COMPONENT);
      INSERT_FORM (1).MAX_REAL := COMPONENT_VALUE (COMPONENT).MAX;
      if COMPONENT in GR_ADMIT..COUPLING then
         INSERT_FORM (5).ACCEPTING := TRUE;
         INSERT_FORM (6).ACCEPTING := TRUE;
      else
         INSERT_FORM (5).ACCEPTING := FALSE;
         INSERT_FORM (6).ACCEPTING := FALSE;
      end if;
      if COMPONENT in CRNT_SOURCE..VOLT_SOURCE then
         INSERT_FORM (7).ACCEPTING := TRUE;
      else
         INSERT_FORM (7).ACCEPTING := FALSE;
      end if;
      Prompts.Read (FORM => INSERT_FORM);
      VALUE     := INSERT_FORM (1).REAL_VALUE;
      FREQUENCY := INSERT_FORM (2).REAL_VALUE;
      HIGH_NODE1 := INSERT_FORM (3).INTEGER_VALUE;
      LOW_NODE1  := INSERT_FORM (4).INTEGER_VALUE;
      HIGH_NODE2 := INSERT_FORM (5).INTEGER_VALUE;
      LOW_NODE2  := INSERT_FORM (6).INTEGER_VALUE;
      PHASE     := INSERT_FORM (7).REAL_VALUE;
      if (VALUE /= 0.0) and (VALUE < COMPONENT_VALUE (COMPONENT).MIN) then
         VALUE := COMPONENT_VALUE (COMPONENT).MIN;
      end if;
      if SCREEN_ROW = 3 then
         Clear;
         Write (ITEM => Screen_Supporter.HEADING1,
            FORM => UNDERLINED);
      end if;
      Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
         HIGH_NODE1 => HIGH_NODE1,
         LOW_NODE1  => LOW_NODE1,
         HIGH_NODE2 => HIGH_NODE2,
         LOW_NODE2  => LOW_NODE2,
         VALUE      => VALUE,
         PHASE      => PHASE,
         ROW        => SCREEN_ROW);
      Insertion;
      Prompts.Read (MESSAGE => "Are there still more components to be"
            & " entered ?",
         ITEM => MORE_COMPONENTS);
      exit when not MORE_COMPONENTS;
      if SCREEN_ROW > 24 then
         SCREEN_ROW := 3;
      end if;
   end loop;
end Insert_Component;

-------------------------------------------------------------------------
-- PURPOSE : Lists all the components of a certain type found in the
--           NETLIST_OBJECT on the main screen.  The components can also
--           be in the form of equivalent inductors or admittances.
-------------------------------------------------------------------------

procedure List_Netlist (ALL_COMPONENTS : in BOOLEAN;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   COMPONENT                          : ELEMENTS;
   SHOW_COMPONENT,      FORM          : ELEMENT_TYPE;
   LIST_COMPONENT                     : BOOLEAN := FALSE;
   ROW_ON_SCREEN                      : POSITIVE := 3;
   VALUE,               PHASE,
   TEMP_VALUE                         : REAL_TYPE;
   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2     : NODE_TYPE;

   LISTING_LIST : LIST_ARRAY_TYPE (1..3) := (
      (TEXT_LENGTH => 6,  TEXT => "R.L.C.",           STATE => TRUE),
      (TEXT_LENGTH => 11, TEXT => "Admittances",      STATE => FALSE),
      (TEXT_LENGTH => 20, TEXT => "Equivalent Inductors", STATE =>FALSE));

begin
   Main.Process_Title (TITLE => "LIST CIRCUIT");
   if ALL_COMPONENTS then
      Prompts.Read (LIST => LISTING_LIST,
         EXCLUSIVE => TRUE,
         HELP_INFO => HP_LIST_NETLIST);
      if LISTING_LIST (1).STATE then
         CIRCUIT_MODE := RLC;
         FORM         := RLC_FORM;
      elsif LISTING_LIST (2).STATE then
         CIRCUIT_MODE := ADMITTANCE;
         FORM         := ADMIT_FORM;
      elsif LISTING_LIST (3).STATE then
         FORM := EQ_INDCT_FORM;
      end if;
   else
      Screen_Supporter.Get_Component (COMPONENT => FORM);
      CIRCUIT_MODE := RLC;
   end if;
   case FORM is
      when CAPS..INDUCT | GR_ADMIT..COUPLING =>
         COMPONENT := FORM;
         SHOW_COMPONENT := FORM;
      when GR_ADMIT | RESISTOR =>
         COMPONENT := GR_ADMIT;
         SHOW_COMPONENT := RESISTOR;
      when REAL_VOLT | IMAG_VOLT | VOLT_SOURCE =>
         COMPONENT := REAL_VOLT;
         SHOW_COMPONENT := VOLT_SOURCE;
      when REAL_CRNT | IMAG_CRNT | CRNT_SOURCE =>
         COMPONENT := REAL_CRNT;
         SHOW_COMPONENT := CRNT_SOURCE;
      when others =>
         LIST_COMPONENT := TRUE;
   end case;
   Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
   loop
      if ROW_ON_SCREEN = 3 then
         Process_Title (TITLE => "LIST OF COMPONENTS");
         Write (ITEM => Screen_Supporter.HEADING1,
            FORM => UNDERLINED);
         Cursor_Next_Line;
      end if;
      if LIST_COMPONENT then
         Outer.List_Branch (LIST_MODE => CIRCUIT_MODE,
            FORM           => FORM,
            ROW            => ROW_ON_SCREEN,
            NETLIST_OBJECT => NETLIST_OBJECT);
      else
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         if (COMPONENT = REAL_VOLT) or (COMPONENT = REAL_CRNT) then
            TEMP_VALUE := VALUE;
            Inner.Read_Netlist (COMPONENT => ELEMENT_TYPE'succ(COMPONENT),
               VALUE          => VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            PHASE := Math_Lib.Arctan (Y => VALUE, X => TEMP_VALUE,
               Y_COORDINATE => TEMP_VALUE,
               X_COORDINATE => VALUE);
            VALUE := Math_Lib.Sqrt (TEMP_VALUE*TEMP_VALUE + VALUE*VALUE);
         end if;

         if abs (VALUE) > MINIMUM_VALUE then
            if (COMPONENT = GR_ADMIT) and (CIRCUIT_MODE = RLC) then
               VALUE := 1.0 / VALUE;
            end if;
            Screen_Supporter.Display_Component (COMPONENT=>SHOW_COMPONENT,
               HIGH_NODE1 => HIGH_NODE1,
               LOW_NODE1  => LOW_NODE1,
               HIGH_NODE2 => HIGH_NODE2,
               LOW_NODE2  => LOW_NODE2,
               VALUE      => VALUE,
               PHASE      => PHASE,
               ROW        => ROW_ON_SCREEN);
         end if;
      end if;

      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      exception
```

```
            when NO_MORE_BRANCHES =>
               exit;
         end;
         if ROW_ON_SCREEN > 24 then
            ROW_ON_SCREEN := 3;
            Status.Wait;
         end if;
      end loop;
      if FORM = EQ_INDUCT_FORM then
         COMPONENT_VALUE (INDUCT).DISPLAY := "L   ";
      end if;
      CIRCUIT_MODE := RLC;
   end List_Netlist;

end Netlist_Supporter;
```

Netlists

```
with  Ada_Extensions;
use   Ada_Extensions;

package Netlists is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This contains the data type definition and methods for
--           instantiations of the NETLIST_TYPE abstract data type.  The
--           netlists object is constructed from three objects :
--              (1) Screen_Supporter object
--              (2) Inner object
--              (3) Outer object
--           The Screen_Supporter and Outer objects can use the low-level
--           services of the User Interface, while the Inner object cannot

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   MAX_NEGATIVE_DECIBEL : constant := -999.0;
   MAX_VOLT_VALUE       : constant := 1.0E18;
   MIN_FREQUENCY        : constant := 1.0E-16;
   MINIMUM_VALUE        : constant := 1.0E-30;
   MIN_VOLT_VALUE       : constant := 1.0E-18;
   DOUBLE_BRANCHES      : constant := 72;
   MAX_BRANCHES         : constant := 36;
   MAX_NODES            : constant := 75;
   MAX_CALCULATIONS     : constant := MAX_NODES;
   MAX_OPTIMUM_VARIABLE : constant := 5;

   type ELEMENT_TYPE is (CAPS,          INDUCT,          GR_ADMIT,
      GI_ADMIT,       REAL_VOLT,       IMAG_VOLT,       REAL_CRNT,
      IMAG_CRNT,      ACT_INDUCT,      CR_ADMIT,        CI_ADMIT,
      CM_ADMIT,       POLE_FREQ,       MT_INDUCT,       COUPLING,
      CRNT_SOURCE,    VOLT_SOURCE,     RESISTOR,        ADMIT_FORM,
      EQ_INDUCT_FORM, PROCEDURE_FORM, RLC_FORM);

   type SEARCH_COMPONENT_TYPE is (GENERAL_COMPONENT, INDUCTIVE_COMPONENT,
      MUTUAL_COMPONENT);

   type NETLIST_TYPE is private;

   subtype ELEMENTS  is ELEMENT_TYPE range CAPS..COUPLING;

   type COMPONENT_UNIT_TYPE is
      record
         UNITS   : STRING (1..4);
         DISPLAY : STRING (1..4);
         MIN,
         MAX     : REAL_TYPE;
      end record;

   type COMPONENT_LIST_TYPE is array (CAPS..RESISTOR) of
      COMPONENT_UNIT_TYPE;

   COMPONENT_VALUE : COMPONENT_LIST_TYPE;

   procedure Set_Component_Values (COMP_VALUES : out COMPONENT_LIST_TYPE);

   type CIRCUIT_MODE_TYPE is (ADMITTANCE, RLC);

   CIRCUIT_MODE : CIRCUIT_MODE_TYPE := RLC;

   package Screen_Supporter is

      HEADING1 : constant STRING := "    Component    Nodes      Value      "
         & " Units          Additional Data              ";

      procedure Display_Component (COMPONENT : in ELEMENT_TYPE;
         HIGH_NODE1, LOW_NODE1,
         HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
         VALUE, PHASES         : in REAL_TYPE;
         ROW                   : in out INTEGER_TYPE);

      procedure Get_Component (COMPONENT : out ELEMENT_TYPE);

   end Screen_Supporter;

   package Inner is

      procedure DR_Inductance_Matrix (NETLIST_OBJECT:in out NETLIST_TYPE);

      procedure Calculate_Mutual_Inductance (HIGH_NODE1, LOW_NODE1,
         HIGH_NODE2, LOW_NODE2 : in out NODE_TYPE;
         MUTUAL_INDUCT_VALUE   :     out REAL_TYPE;
         NETLIST_OBJECT        : in     NETLIST_TYPE);
```

```
      procedure Check_Topology (NETLIST_OBJECT : in NETLIST_TYPE);

      procedure Clear_Branch (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Clear_Locations (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Convert_Admittance_To_RLC (FREQUENCY : in REAL_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Convert_RLC_To_Admittance (FREQUENCY : in REAL_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      function Count_Components(SEARCH_COMPONENT:in SEARCH_COMPONENT_TYPE;
         NETLIST_OBJECT : in NETLIST_TYPE) return NATURAL;

      procedure Delete_Branch (NETLIST_OBJECT : in out NETLIST_TYPE);

      function Empty_Branch (NETLIST_OBJECT : in NETLIST_TYPE)
         return BOOLEAN;

      function Equal_Locations (LOCATION_INDEX : in POSITIVE;
         NETLIST_OBJECT : in NETLIST_TYPE) return BOOLEAN;

      procedure Find_Branch (COMPONENT : in ELEMENTS;
         HIGH_NODE, LOW_NODE : in out NODE_TYPE;
         NETLIST_OBJECT      : in out NETLIST_TYPE);

      procedure Find_Dependent_Source (HIGH_NODE1, LOW_NODE1, HIGH_NODE2,
         LOW_NODE2     : in     NODE_TYPE;
         CREATE        : in     BOOLEAN := TRUE;
         FOUND_SOURCE  :    out BOOLEAN;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Find_Mutuals_and_Coupling (COMPONENT : in ELEMENTS;
         HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
         MUTUAL_VALUE, VALUE : in     REAL_TYPE;
         CREATE              : in     BOOLEAN := TRUE;
         NETLIST_OBJECT      : in out NETLIST_TYPE);

      procedure Find_Node_Maximum (NODE_MAXIMUM : out NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure First_Branch (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Initialize (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Next_Branch (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Read_Netlist (COMPONENT : in ELEMENTS;
         VALUE : out REAL_TYPE;
         HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : out NODE_TYPE;
         NETLIST_OBJECT : in NETLIST_TYPE);

      procedure Retrieve_Location (LOCATION_INDEX : in POSITIVE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Save_Location (LOCATION_INDEX : in POSITIVE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Update_Mutuals_and_Coupling (VALUE : in REAL_TYPE;
         HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Write_Frequency (FREQUENCY : in REAL_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Write_Netlist (COMPONENT : in ELEMENTS;
         VALUE : in REAL_TYPE;
         HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

   end Inner;

   ------------------------------------------------------------------------

   package Outer is

      procedure Change_Drive;

      procedure Change_Directory;

      procedure Get_Frequency (FREQUENCY : out REAL_TYPE;
         NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Initialize_Directory;

      procedure List_Branch (LIST_MODE : in CIRCUIT_MODE_TYPE;
         FORM        : in ELEMENT_TYPE;
         ROW         : in out POSITIVE;
         NETLIST_OBJECT : in NETLIST_TYPE);

      procedure Load_Netlist (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Load_Monte_Netlist (NETLIST_OBJECT1,
         NETLIST_OBJECT2 : in out NETLIST_TYPE);

      procedure Remove_Node (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Renumber_Nodes (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Save_Netlist (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Save_Monte_Netlist (NETLIST_OBJECT1,
         NETLIST_OBJECT2 : in out NETLIST_TYPE);

      procedure Set_Circuit_Title (NETLIST_OBJECT : in out NETLIST_TYPE);

      procedure Show_Node (NETLIST_OBJECT : in out NETLIST_TYPE);

   end Outer;

   ------------------------------------------------------------------------

   BRANCH_OUT_OF_RANGE                : exception;
   CANNOT_CONVERT_WITH_MUTUAL_INDUCTANCES : exception;
   MEANINGLESS_COMPONENT              : exception;
   MEANINGLESS_COMPONENT_CONNECTION   : exception;
   NO_CIRCUIT                         : exception;
   NO_MORE_BRANCHES                   : exception;
   NO_VCCS_CONTROLLING_NODE           : exception;
   NODE_PAIR_DOES_NOT_EXIST           : exception;
   REFERENCE_NODE_MISSING             : exception;
   SPECIFIC_NODE_MISSING              : exception;
   TOO_MANY_BRANCHES                  : exception;

   ------------------------------------------------------------------------

private

   type GENERALS is
      record
         CAPACITOR,                              -- capacitor
         EQUIV_INDUCTOR,                         -- equivalent inductor
         REAL_ADMITTANCE,                        -- real admittance
         IMAG_ADMITTANCE,                        -- imaginary admittance
         REAL_VOLTAGE,                           -- real voltage
         IMAG_VOLTAGE,                           -- imaginary voltage
         REAL_CURRENT,                           -- real current
         IMAG_CURRENT     : REAL_TYPE := 0.0;    -- imaginary current
         HIGHER_NODE,                            -- higher node
         LOWER_NODE       : NODE_TYPE := 0;      -- lower node
      end record;

   type INDUCTORS is
      record
         ACTUAL_INDUCTOR : REAL_TYPE := 0.0;     -- actual inductor
         HIGHER_NODE,                            -- higher node
         LOWER_NODE      : NODE_TYPE := 0;       -- lower node
      end record;

   type VOLTAGE_CONTROL is                       -- voltage controlled
current
      record                                     --    source
         REAL_ADMITTANCE,                        -- real admittance
         IMAG_ADMITTANCE,                        -- imaginary admittance
         MAG_ADMITTANCE,                         -- admittance magnitude
         POLE_FREQUENCY  : REAL_TYPE := 0.0;     -- pole frequency
         HIGHER_CONTROL_NODE,                    -- higher controlling node
         LOWER_CONTROL_NODE,                     -- lower controlling node
         HIGHER_NODE,                            -- higher node
         LOWER_NODE      : NODE_TYPE := 0;       -- lower node
      end record;

   type MUTUAL_INDUCTORS is
      record
         MUTUAL_INDUCTOR,                        -- mutual inductors
         COUPLING        : REAL_TYPE := 0.0;     -- coupling factor
         HIGHER_NODE1,                           -- higher node 1
         LOWER_NODE1,                            -- lower  node 1
         HIGHER_NODE2,                           -- higher node 2
         LOWER_NODE2     : NODE_TYPE := 0;       -- lower  node 2
      end record;

   type COMPONENTS is                            -- components in branch
      record
         GENERAL         : GENERALS;             -- general components
         INDUCTIVE       : INDUCTORS;            -- inductive components
         DEPEND_SOURCE   : VOLTAGE_CONTROL;      -- dependent sources
         MUTUAL_INDUCT   : MUTUAL_INDUCTORS;     -- mutual components
      end record;

   type NETLIST;
   type LINK is access NETLIST;

   type NETLIST is
      record
         NEXT            : LINK;                 -- next branch
         DATA            : COMPONENTS;           -- components in branch
      end record;

   type NETLIST_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE) of LINK;

   type NETLIST_TYPE is
      record
         LIST            : LINK;                 -- present branch in netlist
         HEAD            : LINK;                 -- first branch in netlist
         LOCATION        : NETLIST_ARRAY_TYPE;-- saved pointers to branches
         NODE_MAXIMUM    : NODE_TYPE := 0;    -- the highest node in netlist
         FREQUENCY       : REAL_TYPE := 1000.0;-- present network frequency
         TITLE           : STRING (1..64) := (1..64 => ' ');
                                              -- title of the network
      end record;

   ------------------------------------------------------------------------

end Netlists;

------------------------------------------------------------------------

with  Math_Lib,
      Unchecked_Deallocation;

use   Math_Lib;

package body Netlists is

------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Then these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

------------------------------------------------------------------------
-- PURPOSE : Sets the display symbols and units for the circuit components
--           as well as the maximum and minimum values.
------------------------------------------------------------------------

procedure Set_Component_Values (COMP_VALUES : out COMPONENT_LIST_TYPE) is
```

```
begin
   COMP_VALUES := (
      (UNITS => "F   ", DISPLAY => "F   ", MIN => 1.0E-15, MAX => 1.0E3),
      (UNITS => "H   ", DISPLAY => "H   ", MIN => 1.0E-16, MAX => 1.0E6),
      (UNITS => "MEGA", DISPLAY => "M   ", MIN => 1.0E-9,  MAX => 1.0E4),
      (UNITS => "MEGA", DISPLAY => "MI  ", MIN => 1.0E-12, MAX => 1.0E6),
      (UNITS => "V   ", DISPLAY => "VR  ", MIN => 1.0E-12, MAX => 1.0E12),
      (UNITS => "V   ", DISPLAY => "VI  ", MIN => 1.0E-12, MAX => 1.0E12),
      (UNITS => "A   ", DISPLAY => "IR  ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "A   ", DISPLAY => "II  ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "H   ", DISPLAY => "L   ", MIN => 1.0E-15, MAX => 1.0E6),
      (UNITS => "OHMS", DISPLAY => "OHM ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "MHOS", DISPLAY => "MHO ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "MEGA", DISPLAY => "MA  ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "Hz  ", DISPLAY => "JOHF", MIN => 1.0E-8,  MAX => 1.0E16),
      (UNITS => "H   ", DISPLAY => "M   ", MIN => 1.0E-16, MAX => 1.0E6),
      (UNITS => "    ", DISPLAY => "K   ", MIN => 0.0,   MAX => 0.999_999),
      (UNITS => "A   ", DISPLAY => "I   ", MIN => 1.0E-16, MAX => 1.0E16),
      (UNITS => "V   ", DISPLAY => "V   ", MIN => 1.0E-12, MAX => 1.0E12),
      (UNITS => "Ohms", DISPLAY => "R   ", MIN => 1.0E-4,  MAX => 1.0E9)));
end Set_Component_Values;
```

```
package body Screen_Supporter is separate;
```

```
package body Inner is

   procedure Dispose is new Unchecked_Deallocation (NETLIST, LINK);

   -- Variables used in the BR_Inductance_Matrix subprogram.

   subtype VECTOR1 is POSITIVE range 1..MAX_BRANCHES;
   subtype VECTOR2 is POSITIVE range 1..(MAX_NODES + 1);
   subtype VECTOR3 is POSITIVE range 1..DOUBLE_BRANCHES;

   type EXPAND_TYPE       is array (VECTOR2, VECTOR1) of REAL_TYPE;
   type RIM_TYPE          is array (VECTOR2, VECTOR2) of REAL_TYPE;
   type WORK_TYPE         is array (VECTOR1, VECTOR3) of REAL_TYPE;
   type NODE_CONNECT_TYPE is array (VECTOR1) of NODE_TYPE;
   type NODE_LIST_TYPE    is array (VECTOR2) of NODE_TYPE;

   type MATRIX_TYPE is
      record
         EXPAND : EXPAND_TYPE;
         WORK   : WORK_TYPE;
      end record;

   type LIST_TYPE is
      record
         NODE_CONNECT1 : NODE_CONNECT_TYPE;
         NODE_CONNECT2 : NODE_CONNECT_TYPE;
         NODES         : NODE_LIST_TYPE;
      end record;

   type MATRIX_ACCESS is access MATRIX_TYPE;
   type RIM_ACCESS    is access RIM_TYPE;
   type LIST_ACCESS   is access LIST_TYPE;

   MATRIX     : MATRIX_ACCESS;
   MATRIX_RIM : RIM_ACCESS;
   LIST       : LIST_ACCESS;

-- PURPOSE : Find branch and reciprocal inductance matrices.  A branch
--           inductance matrix is generated  from the data in the netlist.
--           It is converted to an indefinite reciprocal inductance matrix
--           from which are extracted the equivalent inductors  to be used
--           in the nodal analysis solution.
--           For further information, see :
--              L.N.N. Edward, "EIMRIM : A Mutual Inductance Algorithm
--              for use with Nodal Analysis Programs",  International
--              Journal of Electrical Engineering  Education,  Vol 20,
--              No 1, pp 341 - 350, February 1983
-- EXCEPTIONS : STORAGE_ERROR
--              TOO_MANY_BRANCHES

procedure BR_Inductance_Matrix (NETLIST_OBJECT : in out NETLIST_TYPE) is

   CIRCUIT_BRANCHES,    CIRCUIT_MUTUALS,
   NO_OF_BRANCHES,      NO_OF_COLUMNS     : NATURAL;
   DATA_DONE                              : INTEGER_TYPE := 0;

   procedure Dispose_Matrix is new
      Unchecked_Deallocation (MATRIX_TYPE, MATRIX_ACCESS);

   procedure Dispose_Matrix_Rim is new
      Unchecked_Deallocation (RIM_TYPE, RIM_ACCESS);

   procedure Dispose_List is new
      Unchecked_Deallocation (LIST_TYPE, LIST_ACCESS);

   -- PURPOSE : Update the equivalent inductors that have been generated.

   procedure Update_Equivalent_Inductors is

      POINTER                          : LINK := NETLIST_OBJECT.LIST;
      VALUE                            : REAL_TYPE;
      COLUMN, COUNTER                  : POSITIVE;
      TEMP_HIGH_NODE,  TEMP_LOW_NODE   : NODE_TYPE;

   begin
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
      while NETLIST_OBJECT.LIST /= null loop
         NETLIST_OBJECT.LIST.DATA.GENERAL.EQUIV_INDUCTOR := 0.0;
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
      end loop;
      for COUNTER in 1..(NO_OF_COLUMNS - 1) loop
         for COLUMN in (COUNTER + 1)..NO_OF_COLUMNS loop
            TEMP_HIGH_NODE := LIST.NODES (COLUMN);
            TEMP_LOW_NODE  := LIST.NODES (COUNTER);
            if abs (MATRIX_RIM (COUNTER, COLUMN)) > MINIMUM_VALUE then
               VALUE := -1.0 / MATRIX_RIM (COUNTER, COLUMN);
               Find_Branch (COMPONENT => INDUCT,
                  HIGH_NODE      => TEMP_HIGH_NODE,
                  LOW_NODE       => TEMP_LOW_NODE,
                  NETLIST_OBJECT => NETLIST_OBJECT);
               Write_Netlist (COMPONENT => INDUCT,
                  VALUE          => VALUE,
                  HIGH_NODE1     => TEMP_HIGH_NODE,
                  LOW_NODE1      => TEMP_LOW_NODE,
                  HIGH_NODE2     => 0,
                  LOW_NODE2      => 0,
                  NETLIST_OBJECT => NETLIST_OBJECT);
            end if;
         end loop;
      end loop;
      NETLIST_OBJECT.LIST := POINTER;
   end Update_Equivalent_Inductors;

   -- PURPOSE : Find  the  reciprocal inductance matrix  from  the  expand
   --           matrix.

   procedure Find_Reciprocal_Inductance_Matrix is

      POINTER                    : LINK;
      COUNTER1, COUNTER2, ROW    : POSITIVE;

   begin
      for COUNTER1 in 1..NO_OF_COLUMNS loop
         for COUNTER2 in 1..NO_OF_COLUMNS loop
            MATRIX_RIM (COUNTER1, COUNTER2) := 0.0;
         end loop;
      end loop;
      for ROW in 1..NO_OF_COLUMNS loop
         for COUNTER1 in 1..NO_OF_BRANCHES loop
            POINTER := NETLIST_OBJECT.HEAD;
            for COUNTER2 in 1..(LIST.NODE_CONNECT2 (COUNTER1) - 1) loop
               POINTER := POINTER.NEXT;
            end loop;
            if LIST.NODES (ROW) = POINTER.DATA.GENERAL.HIGHER_NODE then
               for COUNTER2 in 1..NO_OF_COLUMNS loop
                  MATRIX_RIM (ROW, COUNTER2) := MATRIX_RIM (ROW, COUNTER2)
                     + MATRIX.EXPAND (COUNTER1, COUNTER2);
               end loop;
            end if;
            if LIST.NODES (ROW) = POINTER.DATA.GENERAL.LOWER_NODE then
               for COUNTER2 in 1..NO_OF_COLUMNS loop
                  MATRIX_RIM (ROW, COUNTER2) := MATRIX_RIM (ROW, COUNTER2)
                     - MATRIX.EXPAND (COUNTER1, COUNTER2);
               end loop;
            end if;
         end loop;
      end loop;
   end Find_Reciprocal_Inductance_Matrix;

   -- PURPOSE : Generates an  expanded  reciprocal inductance  coefficient
   --           matrix,  which,  when used with the node voltage solution,
   --           will yield all the inductive branch currents.

   procedure Find_Expand_Matrix is

      POINTER                          : LINK;
      COUNTER1, COUNTER2, ROW1, ROW2 : POSITIVE;

   begin
      for COUNTER1 in 1..NO_OF_BRANCHES loop
         for COUNTER2 in 1..NO_OF_BRANCHES loop
            MATRIX.EXPAND (COUNTER1, COUNTER2) := 0.0;
         end loop;
      end loop;
      for ROW1 in 1..NO_OF_COLUMNS loop
         for ROW2 in 1..NO_OF_BRANCHES loop
            for COUNTER1 in 1..NO_OF_BRANCHES loop
               POINTER := NETLIST_OBJECT.HEAD;
               for COUNTER2 in 1..(LIST.NODE_CONNECT2 (COUNTER1) - 1) loop
                  POINTER := POINTER.NEXT;
               end loop;
               if LIST.NODES(ROW1) = POINTER.DATA.GENERAL.HIGHER_NODE then
                  MATRIX.EXPAND (ROW2, ROW1) := MATRIX.EXPAND (ROW2, ROW1)
                     + MATRIX.WORK (ROW2, COUNTER1);
               end if;
               if LIST.NODES (ROW1) = POINTER.DATA.GENERAL.LOWER_NODE then
                  MATRIX.EXPAND (ROW2, ROW1) := MATRIX.EXPAND (ROW2, ROW1)
                     - MATRIX.WORK (ROW2, COUNTER1);
               end if;
            end loop;
         end loop;
      end loop;
   end Find_Expand_Matrix;

   -- PURPOSE : Generate a node-column  reference  table.  This is done by
   --           calculating the  number of columns  (NO_OF_COLUMNS) in the
   --           expand matrix (MATRIX.EXPAND).

   procedure Find_Node is

      POINTER                    : LINK;
      COUNTER1, COUNTER2, ROW    : NATURAL;
      TEMP_NODE1                 : NODE_TYPE := 0;

   begin
      NO_OF_COLUMNS := 0;
      for ROW in 1..(NETLIST_OBJECT.NODE_MAXIMUM + 1) loop
         COUNTER1 := 0;
         loop
            COUNTER1 := COUNTER1 + 1;
            POINTER := NETLIST_OBJECT.HEAD;
            for COUNTER2 in 1..(LIST.NODE_CONNECT2 (COUNTER1) - 1) loop
               POINTER := POINTER.NEXT;
            end loop;
            if (POINTER.DATA.GENERAL.LOWER_NODE  = (ROW - 1)) or
               (POINTER.DATA.GENERAL.HIGHER_NODE = (ROW - 1)) then
               if POINTER.DATA.GENERAL.HIGHER_NODE = (ROW - 1) then
                  TEMP_NODE1 := POINTER.DATA.GENERAL.HIGHER_NODE;
```

```
            else
               TEMP_NODE1 := POINTER.DATA.GENERAL.LOWER_NODE;
            end if;
            NO_OF_COLUMNS := NO_OF_COLUMNS + 1;
            LIST.NODES (NO_OF_COLUMNS) := TEMP_NODE1;
            exit;
         end if;
         exit when COUNTER1 >= NO_OF_BRANCHES;
      end loop;
   end loop;
end Find_Node;

-- ------------------------------------------------------------------
-- PURPOSE : Generates the reciprocal inductance coefficient matrix
--           which is obtained from inverting the branch inductance
--           matrix.
-- ------------------------------------------------------------------

procedure Find_Gamma_Matrix is

   COLUMN, COUNTER, ROW  : POSITIVE;
   YX, PM, PIV           : REAL_TYPE;

begin
   for ROW in 1..NO_OF_BRANCHES loop
      for COLUMN in (NO_OF_BRANCHES + 1)..(2 * NO_OF_BRANCHES) loop
         MATRIX.WORK (ROW, COLUMN) := 0.0;
         if (ROW + NO_OF_BRANCHES) = COLUMN then
            MATRIX.WORK (ROW, COLUMN) := 1.0;
         end if;
      end loop;
   end loop;

   for ROW in 1..NO_OF_BRANCHES loop
      PIV := MATRIX.WORK (ROW, ROW);
      for COUNTER in 1..NO_OF_BRANCHES loop
         if ROW /= COUNTER then
            PM := MATRIX.WORK (COUNTER, ROW) / PIV;
            for COLUMN in (ROW + 1)..(NO_OF_BRANCHES + ROW) loop
               MATRIX.WORK (COUNTER, COLUMN)
                  := MATRIX.WORK (COUNTER, COLUMN)
                  - MATRIX.WORK (ROW, COLUMN) * PM;
            end loop;
         end if;
      end loop;
   end loop;

   for ROW in 1..NO_OF_BRANCHES loop
      YX := MATRIX.WORK (ROW, ROW);
      for COLUMN in 1..NO_OF_BRANCHES loop
         MATRIX.WORK (ROW, COLUMN)
            := MATRIX.WORK (ROW, COLUMN + NO_OF_BRANCHES) / YX;
      end loop;
   end loop;
end Find_Gamma_Matrix;

-- ------------------------------------------------------------------
-- PURPOSE : Generates a branch inductance matrix from the data in the
--           Netlist.
-- ------------------------------------------------------------------

procedure Find_Branch_Inductance_Matrix is

   POINTER1,            POINTER2,
   POINTER3                          : LINK;
   COLUMN,              COUNTER1,
   COUNTER2,            ROW,
   NODE1,               NODE2,
   NODE3,               NODE4        : NATURAL := 0;
   FOUND                             : BOOLEAN := FALSE;

begin
   for ROW in 1..NO_OF_BRANCHES loop
      for COLUMN in (ROW + 1)..NO_OF_BRANCHES loop
         MATRIX.WORK (ROW, COLUMN) := 0.0;
      end loop;
   end loop;

   POINTER1 := NETLIST_OBJECT.HEAD;
   for COUNTER2 in 1..CIRCUIT_BRANCHES loop
      if POINTER1.DATA.INDUCTIVE.ACTUAL_INDUCTOR > MINIMUM_VALUE then
         COUNTER1 := COUNTER1 + 1;
         LIST.NODE_CONNECT2 (COUNTER1) := COUNTER2;
         LIST.NODE_CONNECT1 (COUNTER2) := COUNTER1;
         MATRIX.WORK (COUNTER1, COUNTER1)
            := POINTER1.DATA.INDUCTIVE.ACTUAL_INDUCTOR;
      end if;
      POINTER1 := POINTER1.NEXT;
   end loop;

   POINTER3 := NETLIST_OBJECT.HEAD;
   while (POINTER3 /= null) and (not FOUND) loop
      if POINTER3.DATA.MUTUAL_INDUCT.HIGHER_NODE1 /= 0 then
         FOUND := TRUE;
      else
         POINTER3 := POINTER3.NEXT;
      end if;
   end loop;

   for COUNTER1 in 1..CIRCUIT_MUTUALS loop
      NODE1 := 1;
      POINTER1 := NETLIST_OBJECT.HEAD;
      FOUND := FALSE;
      while (POINTER1 /= null) and (not FOUND) loop
         if (POINTER1.DATA.INDUCTIVE.HIGHER_NODE
            = POINTER3.DATA.MUTUAL_INDUCT.HIGHER_NODE1) and
            (POINTER1.DATA.INDUCTIVE.LOWER_NODE
            = POINTER3.DATA.MUTUAL_INDUCT.LOWER_NODE1) then
            FOUND := TRUE;
         else
            POINTER1 := POINTER1.NEXT;
            NODE1 := NODE1 + 1;
         end if;
      end loop;

      if FOUND then
         NODE2 := LIST.NODE_CONNECT1 (NODE1);
      end if;
      NODE2 := 1;
      POINTER2 := NETLIST_OBJECT.HEAD;
      FOUND := FALSE;
      while (POINTER2 /= null) and (not FOUND) loop
         if (POINTER2.DATA.INDUCTIVE.HIGHER_NODE
            = POINTER3.DATA.MUTUAL_INDUCT.HIGHER_NODE2) and
            (POINTER2.DATA.INDUCTIVE.LOWER_NODE
            = POINTER3.DATA.MUTUAL_INDUCT.LOWER_NODE2) then
            FOUND := TRUE;
         else
            POINTER2 := POINTER2.NEXT;
            NODE2 := NODE2 + 1;
         end if;
      end loop;

      if FOUND then
         NODE4 := LIST.NODE_CONNECT1 (NODE2);
      end if;

      if NODE4 > NODE3 then
         MATRIX.WORK (NODE3, NODE4)
            := POINTER3.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR;
      else
         MATRIX.WORK (NODE4, NODE3)
            := POINTER3.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR;
      end if;
      POINTER3 := POINTER3.NEXT;
      FOUND := FALSE;
      while (POINTER3 /= null) and (not FOUND) loop
         if POINTER3.DATA.MUTUAL_INDUCT.HIGHER_NODE1 /= 0 then
            FOUND := TRUE;
         else
            POINTER3 := POINTER1.NEXT;
         end if;
      end loop;
   end loop;

   for ROW in 2..NO_OF_BRANCHES loop
      for COLUMN in 1..(ROW - 1) loop
         MATRIX.WORK (ROW, COLUMN) := MATRIX.WORK (COLUMN, ROW);
      end loop;
   end loop;
end Find_Branch_Inductance_Matrix;

-- ------------------------------------------------------------------

begin  -- BR_Inductance_Matrix
   CIRCUIT_MUTUALS := Count_Components (SEARCH_COMPONENT =>
         MUTUAL_COMPONENT, NETLIST_OBJECT => NETLIST_OBJECT);
   NO_OF_BRANCHES := Count_Components (SEARCH_COMPONENT =>
         INDUCTIVE_COMPONENT, NETLIST_OBJECT => NETLIST_OBJECT);
   CIRCUIT_BRANCHES := Count_Components (SEARCH_COMPONENT =>
         GENERAL_COMPONENT, NETLIST_OBJECT => NETLIST_OBJECT);

   if CIRCUIT_BRANCHES <= MAX_BRANCHES then
      MATRIX_BIN := new BIN_TYPE;
      DATA_DONE  := 1;
      MATRIX     := new MATRIX_TYPE;
      DATA_DONE  := 2;
      LIST       := new LIST_TYPE;
      Find_Branch_Inductance_Matrix;
      Find_Gamma_Matrix;
      Find_Node;
      Find_Expand_Matrix;
      Find_Reciprocal_Inductance_Matrix;
      Update_Equivalent_Inductors;
      Dispose_List (LIST);
      Dispose_Matrix (MATRIX);
      Dispose_Matrix_Bin (MATRIX_BIN);
   else
      raise TOO_MANY_BRANCHES;
   end if;
exception
   when STORAGE_ERROR =>
      if DATA_DONE > 0 then
         Dispose_Matrix_Bin (MATRIX_BIN);
      end if;
      if DATA_DONE = 2 then
         Dispose_Matrix (MATRIX);
      end if;
      raise STORAGE_ERROR;
end BR_Inductance_Matrix;

-- ------------------------------------------------------------------
-- PURPOSE : Calculates the  mutual inductance  in the whole netlist.  The
--           LIST pointer in the NETLIST_OBJECT is not affected.
-- EXCEPTIONS : MEANINGLESS_COMPONENT
--              MEANINGLESS_COMPONENT_CONNECTION
-- ------------------------------------------------------------------

procedure Calculate_Mutual_Inductance (HIGH_NODE1, LOW_NODE1, HIGH_NODE2,
   LOW_NODE2       : in out NODE_TYPE;
   MUTUAL_INDUCT_VALUE :    out REAL_TYPE;
   NETLIST_OBJECT  : in     NETLIST_TYPE) is

   POINTER                          : LINK := NETLIST_OBJECT.HEAD;
   TEMP_HIGH_NODE,      TEMP_LOW_NODE    : NODE_TYPE;
   HENRI_VALUE1,        HENRI_VALUE2     : REAL_TYPE := 0.0;
   FLAG                             : BOOLEAN   := FALSE;

begin
   if (HIGH_NODE1 = LOW_NODE1) or (HIGH_NODE2 = LOW_NODE2) then
      raise MEANINGLESS_COMPONENT_CONNECTION;
   end if;
   if HIGH_NODE1 < LOW_NODE1 then
      Maths.Swap_Node (NODE1 => HIGH_NODE1, NODE2 => LOW_NODE1);
   end if;
   if HIGH_NODE2 < LOW_NODE2 then
      Maths.Swap_Node (NODE1 => HIGH_NODE2, NODE2 => LOW_NODE2);
   end if;
   TEMP_HIGH_NODE := HIGH_NODE1;
   TEMP_LOW_NODE  := LOW_NODE1;
   while (POINTER /= null) and ((HENRI_VALUE1 = 0.0) or
      (HENRI_VALUE2 = 0.0)) loop
      if (TEMP_HIGH_NODE = POINTER.DATA.INDUCTIVE.HIGHER_NODE) and
         (TEMP_LOW_NODE  = POINTER.DATA.INDUCTIVE.LOWER_NODE) then
         if FLAG then
            HENRI_VALUE2 := POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR;
         else
            HENRI_VALUE1 := POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR;
            POINTER := NETLIST_OBJECT.HEAD;
```

```
            FLAG := TRUE;
            TEMP_HIGH_NODE := HIGH_NODE2;
            TEMP_LOW_NODE  := LOW_NODE2;
         end if;
      else
         POINTER := POINTER.NEXT;
      end if;
   end loop;
   if (HENRI_VALUE1 = 0.0) or (HENRI_VALUE2 = 0.0) then
      raise MEANINGLESS_COMPONENT;
   end if;
   MUTUAL_INDUCT_VALUE := Sqrt (abs (HENRI_VALUE1*HENRI_VALUE2));
end Calculate_Mutual_Inductance;

-- PURPOSE : Checks the topology of the netlist. The LIST pointer in
--           NETLIST_OBJECT is not affected.
-- EXCEPTIONS : NO_CIRCUIT
--              NO_VCCS_CONTROLLING_NODE
--              REFERENCE_NODE_MISSING
--              SPECIFIC_NODE_MISSING

procedure Check_Topology (NETLIST_OBJECT : in NETLIST_TYPE) is

   NODE                    : NODE_TYPE := 0;
   POINTER1, POINTER2      : LINK      := NETLIST_OBJECT.HEAD;
   DONE, EXISTS, FOUND, TESTED : BOOLEAN   := FALSE;

begin
   if NETLIST_OBJECT.NODE_MAXIMUM = 0 then
      raise NO_CIRCUIT;
   end if;
   while POINTER1 /= null loop
      EXISTS := FALSE;
      TESTED := FALSE;
      if (POINTER1.DATA.DEPEND_SOURCE.REAL_ADMITTANCE +
          POINTER1.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE +
          POINTER1.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE) > MINIMUM_VALUE then
         EXISTS := TRUE;
         POINTER2 := NETLIST_OBJECT.HEAD;
         while (POINTER2 /= null) and not TESTED loop
            if (POINTER2.DATA.GENERAL.HIGHER_NODE =
                POINTER1.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE) and
               (POINTER2.DATA.GENERAL.LOWER_NODE =
                POINTER1.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE) then
               TESTED := TRUE;
            else
               POINTER1 := POINTER2.NEXT;
            end if;
         end loop;
         if EXISTS and not TESTED then
            raise NO_VCCS_CONTROLLING_NODE;
         elsif TESTED then
            POINTER1 := POINTER1.NEXT;
         end if;
      else
         POINTER1 := POINTER1.NEXT;
      end if;
   end loop;

   loop
      NODE := NODE + 1;
      POINTER1 := NETLIST_OBJECT.HEAD;
      while (POINTER1 /= null) and not DONE loop
         if (POINTER1.DATA.GENERAL.HIGHER_NODE = NODE) and
            (POINTER1.DATA.GENERAL.LOWER_NODE  = 0) then
            DONE := TRUE;
         else
            POINTER1 := POINTER1.NEXT;
         end if;
      end loop;
      exit when (NODE >= NETLIST_OBJECT.NODE_MAXIMUM) or DONE;
   end loop;
   if not DONE then
      raise REFERENCE_NODE_MISSING;
   end if;
   NODE := 0;
   loop
      FOUND := FALSE;
      NODE  := NODE + 1;
      POINTER1 := NETLIST_OBJECT.HEAD;
      while (POINTER1 /= null) and not FOUND loop
         if (NODE = POINTER1.DATA.GENERAL.HIGHER_NODE) or
            (NODE = POINTER1.DATA.GENERAL.LOWER_NODE) then
            FOUND := TRUE;
         else
            POINTER1 := POINTER1.NEXT;
         end if;
      end loop;
      if not FOUND then
         raise SPECIFIC_NODE_MISSING;
      end if;
      exit when NODE >= NETLIST_OBJECT.NODE_MAXIMUM;
   end loop;
end Check_Topology;

-- PURPOSE : Sets all the component fields in the branch pointed to by
--           LIST in the NETLIST_OBJECT pointer to zero.

procedure Clear_Branch (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LIST.DATA :=
      ((0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0, 0), (0.0, 0, 0),
       (0.0, 0.0, 0.0, 0.0, 0, 0, 0, 0), (0.0, 0.0, 0, 0, 0, 0));
end Clear_Branch;

-- PURPOSE : Clears all the LOCATIONS of the NETLIST_OBJECT so that they
--           point to nothing.

procedure Clear_Locations (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LOCATION := (1..MAX_OPTIMUM_VARIABLE => null);
end Clear_Locations;

-- PURPOSE : Convert the representation of components from admittance to
--           RLC form in the whole netlist. The LIST pointer in
--           NETLIST_OBJECT is not affected.
-- EXCEPTIONS : CANNOT_CONVERT_WITH_MUTUAL_INDUCTANCES
--              STORAGE_ERROR
--              TOO_MANY_BRANCHES

procedure Convert_Admittance_To_RLC (FREQUENCY : in REAL_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   MINIMUM_CI_ADMITTANCE : constant := 1.0E-10;
   MINIMUM_VCCS          : constant := 1.0E-20;

   POINTER                          : LINK := NETLIST_OBJECT.HEAD;
   OMEGA                            : REAL_TYPE := 2.0*PI*FREQUENCY;
   RATIO,              TEMP_ADMITTANCE,
   DENOMINATOR                      : REAL_TYPE;

begin
   if Count_Components (SEARCH_COMPONENT => MUTUAL_COMPONENT,
      NETLIST_OBJECT => NETLIST_OBJECT) /= 0 then
      raise CANNOT_CONVERT_WITH_MUTUAL_INDUCTANCES;
   end if;
   while POINTER /= null loop
      TEMP_ADMITTANCE := POINTER.DATA.GENERAL.IMAG_ADMITTANCE;
      if TEMP_ADMITTANCE > 0.0 then  -- capacitive admittance
         if abs(POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR)>MINIMUM_VALUE then
            TEMP_ADMITTANCE := TEMP_ADMITTANCE - 1.0 /
               (OMEGA * POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR);
         end if;
         POINTER.DATA.GENERAL.CAPACITOR := TEMP_ADMITTANCE / OMEGA;
      elsif TEMP_ADMITTANCE < 0.0 then  -- inductive admittance
         DENOMINATOR := OMEGA * POINTER.DATA.GENERAL.CAPACITOR -
            TEMP_ADMITTANCE;
         if abs (DENOMINATOR) > MINIMUM_VALUE then
            POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR :=
               1.0 / (OMEGA * DENOMINATOR);
            POINTER.DATA.INDUCTIVE.HIGHER_NODE :=
               POINTER.DATA.GENERAL.HIGHER_NODE;
            POINTER.DATA.INDUCTIVE.LOWER_NODE  :=
               POINTER.DATA.GENERAL.LOWER_NODE;
         else
            POINTER.DATA.INDUCTIVE.ACTUAL_INDUCTOR := 0.0;
            POINTER.DATA.INDUCTIVE.HIGHER_NODE := 0;
            POINTER.DATA.INDUCTIVE.LOWER_NODE  := 0;
         end if;
         BR_Inductance_Matrix (NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
      if (abs (POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE) +
          abs (POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE))
          > MINIMUM_VCCS then
         if abs (POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE)
            > MINIMUM_CI_ADMITTANCE then
            RATIO := -POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE /
               POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
            POINTER.DATA.DEPEND_SOURCE.POLE_FREQUENCY :=
               FREQUENCY / RATIO;
            POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE :=
               POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE /
               (1.0 + RATIO*RATIO);
         else
            POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE :=
               POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
         end if;
      end if;
      POINTER := POINTER.NEXT;
   end loop;
end Convert_Admittance_To_RLC;

-- PURPOSE : Convert the representation of components from RLC to
--           admittance form in the whole netlist. The LIST pointer in
--           NETLIST_OBJECT is not affected.

procedure Convert_RLC_To_Admittance (FREQUENCY : in REAL_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   POINTER : LINK      := NETLIST_OBJECT.HEAD;
   OMEGA   : REAL_TYPE := 2.0 * PI * FREQUENCY;

begin
   while POINTER /= null loop
      POINTER.DATA.GENERAL.IMAG_ADMITTANCE :=
         OMEGA * POINTER.DATA.GENERAL.CAPACITOR;
      if abs (POINTER.DATA.GENERAL.EQUIV_INDUCTOR) > MINIMUM_VALUE then
         POINTER.DATA.GENERAL.IMAG_ADMITTANCE :=
            POINTER.DATA.GENERAL.IMAG_ADMITTANCE
            - 1.0 / (OMEGA * POINTER.DATA.GENERAL.EQUIV_INDUCTOR);
      end if;
      if POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE /= 0 then
         POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE :=
            POINTER.DATA.DEPEND_SOURCE.GAIN_ADMITTANCE;
         POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE := 0.0;
         if POINTER.DATA.DEPEND_SOURCE.POLE_FREQUENCY > MINIMUM_VALUE then
            POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE :=
               POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE / (1.0 +
               (FREQUENCY/POINTER.DATA.DEPEND_SOURCE.POLE_FREQUENCY)**2);
            POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE :=
               - (FREQUENCY / POINTER.DATA.DEPEND_SOURCE.POLE_FREQUENCY)
               * POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
         end if;
      end if;
      POINTER := POINTER.NEXT;
   end loop;
end Convert_RLC_To_Admittance;

-- PURPOSE : Counts the number of a specified component type in the whole
--           netlist. The LIST pointer in the NETLIST_OBJECT is not
--           affected.

function Count_Components (SEARCH_COMPONENT : in SEARCH_COMPONENT_TYPE;
   NETLIST_OBJECT : in NETLIST_TYPE) return NATURAL is

   COUNTER : NATURAL := 0;
   POINTER : LINK    := NETLIST_OBJECT.HEAD;
```

```
   HIGH_NODE : NODE_TYPE;

begin
   while POINTER /= null loop
      case SEARCH_COMPONENT is
         when GENERAL_COMPONENT =>
            HIGH_NODE := POINTER.DATA.GENERAL.HIGHER_NODE;
         when INDUCTIVE_COMPONENT =>
            HIGH_NODE := POINTER.DATA.INDUCTIVE.HIGHER_NODE;
         when MUTUAL_COMPONENT =>
            HIGH_NODE := POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE1;
      end case;
      if HIGH_NODE /= 0 then
         COUNTER := COUNTER + 1;
      end if;
      POINTER := POINTER.NEXT;
   end loop;
   return COUNTER;
end Count_Components;

-- ------------------------------------------------------------------
-- PURPOSE : Deletes an branch.  The LIST pointer in the NETLIST_OBJECT
--           that is passed in points to the branch to be deleted.  On
--           returning, the LIST pointer points to the next branch.
-- ------------------------------------------------------------------

procedure Delete_Branch (NETLIST_OBJECT : in out NETLIST_TYPE) is

   POINTER : LINK;

begin
   if NETLIST_OBJECT.HEAD /= null then
      if NETLIST_OBJECT.LIST = NETLIST_OBJECT.HEAD then
         NETLIST_OBJECT.HEAD := NETLIST_OBJECT.HEAD.NEXT;
         Dispose (NETLIST_OBJECT.LIST);
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
      else
         POINTER := NETLIST_OBJECT.HEAD;
         while (POINTER.NEXT /= NETLIST_OBJECT.LIST) and
               (POINTER.NEXT /= null) loop
            POINTER := POINTER.NEXT;
         end loop;
         if POINTER.NEXT /= null then
            POINTER.NEXT := POINTER.NEXT.NEXT;
            Dispose (NETLIST_OBJECT.LIST);
            NETLIST_OBJECT.LIST := POINTER.NEXT;
         end if;
      end if;
   end if;
end Delete_Branch;

-- ------------------------------------------------------------------
-- PURPOSE : Checks to see if a branch is empty.  The LIST pointer in the
--           NETLIST_OBJECT in the branch that is checked.
-- EXCEPTIONS : BRANCH_OUT_OF_RANGE
-- ------------------------------------------------------------------

function Empty_Branch (NETLIST_OBJECT : in NETLIST_TYPE) return BOOLEAN is

   COMPONENT : ELEMENTS;
   VALUE     : REAL_TYPE;

begin
   if NETLIST_OBJECT.LIST = null then
      raise BRANCH_OUT_OF_RANGE;
   end if;
   for COMPONENT in CAPS..COUPLING loop
      case COMPONENT is
         when CAPS =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.CAPACITOR;
         when INDUCT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.EQUIV_INDUCTOR;
         when GR_ADMIT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_ADMITTANCE;
         when GI_ADMIT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_ADMITTANCE;
         when REAL_VOLT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_VOLTAGE;
         when IMAG_VOLT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_VOLTAGE;
         when REAL_CURR =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_CURRENT;
         when IMAG_CURR =>
            VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_CURRENT;
         when ACT_INDCT =>
            VALUE := NETLIST_OBJECT.LIST.DATA.INDUCTIVE.ACTUAL_INDUCTOR;
         when GR_ADMIT =>
            VALUE:=NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
         when GI_ADMIT =>
            VALUE:=NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE;
         when GN_ADMIT =>
            VALUE :=NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.MAG_ADMITTANCE;
         when POLE_FREQ =>
            VALUE :=NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.POLE_FREQUENCY;
         when MUT_INDCT =>
            VALUE:=NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR;
         when COUPLING =>
            VALUE := NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.COUPLING;
      end case;
      if (abs (VALUE) > MINIMUM_VALUE) then
         return FALSE;
         exit;
      end if;
   end loop;
   return TRUE;
end Empty_Branch;

-- ------------------------------------------------------------------
-- PURPOSE : Checks to see if the present branch pointed to by LIST is the
--           same branch pointed to by the pointer indexed by
--           LOCATION_INDEX of the LOCATION array in the NETLIST_OBJECT.
-- ------------------------------------------------------------------

function Equal_Locations (LOCATION_INDEX : in POSITIVE;
   NETLIST_OBJECT : in NETLIST_TYPE) return BOOLEAN is

begin
   if NETLIST_OBJECT.LIST = NETLIST_OBJECT.LOCATION (LOCATION_INDEX) then
      return TRUE;
   else
      return FALSE;
   end if;
end Equal_Locations;

-- ------------------------------------------------------------------
-- PURPOSE : Locate the branch between a specified node pair in the
--           netlist.  If the branch cannot be found, a new branch is
--           created. The LIST pointer in the NETLIST_OBJECT points to the
--           found branch.
-- EXCEPTIONS : MEANINGLESS_COMPONENT_CONNECTION
--              NODE_PAIR_DOES_NOT_EXIST
--              STORAGE_ERROR
-- ------------------------------------------------------------------

procedure Find_Branch (COMPONENT : in ELEMENTS;
   HIGH_NODE, LOW_NODE : in out NODE_TYPE;
   NETLIST_OBJECT      : in out NETLIST_TYPE) is

   FOUND : BOOLEAN := FALSE;

begin
   if HIGH_NODE = LOW_NODE then
      raise MEANINGLESS_COMPONENT_CONNECTION;
   end if;
   if HIGH_NODE < LOW_NODE then
      Maths.Swap_Node (NODE1 => HIGH_NODE, NODE2 => LOW_NODE);
   end if;
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while (NETLIST_OBJECT.LIST /= null) and not FOUND loop
      if (HIGH_NODE = NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE) and
         (LOW_NODE  = NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE) then
         FOUND := TRUE;
      else
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
      end if;
   end loop;
   if not FOUND then
      if COMPONENT >= CR_ADMIT then
         raise NODE_PAIR_DOES_NOT_EXIST;
      else
         New_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
   end if;
end Find_Branch;

-- ------------------------------------------------------------------
-- PURPOSE : Locates the branch that contains a dependant source given the
--           nodes related to the dependant source in the netlist.  The
--           LIST pointer in the NETLIST_OBJECT points to the found
--           branch. If the dependant source cannot be found, a new branch
--           is created if CREATE is TRUE, and the FOUND_SOURCE parameter
--           returns a FALSE value.
-- EXCEPTIONS : STORAGE_ERROR
-- ------------------------------------------------------------------

procedure Find_Dependent_Source (HIGH_NODE1, LOW_NODE1, HIGH_NODE2,
   LOW_NODE2     : in     NODE_TYPE;
   CREATE        : in     BOOLEAN := TRUE;
   FOUND_SOURCE  :    out BOOLEAN;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   FOUND : BOOLEAN := FALSE;

begin
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while (NETLIST_OBJECT.LIST /= null) loop
      if (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE = HIGH_NODE1)
         and (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE =
         LOW_NODE1) and
         (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE =
         HIGH_NODE1) and
         (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE =
         LOW_NODE1) then
         FOUND := TRUE;
         exit;
      else
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
      end if;
   end loop;
   if not FOUND then
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
      while (NETLIST_OBJECT.LIST /= null) loop
         if (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE = 0) and
            (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE = 0) and
            (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE=0)
            and (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE
            = 0) then
            FOUND := TRUE;
            exit;
         else
            NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
         end if;
      end loop;
   end if;
   if not FOUND then
      if CREATE then
         New_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
      FOUND_SOURCE := FALSE;
   else
      FOUND_SOURCE := TRUE;
   end if;
end Find_Dependent_Source;

-- ------------------------------------------------------------------
-- PURPOSE : Find the mutuals and coupling factors in the whole netlist.
--           The LIST pointer in the NETLIST_OBJECT is preserved.  If the
--           component cannot be found, a new branch will be created if
--           CREATE is TRUE.
-- EXCEPTIONS : STORAGE_ERROR
-- ------------------------------------------------------------------

procedure Find_Mutuals_and_Coupling (COMPONENT : in ELEMENTS;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
   MUTUAL_VALUE, VALUE : in     REAL_TYPE;
   CREATE              : in     BOOLEAN := TRUE;
   NETLIST_OBJECT      : in out NETLIST_TYPE) is

   MULTIPLIER : constant := 0.999_999;
```

```
   FOUND      : BOOLEAN   := FALSE;
   POINTER    : LINK      := NETLIST_OBJECT.LIST;
   TEMP_VALUE : REAL_TYPE := VALUE;

begin
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while (NETLIST_OBJECT.LIST /= null) loop
      if ((NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE1=HIGH_NODE1)
         and (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE1 =
         LOW_NODE1) and
         (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE2 =
         HIGH_NODE2) and
         (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE2 =
         LOW_NODE2)) then
         FOUND := TRUE;
         exit;
      else
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
      end if;
   end loop;
   if not FOUND then
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
      while (NETLIST_OBJECT.LIST /= null) loop
         if (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE1 = 0) and
            (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE1 = 0) and
            (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE2 = 0) and
            (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE2 = 0) then
            FOUND := TRUE;
            exit;
         else
            NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
         end if;
      end loop;
   end if;

   if not FOUND and CREATE then
      New_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
   end if;

   if CREATE then
      case COMPONENT is
         when MUT_INDCT =>
            if (MULTIPLIER * MUTUAL_VALUE - abs (VALUE)) < 0.0 then
               if VALUE < 0.0 then
                  TEMP_VALUE := - MULTIPLIER * MUTUAL_VALUE;
               else
                  TEMP_VALUE := MULTIPLIER * MUTUAL_VALUE;
               end if;
            end if;
            Write_Netlist (COMPONENT => MUT_INDCT,
               VALUE          => TEMP_VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            TEMP_VALUE := TEMP_VALUE / MUTUAL_VALUE;
            Write_Netlist (COMPONENT => COUPLING,
               VALUE          => TEMP_VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         when COUPLING =>
            if (abs (VALUE) - MULTIPLIER) > 0.0 then
               if VALUE < 0.0 then
                  TEMP_VALUE := - MULTIPLIER;
               else
                  TEMP_VALUE := MULTIPLIER;
               end if;
            end if;
            Write_Netlist (COMPONENT => COUPLING,
               VALUE          => TEMP_VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            TEMP_VALUE := TEMP_VALUE * MUTUAL_VALUE;
            Write_Netlist (COMPONENT => MUT_INDCT,
               VALUE          => TEMP_VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         when others => null;
      end case;
   end if;
   NETLIST_OBJECT.LIST := POINTER;
exception
   when STORAGE_ERROR =>
      NETLIST_OBJECT.LIST := POINTER;
      raise STORAGE_ERROR;
end Find_Mutuals_and_Coupling;

-------------------------------------------------------------------------
-- PURPOSE : Finds the highest numbered node in the whole netlist. The
--           LIST pointer in the NETLIST_OBJECT is not affected.
-------------------------------------------------------------------------

procedure Find_Node_Maximum (NODE_MAXIMUM : out NODE_TYPE;
   NETLIST_OBJECT : in out N_LIST_TYPE) is

   POINTER : LINK := NETLIST_OBJECT.HEAD;

begin
   if NETLIST_OBJECT.HEAD = null then
      NETLIST_OBJECT.NODE_MAXIMUM := 0;
   else
      NETLIST_OBJECT.NODE_MAXIMUM := POINTER.DATA.GENERAL.HIGHER_NODE;
      while POINTER /= null loop
         if POINTER.DATA.GENERAL.HIGHER_NODE >
            NETLIST_OBJECT.NODE_MAXIMUM then
            NETLIST_OBJECT.NODE_MAXIMUM:=POINTER.DATA.GENERAL.HIGHER_NODE;
         end if;
         POINTER := POINTER.NEXT;
      end loop;
   end if;
   NODE_MAXIMUM := NETLIST_OBJECT.NODE_MAXIMUM;
end Find_Node_Maximum;

-------------------------------------------------------------------------
-- PURPOSE : Sets the list pointer in the NETLIST_OBJECT to the first
--           branch in the netlist.
-- EXCEPTIONS : NO_CIRCUIT
-------------------------------------------------------------------------

procedure First_Branch (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   if NETLIST_OBJECT.LIST = null then
      raise NO_CIRCUIT;
   end if;
end First_Branch;

-------------------------------------------------------------------------
-- PURPOSE : Initializes the whole netlist and destroy a network if there
--           was one.
-------------------------------------------------------------------------

procedure Initialize (NETLIST_OBJECT : in out NETLIST_TYPE) is

   POINTER : LINK;

begin
   if NETLIST_OBJECT.NODE_MAXIMUM > 0 then
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
      while NETLIST_OBJECT.LIST /= null loop
         POINTER := NETLIST_OBJECT.LIST.NEXT;
         Dispose (NETLIST_OBJECT.LIST);
         NETLIST_OBJECT.LIST := POINTER;
      end loop;
   end if;
   NETLIST_OBJECT := (LIST => null,
      HEAD        => null,
      LOCATION    => (1..MAX_OPTIMUM_VARIABLE => null),
      NODE_MAXIMUM => 0,
      FREQUENCY   => 1000.0,
      TITLE       => (1..64 => ' '));
end Initialize;

-------------------------------------------------------------------------
-- PURPOSE : Creates a new branch and puts it at the head of the netlist.
-------------------------------------------------------------------------

procedure New_Branch (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LIST := new NETLIST;
   NETLIST_OBJECT.LIST.NEXT := NETLIST_OBJECT.HEAD;
   NETLIST_OBJECT.HEAD := NETLIST_OBJECT.LIST;
end New_Branch;

-------------------------------------------------------------------------
-- PURPOSE : Move the LIST pointer in the NETLIST_OBJECT to the next
--           branch in the netlist.
-- EXCEPTIONS : NO_MORE_BRANCHES
-------------------------------------------------------------------------

procedure Next_Branch (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   if NETLIST_OBJECT.LIST.NEXT = null then
      raise NO_MORE_BRANCHES;
   else
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
   end if;
end Next_Branch;

-------------------------------------------------------------------------
-- PURPOSE : Read the value of a component as well as the nodes to which
--           it is connected.  The component data is read from the branch
--           pointed to by LIST in the NETLIST_OBJECT.
-- EXCEPTIONS : BRANCH_OUT_OF_RANGE
-------------------------------------------------------------------------

procedure Read_Netlist (COMPONENT : in ELEMENTS;
   VALUE : out REAL_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : out NODE_TYPE;
   NETLIST_OBJECT : in NETLIST_TYPE) is

begin
   if NETLIST_OBJECT.LIST = null then
      raise BRANCH_OUT_OF_RANGE;
   end if;
   case COMPONENT is
      when CAPS =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.CAPACITOR;
      when INDUCT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.EQUIV_INDUCTOR;
      when GR_ADMIT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_ADMITTANCE;
      when GI_ADMIT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_ADMITTANCE;
      when REAL_VOLT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_VOLTAGE;
      when IMAG_VOLT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_VOLTAGE;
      when REAL_CRNT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_CURRENT;
      when IMAG_CRNT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_CURRENT;
      when ACT_INDCT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.INDUCTIVE.ACTUAL_INDUCTOR;
      when CR_ADMIT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
      when CI_ADMIT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE;
      when CM_ADMIT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.MAG_ADMITTANCE;
      when POLE_FREQ =>
         VALUE := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.POLE_FREQUENCY;
      when MUT_INDCT =>
         VALUE := NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR;
      when COUPLING =>
         VALUE := NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.COUPLING;
   end case;
```

```
   case COMPONENT is
      when CAPS..IMAG_CRNT =>
         HIGH_NODE1 := NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE;
         LOW_NODE1  := NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE;
         HIGH_NODE2 := 0;
         LOW_NODE2  := 0;
      when ACT_INDCT =>
         HIGH_NODE1 := NETLIST_OBJECT.LIST.DATA.INDUCTIVE.HIGHER_NODE;
         LOW_NODE1  := NETLIST_OBJECT.LIST.DATA.INDUCTIVE.LOWER_NODE;
         HIGH_NODE2 := 0;
         LOW_NODE2  := 0;
      when CR_ADMIT..POLE_FREQ =>
         HIGH_NODE1 := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE;
         LOW_NODE1  := NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE;
         HIGH_NODE2 :=
            NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE;
         LOW_NODE2  :=
            NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE;
      when MUT_INDCT | COUPLING =>
         HIGH_NODE1 :=NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE1;
         LOW_NODE1  :=NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE1;
         HIGH_NODE2 :=NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE2;
         LOW_NODE2  :=NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE2;
      when others => null;
   end case;
end Read_Netlist;

-- PURPOSE : Sets the present branch pointed to by LIST to that pointed to
--           by the pointer indexed by LOCATION_INDEX of the LOCATION
--           array in the NETLIST_OBJECT.

procedure Retrieve_Location (LOCATION_INDEX : in POSITIVE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.LOCATION (LOCATION_INDEX);
end Retrieve_Location;

-- PURPOSE : Sets the pointer indexed by LOCATION_INDEX of the LOCATION
--           array in the NETLIST_OBJECT to the present branch pointed to
--           by LIST.

procedure Save_Location (LOCATION_INDEX : in POSITIVE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.LOCATION (LOCATION_INDEX) := NETLIST_OBJECT.LIST;
end Save_Location;

-- PURPOSE : Updates the mutuals and coupling factors in the whole
--           netlist. The LIST pointer in the NETLIST_OBJECT is preserved.
-- EXCEPTIONS : BRANCH_OUT_OF_RANGE
--              MEANINGLESS_COMPONENT
--              MEANINGLESS_COMPONENT_CONNECTION

procedure Update_Mutuals_and_Coupling (VALUE : in REAL_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   MUTUAL_VALUE    : REAL_TYPE;
   POINTER         : LINK      := NETLIST_OBJECT.LIST;
   TEMP_HIGH_NODE1 : NODE_TYPE := HIGH_NODE1;
   TEMP_LOW_NODE1  : NODE_TYPE := LOW_NODE1;
   TEMP_HIGH_NODE2 : NODE_TYPE := HIGH_NODE2;
   TEMP_LOW_NODE2  : NODE_TYPE := LOW_NODE2;
   TEMP_VALUE      : REAL_TYPE := VALUE;

begin
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while NETLIST_OBJECT.LIST /= null loop
      if (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE1 =
         TEMP_HIGH_NODE1) and
         (NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE1  =
         TEMP_LOW_NODE1) then
         if TEMP_VALUE > MINIMUM_VALUE then
            TEMP_HIGH_NODE2 :=
               NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE2;
            TEMP_LOW_NODE2 :=
               NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE2;
            Calculate_Mutual_Inductance (HIGH_NODE1 => TEMP_HIGH_NODE1,
               LOW_NODE1           => TEMP_LOW_NODE1,
               HIGH_NODE2          => TEMP_HIGH_NODE2,
               LOW_NODE2           => TEMP_LOW_NODE2,
               MUTUAL_INDUCT_VALUE => MUTUAL_VALUE,
               NETLIST_OBJECT      => NETLIST_OBJECT);
            TEMP_VALUE := NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.COUPLING
               * MUTUAL_VALUE;
         else
            TEMP_VALUE := 0.0;
            TEMP_HIGH_NODE2 := 0;
            TEMP_LOW_NODE2  := 0;
            Write_Netlist (COMPONENT => COUPLING,
               VALUE       => TEMP_VALUE,
               HIGH_NODE1  => TEMP_HIGH_NODE1,
               LOW_NODE1   => TEMP_LOW_NODE1,
               HIGH_NODE2  => TEMP_HIGH_NODE2,
               LOW_NODE2   => TEMP_LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         end if;
         Write_Netlist (COMPONENT => MUT_INDCT,
            VALUE       => TEMP_VALUE,
            HIGH_NODE1  => TEMP_HIGH_NODE1,
            LOW_NODE1   => TEMP_LOW_NODE1,
            HIGH_NODE2  => TEMP_HIGH_NODE2,
            LOW_NODE2   => TEMP_LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
   end loop;
   NETLIST_OBJECT.LIST := POINTER;
end Update_Mutuals_and_Coupling;

-- PURPOSE : Changes the present frequency in the NETLIST_OBJECT to
--           FREQUENCY.

procedure Write_Frequency (FREQUENCY : in REAL_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NETLIST_OBJECT.FREQUENCY := FREQUENCY;
end Write_Frequency;

-- PURPOSE : Write the value of a component as well as the nodes to
--           which it is connected. The branch into which this data is
--           written is pointed to by LIST in the NETLIST_OBJECT.
-- EXCEPTIONS : BRANCH_OUT_OF_RANGE

procedure Write_Netlist (COMPONENT : in ELEMENTS;
   VALUE : in REAL_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   COMP_VALUE : REAL_TYPE;

begin
   if NETLIST_OBJECT.LIST = null then
      raise BRANCH_OUT_OF_RANGE;
   elsif ABS (VALUE) < MINIMUM_VALUE then
      COMP_VALUE := 0.0;
   else
      COMP_VALUE := VALUE;
   end if;
   case COMPONENT is
      when CAPS      =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.CAPACITOR := COMP_VALUE;
      when INDUCT    =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.EQUIV_INDUCTOR := COMP_VALUE;
      when CR_ADMIT  =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_ADMITTANCE := COMP_VALUE;
      when CI_ADMIT  =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_ADMITTANCE := COMP_VALUE;
      when REAL_VOLT =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_VOLTAGE := COMP_VALUE;
      when IMAG_VOLT =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_VOLTAGE := COMP_VALUE;
      when REAL_CRNT =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_CURRENT := COMP_VALUE;
      when IMAG_CRNT =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_CURRENT := COMP_VALUE;
      when ACT_INDCT =>
         NETLIST_OBJECT.LIST.DATA.INDUCTIVE.ACTUAL_INDUCTOR := COMP_VALUE;
      when CR_ADMIT  =>
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.REAL_ADMITTANCE :=
            COMP_VALUE;
      when CI_ADMIT  =>
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE :=
            COMP_VALUE;
      when CR_ADMIT  =>
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.MAG_ADMITTANCE :=
            COMP_VALUE;
      when POLE_FREQ =>
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.POLE_FREQUENCY :=
            COMP_VALUE;
      when MUT_INDCT =>
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR :=
            COMP_VALUE;
      when COUPLING  =>
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.COUPLING := COMP_VALUE;
   end case;

   case COMPONENT is
      when CAPS..IMAG_CRNT =>
         NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE := HIGH_NODE1;
         NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE  := LOW_NODE1;
      when ACT_INDCT =>
         NETLIST_OBJECT.LIST.DATA.INDUCTIVE.HIGHER_NODE := HIGH_NODE1;
         NETLIST_OBJECT.LIST.DATA.INDUCTIVE.LOWER_NODE  := LOW_NODE1;
```

```
      when CR_ADMIT..POLE_FREQ =>
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE := HIGH_NODE1;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE  := LOW_NODE1;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE :=
            HIGH_NODE2;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE  :=
            LOW_NODE2;
      when MUT_INDCT | COUPLING =>
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE1 :=HIGH_NODE1;
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE1  :=LOW_NODE1;
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.HIGHER_NODE2 :=HIGH_NODE2;
         NETLIST_OBJECT.LIST.DATA.MUTUAL_INDUCT.LOWER_NODE2  :=LOW_NODE2;
      when others => null;
   end case;
   if (COMP_VALUE = 0.0) and
      (Empty_Branch (NETLIST_OBJECT => NETLIST_OBJECT)) then
      Delete_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
   elsif HIGH_NODE1 > NETLIST_OBJECT.NODE_MAXIMUM then
      NETLIST_OBJECT.NODE_MAXIMUM := HIGH_NODE1;
   end if;
end Write_Netlist;

-------------------------------------------------------------------------------

end Inner;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package body Outer is separate;

-------------------------------------------------------------------------------

end Netlists;

-------------------------------------------------------------------------------


with  Screens;
use   Screens;

separate (Netlists)

package body Screen_Supporter is

   use Main;

-------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------------

   HP_GET_COMPONENT : constant := 110;

   COMPONENT_LIST : LIST_ARRAY_TYPE (1..15) := (
   (TEXT_LENGTH => 8,  TEXT => "Resistor",                  STATE => TRUE),
   (TEXT_LENGTH => 8,  TEXT => "Inductor",                  STATE => FALSE),
   (TEXT_LENGTH => 9,  TEXT => "Capacitor",                 STATE => FALSE),
   (TEXT_LENGTH => 14, TEXT => "Voltage Source",            STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "Real Voltage Source",       STATE => FALSE),
   (TEXT_LENGTH => 24, TEXT => "Imaginary Voltage Source",STATE => FALSE),
   (TEXT_LENGTH => 17, TEXT => "Vc Current Source",         STATE => FALSE),
   (TEXT_LENGTH => 23, TEXT => "-3 dB Frequency of Vccs",  STATE => FALSE),
   (TEXT_LENGTH => 14, TEXT => "Current Source",            STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "Real Current Source",       STATE => FALSE),
   (TEXT_LENGTH => 24, TEXT => "Imaginary Current Source",STATE => FALSE),
   (TEXT_LENGTH => 15, TEXT => "Real Admittand ",           STATE => FALSE),
   (TEXT_LENGTH => 20, TEXT => "Imaginary Admittance",      STATE => FALSE),
   (TEXT_LENGTH => 17, TEXT => "Mutual Inductance",         STATE => FALSE),
   (TEXT_LENGTH => 15, TEXT => "Coupling Factor",           STATE =>FALSE));

-------------------------------------------------------------------------------
-- PURPOSE : Displays the component and it's data on the main screen.
-------------------------------------------------------------------------------

procedure Display_Component (COMPONENT : in ELEMENT_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
   VALUE, PHASE  : in REAL_TYPE;
   ROW           : in out INTEGER_TYPE) is

   NAME : STRING (1..4) := "    ";

begin
   if COMPONENT <= RESISTOR then
      NAME := COMPONENT_VALUE (COMPONENT).DISPLAY;
   end if;
   Cursor_Next_Line;
   Cursor_Set (ROW => ROW, COLUMN => 5);
   ROW := ROW + 1;
   Write (ITEM => NAME);
   Write (ITEM => HIGH_NODE1,  ITEM_WIDTH => 6);
   Write (ITEM => LOW_NODE1,   ITEM_WIDTH => 3);
   Write (ITEM => VALUE,
      WHOLE        => 5,
      FRACTION     => 4,
      ENG_NOTATION => TRUE);
   Write (ITEM => "   " & COMPONENT_VALUE (COMPONENT).UNITS);
   if COMPONENT in CR_ADMIT..CUPLING then
      if COMPONENT in CR_ADMIT..POLE_FREQ then
         Write (ITEM => "   Controlling");
      else
         Write (ITEM => "   Coupled");
      end if;
      Write (ITEM => " nodes ");
      Write (ITEM => HIGH_NODE2,  ITEM_WIDTH => 3);
      Write (ITEM => LOW_NODE2,   ITEM_WIDTH => 3);
   end if;
   if COMPONENT in CRNT_SOURCE..VOLT_SOURCE then
      Write (ITEM => "   Phase =");
      Write (ITEM => PHASE,
         WHOLE     => 4,
         FRACTION  => 3,
         EXPONENT  => 0);
      Write (ITEM => " Degrees");
   end if;
end Display_Component;

-------------------------------------------------------------------------------
-- PURPOSE : Displays a component list on the options submenu and gets a
--           component from the user.
-------------------------------------------------------------------------------

procedure Get_Component (COMPONENT : out ELEMENT_TYPE) is

      ITEM : INTEGER_TYPE;

begin
   Prompts.Read (LIST => COMPONENT_LIST,
      HELP_INFO => HP_GET_COMPONENT);
   for COUNTER in 1..15 loop
      if COMPONENT_LIST (COUNTER).STATE then
         case COUNTER is
            when 1  => COMPONENT := RESISTOR;
            when 2  => COMPONENT := INDUCT;
            when 3  => COMPONENT := CAPS;
            when 4  => COMPONENT := VOLT_SOURCE;
            when 5  => COMPONENT := REAL_VOLT;
            when 6  => COMPONENT := IMAG_VOLT;
            when 7  => COMPONENT := CW_ADMIT;
            when 8  => COMPONENT := POLE_FREQ;
            when 9  => COMPONENT := CRNT_SOURCE;
            when 10 => COMPONENT := REAL_CRNT;
            when 11 => COMPONENT := IMAG_CRNT;
            when 12 => COMPONENT := CR_ADMIT;
            when 13 => COMPONENT := CI_ADMIT;
            when 14 => COMPONENT := MUT_INDCT;
            when 15 => COMPONENT := CUPLING;
         end case;
         exit;
      end if;
   end loop;
end Get_Component;

-------------------------------------------------------------------------------

end Screen_Supporter;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------


with  DOS_Environment,
      Screens,
      Sequential_IO;

use   Screens;

separate (Netlists)

package body Outer is

   use Main;

-------------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------------

   type LOAD_SAVE_RECORD (LOAD : BOOLEAN := TRUE) is
      record
         case LOAD is
            when TRUE =>
               TITLE     : STRING (1..64);
               FREQUENCY : REAL_TYPE;
            when FALSE =>
               COMPS     : COMPONENTS;
         end case;
      end record;

   package Netlist_IO is new Sequential_IO (LOAD_SAVE_RECORD);

   HP_GET_FREQUENCY     : constant := 120;
   HP_GET_FREQUENCY2    : constant := 121;
   HP_LOAD_NETLIST      : constant := 122;
   HP_LOAD_NETLIST2     : constant := 123;
   HP_SAVE_NETLIST      : constant := 124;
   HP_SAVE_NETLIST2     : constant := 125;

   SHOW_FORK : FIELDS_TYPE (1..2) :=
      (1 =>
         (VALUE_LENGTH    => 2,
         TEXT_LENGTH      => 16,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING        => TRUE,
         TEST_AGAINST     => 2,
         ACTUAL_LENGTH    => 1,
         ROW              => 4,
         VALUE            => "1 ",
         TEXT             => "Higher Show Node",
         MAX_INTEGER      => MAX_NODES,
         MIN_INTEGER      => 1,
         INTEGER_VALUE    => 1),
      2 =>
         (VALUE_LENGTH    => 2,
         TEXT_LENGTH      => 16,
         IS_INTEGER_TYPE => TRUE,
         ACCEPTING        => TRUE,
         TEST_AGAINST     => 1,
         ACTUAL_LENGTH    => 1,
```

```
         ROW            => 5,
         VALUE          => "0 ",
         TEXT           => "Power  Show Mode",
         MAX_INTEGER    => MAX_NODES,
         MIN_INTEGER    => 0,
         INTEGER_VALUE  => 0));

   DEVICE_ERROR_MESSAGE : constant STRING := "Device Error.";
   NAME_ERROR_MESSAGE   : constant STRING := "Name Error.";

   PATH : Dos_Environment.PATH_STRING_TYPE;

-------------------------------------------------------------------------
-- PURPOSE : Changes the default directory and shows the new path.
-------------------------------------------------------------------------

procedure Change_Directory is

   LENGTHS : INTEGER_TYPE;

begin
   Process_Title (TITLE => "CHANGE DRIVE");
   Cursor_Set (ROW => 3, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
   Cursor_Next_Line;
   Prompts.Read (MESSAGE => "Enter the new path.",
      ITEM        => PATH,
      ITEM_LENGTH => LENGTHS);
   PATH (LENGTHS + 1) := CHARACTER'val (0);
   Dos_Environment.Change_Directory (PATH_STRING => PATH);
   Cursor_Set (ROW => 3, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
end Change_Directory;

-------------------------------------------------------------------------
-- PURPOSE : Changes the present drive device and shows the new path.
-------------------------------------------------------------------------

procedure Change_Drive is

   LENGTHS : INTEGER_TYPE;

begin
   Process_Title (TITLE => "CHANGE DRIVE");
   Cursor_Set (ROW => 3, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
   Cursor_Next_Line;
   Prompts.Read (MESSAGE => "Enter the new drive letter.",
      ITEM        => PATH,
      ITEM_LENGTH => LENGTHS);
   Dos_Environment.Change_Drive (PATH_STRING => PATH,
      DRIVE => PATH (1));
   Cursor_Set (ROW => 5, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
end Change_Drive;

-------------------------------------------------------------------------
-- PURPOSE : Displays the present frequency on the Prompts screen and then
--           gets a new frequency if the present one is not wanted.
-------------------------------------------------------------------------

procedure Get_Frequency (FREQUENCY : out REAL_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   MIN_FREQUENCY : constant := 1.0E-4;
   MAX_FREQUENCY : constant := 1.0E12;

   DISPLAY_LINE  : Texts.TEXT_TYPE;
   ACCEPTING     : BOOLEAN;

begin
   Texts.Combine (ITEM => "Do you accept the present frequency :",
      TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => NETLIST_OBJECT.FREQUENCY,
      FRACTION  => 5,
      TEXT_LINE => DISPLAY_LINE);
   Prompts.Read (MESSAGE => DISPLAY_LINE.TEXT(1..DISPLAY_LINE.TEXT_INDEX),
      HELP_INFO => HP_GET_FREQUENCY,
      ITEM      => ACCEPTING);
   if not ACCEPTING then
      Prompts.Read (MESSAGE => "Enter the new frequency in Hz.",
         HELP_INFO => HP_GET_FREQUENCY2,
         ITEM      => NETLIST_OBJECT.FREQUENCY);
      Maths.Check_Value (MINIMUM => MIN_FREQUENCY,
         MAXIMUM => MAX_FREQUENCY,
         VALUE   => NETLIST_OBJECT.FREQUENCY);
   end if;
   FREQUENCY := NETLIST_OBJECT.FREQUENCY;
end Get_Frequency;

-------------------------------------------------------------------------
-- PURPOSE : Gets the path.
-------------------------------------------------------------------------

procedure Initialise_Directory is

begin
   Dos_Environment.Get_Drive_Directory (PATH_STRING => PATH);
end Initialise_Directory;

-------------------------------------------------------------------------
-- PURPOSE : Lists the components in a branch of the NETLIST_OBJECT on the
--           main screen.
-------------------------------------------------------------------------

procedure List_Branch (LIST_NODE : in CIRCUIT_NODE_TYPE;
   FORM          : in ELEMENT_TYPE;
   ROW           : in out POSITIVE;
   NETLIST_OBJECT : in NETLIST_TYPE) is

   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2          : NODE_TYPE;
   COMPONENT,           SHOW_COMPONENT     : ELEMENT_TYPE;
   GOT_COMPONENT,       LIST_COMPONENT     : BOOLEAN;
   MAGNITUDE,           PHASE,
   TEMP_VALUE,          VALUE              : REAL_TYPE := 0.0;

begin
   for COMPONENT in CAPS..CUPLING loop
      case COMPONENT is
         when CAPS | ACT_INDCT | CH_ADMIT | POLE_FREQ =>
            LIST_COMPONENT := (LIST_NODE = REG);
         when GI_ADMIT | GR_ADMIT | GI_ADMIT =>
            LIST_COMPONENT := (LIST_NODE = ADMITTANCE);
         when others => null;
      end case;
      Inner.Read_Netlist (COMPONENT => COMPONENT,
         VALUE          => VALUE,
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      if abs (VALUE) > MINIMUM_VALUE then
         GOT_COMPONENT := TRUE;
      else
         GOT_COMPONENT := FALSE;
      end if;
   exception
      when BRANCH_OUT_OF_RANGE => GOT_COMPONENT := FALSE;
   end;
   if GOT_COMPONENT or (COMPONENT = IMAG_VOLT) or
      (COMPONENT = IMAG_CENT) then
      if FORM = EQ_INDCT_FORM then
         COMPONENT_VALUE (INDUCT).DISPLAY := "Leqv";
      end if;
      if COMPONENT = GR_ADMIT then
         LIST_COMPONENT := TRUE;
         if LIST_NODE = RLC then
            COMPONENT_VALUE (GR_ADMIT).DISPLAY := "R   ";
            COMPONENT_VALUE (GR_ADMIT).UNITS   := "Ohms";
            VALUE := 1.0 / VALUE;
         else
            COMPONENT_VALUE (GR_ADMIT).DISPLAY := "YR  ";
            COMPONENT_VALUE (GR_ADMIT).UNITS   := "Mhos";
         end if;
      end if;
      if COMPONENT in REAL_VOLT..IMAG_CENT then
         if COMPONENT in REAL_VOLT..IMAG_VOLT then
            SHOW_COMPONENT := VOLT_SOURCE;
         else
            SHOW_COMPONENT := CENT_SOURCE;
         end if;
         case COMPONENT is
            when REAL_VOLT | REAL_CENT =>
               TEMP_VALUE := VALUE;
            when IMAG_VOLT | IMAG_CENT =>
               MAGNITUDE := Sqrt (TEMP_VALUE*TEMP_VALUE + VALUE*VALUE);
               PHASE := Maths.Angle (X_COORDINATE => TEMP_VALUE,
                  Y_COORDINATE => VALUE);
               TEMP_VALUE := 0.0;
               if (MAGNITUDE > MINIMUM_VALUE) and
                  (FORM /= EQ_INDCT_FORM) then
                  Screen_Supporter.Display_Component (
                     COMPONENT  => SHOW_COMPONENT,
                     HIGH_NODE1 => HIGH_NODE1,
                     LOW_NODE1  => LOW_NODE1,
                     HIGH_NODE2 => HIGH_NODE2,
                     LOW_NODE2  => LOW_NODE2,
                     VALUE      => MAGNITUDE,
                     PHASE      => PHASE,
                     ROW        => ROW);
               end if;
               LIST_COMPONENT := TRUE;

            when others => null;
         end case;
      elsif ((COMPONENT < GR_ADMIT) and (COMPONENT /= INDUCT) and
         (FORM /= EQ_INDCT_FORM) and LIST_COMPONENT) or
         ((COMPONENT = INDUCT) and (FORM = EQ_INDCT_FORM)) then
         Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
            HIGH_NODE1 => HIGH_NODE1,
            LOW_NODE1  => LOW_NODE1,
            HIGH_NODE2 => HIGH_NODE2,
            LOW_NODE2  => LOW_NODE2,
            VALUE      => VALUE,
            PHASE      => PHASE,
            ROW        => ROW);
      end if;

      if (HIGH_NODE2 /= LOW_NODE2) and LIST_COMPONENT and
         (FORM not in EQ_INDCT_FORM..PROCEDURE_FORM) then
         Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
            HIGH_NODE1 => HIGH_NODE1,
            LOW_NODE1  => LOW_NODE1,
            HIGH_NODE1 => HIGH_NODE2,
            LOW_NODE2  => LOW_NODE2,
            VALUE      => VALUE,
            PHASE      => PHASE,
            ROW        => ROW);
      end if;
   end if;
   if ROW > 24 then
      ROW := 3;
      Status.Wait;
      Clear;
      Write (ITEM => Screen_Supporter.HEADING1,
         FORM => UNDERLINED);
   end if;
end loop;
end List_Branch;

-------------------------------------------------------------------------
-- PURPOSE : Gets a name  of a disk file  for loading a netlist  or  monte
--           netlist.  The  existing files  of  netlists or monte are also
--           displayed.
-------------------------------------------------------------------------

procedure Get_Load_File (NETLIST_FILE : in out Netlist_IO.FILE_TYPE;
   MONTE_LOAD : in BOOLEAN) is

   GOT_FILES : BOOLEAN;
   ERROR     : INTEGER_TYPE;
   FILENAME  : Dos_Environment.FILENAME_TYPE;

begin
   Cursor_Set (ROW => 3, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
   Cursor_Next_Line;
```

```
   Dos_Environment.Find_First_File (MONTE => MONTE_LOAD,
      ERROR    => ERROR,
      FILENAME => FILENAME);
   if ERROR /= 0 then
      Errors.Write (MESSAGE => "There are no recognised files in the " &
         "current directory.");
      raise USER_ESCAPE;
   end if;
   while ERROR = 0 loop
      Prompts.Add_To_List (FILE_NAME => FILENAME);
      Dos_Environment.Find_Next_File (ERROR => ERROR,
         FILENAME => FILENAME);
   end loop;
   begin
      Prompts.Read (FILE_NAME => FILENAME,
         HELP_INFO => HP_LOAD_NETLIST);
      Netlist_IO.Open (FILE => NETLIST_FILE,
         MODE => Netlist_IO.IN_FILE,
         NAME => FILENAME);
   exception
      when Netlist_IO.NAME_ERROR =>
         Errors.Write (MESSAGE => "The file cannot be found.",
            HELP_INFO => HP_LOAD_NETLIST);
         raise USER_ESCAPE;
   end;
end Get_Load_File;

-- PURPOSE : Recovers a netlist from a disk file  by reading the values of
--           the components found in the specified file.

procedure Load_Netlist (NETLIST_OBJECT : in out NETLIST_TYPE) is

   NETLIST_FILE                    : Netlist_IO.FILE_TYPE;
   NETLIST_ITEM1                   : LOAD_SAVE_RECORD (LOAD => TRUE);
   NETLIST_ITEM2                   : LOAD_SAVE_RECORD (LOAD => FALSE);
   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2,
   NODE_MAX                        : NODE_TYPE;
   COMPONENT                       : ELEMENTS;
   VALUE                           : REAL_TYPE;

begin
   Process_Title (TITLE => "LOAD CIRCUIT");
   Get_Load_File (NETLIST_FILE => NETLIST_FILE,
      MONTE_LOAD => FALSE);
   Inner.Initialize (NETLIST_OBJECT => NETLIST_OBJECT);
   if not Netlist_IO.End_Of_File (FILE => NETLIST_FILE) then
      Netlist_IO.Read (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM1);
      NETLIST_OBJECT.TITLE     := NETLIST_ITEM1.TITLE;
      NETLIST_OBJECT.FREQUENCY := NETLIST_ITEM1.FREQUENCY;
   end if;
   while not Netlist_IO.End_Of_File (FILE => NETLIST_FILE) loop
      Netlist_IO.Read (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      Inner.New_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      NETLIST_OBJECT.LIST.DATA := NETLIST_ITEM2.COMPS;
      for COMPONENT in ELEMENTS loop
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         if abs (VALUE) > MINIMUM_VALUE then
            Maths.Check_Value (MINIMUM => COMPONENT_VALUE (COMPONENT).MIN,
               MAXIMUM => COMPONENT_VALUE (COMPONENT).MAX,
               VALUE   => VALUE);
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         end if;
      end loop;
   end loop;
   Netlist_IO.Close (FILE => NETLIST_FILE);
   Inner.Find_Node_Maximum (NODE_MAXIMUM => NODE_MAX,
      NETLIST_OBJECT => NETLIST_OBJECT);
exception
   when Netlist_IO.DATA_ERROR | Netlist_IO.DEVICE_ERROR =>
      Netlist_IO.USE_ERROR =>
      Errors.Write (MESSAGE => DEVICE_ERROR_MESSAGE);
end Load_Load_File;

-- PURPOSE : Recovers a  Monte-Carlo  netlist from a disk file  by reading
--           the values found in the specified file.

procedure Load_Monte_Netlist (NETLIST_OBJECT1,
   NETLIST_OBJECT2 : in out NETLIST_TYPE) is

   NETLIST_FILE    : Netlist_IO.FILE_TYPE;
   NETLIST_ITEM2   : LOAD_SAVE_RECORD (LOAD => FALSE);

begin
   Process_Title (TITLE => "LOAD MONTE");
   Get_Load_File (NETLIST_FILE => NETLIST_FILE,
      MONTE_LOAD => TRUE);
   Inner.Initialize (NETLIST_OBJECT => NETLIST_OBJECT1);
   Inner.Initialize (NETLIST_OBJECT => NETLIST_OBJECT2);
   while not Netlist_IO.End_Of_File (FILE => NETLIST_FILE) loop
      Netlist_IO.Read (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      Inner.New_Branch (NETLIST_OBJECT => NETLIST_OBJECT1);
      NETLIST_OBJECT1.LIST.DATA := NETLIST_ITEM2.COMPS;
      Netlist_IO.Read (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      Inner.New_Branch (NETLIST_OBJECT => NETLIST_OBJECT2);
      NETLIST_OBJECT2.LIST.DATA := NETLIST_ITEM2.COMPS;
   end loop;
   Netlist_IO.Close (FILE => NETLIST_FILE);
exception
   when Netlist_IO.DATA_ERROR | Netlist_IO.DEVICE_ERROR =>
      Netlist_IO.USE_ERROR =>
      Errors.Write (MESSAGE => DEVICE_ERROR_MESSAGE);
end Load_Monte_Netlist;

-- PURPOSE : Removes a node from the  NETLIST_OBJECT.  The components that
--           are deleted are displayed on the main screen.
-- EXCEPTIONS : ATTEMPTED_REFERENCE_NODE_REMOVAL
--              DELETE_NODE_DOES_NOT_EXIST
--              NO_CIRCUIT

procedure Remove_Node (NETLIST_OBJECT : in out NETLIST_TYPE) is

   BRANCH_DELETED,      ERROR                : BOOLEAN  := TRUE;
   SCREEN_ROW                                : POSITIVE := 3;
   NODE_MAX                                  : NODE_TYPE;
   DELETE_NODE                               : INTEGER_TYPE;

   ATTEMPTED_REFERENCE_NODE_REMOVAL          : exception;
   DELETE_NODE_DOES_NOT_EXIST                : exception;

begin
   if NETLIST_OBJECT.NODE_MAXIMUM = 0 then
      raise NO_CIRCUIT;
   end if;
   Process_Title (TITLE => "REMOVE NODE");
   Prompts.Read (MESSAGE => "Enter the Node to delete.",
      ITEM => DELETE_NODE);
   if DELETE_NODE = 0 then
      raise ATTEMPTED_REFERENCE_NODE_REMOVAL;
   elsif DELETE_NODE not in 1..NETLIST_OBJECT.NODE_MAXIMUM then
      raise DELETE_NODE_DOES_NOT_EXIST;
   end if;
   Write (ITEM => Screen_Supporter.HEADING1,
      FORM => UNDERLINED);
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while NETLIST_OBJECT.LIST /= null loop
      BRANCH_DELETED := FALSE;
      if (NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE = DELETE_NODE) or
         (NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE = DELETE_NODE) then
         ERROR := FALSE;
         List_Branch (LIST_NODE => RLC,
            FORM           => PROCEDURES_FORM,
            ROW            => SCREEN_ROW,
            NETLIST_OBJECT => NETLIST_OBJECT);
         if (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE = 0) and
            (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE = 0) then
            Inner.Delete_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
            BRANCH_DELETED := TRUE;
         else
            NETLIST_OBJECT.LIST.DATA.GENERAL.CAPACITOR      := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.EQUIV_INDUCTOR := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_ADMITTANCE:= 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_ADMITTANCE:= 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_VOLTAGE   := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_VOLTAGE   := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.REAL_CURRENT   := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.IMAG_CURRENT   := 0.0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE    := 0;
            NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE     := 0;
         end if;
      end if;
      if not BRANCH_DELETED then
         NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
      end if;
   end loop;
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while NETLIST_OBJECT.LIST /= null loop
      if (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE=DELETE_NODE)
         or (NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE =
         DELETE_NODE) then
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.REAL_ADMITTANCE := 0.0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE := 0.0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.MAG_ADMITTANCE  := 0.0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.POLE_FREQUENCY  := 0.0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE     := 0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE      := 0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE := 0;
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE  := 0;
      end if;
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
   end loop;
   if ERROR then
      raise DELETE_NODE_DOES_NOT_EXIST;
   end if;
   Inner.Find_Node_Maximum (NODE_MAXIMUM => NODE_MAX,
      NETLIST_OBJECT => NETLIST_OBJECT);
exception
   when ATTEMPTED_REFERENCE_NODE_REMOVAL =>
      Errors.Write (MESSAGE => "The reference node cannot be deleted.");
   when DELETE_NODE_DOES_NOT_EXIST =>
      Errors.Write (MESSAGE => "The node to be deleted does not exist");
end Remove_Node;

-- PURPOSE : Renumbers an  existing  node.  Certain  voltage  and  current
--           directions are changed. The LIST pointer in NETLIST_OBJECT is
--           not affected.
-- EXCEPTIONS : NEW_NODE_ALREADY_EXISTS
--              NO_CIRCUIT
--              OLD_NODE_DOES_NOT_EXIST

procedure Renumber_Nodes (NETLIST_OBJECT : in out NETLIST_TYPE) is

   NEW_NODE,            OLD_NODE            : INTEGER_TYPE;
   SWITCHED                                 : BOOLEAN;
   POINTER                                  : LINK := NETLIST_OBJECT.HEAD;
   NODE_MAX                                 : NODE_TYPE;

   NEW_NODE_ALREADY_EXISTS                  : exception;
   OLD_NODE_DOES_NOT_EXIST                  : exception;

-- PURPOSE : Checks to see if the specified is  found  at  all  in  the
--           whole netlist. The LIST pointer in the  NETLIST_OBJECT  is
--           not affected.
```

```
   function Node_Exists (NODE : in NODE_TYPE) return BOOLEAN is

      NODE_MAXIMUM : constant := 100;
      POINTER      : LINK     := NETLIST_OBJECT.HEAD;

   begin
      while (POINTER/=null) and (NODE <= NODE_MAXIMUM) and (NODE > 0) loop
         if (NODE = POINTER.DATA.GENERAL.HIGHER_NODE) or
            (NODE = POINTER.DATA.GENERAL.LOWER_NODE) or
            (NODE = POINTER.DATA.INDUCTIVE.HIGHER_NODE) or
            (NODE = POINTER.DATA.INDUCTIVE.LOWER_NODE) or
            (NODE = POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE) or
            (NODE = POINTER.DATA.DEPEND_SOURCE.LOWER_NODE) then
            return TRUE;
         end if;
         POINTER := POINTER.NEXT;
      end loop;
      return FALSE;
   end Node_Exists;
```

```
begin  -- Renumber_Nodes
   if NETLIST_OBJECT.NODE_MAXIMUM = 0 then
      raise NO_CIRCUIT;
   end if;
   Process_Title (TITLE => "RENUMBER NODE");
   loop
      Prompts.Read (MESSAGE => "Enter the old node number.",
         ITEM => OLD_NODE);
      Prompts.Read (MESSAGE => "Enter the new node number.",
         ITEM => NEW_NODE);
      if (OLD_NODE in 1..MAX_NODES) and (NEW_NODE in 1..MAX_NODES) then
         exit;
      else
         Errors.Write (MESSAGE => "The node numbers are incorrect");
      end if;
   end loop;
   if not Node_Exists (OLD_NODE) then
      raise OLD_NODE_DOES_NOT_EXIST;
   elsif Node_Exists (NEW_NODE) then
      raise NEW_NODE_ALREADY_EXISTS;
   end if;

   while POINTER /= null loop
      if OLD_NODE = POINTER.DATA.GENERAL.HIGHER_NODE then
         POINTER.DATA.GENERAL.HIGHER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.GENERAL.LOWER_NODE then
         POINTER.DATA.GENERAL.LOWER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.INDUCTIVE.HIGHER_NODE then
         POINTER.DATA.INDUCTIVE.HIGHER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.INDUCTIVE.LOWER_NODE then
         POINTER.DATA.INDUCTIVE.LOWER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE then
         POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.DEPEND_SOURCE.LOWER_NODE then
         POINTER.DATA.DEPEND_SOURCE.LOWER_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE then
         POINTER.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE then
         POINTER.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE1 then
         POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE1 := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE1 then
         POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE1 := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE2 then
         POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE2 := NEW_NODE;
      end if;
      if OLD_NODE = POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE2 then
         POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE2 := NEW_NODE;
      end if;
      POINTER := POINTER.NEXT;
   end loop;

   POINTER := NETLIST_OBJECT.HEAD;
   while POINTER /= null loop
      if (abs (POINTER.DATA.GENERAL.REAL_VOLTAGE) +
         abs (POINTER.DATA.GENERAL.IMAG_VOLTAGE)) > MINIMUM_VALUE then
         if POINTER.DATA.GENERAL.HIGHER_NODE <
            POINTER.DATA.GENERAL.LOWER_NODE then
            POINTER.DATA.GENERAL.REAL_VOLTAGE :=
               -1.0 * POINTER.DATA.GENERAL.REAL_VOLTAGE;
            POINTER.DATA.GENERAL.IMAG_VOLTAGE :=
               -1.0 * POINTER.DATA.GENERAL.IMAG_VOLTAGE;
            Maths.Swap_Node (NODE1 => POINTER.DATA.GENERAL.HIGHER_NODE,
               NODE2 => POINTER.DATA.GENERAL.LOWER_NODE);
         end if;
      end if;

      if (abs (POINTER.DATA.GENERAL.REAL_CURRENT) +
         abs (POINTER.DATA.GENERAL.IMAG_CURRENT)) > MINIMUM_VALUE then
         if POINTER.DATA.GENERAL.HIGHER_NODE <
            POINTER.DATA.GENERAL.LOWER_NODE then
            POINTER.DATA.GENERAL.REAL_CURRENT :=
               -1.0 * POINTER.DATA.GENERAL.REAL_CURRENT;
            POINTER.DATA.GENERAL.IMAG_CURRENT :=
               -1.0 * POINTER.DATA.GENERAL.IMAG_CURRENT;
            Maths.Swap_Node (NODE1 => POINTER.DATA.GENERAL.HIGHER_NODE,
               NODE2 => POINTER.DATA.GENERAL.LOWER_NODE);
         end if;
      end if;

      if POINTER.DATA.GENERAL.HIGHER_NODE <
         POINTER.DATA.GENERAL.LOWER_NODE then
         Maths.Swap_Node (NODE1 => POINTER.DATA.GENERAL.HIGHER_NODE,
            NODE2 => POINTER.DATA.GENERAL.LOWER_NODE);
      end if;

      SWITCHED := FALSE;
      if abs (POINTER.DATA.DEPEND_SOURCE.MAG_ADMITTANCE) > MINIMUM_VALUE then
         if POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE <
            POINTER.DATA.DEPEND_SOURCE.LOWER_NODE then
            SWITCHED := TRUE;
            Maths.Swap_Node (
               NODE1 => POINTER.DATA.DEPEND_SOURCE.HIGHER_NODE,
               NODE2 => POINTER.DATA.DEPEND_SOURCE.LOWER_NODE);
         end if;
         if POINTER.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE <
            POINTER.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE then
            SWITCHED := TRUE;
            Maths.Swap_Node (
               NODE1 => POINTER.DATA.DEPEND_SOURCE.HIGHER_CONTROL_NODE,
               NODE2 => POINTER.DATA.DEPEND_SOURCE.LOWER_CONTROL_NODE);
         end if;
         if SWITCHED then
            POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE :=
               -1.0 * POINTER.DATA.DEPEND_SOURCE.REAL_ADMITTANCE;
            POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE :=
               -1.0 * POINTER.DATA.DEPEND_SOURCE.IMAG_ADMITTANCE;
            POINTER.DATA.DEPEND_SOURCE.MAG_ADMITTANCE :=
               -1.0 * POINTER.DATA.DEPEND_SOURCE.MAG_ADMITTANCE;
         end if;
      end if;

      SWITCHED := FALSE;
      if abs (POINTER.DATA.MUTUAL_INDUCT.COUPLING) > MINIMUM_VALUE then
         if POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE1 <
            POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE1 then
            SWITCHED := TRUE;
            Maths.Swap_Node (
               NODE1 => POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE1,
               NODE2 => POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE1);
         end if;
         if POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE2 <
            POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE2 then
            SWITCHED := TRUE;
            Maths.Swap_Node (
               NODE1 => POINTER.DATA.MUTUAL_INDUCT.HIGHER_NODE2,
               NODE2 => POINTER.DATA.MUTUAL_INDUCT.LOWER_NODE2);
         end if;
         if SWITCHED then
            POINTER.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR :=
               -1.0 * POINTER.DATA.MUTUAL_INDUCT.MUTUAL_INDUCTOR;
            POINTER.DATA.MUTUAL_INDUCT.COUPLING :=
               -1.0 * POINTER.DATA.MUTUAL_INDUCT.COUPLING;
         end if;
      end if;

      POINTER := POINTER.NEXT;
   end loop;
   Inner.Find_Node_Maximum (NODE_MAXIMUM => NODE_MAX,
      NETLIST_OBJECT => NETLIST_OBJECT);
exception
   when NEW_NODE_ALREADY_EXISTS =>
      Errors.Write (MESSAGE => "The new node already exists.");
   when OLD_NODE_DOES_NOT_EXIST =>
      Errors.Write (MESSAGE => "The old node does not exist.");
end Renumber_Nodes;
```

```
-- PURPOSE : Gets a name  of a disk file  for saving  a netlist  or  monte
--           netlist.  The  existing files  of  netlists or monte are also
--           displayed.
```

```
procedure Get_Save_File (NETLIST_FILE : in out Netlist_IO.FILE_TYPE;
   MONTE_SAVE : in BOOLEAN) is

   SCREEN_ITEM                          : INTEGER_TYPE := 0;
   ERROR                                : INTEGER_TYPE;
   FILENAME                             : Dos_Environment.FILENAME_TYPE;
   CIRCUIT_LINE                         : Texts.TEXT_TYPE;
   GOT_FILES,          OVERWRITE        : BOOLEAN;
   EXTENSION                            : STRING (1..4);

   DISK_MESSAGE : constant STRING := "Enter the name of the file " &
      "(but not the extension) to be saved.";

begin
   if MONTE_SAVE then
      EXTENSION := ".MNT";
   else
      EXTENSION := ".NTL";
   end if;
   Cursor_Set (ROW => 3, COLUMN => 7);
   Write (ITEM => PATH, FORM => UNDERLINED);
   Cursor_Next_Line;
   Dos_Environment.Find_First_File (MONTE => MONTE_SAVE,
      ERROR    => ERROR,
      FILENAME => FILENAME);
   loop
      if ERROR = 0 then
         if SCREEN_ITEM < 100 then
            Cursor_Set (ROW => SCREEN_ITEM / 5 + 5,
               COLUMN => (SCREEN_ITEM rem 5) * 15 + 3);
            Write (ITEM => FILENAME);
            SCREEN_ITEM := SCREEN_ITEM + 1;
         else
            Status.Wait;
            SCREEN_ITEM := 1;
            Clear;
            Cursor_Set (ROW => 3, COLUMN => 7);
            Write (ITEM => PATH, FORM => UNDERLINED);
         end if;
      else
         exit;
      end if;
      Dos_Environment.Find_Next_File (ERROR => ERROR,
         FILENAME => FILENAME);
   end loop;
   loop
      begin
         Prompts.Read (MESSAGE => DISK_MESSAGE,
            HELP_INFO   => BP_SAVE_NETLIST,
            ITEM        => CIRCUIT_LINE.TEXT,
            ITEM_LENGTH => CIRCUIT_LINE.TEXT_INDEX);
         Texts.Combine (ITEM => EXTENSION,
            TEXT_LINE => CIRCUIT_LINE);
         Netlist_IO.Open (FILE => NETLIST_FILE,
```

```
            MODE => Netlist_IO.IN_FILE,
            NAME => CIRCUIT_LINE.TEXT (1..CIRCUIT_LINE.TEXT_INDEX));
         Netlist_IO.Close (FILE => NETLIST_FILE);
         Prompts.Read (MESSAGE => "The file already exists. Do you want"
            & " to overwrite it ?",
            HELP_INFO => HP_SAVE_NETLIST2,
            ITEM      => OVERWRITE);
         if OVERWRITE then
            Netlist_IO.Open (FILE => NETLIST_FILE,
               MODE => Netlist_IO.OUT_FILE,
               NAME => CIRCUIT_LINE.TEXT (1..CIRCUIT_LINE.TEXT_INDEX));
            exit;
         end if;
      exception
         when Netlist_IO.NAME_ERROR =>
            Netlist_IO.Create (FILE => NETLIST_FILE,
               NAME => CIRCUIT_LINE.TEXT (1..CIRCUIT_LINE.TEXT_INDEX));
            exit;
      end;
   end loop;
end Get_Save_File;

-- ------------------------------------------------------------------
-- PURPOSE : Saves the Netlist by saving the values of the components to a
--           specified disk file.
-- EXCEPTIONS : NO_CIRCUIT
-- ------------------------------------------------------------------

procedure Save_Netlist (NETLIST_OBJECT : in out NETLIST_TYPE) is

   NETLIST_FILE  : Netlist_IO.FILE_TYPE;
   NETLIST_ITEM1 : LOAD_SAVE_RECORD (LOAD => TRUE);
   NETLIST_ITEM2 : LOAD_SAVE_RECORD (LOAD => FALSE);

begin
   Process_Title (TITLE => "SAVE CIRCUIT");
   if NETLIST_OBJECT.NODE_MAXIMUM = 0 then
      raise NO_CIRCUIT;
   end if;
   Get_Save_File (NETLIST_FILE => NETLIST_FILE,
      MONTE_SAVE => FALSE);
   NETLIST_ITEM1.FREQUENCY := NETLIST_OBJECT.FREQUENCY;
   NETLIST_ITEM1.TITLE     := NETLIST_OBJECT.TITLE;
   Netlist_IO.Write (FILE => NETLIST_FILE,
      ITEM => NETLIST_ITEM1);
   Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
   loop
      NETLIST_ITEM2.COMPS := NETLIST_OBJECT.LIST.DATA;
      Netlist_IO.Write (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
   Netlist_IO.Close (FILE => NETLIST_FILE);
exception
   when Netlist_IO.DATA_ERROR | Netlist_IO.DEVICE_ERROR |
      Netlist_IO.USE_ERROR =>
      Errors.Write (MESSAGE => DEVICE_ERROR_MESSAGE);
   when Netlist_IO.NAME_ERROR =>
      Errors.Write (MESSAGE => NAME_ERROR_MESSAGE);
end Save_Netlist;

-- ------------------------------------------------------------------
-- PURPOSE : Saves the Monte-Carlo netlist by saving the values to a
--           specified disk file.
-- ------------------------------------------------------------------

procedure Save_Monte_Netlist (NETLIST_OBJECT1,
   NETLIST_OBJECT2 : in out NETLIST_TYPE) is

   NETLIST_FILE  : Netlist_IO.FILE_TYPE;
   NETLIST_ITEM2 : LOAD_SAVE_RECORD (LOAD => FALSE);

begin
   Process_Title (TITLE => "SAVE MONTE");
   Get_Save_File (NETLIST_FILE => NETLIST_FILE,
      MONTE_SAVE => TRUE);
   Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT1);
   Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT2);
   loop
      NETLIST_ITEM2.COMPS := NETLIST_OBJECT1.LIST.DATA;
      Netlist_IO.Write (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      NETLIST_ITEM2.COMPS := NETLIST_OBJECT2.LIST.DATA;
      Netlist_IO.Write (FILE => NETLIST_FILE,
         ITEM => NETLIST_ITEM2);
      begin
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT1);
         Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT2);
      exception
         when NO_MORE_BRANCHES => exit;
      end;
   end loop;
   Netlist_IO.Close (FILE => NETLIST_FILE);
exception
   when Netlist_IO.DATA_ERROR | Netlist_IO.DEVICE_ERROR |
      Netlist_IO.USE_ERROR =>
      Errors.Write (MESSAGE => DEVICE_ERROR_MESSAGE);
   when Netlist_IO.NAME_ERROR =>
      Errors.Write (MESSAGE => NAME_ERROR_MESSAGE);
end Save_Monte_Netlist;

-- ------------------------------------------------------------------
-- PURPOSE : Displays the title of the circuit,  and if desired, it can be
--           changed.
-- ------------------------------------------------------------------

procedure Set_Circuit_Title (NETLIST_OBJECT : in out NETLIST_TYPE) is

   RESPONSE      : BOOLEAN;
   STRING_LENGTH : INTEGER_TYPE;

begin
   Main.Process_Title (TITLE => "CIRCUIT TITLE");
   Main.Cursor_Set (ROW => 3, COLUMN => 8);
   Main.Write (ITEM => NETLIST_OBJECT.TITLE, FORM => UNDERLINED);
   Prompts.Read (MESSAGE => "DO you want to change it?",
      ITEM => RESPONSE);
   if RESPONSE then
      Prompts.Read (MESSAGE => "Enter the title for the circuit.",
         ITEM        => NETLIST_OBJECT.TITLE,
         ITEM_LENGTH => STRING_LENGTH);
   end if;
   Main.Clear;
end Set_Circuit_Title;

-- ------------------------------------------------------------------
-- PURPOSE : Lists all the components that are connected between the
--           LOW_SHOW_NODE and the HIGH_SHOW_NODE.
-- EXCEPTIONS : NO_CIRCUIT
-- ------------------------------------------------------------------

procedure Show_Node (NETLIST_OBJECT : in out NETLIST_TYPE) is

   PHASE : constant := 0.0;

   COMPONENT                              : ELEMENTS;
   DISPLAY_LINE                           : Texts.TEXT_TYPE;
   SCREEN_ROW                             : POSITIVE  := 3;
   VALUE                                  : REAL_TYPE;
   HIGH_SHOW_NODE,      LOW_SHOW_NODE,
   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2         : NODE_TYPE;

begin
   if NETLIST_OBJECT.NODE_MAXIMUM = 0 then
      raise NO_CIRCUIT;
   end if;
   Prompts.Read (FORM => SHOW_FORM);
   HIGH_SHOW_NODE := SHOW_FORM (1).INTEGER_VALUE;
   LOW_SHOW_NODE  := SHOW_FORM (2).INTEGER_VALUE;
   if HIGH_SHOW_NODE < LOW_SHOW_NODE then
      Maths.Swap_Node (NODE1 => HIGH_SHOW_NODE,
         NODE2 => LOW_SHOW_NODE);
   end if;

   Texts.Combine (ITEM => "SHOW NODE PAIR", TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => HIGH_SHOW_NODE,
      ITEM_WIDTH => 3,
      TEXT_LINE  => DISPLAY_LINE);
   Texts.Combine (ITEM => " -", TEXT_LINE => DISPLAY_LINE);
   Texts.Combine (ITEM => LOW_SHOW_NODE,
      ITEM_WIDTH => 3,
      TEXT_LINE  => DISPLAY_LINE);
   Texts.Combine (ITEM => " ", TEXT_LINE => DISPLAY_LINE);
   Process_Title (TITLE => DISPLAY_LINE.TEXT(1..DISPLAY_LINE.TEXT_INDEX));
   Write (ITEM => Screen_Supporter.HEADING1,
      FORM => UNDERLINED);

   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
   while NETLIST_OBJECT.LIST /= null loop
      if (HIGH_SHOW_NODE=NETLIST_OBJECT.LIST.DATA.GENERAL.HIGHER_NODE) and
         (LOW_SHOW_NODE=NETLIST_OBJECT.LIST.DATA.GENERAL.LOWER_NODE) then
         List_Branch (LIST_NODE => RLC,
            FORM          => PROCEDURE_FORM,
            ROW           => SCREEN_ROW,
            NETLIST_OBJECT => NETLIST_OBJECT);
      end if;

      if (HIGH_SHOW_NODE =
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.HIGHER_NODE) and
         (LOW_SHOW_NODE =
         NETLIST_OBJECT.LIST.DATA.DEPEND_SOURCE.LOWER_NODE) then
         for COMPONENT in CR_ADMIT..POLE_FREQ loop
            Inner.Read_Netlist (COMPONENT => COMPONENT,
               VALUE       => VALUE,
               HIGH_NODE1  => HIGH_NODE1,
               LOW_NODE1   => LOW_NODE1,
               HIGH_NODE2  => HIGH_NODE2,
               LOW_NODE2   => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            if (abs (VALUE) > MINIMUM_VALUE) and
               (HIGH_NODE1 /= LOW_NODE1) then
               Screen_Supporter.Display_Component (COMPONENT => COMPONENT,
                  HIGH_NODE1 => HIGH_NODE1,
                  LOW_NODE1  => LOW_NODE1,
                  HIGH_NODE2 => HIGH_NODE2,
                  LOW_NODE2  => LOW_NODE2,
                  VALUE      => VALUE,
                  PHASE      => PHASE,
                  ROW        => SCREEN_ROW);
            end if;
         end loop;
      end if;
      NETLIST_OBJECT.LIST := NETLIST_OBJECT.LIST.NEXT;
   end loop;
   NETLIST_OBJECT.LIST := NETLIST_OBJECT.HEAD;
end Show_Node;

-- ------------------------------------------------------------------

end Outer;
```

Optimise

```
with  Ada_Extensions,
      Netlists;

use   Ada_Extensions,
      Netlists;

package Optimiser is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This optimises a circuit's performance by tuning component
--           values and/or frequency.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Clear (NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Find_Bandwidth (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Insert (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Maximise (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Minimise (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Set (NODE_MAX : in NODE_TYPE;
      NETLIST_OBJECT : in out NETLIST_TYPE);

   procedure Show (NETLIST_OBJECT : in out NETLIST_TYPE);

end Optimiser;
```

```
with  Control_Solver,
      Math_Lib,
      Netlist_Supporter,
      Screens;

use   Screens;

package body Optimiser is

   use Screens.Main;

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   HP_CHANGE_OPTIMISE : constant := 160;
   HP_OPTIMISING      : constant := 161;
   HP_OPTIMISING2     : constant := 162;
   HP_TUNE            : constant := 163;
   HP_TUNE2           : constant := 164;
   HP_OPTIM_INSERT    : constant := 165;
   HP_OPTIM_INSERT2   : constant := 166;

   DELTA_INCREMENT  : constant := 0.1;
   MINIMUM_ACCURACY : constant := 1.0E-10;
   MINIMUM_ERROR    : constant := 1.0E-6;
   MIN_F            : constant := 1.0E-35;
   MINIMUM_F        : constant := -700.0;

   type OPTIMISE1_TYPE is (FIND,  MAXI,  MINI,  SETO);
   type OPTIMISE2_TYPE is (MAXIM, MINIM, SETT);
   type RESULTS_TYPE   is (UNCHN, IMAGN, MAGNI, PHASE, REALS);

   type INTEGER_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE)
      of INTEGER_TYPE;
   type REAL_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE) of REAL_TYPE;
   type NX_ARRAY_TYPE   is array (1..MAX_OPTIMUM_VARIABLE) of ELEMENTS;
```

```
   ANGLE_E,             ANGLE_Y,
   ANGLE_Y,             MAGN_E,
   MAGN_Y,              MAGN_Y,
   PERTUB,              SC,
   XX                                : REAL_ARRAY_TYPE;
   IDEG,                LPAR         : INTEGER_ARRAY_TYPE;
   NX                                : NX_ARRAY_TYPE;

   MUTUAL_VALUE,        PHASE,
   TEMP_VALUE1,         VALUE1       : REAL_TYPE;

   NN,                  NEE,
   NTT,                 NZI,
   NZI,                 NTYI,
   NTYA,                NTYAI,
   NTYAR,               NTYR,
   NTI                               : INTEGER_TYPE;
   RESULTS                           : RESULTS_TYPE;
   COMPONENT                         : ELEMENTS;
   VISIBLE_COMPONENT                 : ELEMENT_TYPE;
   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2    : NODE_TYPE;

OPTIMISE_FORM1 : FIELDS_TYPE (1..7) :=
   (1 =>
      (VALUE_LENGTH  => 2,
      TEXT_LENGTH    => 13,
      IS_INTEGER_TYPE => TRUE,
      ACCEPTING      => TRUE,
      TEST_AGAINST   => 2,
      ACTUAL_LENGTH  => 1,
      ROW            => 4,
      VALUE          => "1 ",
      TEXT           => "Higher Node 1",
      MAX_INTEGER    => MAX_NODES,
      MIN_INTEGER    => 1,
      INTEGER_VALUE  => 1),
   2 =>
      (VALUE_LENGTH  => 2,
      TEXT_LENGTH    => 12,
      IS_INTEGER_TYPE => TRUE,
      ACCEPTING      => TRUE,
      TEST_AGAINST   => 1,
      ACTUAL_LENGTH  => 1,
      ROW            => 5,
      VALUE          => "0 ",
      TEXT           => "Lower Node 1",
      MAX_INTEGER    => MAX_NODES,
      MIN_INTEGER    => 0,
      INTEGER_VALUE  => 0),
   3 =>
      (VALUE_LENGTH  => 2,
      TEXT_LENGTH    => 13,
      IS_INTEGER_TYPE => TRUE,
      ACCEPTING      => FALSE,
      TEST_AGAINST   => 4,
      ACTUAL_LENGTH  => 1,
      ROW            => 6,
      VALUE          => "1 ",
      TEXT           => "Higher Node 2",
      MAX_INTEGER    => MAX_NODES,
      MIN_INTEGER    => 1,
      INTEGER_VALUE  => 1),
   4 =>
      (VALUE_LENGTH  => 2,
      TEXT_LENGTH    => 12,
      IS_INTEGER_TYPE => TRUE,
      ACCEPTING      => FALSE,
      TEST_AGAINST   => 3,
      ACTUAL_LENGTH  => 1,
      ROW            => 7,
      VALUE          => "0 ",
      TEXT           => "Lower Node 2",
      MAX_INTEGER    => MAX_NODES,
      MIN_INTEGER    => 0,
      INTEGER_VALUE  => 0),
   5 =>
      (VALUE_LENGTH  => 15,
      TEXT_LENGTH    => 19,
      IS_INTEGER_TYPE => FALSE,
      ACCEPTING      => TRUE,
      TEST_AGAINST   => 0,
      ACTUAL_LENGTH  => 3,
      ROW            => 8,
      VALUE          => "1.0e3           ",
      TEXT           => "Impedance Frequency",
      MAX_REAL       => 1.0e25,
      MIN_REAL       => MINIMUM_VALUE,
      REAL_VALUE     => 1000.0),
   6 =>
      (VALUE_LENGTH  => 15,
      TEXT_LENGTH    => 19,
      IS_INTEGER_TYPE => FALSE,
      ACCEPTING      => FALSE,
      TEST_AGAINST   => 0,
      ACTUAL_LENGTH  => 4,
      ROW            => 7,
      VALUE          => "90.0            ",
      TEXT           => "Required % Accuracy",
      MAX_REAL       => 1.0e25,
      MIN_REAL       => MINIMUM_ACCURACY,
      REAL_VALUE     => 90.0),
   7 =>
      (VALUE_LENGTH  => 15,
      TEXT_LENGTH    => 14,
      IS_INTEGER_TYPE => FALSE,
      ACCEPTING      => FALSE,
      TEST_AGAINST   => 0,
      ACTUAL_LENGTH  => 6,
      ROW            => 7,
      VALUE          => "1.0e-2          ",
      TEXT           => "Absolute Error",
      MAX_REAL       => 1.0e25,
      MIN_REAL       => MINIMUM_ERROR,
```

```
            REAL_VALUE     => 0.01)};

   OPTIMISE_LIST( : LIST_ARRAY_TYPE (1..1) := (1 =>
   (TEXT_LENGTH => 15, TEXT => "Interim results", STATE => TRUE)};

   OPTIMISE_LIST2 : LIST_ARRAY_TYPE (1..6) := (
      (TEXT_LENGTH => 24,  TEXT => "Voltage source magnitude",
         STATE => TRUE),
      (TEXT_LENGTH => 24,  TEXT => "Current source magnitude",
         STATE => FALSE),
      (TEXT_LENGTH => 20,  TEXT => "Admittance magnitude",
         STATE => FALSE),
      (TEXT_LENGTH => 26,  TEXT => "Voltage source phase angle",
         STATE => FALSE),
      (TEXT_LENGTH => 25,  TEXT => "Current source phase angle",
         STATE => FALSE),
      (TEXT_LENGTH => 22,  TEXT => "Admittance phase angle",
         STATE => FALSE));

-------------------------------------------------------------------------
-- PURPOSE : Adjusts the magnitude and phase of a component.
-------------------------------------------------------------------------

procedure Adjust_Magnitude_Phase (COMPONENT : in ELEMENTS;
   VALUE, MAGNITUDE, DEGREE, XC                : in REAL_TYPE;
   HIGH_NODE1, LOW_NODE1, HIGH_NODE2, LOW_NODE2 : in NODE_TYPE;
   NETLIST_OBJECT                              : in out NETLIST_TYPE) is

begin
   if abs (DEGREE) > 0.0 then
      Inner.Write_Netlist (COMPONENT => COMPONENT,
         VALUE          => XC * Math_Lib.Cos (DEGREE),
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
      Inner.Write_Netlist (COMPONENT => ELEMENTS'succ (COMPONENT),
         VALUE          => XC * Math_Lib.Sin (DEGREE),
         HIGH_NODE1     => HIGH_NODE1,
         LOW_NODE1      => LOW_NODE1,
         HIGH_NODE2     => HIGH_NODE2,
         LOW_NODE2      => LOW_NODE2,
         NETLIST_OBJECT => NETLIST_OBJECT);
   else
      if abs (MAGNITUDE) > 0.0 then
         Inner.Write_Netlist (COMPONENT => COMPONENT,
            VALUE          => MAGNITUDE * Math_Lib.Cos (XC),
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Write_Netlist (COMPONENT => ELEMENTS'succ (COMPONENT),
            VALUE          => MAGNITUDE * Math_Lib.Sin (XC),
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
      else
         Inner.Write_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
   end if;
end Adjust_Magnitude_Phase;

-------------------------------------------------------------------------
-- PURPOSE : Clears all the variables that were used in the optimisation
--           process.
-------------------------------------------------------------------------

procedure Clear_Tune (NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   NN    := 0;
   NTE   := 0;
   NTT   := 0;
   NTI   := 0;
   NTV   := 0;
   NTVR  := 0;
   NTVI  := 0;
   NTIM  := 0;
   NTIMR := 0;
   NTIMI := 0;
   NTX   := 0;
   XX       := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   I_VA     := (1..MAX_OPTIMUM_VARIABLE => 0);
   P_JUMP   := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   MAGN_E   := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   MAGN_I   := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   MAGN_V   := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   ANGLE_E  := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   ANGLE_I  := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   ANGLE_V  := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   IDEG     := (1..MAX_OPTIMUM_VARIABLE => 0);
   Inner.Clear_Locations (NETLIST_OBJECT => NETLIST_OBJECT);
end Clear_Tune;

-------------------------------------------------------------------------
-- PURPOSE : Performs optimisation on the magnitude or phase of a voltage,
--           current or admittance.
-------------------------------------------------------------------------

procedure Change_Optimise (NETLIST_OBJECT : in out NETLIST_TYPE) is

   COMPONENT_DOES_NOT_EXIST : exception;

   MINIMUM_ADJUST_PHASE : constant := 1.0E-3;
   MINIMUM_PHASE        : constant := 1.0E-25;

   HIGH_NODE1,          LOW_NODE1,
   HIGH_NODE2,          LOW_NODE2          : NODE_TYPE;
   OPTION,              COUNTER,
   SCREEN_ROW                              : INTEGER_TYPE;
   MAGNITUDE,           PHASE,
   TEMP_VALUE,          VALUE              : REAL_TYPE;
   COMPONENT,           VISIBLE_COMPONENT  : ELEMENT_TYPE;

begin
   Write (ITEM => Screen_Supporter.HEADING1,
      FOR => UNDERLINED);
   loop
      begin
         Prompts.Read (FROM => OPTIMISE_FORM1 (1..2),
            LIST      => OPTIMISE_LIST2,
            HELP_INFO => HP_CHANGE_OPTIMISE);
         HIGH_NODE1 := OPTIMISE_FORM1 (1).INTEGER_VALUE;
         LOW_NODE1  := OPTIMISE_FORM1 (2).INTEGER_VALUE;
         for COUNTER in 1..OPTIMISE_LIST2'last loop
            if OPTIMISE_LIST2 (COUNTER).STATE then
               OPTION := COUNTER;
               exit;
            end if;
         end loop;
         case OPTION is
            when 1 | 4 =>
               COMPONENT := REAL_VOLT;
               VISIBLE_COMPONENT := VOLT_SOURCE;
            when 2 | 5 =>
               COMPONENT := REAL_CRNT;
               VISIBLE_COMPONENT := CRNT_SOURCE;
            when 3 | 6 =>
               COMPONENT := GR_ADMIT;
               VISIBLE_COMPONENT := GR_ADMIT;
               COMPONENT_VALUE (GR_ADMIT).DISPLAY := 'Y   ';
            when others => null;
         end case;
         Inner.Find_Branch (COMPONENT => NOT_INDUCT,
            HIGH_NODE  => HIGH_NODE1,
            LOW_NODE   => LOW_NODE1,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => COMPONENT,
            VALUE          => TEMP_VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Read_Netlist (COMPONENT => ELEMENTS'succ (COMPONENT),
            VALUE          => VALUE,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         if (abs (TEMP_VALUE) + abs (VALUE)) > MINIMUM_VALUE then
            PHASE := Maths_Angle (X_COORDINATE => TEMP_VALUE,
               Y_COORDINATE => VALUE) / RADIAN;
            MAGNITUDE := Math_Lib.Sqrt (TEMP_VALUE * TEMP_VALUE
               + VALUE * VALUE);
            NN := NN + 1;
            Cursor_Set (ROW => 6, COLUMN => 3);

            case OPTION is
               when 1..3 =>
                  XX (NN) := MAGNITUDE;
                  Write (ITEM => "Optimising magnitude");

               when 4..6 =>
                  Write (ITEM => "Optimising phase");
                  XX (NN) := PHASE;
                  if abs (XX (NN)) < MINIMUM_ADJUST_PHASE then
                     if XX (NN) < 0.0 then
                        XX (NN) := -MINIMUM_ADJUST_PHASE;
                     else
                        XX (NN) := MINIMUM_ADJUST_PHASE;
                     end if;
                  end if;

               when others => null;
            end case;

            RETURN (NN) := 1.0E-1 * XX (NN);
            LPAR (NN) := 11;
            MX (NN) := COMPONENT;
            Inner.Save_Location (LOCATION_INDEX => NN,
               NETLIST_OBJECT => NETLIST_OBJECT);
            case OPTION is
               when 1 =>
                  ANGLE_E (NN) := PHASE + MINIMUM_PHASE;
                  NTE := 1;

               when 2 =>
                  ANGLE_I (NN) := PHASE + MINIMUM_PHASE;
                  NTI := 1;

               when 3 =>
                  ANGLE_V (NN) := PHASE + MINIMUM_PHASE;
                  NTV := 1;

               when 4 =>
                  MAGN_E (NN) := MAGNITUDE;
                  NTE := 1;
                  IDEG (NN) := 0;

               when 5 =>
                  MAGN_I (NN) := MAGNITUDE;
                  NTI := 1;
                  IDEG (NN) := 1;

               when 6 =>
                  MAGN_V (NN) := MAGNITUDE;
                  NTV := 1;
                  IDEG (NN) := 2;

               when others => null;
            end case;
            SCREEN_ROW := 3;
            Screen_Supporter.Display_Component (
               COMPONENT   => VISIBLE_COMPONENT,
               HIGH_NODE1 => HIGH_NODE1,
               LOW_NODE1  => LOW_NODE1,
               HIGH_NODE2 => HIGH_NODE2,
               LOW_NODE2  => LOW_NODE2,
```

```
            VALUE      => MAGNITUDE,
            PHASES     => PHASE,
            ROW        => SCREEN_ROW);
         exit;
      else
         raise COMPONENT_DOES_NOT_EXIST;
      end if;
   exception
      when NODE_PAIR_DOES_NOT_EXIST | COMPONENT_DOES_NOT_EXIST =>
         Errors.Write (MESSAGE => "Such a component does not exist.");
   end;
 end loop;
end Change_Optimise;

-- PURPOSE : Performs optimisation on a component.

procedure Optimise_Component (NODE_MAX : in NODE_TYPE;
  NETLIST_OBJECT : in out NETLIST_TYPE) is

  COMPONENT_DOES_NOT_EXIST : exception;

  VALUE                              : REAL_TYPE;
  SCREEN_ROW                         : INTEGER_TYPE;
  HIGH_NODE1,          LOW_NODE1,
  HIGH_NODE2,          LOW_NODE2     : NODE_TYPE;
  COMPONENT,           VISIBLE_COMPONENT : ELEMENT_TYPE;
  FOUND_SOURCE                       : BOOLEAN;

begin
  Write (ITEM => Screen_Supporter.HEADING1,
    FORM => UNDERLINED);
  Cursor_Next_Line;
  loop
    begin
      Screen_Supporter.Get_Component (COMPONENT => VISIBLE_COMPONENT);
      case VISIBLE_COMPONENT is
        when INDUCT     => COMPONENT := ACT_INDCT;
        when RESISTOR   => COMPONENT := CR_ADMIT;
        when CRNT_SOURCE => COMPONENT := REAL_CRNT;
        when VOLT_SOURCE => COMPONENT := REAL_VOLT;
        when others     => COMPONENT := VISIBLE_COMPONENT;
      end case;
      if COMPONENT in CR_ADMIT..POLE_FREQ then
        OPTIMISE_FORM1 (3).ACCEPTING := TRUE;
        OPTIMISE_FORM1 (4).ACCEPTING := TRUE;
      else
        OPTIMISE_FORM1 (3).ACCEPTING := FALSE;
        OPTIMISE_FORM1 (4).ACCEPTING := FALSE;
      end if;
      Prompts.Read (FORM => OPTIMISE_FORM1 (1..4));
      HIGH_NODE1 := OPTIMISE_FORM1 (1).INTEGER_VALUE;
      LOW_NODE1  := OPTIMISE_FORM1 (2).INTEGER_VALUE;
      HIGH_NODE2 := OPTIMISE_FORM1 (3).INTEGER_VALUE;
      LOW_NODE2  := OPTIMISE_FORM1 (4).INTEGER_VALUE;

      Inner.Find_Branch (COMPONENT => NOT_INDCT,
        HIGH_NODE      => HIGH_NODE1,
        LOW_NODE       => LOW_NODE1,
        NETLIST_OBJECT => NETLIST_OBJECT);
      if COMPONENT in CR_ADMIT..POLE_FREQ then
        Inner.Find_Branch (COMPONENT => NOT_INDCT,
          HIGH_NODE      => HIGH_NODE2,
          LOW_NODE       => LOW_NODE2,
          NETLIST_OBJECT => NETLIST_OBJECT);
        Inner.Find_Dependant_Source (HIGH_NODE1 => HIGH_NODE1,
          LOW_NODE1      => LOW_NODE1,
          HIGH_NODE2     => HIGH_NODE2,
          LOW_NODE2      => LOW_NODE2,
          CREATE         => FALSE,
          FOUND_SOURCE   => FOUND_SOURCE,
          NETLIST_OBJECT => NETLIST_OBJECT);
      elsif COMPONENT in MUT_INDCT..COUPLING then
        Inner.Find_Mutuals_And_Coupling (COMPONENT => COMPONENT,
          HIGH_NODE1     => HIGH_NODE1,
          LOW_NODE1      => LOW_NODE1,
          HIGH_NODE2     => HIGH_NODE2,
          LOW_NODE2      => LOW_NODE2,
          MUTUAL_VALUE   => MUTUAL_VALUE,
          VALUE          => VALUE,
          CREATE         => FALSE,
          NETLIST_OBJECT => NETLIST_OBJECT);
      end if;
      Inner.Read_Netlist (COMPONENT => COMPONENT,
        VALUE       => VALUE,
        HIGH_NODE1  => HIGH_NODE1,
        LOW_NODE1   => LOW_NODE1,
        HIGH_NODE2  => HIGH_NODE2,
        LOW_NODE2   => LOW_NODE2,
        NETLIST_OBJECT => NETLIST_OBJECT);
      if abs (VALUE) > MINIMUM_VALUE then
        NN := NN + 1;
        XX (NN) := VALUE;
        Inner.Save_Location (LOCATION_INDEX => NN,
          NETLIST_OBJECT => NETLIST_OBJECT);
        LPAR (NN) := 1;
        PTURB (NN) := DELTA_INCREMENT * XX (NN);
        KK (NN) := COMPONENT;
        case COMPONENT is
          when CAPS | ACT_INDCT | CR_ADMIT..COUPLING =>
            NTI := 1;
          when CR_ADMIT =>
            NTZ  := 1;
            NTYR := 1;
          when CI_ADMIT =>
            NTYI := 1;
          when REAL_VOLT | IMAG_VOLT =>
            NTE := 1;
          when REAL_CRNT | IMAG_CRNT =>
            NTI := 1;
          when CR_ADMIT =>
            NTYMR := 1;
          when CI_ADMIT =>
            NTYMI := 1;
          when others => null;
        end case;
        if (VISIBLE_COMPONENT = RESISTOR) and (VALUE > 0.0) then
          VALUE := 1.0 / VALUE;
        end if;
        SCREEN_ROW := 3;
        Screen_Supporter.Display_Component (
          COMPONENT  => VISIBLE_COMPONENT,
          HIGH_NODE1 => HIGH_NODE1,
          LOW_NODE1  => LOW_NODE1,
          HIGH_NODE2 => HIGH_NODE2,
          LOW_NODE2  => LOW_NODE2,
          VALUE      => VALUE,
          PHASES     => PHASE,
          ROW        => SCREEN_ROW);
        exit;
      else
        raise COMPONENT_DOES_NOT_EXIST;
      end if;
    exception
      when NODE_PAIR_DOES_NOT_EXIST | COMPONENT_DOES_NOT_EXIST =>
        Errors.Write (MESSAGE => "Such a component does not exists.");
    end;
  end loop;
end Optimise_Component;

-- PURPOSE : Main optimising (tuning) procedure that is called by several
--           others and does the maximising, minimising, setting, etc.

procedure Tune (NODE_MAX : in NODE_TYPE;
  OPTIMISE1      : In     OPTIMISE1_TYPE;
  NETLIST_OBJECT : in out NETLIST_TYPE) is

  IBW,                   NODE1,
  NODE2,                 NODE3,
  NODE4                                : INTEGER_TYPE;

  ABSENT,                ACCURACY,
  FREQ_TUNE,             VALUE         : REAL_TYPE;
  SHOW_CALC                            : BOOLEAN;

  PARAMETER_LIST : LIST_ARRAY_TYPE (1..5) := (
    (TEXT_LENGTH => 7,  TEXT => "Decibel",   STATE => TRUE),
    (TEXT_LENGTH => 9,  TEXT => "Imaginary", STATE => FALSE),
    (TEXT_LENGTH => 9,  TEXT => "Magnitude", STATE => FALSE),
    (TEXT_LENGTH => 5,  TEXT => "Phase",     STATE => FALSE),
    (TEXT_LENGTH => 4,  TEXT => "Real",      STATE => FALSE));

-- PURPOSE : Adjusts the netlist for optimising.

  procedure Optimum_Analysis (F, FREQ_O : in out REAL_TYPE;
    LL, NODE1, NODE2 : in out INTEGER_TYPE;
    OPTIMISE2        : in OPTIMISE2_TYPE;
    RESULTS          : in RESULTS_TYPE) is

    VOLT_REAL,           VOLT_IMAG          : REAL_TYPE;
    ABORT_SUBPROGRAM                        : BOOLEAN := FALSE;
    COUNTER                                 : INTEGER_TYPE;

  begin
    for COUNTER in 1..NN loop
      LL := LPAR (COUNTER);
      if (l - (LL - 1) / 60) /= 0 then
        XX (COUNTER) := abs (XX (COUNTER));
      end if;
      if LL <= 0 then
        FREQ_O := XX (COUNTER);
      elsif OPTIMISE1 /= FIND then
        Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
        COMPONENT := KK (COUNTER);
        while not Inner.Equal_Locations (LOCATION_INDEX => COUNTER,
          NETLIST_OBJECT => NETLIST_OBJECT) loop
          begin
            Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
          exception
            when NO_MORE_BRANCHES => exit;
          end;
        end loop;
        Inner.Read_Netlist (COMPONENT => COMPONENT,
          VALUE       => VALUE1,
          HIGH_NODE1  => HIGH_NODE1,
          LOW_NODE1   => LOW_NODE1,
          HIGH_NODE2  => HIGH_NODE2,
          LOW_NODE2   => LOW_NODE2,
          NETLIST_OBJECT => NETLIST_OBJECT);
        VALUE1 := XX (COUNTER);
        case KK (COUNTER) is
          when MUT_INDCT | COUPLING =>
            begin
              Inner.Calculate_Mutual_Inductance (
                HIGH_NODE1          => HIGH_NODE1,
                LOW_NODE1           => LOW_NODE1,
                HIGH_NODE2          => HIGH_NODE2,
                LOW_NODE2           => LOW_NODE2,
                MUTUAL_INDUCT_VALUE => MUTUAL_VALUE,
                NETLIST_OBJECT      => NETLIST_OBJECT);
              Inner.Find_Mutuals_And_Coupling(COMPONENT=>COMPONENT,
                HIGH_NODE1      => HIGH_NODE1,
                LOW_NODE1       => LOW_NODE1,
                HIGH_NODE2      => HIGH_NODE2,
                LOW_NODE2       => LOW_NODE2,
                MUTUAL_VALUE    => MUTUAL_VALUE,
                VALUE           => VALUE1,
                CREATE          => TRUE,
                NETLIST_OBJECT => NETLIST_OBJECT);
              Inner.BR_Inductance_Matrix (
                NETLIST_OBJECT => NETLIST_OBJECT);
            exception
              when NODE_PAIR_DOES_NOT_EXIST => null;
            end;

          when ACT_INDCT =>
            Maths.Check_Value (MINIMUM=>COMPONENT_VALUE(INDUCT).MIN,
              MAXIMUM => COMPONENT_VALUE (INDUCT).MAX,
              VALUE   => VALUE1);
            XX (COUNTER) := VALUE1;
            Inner.Write_Netlist (COMPONENT => COMPONENT,
              VALUE       => VALUE1,
              HIGH_NODE1  => HIGH_NODE1,
              LOW_NODE1   => LOW_NODE1,
              HIGH_NODE2  => HIGH_NODE2,
```

```
            LOW_NODE2       => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);
         Inner.Update_Mutuals_and_Coupling (VALUE => VALUE1,
            HIGH_NODE1      => HIGH_NODE1,
            LOW_NODE1       => LOW_NODE1,
            HIGH_NODE2      => HIGH_NODE2,
            LOW_NODE2       => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT );
         Inner.HR_Inductance_Matrix (
            NETLIST_OBJECT => NETLIST_OBJECT);

      when CAPS =>
         Maths.Check_Value (MINIMUM => COMPONENT_VALUE(CAPS).MIN,
            MAXIMUM => COMPONENT_VALUE (CAPS).MAX,
            VALUE   => VALUE1);
         XX (COUNTER) := VALUE1;
         Inner.Write_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE1,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);

      when CR_ADMIT =>
         Maths.Check_Value (
            MINIMUM => COMPONENT_VALUE (RESISTOR).MIN,
            MAXIMUM => COMPONENT_VALUE (RESISTOR).MAX,
            VALUE   => VALUE1);
         XX (COUNTER) := VALUE1;
         Inner.Write_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE1,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);

      when CR_ADMIT..CR_ADMIT =>
         Inner.Write_Netlist (COMPONENT => COMPONENT,
            VALUE          => VALUE1,
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);

      when GI_ADMIT =>
         Adjust_Magnitude_Phase (COMPONENT =>'COMPONENT,
            VALUE          => VALUE1,
            MAGNITUDE      => MAGN_Y (COUNTER),
            DEGREE         => ANGLE_Y (COUNTER),
            XC             => XX (COUNTER),
            HIGH_NODE1     => HIGH_NODE1,
            LOW_NODE1      => LOW_NODE1,
            HIGH_NODE2     => HIGH_NODE2,
            LOW_NODE2      => LOW_NODE2,
            NETLIST_OBJECT => NETLIST_OBJECT);

      when REAL_VOLT =>
         if (abs (MAGN_E (COUNTER)) + abs (ANGLE_E (COUNTER)))
            < MINIMUM_VALUE then
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         else
            Adjust_Magnitude_Phase (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               MAGNITUDE      => MAGN_E (COUNTER),
               DEGREE         => ANGLE_E (COUNTER),
               XC             => XX (COUNTER),
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         end if;

      when IMAG_VOLT =>
         if (abs (MAGN_E (COUNTER)) + abs (ANGLE_E (COUNTER)))
            < MINIMUM_VALUE then
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         end if;

      when REAL_CURR =>
         if (abs (MAGN_I (COUNTER)) + abs (ANGLE_I (COUNTER))) <
            MINIMUM_VALUE then
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         else
            Adjust_Magnitude_Phase (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               MAGNITUDE      => MAGN_I (COUNTER),
               DEGREE         => ANGLE_I (COUNTER),
               XC             => XX (COUNTER),
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
         end if;

      when IMAG_CURR =>
         if (abs (MAGN_I (COUNTER)) + abs (ANGLE_I (COUNTER)))
            < MINIMUM_VALUE then
               Inner.Write_Netlist (COMPONENT => COMPONENT,
                  VALUE          => VALUE1,
                  HIGH_NODE1     => HIGH_NODE1,
                  LOW_NODE1      => LOW_NODE1,
                  HIGH_NODE2     => HIGH_NODE2,
                  LOW_NODE2      => LOW_NODE2,
                  NETLIST_OBJECT => NETLIST_OBJECT);
            end if;

         when POLE_FREQ =>
            VALUE1 := abs (VALUE1);
            if VALUE1 > COMPONENT_VALUE (POLE_FREQ).MAX then
               VALUE1 := 0.0;
            elsif VALUE1 < COMPONENT_VALUE (POLE_FREQ).MIN then
               VALUE1 := 1.0 / COMPONENT_VALUE (POLE_FREQ).MIN;
            end if;
            Inner.Write_Netlist (COMPONENT => COMPONENT,
               VALUE          => VALUE1,
               HIGH_NODE1     => HIGH_NODE1,
               LOW_NODE1      => LOW_NODE1,
               HIGH_NODE2     => HIGH_NODE2,
               LOW_NODE2      => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);

         when others => null;
         end case;
      end if;
   end loop;

   Control_Solver.Solve_AC_Network (NODE => RLC,
      AC_FREQUENCY    => FREQ_0,
      CALCULATE       => TRUE,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Control_Solver.Get_Voltages (LOW_NODE => NODE2,
      HIGH_NODE => NODE1,
      REAL_VOLT => VOLT_REAL,
      IMAG_VOLT => VOLT_IMAG);
   F := Math_Lib.Sqrt (
      X => VOLT_REAL * VOLT_REAL + VOLT_IMAG * VOLT_IMAG);

   case RESULTS is
      when MAGNI =>
         F := Math_Lib.Sqrt (
            X => VOLT_REAL * VOLT_REAL + VOLT_IMAG * VOLT_IMAG);
      when REALS =>
         F := VOLT_REAL;
      when DECIB =>
         if F > MIN_F then
            F := 20.0 * Maths.Log (F);
         else
            F := MINIMUM_F;
         end if;
      when IMAGN =>
         F := VOLT_IMAG;
      when PHASES =>
         if (abs (VOLT_REAL) < MINIMUM_VALUE) and
            (abs (VOLT_IMAG) < MINIMUM_VALUE) then
            F := 0.0;
         else
            F := Maths.Angle (X_COORDINATE => VOLT_REAL,
               Y_COORDINATE => VOLT_IMAG) / RADIAN;
         end if;
      when others => null;
   end case;

   case OPTIMISE2 is
      when MAXIN  => F := -F;
      when SETP   => F := abs (VALUE - F);
      when others => null;
   end case;
end Optimum_Analysis;

-- ------------------------------------------------------------------
-- PURPOSE : Optimising using the Simplex algorithm.
-- ------------------------------------------------------------------

procedure Optimising (OPTIMISE1 : in OPTIMISE1_TYPE;
   SHOW_CALC       : in BOOLEAN;
   ACCURACY, ABSERR : in REAL_TYPE;
   NODE1, NODE2     : in INTEGER_TYPE) is

   MAXIMUM_ITERATIONS    : constant := 100;
   MAXIMUM_NEGATIVE_PH2 : constant := -1.0E30;
   MAX_OPTIMUM_P1        : constant := 4;

   type REAL_MATRIX_TYPE is array (1..MAX_OPTIMUM_P1,1..MAX_OPTIMUM_P1)
      of REAL_TYPE;

   S : REAL_MATRIX_TYPE;

   NODES1                                : INTEGER_TYPE := NODE1;
   NODES2                                : INTEGER_TYPE := NODE2;
   OPTIMISE2                             : OPTIMISE2_TYPE;
   SCREEN_ROW                            : INTEGER_TYPE := 8;

   SCREEN2,
   COUNT,                 COUNT1,
   COUNT2,                COUNT3,
   COUNT4,                FNN,
   L2,                    NN_PLUS1       : INTEGER_TYPE;

   ANSWER,                BAND_WIDTH,
   BURP,                  ERROR1,
   F,                     F1,
   F0,                    F02,
   F_LOWER,               F_CENTRE,
   F_UPPER,               SUM,
   TEN_F,
   VOLT_REAL,             VOLT_IMAG      : REAL_TYPE;
   FREQ_SAVE,             SAVE_F         : REAL_TYPE := FREQ_TUNE;

   ABORT_OP,              CIRCUIT_OPTIMISED,
   DONE,                  MORE_ITERATIONS,
   NO_ERRORS                             : BOOLEAN;

begin
   S := (1..(MAX_OPTIMUM_VARIABLE + 1) => (1..(MAX_OPTIMUM_VARIABLE+1)
      => 0.0));
   SC := (1..MAX_OPTIMUM_VARIABLE => 0.0);
   FNN := NN;
```

```
      Cursor_Next_Line;
      Cursor_Set (ROW => 3, COLUMN => 3);
      case OPTIMISE1 is
         when FIND =>
            Write (ITEM => "Node Diff to Find Bandwidth are");
            Write (ITEM => NODES1, ITEM_WIDTH => 3);
            Write (ITEM => " -");
            Write (ITEM => NODES2, ITEM_WIDTH => 3);
            Cursor_Next_Line;
            Write (ITEM => "  Accuracy will be within");
            Write (ITEM => ACCURACY * 100.0,
               WHOLE         => 4,
               FRACTION      => 5);
            Write (ITEM => "  %");
            OPTIMISE2 := MAXIN;
            RESULTS := Decib;
         when MINI =>
            Write (ITEM => "Node Diff to Minimise are");
            Write (ITEM => NODES1, ITEM_WIDTH => 3);
            Write (ITEM => " -");
            Write (ITEM => NODES2, ITEM_WIDTH => 3);
            Cursor_Next_Line;
            Write (ITEM => "  Error test is");
            Write (ITEM => ABSERR,
               WHOLE         => 4,
               FRACTION      => 5);
            OPTIMISE2 := MININ;
         when MAXI =>
            Write (ITEM => "Node Diff to Maximise are");
            Write (ITEM => NODES1, ITEM_WIDTH => 3);
            Write (ITEM => " -");
            Write (ITEM => NODES2, ITEM_WIDTH => 3);
            Cursor_Next_Line;
            Write (ITEM => "  Accuracy will be within");
            Write (ITEM => ACCURACY * 100.0,
               WHOLE         => 4,
               FRACTION      => 5);
            Write (ITEM => "  %");
            OPTIMISE2 := MAXIN;
         when SETO =>
            Write (ITEM => "Node Diff to SET are");
            Write (ITEM => NODES1, ITEM_WIDTH => 3);
            Write (ITEM => " -");
            Write (ITEM => NODES2, ITEM_WIDTH => 3);
            Cursor_Next_Line;
            Write (ITEM => "  Accuracy will be within");
            Write (ITEM => ACCURACY * 100.0,
               WHOLE         => 4,
               FRACTION      => 5);
            Write (ITEM => "  %");
            OPTIMISE2 := SETT;
         when others => null;
      end case;
      Cursor_Next_Line;
      Cursor_Set (ROW => 7, COLUMN => 10);
      Write (ITEM => "    Count      Value          Burp        Error "
         & "Value     ",
         FORM => UNDERLINED);
      Cursor_Next_Line;
      Status.Write (MESSAGE => "Note : Strike any key to stop the " &
         "optimisation. Press the Ok button to continue.");
      if NODES2 < 0 then
         NODES2 := -NODES2;
      end if;

      loop
         NN_PLUS1 := NN + 1;
         for COUNT1 in 1..NN loop
            for COUNT2 in 1..NN_PLUS1 loop
               S (COUNT1, COUNT2) := XX (COUNT1);
            end loop;
            S (COUNT1, COUNT1) := XX (COUNT1) + SRTURN (COUNT1);
         end loop;
         COUNT2 := 1;

         loop
            for COUNT1 in 1..NN loop
               XX (COUNT1) := S (COUNT1, COUNT2);
            end loop;
            Optimum_Analysis (F => F,
               FREQ_O      => FREQ_SAVO,
               LL          => LL,
               NODE1       => NODES1,
               NODE2       => NODES2,
               OPTIMISE2 => OPTIMISE2,
               RESULTS     => RESULTS);
            Central_Solver.Get_Voltages (LOW_NODE => NODES2,
               HIGH_NODE => NODES1,
               REAL_VOLT => VOLT_REAL,
               IMAG_VOLT => VOLT_IMAG);
            ABORT_OP := (abs(VOLT_REAL) + abs(VOLT_IMAG)) < MINIMUM_VALUE;
            if not ABORT_OP then
               S (NN_PLUS1, COUNT2) := F;
            end if;
            COUNT2 := COUNT2 + 1;
            exit when (COUNT2 > NN_PLUS1) or ABORT_OP;
         end loop;

         if ABORT_OP then
            Errors.Write (MESSAGE => "The node differential voltage is " &
               "too small.",
               HELP_INFO => HP_OPTIMISING);
         else
            loop
               COUNT := 1;
               loop
                  FL := S (NN_PLUS1, 1);
                  FH := FL;
                  FH2 := MAXIMUM_NEGATIVE_FH2;
                  COUNT3 := 1;
                  COUNT4 := 1;
                  for COUNT2 in 2..NN_PLUS1 loop
                     BURP := S (NN_PLUS1, COUNT2);
                     if FH < BURP then
                        FH2 := FH;
                        FH  := BURP;
                        COUNT3 := COUNT2;
                     else
                        if (FH2 < BURP) then
                           FH2 := BURP;
```

```
                        end if;
                        if (FL > BURP) then
                           FL := BURP;
                           COUNT4 := COUNT2;
                        end if;
                     end if;
                  end loop;
                  CIRCUIT_OPTIMISED := FALSE;
                  case OPTIMISE2 is
                     when MAXIN =>
                        ERRVAL := ACCURACY * F;
                        if abs (FH - FL) < abs (ACCURACY * F) then
                           CIRCUIT_OPTIMISED := TRUE;
                        end if;
                     when SETT =>
                        ERRVAL := ACCURACY * VALUE;
                        if abs (F) <= abs (ACCURACY * VALUE) then
                           CIRCUIT_OPTIMISED := TRUE;
                        end if;
                     when MININ =>
                        ERRVAL := ABSERR;
                        if (FH - FL - ABSERR) < 0.0 then
                           CIRCUIT_OPTIMISED := TRUE;
                        end if;
                     when others => null;
                  end case;
                  if Show_Calc then
                     Cursor_Set (ROW => SCREEN_ROW, COLUMN => 14);
                     Write (ITEM => COUNT, ITEM_WIDTH => 3);
                     Write (ITEM => F,
                        WHOLE         => 5,
                        FRACTION      => 5,
                        ENG_NOTATION => TRUE);
                     Write (ITEM => BURP,
                        WHOLE         => 4,
                        FRACTION      => 5,
                        ENG_NOTATION => TRUE);
                     Write (ITEM => ERRVAL,
                        WHOLE         => 4,
                        FRACTION      => 5,
                        ENG_NOTATION => TRUE);
                     Cursor_Next_Line;
                  end if;
                  if (abs (FH) < MINIMUM_VALUE) and
                     (abs (FL) < MINIMUM_VALUE) then
                     Errors.Write (MESSAGE =>"No solution can be found.");
                     SCREEN_ROW := SCREEN_ROW + 1;
                     COUNT := MAXIMUM_ITERATIONS + 1;  -- Error exit
                  end if;
                  if not CIRCUIT_OPTIMISED then
                     -- Compute the centroid
                     for COUNT1 in 1..NN loop
                        SUM := 0.0;
                        for COUNT2 in 1..NN_PLUS1 loop
                           SUM := SUM + S (COUNT1, COUNT2);
                        end loop;
                        SC (COUNT1) := (SUM - S (COUNT1, COUNT3)) /
                           REAL_TYPE (FNN);
                     end loop;
                     -- Reflect the Simplex
                     for COUNT1 in 1..NN loop
                        XX (COUNT1) := 2.0 * abs (SC (COUNT1)) -
                           S (COUNT1, COUNT3);
                     end loop;
                     Optimum_Analysis (F => F,
                        FREQ_O      => FREQ_SAVO,
                        LL          => LL,
                        NODE1       => NODES1,
                        NODE2       => NODES2,
                        OPTIMISE2 => OPTIMISE2,
                        RESULTS     => RESULTS);
                     DONE := FALSE;
                     if (F < FL) and not DONE then
                        for COUNT1 in 1..NN loop
                           BURP := XX (COUNT1);
                           S (COUNT1, COUNT3) := BURP;
                           XX (COUNT1) := BURP + BURP - SC (COUNT1);
                        end loop;
                        S (NN_PLUS1, COUNT3) := F;
                        FL := F;
                        Optimum_Analysis (F => F,
                           FREQ_O      => FREQ_SAVO,
                           LL          => LL,
                           NODE1       => NODES1,
                           NODE2       => NODES2,
                           OPTIMISE2 => OPTIMISE2,
                           RESULTS     => RESULTS);
                        if F < FL then
                           for COUNT1 in 1..NN loop
                              S (COUNT1, COUNT3) := XX (COUNT1);
                           end loop;
                           S (NN_PLUS1, COUNT3) := F;
                        end if;
                        DONE := TRUE;
                     end if;
                     if (F >= FL) and (F < FH2) and not DONE then
                        for COUNT1 in 1..NN loop
                           S (COUNT1, COUNT3) := XX (COUNT1);
                        end loop;
                        S (NN_PLUS1, COUNT3) := F;
                        DONE := TRUE;
                     end if;
                     if (F >= FL) and (F >= FH2) and not DONE then
                        if F < FH then
                           for COUNT1 in 1..NN loop
                              S (COUNT1, COUNT3) := XX (COUNT1);
                           end loop;
                           S (NN_PLUS1, COUNT3) := F;
                           FH := F;
                        end if;
                        -- Contract the Simplex
                        for COUNT1 in 1..NN loop
                           XX (COUNT1) := 0.5 *
                              (S (COUNT1, COUNT3) + SC (COUNT1));
                        end loop;
                        Optimum_Analysis (F => F,
                           FREQ_O      => FREQ_SAVO,
                           LL          => LL,
                           NODE1       => NODES1,
                           NODE2       => NODES2,
```

```
                    OPTIMISE2 => OPTIMISE2,
                    RESULTS   => RESULTS);
                 if F >= FM then
                    for COUNT2 in 1..MM_PLUS1 loop
                       if COUNT2 /= COUNT4 then
                          -- shrink the simplex about the lowest value
                          for COUNT1 in 1..MM loop
                             BURP := 0.5 * (S (COUNT1, COUNT2)
                                + S (COUNT1, COUNT4));
                             S (COUNT1, COUNT2) := BURP;
                             XX (COUNT1) := BURP;
                          end loop;
                          Optimum_Analysis (F => F,
                             FREQ_O    => FREQ_SAVE,
                             LL        => LL,
                             NODE1     => NODES1,
                             NODE2     => NODES2,
                             OPTIMISE2 => OPTIMISE2,
                             RESULTS   => RESULTS);
                          S (MM_PLUS1, COUNT2) := F;
                       end if;
                    end loop;
                 else
                    for COUNT1 in 1..MM loop
                       S (COUNT1, COUNT3) := XX (COUNT1);
                    end loop;
                    S (MM_PLUS1, COUNT3) := F;
                 end if;
                 DONE := TRUE;
              end if;
           end if;
           COUNT := COUNT + 1;
           exit when (COUNT > MAXIMUM_ITERATIONS) or
              CIRCUIT_OPTIMISED or Halt_Processing;
        end loop;
        if not CIRCUIT_OPTIMISED then
           Prompts.Read (MESSAGE => "No Convergence. Do you want "
              & "more iterations ?",
              HELP_INFO => HP_OPTIMISING2,
              ITEM      => MORE_ITERATIONS);
        end if;
        exit when CIRCUIT_OPTIMISED or not MORE_ITERATIONS;
     end loop;
     if not CIRCUIT_OPTIMISED and not MORE_ITERATIONS then
        IGN := 1;
     end if;

     if OPTIMISE1 = FIND then
        case IGN is
           when 1 =>
              Cursor_Next_Line;
              F_UPPER := S (1, COUNT4);
              BAND_WIDTH := abs (F_UPPER - F_LOWER);
              Cursor_Set (ROW => SCREEN_ROW + 2, COLUMN => 3);
              Write (ITEM => "  Centre Frequency    =");
              Write (ITEM => F_CENTRE,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Write (ITEM => " Hz    ");
              Write (ITEM => ANSWER,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Write (ITEM => " dBis");
              Cursor_Next_Line;
              if F_LOWER > F_UPPER then
                 TEMP_F  := F_LOWER;
                 F_LOWER := F_UPPER;
                 F_UPPER := TEMP_F;
              end if;
              Write (ITEM => "  Lower 3db Frequency =");
              Write (ITEM => F_LOWER,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Cursor_Next_Line;
              Write (ITEM => "  Upper 3db Frequency =");
              Write (ITEM => F_UPPER,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Cursor_Next_Line;
              Write (ITEM => "  Band Width          =");
              Write (ITEM => BAND_WIDTH,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Cursor_Next_Line;
              Write (ITEM => "  Q Factor            =");
              Write (ITEM => F_CENTRE / BAND_WIDTH,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              Cursor_Next_Line;
              if COUNT > MAXIMUM_ITERATIONS then
                 Inner.Write_Frequency (FREQUENCY => SAVE_F,
                    NETLIST_OBJECT => NETLIST_OBJECT);
              else
                 Inner.Write_Frequency (FREQUENCY => F_CENTRE,
                    NETLIST_OBJECT => NETLIST_OBJECT);
              end if;

           when 0 =>
              F_CENTRE := S (1, COUNT4);
              ANSWER := -S (2, COUNT4);
              VALUE := ANSWER - 3.0;
              PERTURB (1) := 2.0E-2 * F_CENTRE;
              XX (1) := 0.0E-1 * F_CENTRE;
              IGN := -1;
              OPTIMISE2 := SETT;
              Cursor_Set (ROW => 22, COLUMN => 3);
              Write (ITEM => "Found centre frequency.");
              Cursor_Next_Line;
              if COUNT < MAXIMUM_ITERATIONS then
                 MORE_ITERATIONS := TRUE;
              end if;
              CIRCUIT_OPTIMISED := False;

           when -1 =>
              F_LOWER := S (1, COUNT4);
              BAND_WIDTH := F_CENTRE - F_LOWER;
              XX (1) := F_CENTRE + BAND_WIDTH;
              if XX (1) < 0.0 then
                 XX (1) := 1.0E-1 * F_CENTRE;
              end if;
              PERTURB (1) := -2.0E-2 * BAND_WIDTH;
              Cursor_Set (ROW => 22, COLUMN => 3);
              Write (ITEM => "Found lower frequency. ");
              Cursor_Next_Line;
              IGN := 1;
              CIRCUIT_OPTIMISED := False;
              if COUNT < MAXIMUM_ITERATIONS then
                 MORE_ITERATIONS := TRUE;
              end if;
           when others => null;
        end case;
     else
        Cursor_Next_Line;
        Cursor_Set (ROW => SCREEN_ROW + 2, COLUMN => 3);
        Write (ITEM => "The tuned components now values are :");
        Cursor_Next_Line;
        Write (ITEM => Screen_Supporter.HEADING1,
           FORM => UNDERLINED);
        for COUNT in 1..MM loop
           ANSWER := S (COUNT, COUNT4);
           if LEAK (COUNT) < 6 then
              Cursor_Next_Line;
              Write (ITEM => "  Adjusted Frequency =");
              Write (ITEM => ANSWER,
                 WHOLE      => 4,
                 FRACTION   => 5,
                 ENG_NOTATION => TRUE);
              SCREEN_ROW := SCREEN_ROW + 1;
              if COUNT > MAXIMUM_ITERATIONS then
                 Inner.Write_Frequency (FREQUENCY => SAVE_F,
                    NETLIST_OBJECT => NETLIST_OBJECT);
              else
                 Inner.Write_Frequency (FREQUENCY => ANSWER,
                    NETLIST_OBJECT => NETLIST_OBJECT);
              end if;
           else
              Inner.First_Branch (NETLIST_OBJECT=>NETLIST_OBJECT);
              NO_ERRORS := TRUE;
              while NO_ERRORS and not Inner.Equal_Locations (
                 LOCATION_INDEX => COUNT,
                 NETLIST_OBJECT => NETLIST_OBJECT) loop
                 begin
                    Inner.Next_Branch (
                       NETLIST_OBJECT => NETLIST_OBJECT);
                 exception
                    when NO_MORE_BRANCHES =>
                       NO_ERRORS := FALSE;
                 end;
              end loop;
              if NO_ERRORS then
                 Netlists.Inner.Read_Netlist (COMPONENT=>MX(COUNT),
                    VALUE       => VALUE1,
                    HIGH_NODE1  => HIGH_NODE1,
                    LOW_NODE1   => LOW_NODE1,
                    HIGH_NODE2  => HIGH_NODE2,
                    LOW_NODE2   => LOW_NODE2,
                    NETLIST_OBJECT => NETLIST_OBJECT);
              end if;
              if (abs (MAGN_E (COUNT)) + abs (MAGN_X (COUNT))
                 + abs (MAGN_Y (COUNT)) + abs (ANGLE_E (COUNT))
                 + abs (ANGLE_X (COUNT)) + abs (ANGLE_Y (COUNT)))
                 < MINIMUM_VALUE then
                 case MX (COUNT) is
                    when CAPS | GR_ADMIT..COUPLING =>
                       Inner.Write_Netlist (COMPONENT => MX(COUNT),
                          VALUE       => ANSWER,
                          HIGH_NODE1  => HIGH_NODE1,
                          LOW_NODE1   => LOW_NODE1,
                          HIGH_NODE2  => HIGH_NODE2,
                          LOW_NODE2   => LOW_NODE2,
                          NETLIST_OBJECT => NETLIST_OBJECT);
                       if (MX (COUNT)= GR_ADMIT) and
                          (CIRCUIT_MODE = RLC) and
                          (VALUE1 > 0.0) then
                          ANSWER := 1.0 / ANSWER;
                          VISIBLE_COMPONENT := RESISTOR;
                       else
                          VISIBLE_COMPONENT := MX (COUNT);
                       end if;
                       SCREEN2 := SCREEN_ROW + 4;
                       SCREEN_ROW := SCREEN_ROW + 1;
                       Screen_Supporter.Display_Component (
                          COMPONENT  => VISIBLE_COMPONENT,
                          HIGH_NODE1 => HIGH_NODE1,
                          LOW_NODE1  => LOW_NODE1,
                          HIGH_NODE2 => HIGH_NODE2,
                          LOW_NODE2  => LOW_NODE2,
                          VALUE      => ANSWER,
                          PHASES     => PHASE,
                          ROW        => SCREEN2);
                    when others => null;
                 end case;
              else
                 COMPONENT := MX (COUNT);
                 case MX (COUNT) is
                    when REAL_VOLT =>
                       Adjust_Magnitude_Phase(
                          COMPONENT     => COMPONENT,
                          VALUE         => VALUE1,
                          MAGNITUDE     => MAGN_E (COUNT),
                          DEGREE        => ANGLE_E (COUNT),
                          XC            => ANSWER,
                          HIGH_NODE1    => HIGH_NODE1,
                          LOW_NODE1     => LOW_NODE1,
                          HIGH_NODE2    => HIGH_NODE2,
                          LOW_NODE2     => LOW_NODE2,
                          NETLIST_OBJECT => NETLIST_OBJECT);

                    when REAL_CAPT =>
                       Adjust_Magnitude_Phase (
                          COMPONENT     => COMPONENT,
                          VALUE         => VALUE1,
                          MAGNITUDE     => MAGN_X (COUNT),
                          DEGREE        => ANGLE_X (COUNT),
```

```
                        XC              => ANSWER,
                        HIGH_NODE1      => HIGH_NODE1,
                        LOW_NODE1       => LOW_NODE1,
                        HIGH_NODE2      => HIGH_NODE2,
                        LOW_NODE2       => LOW_NODE2,
                        NETLIST_OBJECT  => NETLIST_OBJECT);

                  when GI_ADMIT =>
                     Adjust_Magnitude_Phase (
                        COMPONENT       => COMPONENT,
                        VALUE           => VALUE1,
                        MAGNITUDE       => MAGN_Y (COUNT),
                        DEGREE          => ANGLE_Y (COUNT),
                        XC              => ANSWER,
                        HIGH_NODE1      => HIGH_NODE1,
                        LOW_NODE1       => LOW_NODE1,
                        HIGH_NODE2      => HIGH_NODE2,
                        LOW_NODE2       => LOW_NODE2,
                        NETLIST_OBJECT  => NETLIST_OBJECT);

                  when others => null;
               end case;
               if (NX (COUNT) = REAL_VOLT) or
                  (NX (COUNT) = REAL_CURT) or
                  (NX (COUNT) = GI_ADMIT) then
                  Cursor_Set (ROW => SCREEN_ROW, COLUMN => 2);
                  SCREEN_ROW := SCREEN_ROW + 1;
                  if (abs (MAGN_X (COUNT)) + abs (MAGN_Y (COUNT))
                     + abs (MAGN_Y (COUNT)) > MININUM_VALUE then
                     Write (ITEM => "Adjusting Phase");
                  end if;
                  if (abs (ANGLE_X(COUNT)) + abs (ANGLE_Y(COUNT))
                     + abs (ANGLE_Y(COUNT)) > MININUM_VALUE then
                     Write (ITEM => "Adjusting Magnitude");
                  end if;
                  Cursor_Next_Line;
                  COMPONENT := NX (COUNT);
                  Inner.Read_Netlist (COMPONENT => COMPONENT,
                     VALUE          => TEMP_VALUE1,
                     HIGH_NODE1     => HIGH_NODE1,
                     LOW_NODE1      => LOW_NODE1,
                     HIGH_NODE2     => HIGH_NODE2,
                     LOW_NODE2      => LOW_NODE2,
                     NETLIST_OBJECT => NETLIST_OBJECT);
                  COMPONENT := ELEMENTS'succ (COMPONENT);
                  Inner.Read_Netlist (COMPONENT => COMPONENT,
                     VALUE          => VALUE1,
                     HIGH_NODE1     => HIGH_NODE1,
                     LOW_NODE1      => LOW_NODE1,
                     HIGH_NODE2     => HIGH_NODE2,
                     LOW_NODE2      => LOW_NODE2,
                     NETLIST_OBJECT => NETLIST_OBJECT);
                  if (abs (TEMP_VALUE1) + abs (VALUE1))
                     > MININUM_VALUE then
                     PHASE := Maths.Angle (
                        X_COORDINATE => TEMP_VALUE1,
                        Y_COORDINATE => VALUE1);
                     VALUE1 := Math_Lib.Sqrt (
                        TEMP_VALUE1*TEMP_VALUE1 + VALUE1*VALUE1);
                     case COMPONENT is
                        when IMAG_VOLT =>
                           VISIBLE_COMPONENT := VOLT_SOURCE;
                        when REAL_CURT =>
                           VISIBLE_COMPONENT := CURT_SOURCE;
                        when GI_ADMIT =>
                           VISIBLE_COMPONENT := RESISTOR;
                        when others =>
                           VISIBLE_COMPONENT := COMPONENT;
                     end case;
                     SCREEN2 := SCREEN_ROW + 4;
                     Screen_Supporter.Display_Component (
                        COMPONENT  => VISIBLE_COMPONENT,
                        HIGH_NODE1 => HIGH_NODE1,
                        LOW_NODE1  => LOW_NODE1,
                        HIGH_NODE2 => HIGH_NODE2,
                        LOW_NODE2  => LOW_NODE2,
                        VALUE      => VALUE1,
                        PHASE      => PHASE,
                        ROW        => SCREEN2);
                  end if;
               end if;
            end if;
         end if;
      end if;
   end loop;
   case OPTIMISE2 is
      when MAXIM  => F := -F;
      when SETT   => F := VALUE - F;
      when others => null;
   end case;
   Central_Solver.Get_Voltages (LOW_NODE => NODE2,
      HIGH_NODE => NODE1,
      REAL_VOLT => VOLT_REAL,
      IMAG_VOLT => VOLT_IMAG);
   F := Maths.Find_Magnitude (REAL_VALUE => VOLT_REAL,
      IMAG_VALUE => VOLT_IMAG);
   Cursor_Next_Line;
   Cursor_Set (ROW => 24, COLUMN => 3);
   Write (ITEM => "V ");
   Write (ITEM => NODE1, ITEM_WIDTH => 3);
   Write (ITEM => " -");
   Write (ITEM => NODE2, ITEM_WIDTH => 3);

   case RESULTS is
      when MAGNI =>
         F := Maths.Find_Magnitude (REAL_VALUE => VOLT_REAL,
            IMAG_VALUE => VOLT_IMAG);
      when REALS =>
         F := VOLT_REAL;
      when DECIB =>
         if F > MIN_F then
            F := 20.0 * Maths.Log (F);
         else
            F := MININUM_F;
         end if;
      when IMAGN =>
         F := VOLT_IMAG;
      when PHASES =>
         if (abs (VOLT_REAL) < MININUM_VALUE) and
            (abs (VOLT_IMAG) < MININUM_VALUE) then
            F := 0.0;
         else
            F := Maths.Angle (X_COORDINATE => VOLT_REAL,
               Y_COORDINATE => VOLT_IMAG) / RADIAN;
         end if;
      when others => null;
   end case;

   case RESULTS is
      when MAGNI | REALS | IMAGN =>
         Write (ITEM => F,
            WHOLE       => 4,
            FRACTION    => 5,
            ENG_NOTATION => TRUE));
         Write (ITEM => " Volt");
      when DECIB =>
         Write (ITEM => F,
            WHOLE       => 4,
            FRACTION    => 5,
            ENG_NOTATION => TRUE);
         Write (ITEM => " Decibels");
      when PHASES =>
         Write (ITEM => "   PHASE =");
         Write (ITEM => F * RADIAN,
            WHOLE    => 4,
            FRACTION => 5,
            EXPONENT => 0);
         Write (ITEM => " Degrees");
      when others => null;
   end case;
end if;
exit when ((OPTIMISE1 = FIND) and CIRCUIT_OPTIMISED and (ISW=1))
   or ABORT_OP or (not CIRCUIT_OPTIMISED and not MORE_ITERATIONS)
   or (CIRCUIT_OPTIMISED and (OPTIMISE1 in MAXI..SETO));
end loop;
end Optimising;
```

```
begin -- Tune
   Process_Title (TITLE => "OPTIMISE");
   if (OPTIMISE1 in MINI..SETO) and ((NTR + NTF + NTI + NTE) = 0) then
      Errors.Write (MESSAGE => "There are no tuned variables.");
   else
      OPTIMISE_FORM1 (3).ACCEPTING := FALSE;
      OPTIMISE_FORM1 (4).ACCEPTING := FALSE;
      case OPTIMISE1 is
         when MAXI | SETO | FIND =>
            OPTIMISE_FORM1 (6).ACCEPTING := TRUE;
            OPTIMISE_FORM1 (7).ACCEPTING := FALSE;
         when MINI =>
            OPTIMISE_FORM1 (6).ACCEPTING := FALSE;
            OPTIMISE_FORM1 (7).ACCEPTING := TRUE;
         when others => null;
      end case;
      Prompts.Read (FORM => OPTIMISE_FORM1,
         LIST        => OPTIMISE_LIST1,
         EXCLUSIVE => FALSE,
         HELP_INFO => HP_TUNE);
      NODE1      := OPTIMISE_FORM1 (1).INTEGER_VALUE;
      NODE2      := OPTIMISE_FORM1 (2).INTEGER_VALUE;
      FREQ_TUNE := OPTIMISE_FORM1 (5).REAL_VALUE;
      SHOW_CALC := OPTIMISE_LIST1 (1).STATE;
      if NODE1 < NODE2 then
         Maths.Swop_Node (NODE1 => NODE1,
            NODE2 => NODE2);
      end if;
      case OPTIMISE1 is
         when MAXI | SETO | FIND =>
            ACCURACY := OPTIMISE_FORM1 (6).REAL_VALUE;
            ACCURACY := 1.0 - 1.0E-2 * ACCURACY;
         when MINI =>
            ABSERR := OPTIMISE_FORM1 (7).REAL_VALUE;
         when others => null;
      end case;
      if OPTIMISE1 in MAXI..SETO then
         Prompts.Read (LIST => PARAMETER_LIST,
            HELP_INFO => HP_TUNE2);
         for COUNTER in 1..PARAMETER_LIST'last loop
            if PARAMETER_LIST (COUNTER).STATE then
               RESULTS := RESULTS_TYPE'val (COUNTER - 1);
               exit;
            end if;
         end loop;
      end if;

      case OPTIMISE1 is
         when FIND =>
            IDN := 0;
            NN := 1;
            XX (NN) := FREQ_TUNE;
            PERTURB (NN) := 0.1 * XX (NN);
            LVAR (NN) := -1;
            Process_Title (TITLE => "FINDING THE BANDWIDTH");
         when MAXI =>
            Process_Title (TITLE => "MAXIMISING");
         when MINI =>
            Process_Title (TITLE => "MINIMISING");
         when SETO =>
            Prompts.Read (MESSAGE => "Enter the value to be set.",
               ITEM => VALUE);
            if RESULTS = PHASES then
               VALUE := VALUE / RADIAN;
            end if;
            Process_Title (TITLE => "SETTING");
         when others => null;
      end case;
      Optimising (OPTIMISE1 => OPTIMISE1,
         SHOW_CALC => SHOW_CALC,
         ACCURACY  => ACCURACY,
         ABSERR    => ABSERR,
         NODE1     => NODE1,
         NODE2     => NODE2);
   end if;
end Tune;
```

```
-- PURPOSE : Clears all the optimisation variables if the user agrees.
```

```
procedure Clear (NETLIST_OBJECT : in out NETLIST_TYPE) is

   RESPONSE : BOOLEAN;

begin
   Prompts.Read (MESSAGE => "Do you want to clear the Optimisation "
      & "Variables ?",
      ITEM => RESPONSE);
   if RESPONSE then
      Clear_Tune (NETLIST_OBJECT => NETLIST_OBJECT);
   end if;
end Clear;

-- PURPOSE : Finds the bandwidth, upper, centre and lower frequencies, as
--           well as maximum gain and the Q-factor of a bandpass response
--           circuit.

procedure Find_Bandwidth (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   Tune (NODE_MAX => NODE_MAX,
      OPTIMISE1      => FIND,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Clear_Tune (NETLIST_OBJECT => NETLIST_OBJECT);
end Find_Bandwidth;

-- PURPOSE : Defines the variables that are to be used in the optimisation
--           process : the component's value, it's magnitude or phase,  or
--           the operating frequency.

procedure Insert (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

   INSERT_LIST : LIST_ARRAY_TYPE (1..3) := (
      (TEXT_LENGTH => 9,   TEXT => "Component",         STATE => TRUE),
      (TEXT_LENGTH => 18,  TEXT => "Magnitude or Phase", STATE => FALSE),
      (TEXT_LENGTH => 19,  TEXT => "Operating Frequency", STATE =>FALSE));

   COUNTER        : NATURAL;
   FREQ_MAIN_TUNE : REAL_TYPE;

begin
   if NN >= MAX_OPTIMUM_VARIABLE then
      Errors.Write (MESSAGE => "The maximum number of tuned variables "
         & "has already been selected.");
   else
      Process_Title (TITLE => "TUNE VARIABLES");
      Prompts.Read (LIST => INSERT_LIST,
         HELP_INFO => HP_OPTIN_INSERT);
      for COUNTER in 1..INSERT_LIST'last loop
         if INSERT_LIST (COUNTER).STATE then
            case COUNTER is
               when 1 =>
                  Optimise_Component (NODE_MAX => NODE_MAX,
                     NETLIST_OBJECT => NETLIST_OBJECT);
               when 2 =>
                  Change_Optimise (NETLIST_OBJECT => NETLIST_OBJECT);
               when 3 =>
                  if NTF = 1 then
                     Errors.Write (MESSAGE => "The Frequency is already"
                        & " selected.",
                        HELP_INFO => HP_OPTIN_INSERT2);
                  else
                     Outer.Get_Frequency (
                        FREQUENCY      => FREQ_MAIN_TUNE,
                        NETLIST_OBJECT => NETLIST_OBJECT);
                     NN := NN + 1;
                     XX (NN) := FREQ_MAIN_TUNE;
                     RTUNE (NN) := DELTA_INCREMENT * XI (NN);
                     LPAR (NN) := -1;
                     NTF := 1;
                     Cursor_Next_Line;
                     Cursor_Set (ROW => 3, COLUMN => 5);
                     Write ("Tuning Frequency, starting at");
                     Write (ITEM => FREQ_MAIN_TUNE,
                        FRACTION      => 5,
                        ENG_NOTATION => TRUE);
                     Write (ITEM => " Hz");
                     Cursor_Set (ROW => 4, COLUMN => 3);
                  end if;
               when others => null;
            end case;
            exit;
         end if;
      end loop;
   end if;
end Insert;

-- PURPOSE : Maximises  the response  of a  specific  node pair  by tuning
--           selected component values and / or frequencies.

procedure Maximise (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   Tune (NODE_MAX => NODE_MAX,
      OPTIMISE1      => MAXI,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Clear_Tune (NETLIST_OBJECT => NETLIST_OBJECT);
end Maximise;

-- PURPOSE : Minimises  the response  of a  specific  node pair  by tuning
--           selected component values and / or frequencies.

procedure Minimise (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   Tune (NODE_MAX => NODE_MAX,
      OPTIMISE1      => MINI,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Clear_Tune (NETLIST_OBJECT => NETLIST_OBJECT);
end Minimise;

-- PURPOSE : Tunes  component  values  and / or  frequency  so  that  the
--           response of the selected node pair  will be set at a specific
--           value.

procedure Set (NODE_MAX : in NODE_TYPE;
   NETLIST_OBJECT : in out NETLIST_TYPE) is

begin
   Tune (NODE_MAX => NODE_MAX,
      OPTIMISE1      => SETO,
      NETLIST_OBJECT => NETLIST_OBJECT);
   Clear_Tune (NETLIST_OBJECT => NETLIST_OBJECT);
end Set;

-- PURPOSE : Displays  the list  of variables  that are to be used  in the
--           optimisation.

procedure Show (NETLIST_OBJECT : in out NETLIST_TYPE) is

   SCREEN_ROW                    : POSITIVE := 3;
   NO_ERRORS                     : BOOLEAN;
   COMPONENT,        VISIBLE_COMPONENT : ELEMENT_TYPE;
   COUNT                         : INTEGER_TYPE;

begin
   Process_Title (TITLE => "TUNE VARIABLES");
   Write (ITEM => Screen_Supporter.HEADING1,
      FORM => UNDERLINED);
   Cursor_Ne t_Line;
   for COUNT in 1..NN loop
      Inner.First_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
      if LPAR (COUNT) > 0 then
         COMPONENT := MX (COUNT);
         while not Inner.Equal_Locations (LOCATION_INDEX => COUNT,
            NETLIST_OBJECT => NETLIST_OBJECT) loop
            begin
               Inner.Next_Branch (NETLIST_OBJECT => NETLIST_OBJECT);
            exception
               when NO_MORE_BRANCHES => exit;
            end;
         end loop;
         begin
            Inner.Read_Netlist (COMPONENT => COMPONENT,
               VALUE       => VALUE1,
               HIGH_NODE1  => HIGH_NODE1,
               LOW_NODE1   => LOW_NODE1,
               HIGH_NODE2  => HIGH_NODE2,
               LOW_NODE2   => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            NO_ERRORS := TRUE;
         exception
            when BRANCH_OUT_OF_RANGE =>
               NO_ERRORS := FALSE;
         end;
         if (COMPONENT = GR_ADMIT) and (CIRCUIT_MODE = RLC) and
            (VALUE1 > 0.0) then
            VALUE1 := 1.0 / VALUE1;
            VISIBLE_COMPONENT := RESISTOR;
         else
            VISIBLE_COMPONENT := COMPONENT;
         end if;
         if ((COMPONENT = REAL_VOLT) or (COMPONENT = REAL_CRNT) or
            (COMPONENT = GI_ADMIT)) and
            ((abs (MAGN_E (COUNT)) + abs (MAGN_I (COUNT))
            + abs (MAGN_Y (COUNT)) + abs (ANGLE_E (COUNT))
            + abs (ANGLE_I (COUNT)) + abs (ANGLE_Y (COUNT)))
            > MINIMUM_VALUE) then
            Cursor_Set (ROW => SCREEN_ROW, COLUMN => 3);
            if (abs (MAGN_E (COUNT)) + abs (MAGN_I (COUNT))
               + abs (MAGN_Y (COUNT))) > MINIMUM_VALUE then
               Write (ITEM => "Adjusting Phase");
            end if;
            if (abs (ANGLE_E (COUNT)) + abs (ANGLE_I (COUNT))
               + abs (ANGLE_Y (COUNT))) > MINIMUM_VALUE then
               Write (ITEM => "Adjusting Magnitude");
            end if;
            SCREEN_ROW := SCREEN_ROW + 1;
            Cursor_Next_Line;
            TEMP_VALUE1 := VALUE1;
            Inner.Read_Netlist (COMPONENT => ELEMENTS'succ (COMPONENT),
               VALUE       => VALUE1,
               HIGH_NODE1  => HIGH_NODE1,
               LOW_NODE1   => LOW_NODE1,
               HIGH_NODE2  => HIGH_NODE2,
               LOW_NODE2   => LOW_NODE2,
               NETLIST_OBJECT => NETLIST_OBJECT);
            if (abs (TEMP_VALUE1) + abs (VALUE1)) > MINIMUM_VALUE then
               PHASE := Maths.Angle (X_COORDINATE => TEMP_VALUE1,
                  Y_COORDINATE => VALUE1);
               VALUE1 := Math_Lib.sqrt (
                  TEMP_VALUE1 * TEMP_VALUE1 + VALUE1 * VALUE1);
               case COMPONENT is
                  when IMAG_VOLT => VISIBLE_COMPONENT := VOLT_SOURCE;
                  when REAL_CRNT => VISIBLE_COMPONENT := CRNT_SOURCE;
                  when GI_ADMIT  => VISIBLE_COMPONENT := RESISTOR;
                  when others    => VISIBLE_COMPONENT := COMPONENT;
               end case;
               NO_ERRORS := TRUE;
            end if;
         end if;
         if NO_ERRORS and (LPAR (COUNT) > 0) then
            Screen_Supporter.Display_Component (
               COMPONENT  => VISIBLE_COMPONENT,
               HIGH_NODE1 => HIGH_NODE1,
               LOW_NODE1  => LOW_NODE1,
               HIGH_NODE2 => HIGH_NODE2,
               LOW_NODE2  => LOW_NODE2,
               VALUE      => VALUE1,
               PHASES     => PHASE,
               ROW        => SCREEN_ROW);
         end if;
      end if;
      if LPAR (COUNT) < 0 then
```

```
      Cursor_Set (ROW => SCREEN_ROW, COLUMN => 3);
      Write (ITEM => "Optimising frequency =");
      Write (ITEM => XX (COUNT),
        WHOLE      => 4,
        FRACTION   => 5,
        ENG_NOTATION => TRUE);
      Cursor_Next_Line;
      SCREEN_ROW := SCREEN_ROW + 1;
    end if;
  end loop;
end Show;
```

```
end Optimiser;
```

Scheduler

```
package Scheduler is

-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This does the scheduling of all the process in the program.
--           It also instructs the Menus object from which menu options
--           can be made selectable or non-selectable.

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   procedure Initialise;

   procedure Scheduling;

end Scheduler;
```

```
with  Ada_Extensions,
      Analysers,
      Central_Solver,
      Drawer,
      Graphs,
      Menus,
      Monte,
      Netlists,
      Netlist_Supporter,
      Optimiser,
      Screens,
      Tutorial;

use   Ada_Extensions,
      Menus,
      Netlists,
      Screens;

package body Scheduler is

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

   NETLIST                                 : NETLIST_TYPE;
   NODE_MAXIMUM                            : NODE_TYPE := 0;

-- PURPOSE : Saves the present circuit to disk if desired.

procedure Save_Present_Circuit (GOT_CIRCUIT : in BOOLEAN) is

   RESPONSE : BOOLEAN;

begin
   if GOT_CIRCUIT then
      Prompts.Read (MESSAGE =>"Do you want to save the current circuit ?",
        ITEM => RESPONSE);
      if RESPONSE then
        Outer.Save_Netlist (NETLIST_OBJECT => NETLIST);
      end if;
   end if;
   Inner.Initialize (NETLIST_OBJECT => NETLIST);
end Save_Present_Circuit;

-- PURPOSE : Makes Monte operations selectable or non-selectable.

procedure Valid_Monte_Operations (MODE : in BOOLEAN) is

   type VALID_TYPE is array (1..6) of INTEGER_TYPE;

   VALID : constant VALID_TYPE :=
     (OP_MONTE_CLEAR,
      OP_MONTE_PERFORM,
      OP_MONTE_SAVE,
      OP_MONTE_SHOW,
      OP_MONTE_WORST,
      OP_MONTE_WORST2);
```

```
   COUNTER : INTEGER_TYPE;

begin
   for COUNTER in 1..VALID'last loop
      Change_Menu_Option (OPERATION => VALID (COUNTER),
         STATE => NODE);
   end loop;
end Valid_Monte_Operations;

-- PURPOSE : Makes optimisation operations selectable or non-selectable.

procedure Valid_Optimisation_Operations (NODE : in BOOLEAN) is

   type VALID_TYPE is array (1..5) of INTEGER_TYPE;

   VALID : constant VALID_TYPE :=
      (OP_OPTIM_CLEAR,
       OP_OPTIM_MAXIMISE,
       OP_OPTIM_MINIMISE,
       OP_OPTIM_SET,
       OP_OPTIM_SHOW);

   COUNTER : INTEGER_TYPE;

begin
   for COUNTER in 1..VALID'last loop
      Change_Menu_Option (OPERATION => VALID (COUNTER),
         STATE => NODE);
   end loop;
end Valid_Optimisation_Operations;

-- PURPOSE : Makes certain menu operations selectable or non-selectable
--           depending if there is a circuit or not.

procedure Valid_Operations (NODE : in BOOLEAN) is

   type VALID_TYPE is array (1..25) of INTEGER_TYPE;

   VALID : constant VALID_TYPE :=
      (OP_ADMITTANCE,
       OP_BRANCH_CURRENTS,
       OP_CHANGE_MAG_PH,
       OP_CONVERT_TO_ADMITTANCE,
       OP_CONVERT_TO_RLC,
       OP_DEFINE,
       OP_DELETE_NODE,
       OP_DRAW,
       OP_FOURIER,
       OP_LIST_COMPONENTS,
       OP_MONTE_INSERT,
       OP_MONTE_RECOVER,
       OP_NODE_VOLTAGES,
       OP_OPTIM_BANDWIDTH,
       OP_OPTIM_INSERT,
       OP_PHASOR,
       OP_PLOTS,
       OP_POLAR,
       OP_POWER,
       OP_RENUMBER_NODE,
       OP_SAVE,
       OP_SENSITIVITY,
       OP_SHOW_BRANCH,
       OP_SHOW_COMPONENT,
       OP_SHOW_NODES);

   COUNTER : INTEGER_TYPE;

begin
   for COUNTER in 1..VALID'last loop
      Change_Menu_Option (OPERATION => VALID (COUNTER),
         STATE => NODE);
   end loop;
end Valid_Operations;

-- PURPOSE : Initialises the Scheduler object by initialising certain
--           other objects which need initialising.

procedure Initialize is

begin
   Tutorial.Initialize_Tutorial_Path;
   Graphs.Setup;
   Set_Component_Values (COMP_VALUES => COMPONENT_VALUE);
   Inner.Initialize (NETLIST_OBJECT => NETLIST);
   Outer.Initialize_Directory;
end Initialize;

-- PURPOSE : Schedules the different process and determines which menu
--           options are available and which are not.

procedure Scheduling is

   ------------------------------------------------------------------

   OP_IGNORE : constant := 10000;

   PREVIOUS_OPERATION                    : OPERATION_TYPE := OP_IGNORE;
   OPERATION                             : OPERATION_TYPE;
   CHECK_CIRCUIT,      CHECK_NODE,
   GOT_CIRCUIT                           : BOOLEAN := FALSE;
   FREQUENCY                             : REAL_TYPE;

   ------------------------------------------------------------------

begin
   loop
      Get_Menu_Operation (OPERATION => OPERATION);
      begin
         begin
            if (PREVIOUS_OPERATION in OP_MONTE_CLEAR..OP_MONTE_WORST2) and
               (OPERATION not in OP_MONTE_CLEAR..OP_MONTE_WORST2) then
               Monte.Destroy_Data_Structure;
               Valid_Monte_Operations (NODE => FALSE);
            end if;
            if CHECK_CIRCUIT then
               case OPERATION is
                  when OP_ADMITTANCE | OP_BRANCH_CURRENTS | OP_DEFINE |
                     OP_FOURIER | OP_MONTE_CLEAR..OP_MONTE_WORST2 |
                     OP_NODE_VOLTAGES..OP_POWER | OP_SENSITIVITY |
                     OP_SHOW_NODES =>
                     Inner.Check_Topology (NETLIST_OBJECT => NETLIST);
                     Analysers.All_Calculation_Modes (
                        NODE_MAX => NODE_MAXINUM);
                     CHECK_CIRCUIT := FALSE;
                  when others => null;
               end case;
            end if;
         exception
            when NO_CIRCUIT =>
               Errors.Write(MESSAGE=>"There is no circuit to work with.");
               OPERATION := OP_IGNORE;
            when NO_VCCS_CONTROLLING_NODE =>
               Errors.Write (MESSAGE => "There are no controlling nodes " &
                  & "for a voltage controlled current source.");
               OPERATION := OP_IGNORE;
            when REFERENCE_NODE_MISSING =>
               Errors.Write (MESSAGE => "The reference node is missing.");
               OPERATION := OP_IGNORE;
            when SPECIFIC_NODE_MISSING =>
               Errors.Write (MESSAGE => "A specific node is missing.");
               OPERATION := OP_IGNORE;
         end;

         begin
            case OPERATION is
               when OP_ADMITTANCE =>
                  Prompts.Read (MESSAGE => "Enter the frequency (",
                     ITEM => FREQUENCY);
                  Control_Solver.Solve_AC_Network (NODE => NLC,
                     AC_FREQUENCY    => FREQUENCY,
                     CALCULATE       => FALSE,
                     NETLIST_OBJECT => NETLIST);

               when OP_BRANCH_CURRENTS =>
                  Analysers.Calculate_Current (NODE_MAX => NODE_MAXIMUM,
                     NETLIST_OBJECT => NETLIST);

               when OP_CHANGE_MAG_PH =>
                  Netlist_Supporter.Change_Magnitude_Phase (
                     NETLIST_OBJECT => NETLIST);

               when OP_CONVERT_TO_ADMITTANCE =>
                  Main.Process_Title (TITLE => "CONVERT TO ADMITTANCE");
                  Prompts.Read (MESSAGE => "Enter the frequency for the" &
                     " conversion:",
                     ITEM => FREQUENCY);
                  Inner.Convert_RLC_To_Admittance (
                     FREQUENCY      => FREQUENCY,
                     NETLIST_OBJECT => NETLIST);

               when OP_CONVERT_TO_RLC =>
                  Main.Process_Title (TITLE => "CONVERT TO RLC");
                  Prompts.Read (MESSAGE => "Enter the frequency for the" &
                     " conversion.",
                     ITEM => FREQUENCY);
                  declare
                     ERROR1 : constant STRING := "The conversion was " &
                        "aborted because there ";
                  begin
                     Inner.Convert_Admittance_To_RLC (
                        FREQUENCY      => FREQUENCY,
                        NETLIST_OBJECT => NETLIST);
                  exception
                     when CANNOT_CONVERT_WITH_MUTUAL_INDUCTANCES =>
                        Errors.Write (MESSAGE => ERROR1 &
                           "are mutual inductances present.");
                     when STORAGE_ERROR =>
                        Errors.Write (MESSAGE => ERROR1 &
                           "was not enough memory.");
                     when TOO_MANY_BRANCHES =>
                        Errors.Write (MESSAGE => ERROR1 &
                           "are too many branches.");
                  end;

               when OP_DEFINE =>
                  Analysers.Set_Calculation_Modes(NODE_MAX=>NODE_MAXIMUM);

               when OP_DELETE_NODE =>
                  Outer.Remove_Node (NETLIST_OBJECT => NETLIST);
                  CHECK_NODE := TRUE;

               when OP_DRAW =>
                  Drawer.Draw_Circuit (NODE_MAX => NODE_MAXIMUM,
                     NETLIST_OBJECT => NETLIST);

               when OP_EXTRA_RESULTS =>
                  Main.Open_Stream_Devices;

               when OP_FOURIER =>
                  Analysers.Plotter.Fourier (NODE_MAX => NODE_MAXIMUM,
                     NETLIST_OBJECT => NETLIST);

               when OP_INSERT_COMPONENT =>
                  CHECK_NODE := TRUE;
                  Netlist_Supporter.Insert_Component (
                     NETLIST_OBJECT => NETLIST);

               when OP_LIST_COMPONENTS =>
                  Netlist_Supporter.List_Netlist (ALL_COMPONENTS => TRUE,
                     NETLIST_OBJECT => NETLIST);

               when OP_LOAD =>
                  Save_Present_Circuit (GOT_CIRCUIT => GOT_CIRCUIT);
                  CHECK_NODE := TRUE;
                  Outer.Load_Netlist (NETLIST_OBJECT => NETLIST);

               when OP_MONTE_CLEAR =>
                  Monte.Clear_Variables;
                  Valid_Monte_Operations (NODE => FALSE);

               when OP_MONTE_INSERT =>
                  Monte.Insert (NODE_MAX => NODE_MAXIMUM,
```

```
            NETLIST1 => NETLIST);
         Valid_Monte_Operations (MODE => TRUE);

      when OP_MONTE_PERFORM =>
         Monte.Perform (NODE_MAX => NODE_MAXIMUM,
            NETLIST => NETLIST);

      when OP_MONTE_RECOVER =>
         Monte.Load (NETLIST => NETLIST);
         Valid_Monte_Operations (MODE => TRUE);

      when OP_MONTE_SAVE =>
         Monte.Save;

      when OP_MONTE_SHOW =>
         Monte.Show (NETLIST1 => NETLIST);

      when OP_MONTE_TEST =>
         Monte.Test_Random;

      when OP_MONTE_WORST =>
         Monte.Worst (SINGLE => TRUE,
            NODE_MAX => NODE_MAXIMUM,
            NETLIST1 => NETLIST);

      when OP_MONTE_WORST2 =>
         Monte.Worst (SINGLE => FALSE,
            NODE_MAX => NODE_MAXIMUM,
            NETLIST1 => NETLIST);

      when OP_NEW_NETWORK =>
         Save_Present_Circuit (GOT_CIRCUIT => GOT_CIRCUIT);
         CIRCUIT_NODE := NIL;
         Set_Component_Values (COMP_VALUES => COMPONENT_VALUE);
         CHECK_NODE := TRUE;

      when OP_NODE_VOLTAGES =>
         Analysers.Calculate_Voltage (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_OPTIM_BANDWIDTH =>
         Optimiser.Find_Bandwidth (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_OPTIM_CLEAR =>
         Optimiser.Clear (NETLIST_OBJECT => NETLIST);
         Valid_Optimisation_Operations (MODE => FALSE);

      when OP_OPTIM_INSERT =>
         Optimiser.Insert (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);
         Valid_Optimisation_Operations (MODE => TRUE);

      when OP_OPTIM_MAXIMISE =>
         Optimiser.Maximise (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);
         Valid_Optimisation_Operations (MODE => FALSE);

      when OP_OPTIM_MINIMISE =>
         Optimiser.Minimise (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);
         Valid_Optimisation_Operations (MODE => FALSE);

      when OP_OPTIM_SET =>
         Optimiser.Set (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);
         Valid_Optimisation_Operations (MODE => FALSE);

      when OP_OPTIM_SHOW =>
         Optimiser.Show (NETLIST_OBJECT => NETLIST);

      when OP_PHASOR =>
         Analysers.Plotter.Phasors (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_PLOTS =>
         Analysers.Plotter.Circuit_Response (
            NODE_MAX       => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_POLAR =>
         Analysers.Plotter.Polar (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_POWER =>
         Analysers.Calculate_Power (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_QUIT =>
         Save_Present_Circuit (GOT_CIRCUIT => GOT_CIRCUIT);
         exit;

      when OP_RENUMBER_NODE =>
         Outer.Renumber_Nodes (NETLIST_OBJECT => NETLIST);
         CHECK_NODE := TRUE;

      when OP_SAVE =>
         Outer.Save_Netlist (NETLIST_OBJECT => NETLIST);

      when OP_SENSITIVITY =>
         Analysers.Sensitivity (NODE_MAX => NODE_MAXIMUM,
            NETLIST_OBJECT => NETLIST);

      when OP_SET_DIRECTORY =>
         Outer.Change_Directory;

      when OP_SET_DRIVE =>
         Outer.Change_Drive;

      when OP_SHOW_BRANCH =>
         Outer.Show_Node (NETLIST_OBJECT => NETLIST);

      when OP_SHOW_COMPONENT =>
         Netlist_Supportar.List_Netlist (ALL_COMPONENTS => FALSE,
            NETLIST_OBJECT => NETLIST);

      when OP_SHOW_NODES =>
         Analysers.Show_Nodes;

      when OP_TITLE =>
            Outer.Set_Circuit_Title (NETLIST_OBJECT => NETLIST);

         when OP_TUTORIAL =>
            Tutorial.Show;

         when others =>
            null;
      end case;
   exception
      when MEANINGLESS_COMPONENT =>
         Errors.Write(
            MESSAGE => "A meaningless component was found.");
      when NO_CIRCUIT =>
         Errors.Write (
            MESSAGE => "There is no circuit to work with.");
      when NODE_PAIR_DOES_NOT_EXIST =>
         Errors.Write (MESSAGE => "The node pair does not exist.");
      when STORAGE_ERROR =>
         Errors.Write (MESSAGE => "There is not enough memory to "
            & "do the process.");
   end;
 exception
   when USER_ESCAPE =>
      null;
   when others =>
      Errors.Write (MESSAGE => "An undefined error occurred.");
 end;
 if CHECK_NODE then
    CHECK_NODE := FALSE;
    Inner.Find_Node_Maximum (NODE_MAXIMUM => NODE_MAXIMUM,
       NETLIST_OBJECT => NETLIST);
    if (NODE_MAXIMUM = 0) and GOT_CIRCUIT then
       Valid_Operations (MODE => FALSE);
       GOT_CIRCUIT := FALSE;
    elsif (NODE_MAXIMUM > 0) then
       if not GOT_CIRCUIT then
          Valid_Operations (MODE => TRUE);
          GOT_CIRCUIT := TRUE;
       end if;
       CHECK_CIRCUIT := TRUE;
    end if;
 end if;
 PREVIOUS_OPERATION := OPERATION;
end loop;
Main.Process_Title (TITLE => "ELECTOR");
end Scheduling;
```

```
end Scheduler;
```

Screen_Objects

```
-- $ ----------------------------------------------------------------------

with  Ada_Extensions,
      Dos_Environment,
      Inputter;

use   Ada_Extensions,
      Dos_Environment,
      Inputter;

package Screen_Objects is

-- ----------------------------------------------------------------------
-- Author  : Warren Buckle
-- Date    : June 1991
-- Purpose : This  contains a  collection of objects  from which the  user
--           interface is constructed.
-- ----------------------------------------------------------------------

-- ----------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-- ----------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-- ----------------------------------------------------------------------

   OP_DONE    : constant := 1;       OP_OK    : constant := 2;
   OP_MOVE    : constant := 3;       OP_HELP  : constant := 4;
   OP_CANCEL  : constant := 5;       OP_TRUE  : constant := 6;
   OP_FALSE   : constant := 7;       OP_MOVED : constant := 8;

   OP_NON_ACCEPTING : constant := 9;

   -- -------------------------------------------------------------------

   package Buttons is

      type BUTTON_TYPE is private;


      procedure Create (BUTTON : in out BUTTON_TYPE;
         NAME          : in STRING;
         OPERATION     : in OPERATION_TYPE;
         COLOURS       : in SCREEN_COLOUR_TYPE;
         SELECT_COLOURS : in SCREEN_COLOUR_TYPE;
         UNDER_COLOURS : in SCREEN_COLOUR_TYPE;
         ROW           : in ROW_TYPE;
         COLUMN        : in COLUMN_TYPE;
         HOT_KEY       : in KEY_VALUES_TYPE);

      procedure Draw (BUTTON : in BUTTON_TYPE);

      procedure Handle_Input (BUTTON : in BUTTON_TYPE;
         INPUTS : in out INPUT_TYPE);

      procedure Press (BUTTON : in BUTTON_TYPE);


   private

      type BUTTON_TYPE is
         record
            NAME_LENGTH    : NATURAL;
            NAME           : STRING (1..15);
            OPERATION      : OPERATION_TYPE;
            COLOURS        : SCREEN_COLOUR_TYPE;
            SELECT_COLOURS : SCREEN_COLOUR_TYPE;
            UNDER_COLOURS  : SCREEN_COLOUR_TYPE;
            ROW            : ROW_TYPE;
            COLUMN         : COLUMN_TYPE;
            HOT_KEY        : KEY_VALUES_TYPE;
         end record;

   end Buttons;

   -- -------------------------------------------------------------------

   package Boolean_Buttons is

      type BOOLEAN_BUTTON_TYPE is private;


      procedure Create (BL_BUTTON : in out BOOLEAN_BUTTON_TYPE;
         COLOURS       : in SCREEN_COLOUR_TYPE;
         SELECT_COLOURS : in SCREEN_COLOUR_TYPE;
         UNDER_COLOURS : in SCREEN_COLOUR_TYPE;
         TRUE_ROW      : in ROW_TYPE;
         TRUE_COLUMN   : in COLUMN_TYPE;
         FALSE_ROW     : in ROW_TYPE;
         FALSE_COLUMN  : in COLUMN_TYPE);

      procedure Draw (BL_BUTTON : in BOOLEAN_BUTTON_TYPE);

      procedure Handle_Input (BL_BUTTON : in BOOLEAN_BUTTON_TYPE;
         INPUTS : in out INPUT_TYPE);


   private

      type BOOLEAN_BUTTON_TYPE is
         record
            BT_TRUE  : Buttons.BUTTON_TYPE;
            BT_FALSE : Buttons.BUTTON_TYPE;
         end record;

end Boolean_Buttons;

-- ----------------------------------------------------------------------

package Input_Lines is

   type LINE_TYPE is private;


   procedure Create (LINER : in out LINE_TYPE;
      ROW           : in ROW_TYPE;
      COLUMN        : in COLUMN_TYPE;
      OFFSET_ROW    : in ROW_TYPE;
      OFFSET_COLUMN : in COLUMN_TYPE;
      TOTAL_LENGTH  : in COLUMN_TYPE;
      DISPLAY_LENGTH : in COLUMN_TYPE;
      COLOURS       : in SCREEN_COLOUR_TYPE);

   procedure Draw (LINER : in LINE_TYPE);

   procedure Get_Line (LINER : in LINE_TYPE; LINE : out STRING;
      LENGTH : out NATURAL);

   procedure Handle_Input (LINER : in out LINE_TYPE;
      INPUTS : in out INPUT_TYPE);

   procedure Reset_Line (LINER : in out LINE_TYPE);

private

   type LINE_TYPE is
      record
         MAX_LENGTH     : NATURAL;
         LINE           : STRING (1..64);
         ACTUAL_LENGTH  : POSITIVE;
         DISPLAY_LENGTH : POSITIVE;
         START_DISPLAY  : POSITIVE;
         CURSOR         : POSITIVE;
         INSERTING      : BOOLEAN;
         ROW            : ROW_TYPE;
         COLUMN         : COLUMN_TYPE;
         COLOUR         : SCREEN_COLOUR_TYPE;
      end record;

end Input_Lines;

-- ----------------------------------------------------------------------

package Pick_Lists is

   type LIST_TYPE   is private;
   type LIST_D_TYPE is private;


   procedure Create (LISTER : in out LIST_TYPE;
      ROW          : in ROW_TYPE;
      COLUMN       : in COLUMN_TYPE;
      LIST_DATA    : in LIST_ARRAY_TYPE;
      EXCLUSIVE    : in BOOLEAN;
      WIDTHS       : in POSITIVE;
      COLOURS      : in SCREEN_COLOUR_TYPE;
      BAR_COLOURS  : in SCREEN_COLOUR_TYPE);

   procedure Create (LISTER : in out LIST_D_TYPE;
      ROW              : in ROW_TYPE;
      COLUMN           : in COLUMN_TYPE;
      DISPLAY_LENGTH   : in NATURAL;
      COLOURS          : in SCREEN_COLOUR_TYPE;
      BAR_COLOURS      : in SCREEN_COLOUR_TYPE;
      SCROLLER_COLOURS : in SCREEN_COLOUR_TYPE);

   procedure Create_List (LISTER : in out LIST_D_TYPE;
      NAME : in STRING);

   procedure Destroy_List (LISTER : in out LIST_D_TYPE);

   procedure Draw (LISTER : in LIST_TYPE;
      LIST_DATA : in LIST_ARRAY_TYPE);

   procedure Draw (LISTER : in LIST_D_TYPE);

   procedure Get_Name (LISTER : in out LIST_D_TYPE;
      ITEM : out STRING);

   procedure Handle_Input (LISTER : in out LIST_TYPE;
      LIST_DATA : in out LIST_ARRAY_TYPE;
      INPUTS    : in out INPUT_TYPE);

   procedure Handle_Input (LISTER : in out LIST_D_TYPE;
      INPUTS : in out INPUT_TYPE);


private

   type LIST_TYPE is
      record
         LENGTHS     : NATURAL;
         EXCLUSIVE   : BOOLEAN;
         ACTIVE      : BOOLEAN;
         WIDTHS      : POSITIVE;
         CHOICE      : POSITIVE;
         TOP_ROW     : ROW_TYPE;
         LEFT_SIDE   : COLUMN_TYPE;
         COLOUR      : SCREEN_COLOUR_TYPE;
         BAR_COLOUR  : SCREEN_COLOUR_TYPE;
      end record;

   type LIST_DYNAMIC_TYPE;

   type NAME_LINK is access LIST_DYNAMIC_TYPE;

   type LIST_DYNAMIC_TYPE is
      record
         NAME : STRING (1..12);
         NEXT : NAME_LINK;
         BACK : NAME_LINK;
      end record;

   type LIST_D_TYPE is
```

```
      record
         START          : NAME_LINK;
         PRESENT_TOP    : NAME_LINK;
         CHOICE_POINTER : NAME_LINK;
         MAX_LENGTH     : NATURAL;
         PRESENT_LENGTH : NATURAL;
         LENGTHS        : NATURAL;
         CHOICE         : POSITIVE;
         TOP_ROW        : ROW_TYPE;
         LEFT_SIDE      : COLUMN_TYPE;
         COLOUR         : SCREEN_COLOUR_TYPE;
         BAR_COLOUR     : SCREEN_COLOUR_TYPE;
         SCROLLER_COLOUR : SCREEN_COLOUR_TYPE;
      end record;

end Pick_Lists;
```

```
package Forms is

   type FORM_TYPE is private;

   INTEGER_ERROR       : exception;
   INTEGER_MIN_ERROR   : exception;
   INTEGER_MAX_ERROR   : exception;
   INTEGER_EQUAL_ERROR : exception;
   REAL_ERROR          : exception;
   REAL_MIN_ERROR      : exception;
   REAL_MAX_ERROR      : exception;

   ERROR_FIELD         : INTEGER_TYPE;


   procedure Create (FORM : in out FORM_TYPE;
      FORM_DATA     : in FIELDS_TYPE;
      OFFSET_ROW    : in ROW_TYPE;
      OFFSET_COLUMN : in COLUMN_TYPE;
      TEXT_COLUMN   : in COLUMN_TYPE;
      VALUE_COLUMN  : in COLUMN_TYPE;
      ACTIVE_COLOUR : in SCREEN_COLOUR_TYPE;
      TEXT_COLOUR   : in SCREEN_COLOUR_TYPE;
      FIELD_COLOUR  : in SCREEN_COLOUR_TYPE);

   procedure Draw (FORM : in FORM_TYPE;
      FORM_DATA : in FIELDS_TYPE);

   procedure Handle_Input (FORM : in out FORM_TYPE;
      FORM_DATA : in out FIELDS_TYPE;
      INPUTS    : in out INPUT_TYPE);

private

   type FORM_TYPE is
      record
         ACTIVE_FIELD  : POSITIVE;
         CURSOR_ROW    : ROW_TYPE;
         CURSOR_COLUMN : COLUMN_TYPE;
         TEXT_COLUMN   : COLUMN_TYPE;
         VALUE_COLUMN  : COLUMN_TYPE;
         ACTIVE        : BOOLEAN;
         INSERTING     : BOOLEAN;
         ACTIVE_COLOUR : SCREEN_COLOUR_TYPE;
         FIELD_COLOUR  : SCREEN_COLOUR_TYPE;
         TEXT_COLOUR   : SCREEN_COLOUR_TYPE;
      end record;

end Forms;
```

```
package Windows is

   type WINDOW_TYPE is private;

   WINDOW_TOO_LARGE : exception;


   procedure Close;

   procedure Create (WINDOW : in out WINDOW_TYPE;
      NAME       : in STRING;
      KEYS_TEXT  : in STRING;
      TOP_ROW    : in ROW_TYPE;
      LEFT_SIDE  : in COLUMN_TYPE;
      LENGTHS    : in ROW_TYPE;
      WIDTHS     : in COLUMN_TYPE;
      TEXT_COLOUR  : in SCREEN_COLOUR_TYPE;
      FRAME_COLOUR : in SCREEN_COLOUR_TYPE;
      TITLE_COLOUR : in SCREEN_COLOUR_TYPE);

   procedure Draw (WINDOW : in WINDOW_TYPE;
      TEXT     : in STRING;
      TEXT_ROW : in ROW_TYPE);

   procedure Handle_Input (WINDOW : in out WINDOW_TYPE;
      INPUTS : in out INPUT_TYPE);

   procedure Paragrapher (WINDOW : in WINDOW_TYPE;
      LINE : in STRING;
      ROW  : in ROW_TYPE);

   procedure Update (WINDOW : in WINDOW_TYPE;
      TEXT     : in STRING;
      TEXT_ROW : in ROW_TYPE);


private

   type WINDOW_TYPE is
      record
         NAME_LENGTH  : NATURAL;
         KEYS_LENGTH  : NATURAL;
         NAME         : STRING (1..15);
         KEYS_TEXT    : STRING (1..74);
         TOP_ROW      : ROW_TYPE;
         LEFT_SIDE    : COLUMN_TYPE;
         LENGTHS      : ROW_TYPE;
         WIDTHS       : COLUMN_TYPE;
         KEY_MOVING   : BOOLEAN := FALSE;
         MOUSE_MOVING : BOOLEAN := FALSE;
         TEXT_COLOUR  : SCREEN_COLOUR_TYPE;
         FRAME_COLOUR : SCREEN_COLOUR_TYPE;
         TITLE_COLOUR : SCREEN_COLOUR_TYPE;
      end record;

   end Windows;
```

```
end Screen_Objects;
```

```
with  Unchecked_Deallocation;

package body Screen_Objects is

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can  be  introduced  again by
-- omitting the SUPPRESS pragma for that check.

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);
```

```
package body Buttons is

-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           button object.

procedure Create (BUTTON : in out BUTTON_TYPE;
   NAME           : in STRING;
   OPERATION      : in OPERATION_TYPE;
   COLOURS        : in SCREEN_COLOUR_TYPE;
   SELECT_COLOURS : in SCREEN_COLOUR_TYPE;
   UNDER_COLOURS  : in SCREEN_COLOUR_TYPE;
   ROW            : in ROW_TYPE;
   COLUMN         : in COLUMN_TYPE;
   HOT_KEY        : in KEY_VALUES_TYPE) is

begin
   BUTTON.NAME_LENGTH    := NAME'length;
   BUTTON.NAME (1..NAME'length) := NAME;
   BUTTON.OPERATION      := OPERATION;
   BUTTON.COLOURS        := COLOURS;
   BUTTON.SELECT_COLOURS := SELECT_COLOURS;
   BUTTON.ROW            := ROW;
   BUTTON.COLUMN         := COLUMN;
   BUTTON.HOT_KEY        := HOT_KEY;
   BUTTON.UNDER_COLOURS  := UNDER_COLOURS;
end Create;

-- PURPOSE : Draws the  button object in the  "unpressed"  state  into the
--           window.

procedure Draw (BUTTON : in BUTTON_TYPE) is

begin
   String_To_Window_Buffer (MESSAGE => BUTTON.NAME(1..BUTTON.NAME_LENGTH),
      ROW     => BUTTON.ROW,
      COLUMN  => BUTTON.COLUMN,
      COLOURS1 => BUTTON.COLOURS,
      COLOURS2 => BUTTON.SELECT_COLOURS);
end Draw;

-- PURPOSE : Draws  the  button object  in  the  "pressed"  state  into  the
--           window.

procedure Press (BUTTON : in BUTTON_TYPE) is

begin
   String_To_Window_Buffer (MESSAGE => BUTTON.NAME(1..BUTTON.NAME_LENGTH),
      ROW     => BUTTON.ROW,
      COLUMN  => BUTTON.COLUMN,
      COLOURS1 => BUTTON.UNDER_COLOURS,
      COLOURS2 => BUTTON.UNDER_COLOURS);
end Press;

-- PURPOSE : Handles  input  that  is  directed  to  this  button  object,
--           otherwise it ignores it.

procedure Handle_Input (BUTTON : in BUTTON_TYPE;
   INPUTS : in out INPUT_TYPE) is

   PRESSED : BOOLEAN := FALSE;

begin
   if INPUTS.GOT_MOUSE and (INPUTS.BUTTON = LEFT_BUTTON) then
      if (INPUTS.Y = BUTTON.ROW) and (INPUTS.X in
         BUTTON.COLUMN..(BUTTON.COLUMN + BUTTON.NAME_LENGTH - 1)) then
         PRESSED := TRUE;
      end if;
   elsif INPUTS.KEY = BUTTON.HOT_KEY then
      PRESSED := TRUE;
   end if;
   if PRESSED then
      Press (BUTTON => BUTTON);
      Window_Buffer_To_Screen;
      delay 0.35;
      Draw (BUTTON => BUTTON);
```

```
      Window_Buffer_To_Screen;
      delay 0.15;
      INPUTS.OPERATION := BUTTON.OPERATION;
   end if;
end Handle_Input;

-------------------------------------------------------------------------

end Buttons;

-------------------------------------------------------------------------
-------------------------------------------------------------------------

package body Boolean_Buttons is

-------------------------------------------------------------------------
-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           boolean button object.
-------------------------------------------------------------------------

procedure Create (BL_BUTTON : in out BOOLEAN_BUTTON_TYPE;
   COLOURS        : in SCREEN_COLOUR_TYPE;
   SELECT_COLOURS : in SCREEN_COLOUR_TYPE;
   UNDER_COLOURS  : in SCREEN_COLOUR_TYPE;
   TRUE_ROW       : in ROW_TYPE;
   TRUE_COLUMN    : in COLUMN_TYPE;
   FALSE_ROW      : in ROW_TYPE;
   FALSE_COLUMN   : in COLUMN_TYPE) is

begin
   Buttons.Create (BUTTON => BL_BUTTON.BT_TRUE,
      NAME          => " -Y-es ",
      OPERATION     => OP_TRUE,
      COLOURS       => COLOURS,
      SELECT_COLOURS => SELECT_COLOURS,
      UNDER_COLOURS  => UNDER_COLOURS,
      ROW           => TRUE_ROW,
      COLUMN        => TRUE_COLUMN,
      HOT_KEY       => KB_ALT_Y);
   Buttons.Create (BUTTON => BL_BUTTON.BT_FALSE,
      NAME          => " -N-o ",
      OPERATION     => OP_FALSE,
      COLOURS       => COLOURS,
      SELECT_COLOURS => SELECT_COLOURS,
      UNDER_COLOURS  => UNDER_COLOURS,
      ROW           => FALSE_ROW,
      COLUMN        => FALSE_COLUMN,
      HOT_KEY       => KB_ALT_N);
end Create;

-------------------------------------------------------------------------
-- PURPOSE : Draws the boolean_button object into the window.
-------------------------------------------------------------------------

procedure Draw (BL_BUTTON : in BOOLEAN_BUTTON_TYPE) is

begin
   Buttons.Draw (BUTTON => BL_BUTTON.BT_TRUE);
   Buttons.Draw (BUTTON => BL_BUTTON.BT_FALSE);
end Draw;

-------------------------------------------------------------------------
-- PURPOSE : Handles input that is directed to this boolean_button object,
--           otherwise it ignores it.
-------------------------------------------------------------------------

procedure Handle_Input (BL_BUTTON : in BOOLEAN_BUTTON_TYPE;
   INPUTS : in out INPUT_TYPE) is

begin
   Buttons.Handle_Input (BUTTON => BL_BUTTON.BT_TRUE, INPUTS => INPUTS);
   if INPUTS.OPERATION = OP_DO then
      Buttons.Handle_Input (BUTTON => BL_BUTTON.BT_FALSE,
         INPUTS => INPUTS);
   end if;
end Handle_Input;

-------------------------------------------------------------------------

end Boolean_Buttons;

-------------------------------------------------------------------------
-------------------------------------------------------------------------

package body Input_Lines is

   OFFSET_Y : ROW_TYPE;
   OFFSET_X : COLUMN_TYPE;

-------------------------------------------------------------------------
-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           input_line object.
-------------------------------------------------------------------------

procedure Create (LINER : in out LINE_TYPE;
   ROW            : in ROW_TYPE;
   COLUMN         : in COLUMN_TYPE;
   OFFSET_ROW     : in ROW_TYPE;
   OFFSET_COLUMN  : in COLUMN_TYPE;
   TOTAL_LENGTH   : in COLUMN_TYPE;
   DISPLAY_LENGTH : in COLUMN_TYPE;
   COLOURS        : in SCREEN_COLOUR_TYPE) is

begin
   LINER.MAX_LENGTH := TOTAL_LENGTH;
   LINER.LINE (1..TOTAL_LENGTH) := (1..TOTAL_LENGTH => ' ');
   LINER.DISPLAY_LENGTH := DISPLAY_LENGTH;
   LINER.ACTUAL_LENGTH  := 1;
   LINER.START_DISPLAY  := 1;
   LINER.CURSOR         := 1;
   LINER.INSERTING      := FALSE;
   LINER.ROW            := ROW;
   LINER.COLUMN         := COLUMN;
   LINER.COLOUR         := COLOURS;
   OFFSET_Y             := OFFSET_ROW;
   OFFSET_X             := OFFSET_COLUMN;
end Create;

-------------------------------------------------------------------------
-- PURPOSE : Draws the input_line object into the window.
-------------------------------------------------------------------------

procedure Draw (LINER : in LINE_TYPE) is

begin
   String_To_Window_Buffer (MESSAGE => LINER.LINE (LINER.START_DISPLAY..
      (LINER.START_DISPLAY + LINER.DISPLAY_LENGTH - 1));
      ROW      => LINER.ROW,
      COLUMN   => LINER.COLUMN + 4,
      COLOURS1 => LINER.COLOUR,
      COLOURS2 => LINER.COLOUR);
   String_To_Window_Buffer (MESSAGE => "   ",
      ROW      => LINER.ROW,
      COLUMN   => LINER.COLUMN,
      COLOURS1 => LINER.COLOUR,
      COLOURS2 => LINER.COLOUR);
   if LINER.START_DISPLAY > 1 then
      Character_To_Window_Buffer (CHARACTER_CODE => 17,
         COUNT     => 1,
         ROW       => LINER.ROW,
         COLUMN    => LINER.COLUMN + 1,
         COLOURS   => LINER.COLOUR);
   end if;
   String_To_Window_Buffer (MESSAGE => "   ",
      ROW      => LINER.ROW,
      COLUMN   => LINER.COLUMN + LINER.DISPLAY_LENGTH + 3,
      COLOURS1 => LINER.COLOUR,
      COLOURS2 => LINER.COLOUR);
   if (LINER.DISPLAY_LENGTH + LINER.START_DISPLAY) <
      LINER.ACTUAL_LENGTH then
      Character_To_Window_Buffer (CHARACTER_CODE => 16,
         COUNT     => 1,
         ROW       => LINER.ROW,
         COLUMN    => LINER.COLUMN + LINER.DISPLAY_LENG,     f
         COLOURS   => LINER.COLOUR);
   end if;
   Set_Cursor (ROW => LINER.ROW + OFFSET_Y,
      COLUMN => LINER.COLUMN + OFFSET_X + 3 + LINER.CURSOR -
      LINER.START_DISPLAY);
end Draw;

-------------------------------------------------------------------------
-- PURPOSE : Returns the string that has been got from the user by the
--           input_line object.
-------------------------------------------------------------------------

procedure Get_Line (LINER : in LINE_TYPE; LINE : out STRING;
   LENGTHS : out NATURAL) is

   TEMP_LENGTH : NATURAL;

begin
   TEMP_LENGTH := LINER.ACTUAL_LENGTH - 1;
   LINE (1..TEMP_LENGTH) := LINER.LINE (1..TEMP_LENGTH);
   LENGTHS := TEMP_LENGTH;
end Get_Line;

-------------------------------------------------------------------------
-- PURPOSE : Handles input that is directed to this input_line object,
--           otherwise it ignores it.
-------------------------------------------------------------------------

procedure Handle_Input (LINER : in out LINE_TYPE;
   INPUTS : in out INPUT_TYPE) is

   CHARACTER_CODE : CHARACTER;
   CHANGE         : BOOLEAN := TRUE;

   -------------------------------------------------------------------
   -- PURPOSE : Moves the cursor to the left if there is room.
   -------------------------------------------------------------------

   procedure Move_Cursor_Left is

   begin
      if LINER.CURSOR > 1 then
         LINER.CURSOR := LINER.CURSOR - 1;
         if LINER.CURSOR <= LINER.START_DISPLAY then
            LINER.START_DISPLAY := LINER.CURSOR;
         end if;
      end if;
   end Move_Cursor_Left;

   -------------------------------------------------------------------
   -- PURPOSE : Moves the cursor to the right if there room.
   -------------------------------------------------------------------

   procedure Move_Cursor_Right is

   begin
      if LINER.CURS      LINER.ACTUAL_LENGTH then
         LINER.CURS   := LINER.CURSOR + 1;
         if ((LINER.CURSOR - LINER.START_DISPLAY) >=
            LINER.DISPLAY_LENGTH) then
            LINER.START_DISPLAY := LINER.CURSOR - LINER.DISPLAY_LENGTH+1;
         end if;
      end if;
   end Move_Cursor_Right;

   -------------------------------------------------------------------

begin
   if INPUTS.OPERATION = OP_MOVED then
      OFFSET_Y := INPUTS.Y;
      OFFSET_X := INPUTS.X;
      Set_Cursor (ROW => LINER.ROW + OFFSET_Y,
         COLUMN => LINER.COLUMN + OFFSET_X + 3 + LINER.CURSOR -
         LINER.START_DISPLAY);
   end if;
   case INPUTS.KEY is
      when KB_LEFT =>
         Move_Cursor_Left;
      when KB_RIGHT =>
         Move_Cursor_Right;
      when KB_BACKSPACE =>
         if LINER.CURSOR > 1 then
            LINER.LINE ((LINER.CURSOR - 1)..(LINER.ACTUAL_LENGTH - 1)) :=
               LINER.LINE (LINER.CURSOR..LINER.ACTUAL_LENGTH);
            LINER.LINE (LINER.ACTUAL_LENGTH) := ' ';
            LINER.ACTUAL_LENGTH := LINER.ACTUAL_LENGTH - 1;
```

```
            Move_Cursor_Left;
         end if;
      when KB_DEL =>
         LINER.LINE (LINER.CURSOR..(LINER.ACTUAL_LENGTH - 1)) :=
            LINER.LINE ((LINER.CURSOR + 1)..LINER.ACTUAL_LENGTH);
         LINER.LINE (LINER.ACTUAL_LENGTH) := ' ';
         if LINER.CURSOR < LINER.ACTUAL_LENGTH then
            LINER.ACTUAL_LENGTH := LINER.ACTUAL_LENGTH - 1;
         end if;
      when KB_INST =>
         LINER.INSERTING := not LINER.INSERTING;
         if LINER.INSERTING then
            Change_Cursor_Shape (START_LINE => 7,
               ENDING_LINE => 0,
               DISPLAY     => TRUE);
         else
            Change_Cursor_Shape (START_LINE => 7,
               ENDING_LINE => 8,
               DISPLAY     => TRUE);
         end if;
         CHANGE := FALSE;
      when KB_SPACE|KB_PLUS| KB_MINUS..KB_COLON |KB_UPPER_A..KB_LOWER_Z =>
         CHARACTER_CODE := CHARACTER'val (INPUTS.KEY);
         if (LINER.ACTUAL_LENGTH /= LINER.MAX_LENGTH) then
            if LINER.CURSOR = LINER.ACTUAL_LENGTH then
               LINER.ACTUAL_LENGTH := LINER.ACTUAL_LENGTH + 1;
            elsif LINER.INSERTING then
               LINER.LINE ((LINER.CURSOR + 1)..(LINER.ACTUAL_LENGTH + 1))
                  := LINER.LINE (LINER.CURSOR..LINER.ACTUAL_LENGTH);
               LINER.ACTUAL_LENGTH := LINER.ACTUAL_LENGTH + 1;
            end if;
            LINER.LINE (LINER.CURSOR) := CHARACTER_CODE;
            Move_Cursor_Right;
         elsif (not LINER.INSERTING) and
            (LINER.CURSOR /= LINER.MAX_LENGTH) then
            LINER.LINE (LINER.CURSOR) := CHARACTER_CODE;
            Move_Cursor_Right;
         end if;
      when others =>
         CHANGE := FALSE;
   end case;
   if CHANGE then
      Draw (LINER => LINER);
      Window_Buffer_To_Screen;
      INPUTS.OPERATION := DP_DONE;
   end if;
end Handle_Input;

-- PURPOSE : Resets the  Input_Line  object  and erases any text  that had
--           been entered.

procedure Reset_Line (LINER : in out LINE_TYPE) is

begin
   LINER.LINE (1..LINER.MAX_LENGTH) := (1..LINER.MAX_LENGTH => ' ');
   LINER.ACTUAL_LENGTH := 1;
   LINER.START_DISPLAY := 1;
   LINER.CURSOR        := 1;
   LINER.INSERTING     := FALSE;
end Reset_Line;

end Input_Lines;

package body Pick_Lists is

   NEW_PRESS : BOOLEAN := TRUE;

   STRING_LENGTH : constant := 12;

-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           static pick_list object.

procedure Create (LISTER : in out LIST_TYPE;
   ROW          : in ROW_TYPE;
   COLUMN       : in COLUMN_TYPE;
   LIST_DATA    : in LIST_ARRAY_TYPE;
   EXCLUSIVE    : in BOOLEAN;
   WIDTHS       : in POSITIVE;
   COLOURS      : in SCREEN_COLOUR_TYPE;
   BAR_COLOURS  : in SCREEN_COLOUR_TYPE) is

begin
   LISTER.LENGTHS    := LIST_DATA'last;
   LISTER.ACTIVE     := FALSE;
   LISTER.EXCLUSIVE  := EXCLUSIVE;
   LISTER.CHOICE     := 1;
   LISTER.WIDTHS     := WIDTHS;
   LISTER.TOP_ROW    := ROW;
   LISTER.LEFT_SIDE  := COLUMN;
   LISTER.COLOUR     := COLOURS;
   LISTER.BAR_COLOUR := BAR_COLOURS;
   if LISTER.EXCLUSIVE then
      for COUNTER in 1..LISTER.LENGTHS loop
         if LIST_DATA (COUNTER).STATE then
            LISTER.CHOICE := COUNTER;
         end if;
      end loop;
   end if;
end Create;

-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           dynamic pick_list object.

procedure Create (LISTER : in out LIST_D_TYPE;
   ROW              : in ROW_TYPE;
   COLUMN           : in COLUMN_TYPE;
   DISPLAY_LENGTH   : in NATURAL;
   COLOURS          : in SCREEN_COLOUR_TYPE;
   BAR_COLOURS      : in SCREEN_COLOUR_TYPE;
   SCROLLER_COLOURS : in SCREEN_COLOUR_TYPE) is

begin
   LISTER.PRESENT_LENGTH  := 1;
   LISTER.CHOICE          := 1;
   LISTER.LENGTHS         := DISPLAY_LENGTH;
   LISTER.TOP_ROW         := ROW;
   LISTER.LEFT_SIDE       := COLUMN;
   LISTER.COLOUR          := COLOURS;
   LISTER.BAR_COLOUR      := BAR_COLOURS;
   LISTER.SCROLLER_COLOUR := SCROLLER_COLOURS;
end Create;

-- PURPOSE : Sets all  the  data  for the list  of the  dynamic  pick_list
--           object.

procedure Create_List (LISTER : in out LIST_D_TYPE;
   NAME : in STRING) is

   POINTER : NAME_LINK;

begin
   LISTER.CHOICE_POINTER      := new LIST_DYNAMIC_TYPE;
   LISTER.CHOICE_POINTER.NAME := NAME;
   LISTER.CHOICE_POINTER.NEXT := LISTER.START;
   if LISTER.START /= null then
      LISTER.START.BACK := LISTER.CHOICE_POINTER;
   end if;
   LISTER.START       := LISTER.CHOICE_POINTER;
   LISTER.PRESENT_TOP := LISTER.START;
   LISTER.MAX_LENGTH  := LISTER.MAX_LENGTH + 1;
   LISTER.CHOICE      := 1;
end Create_List;

-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           pick_list object.

procedure Destroy_List (LISTER : in out LIST_D_TYPE) is

   procedure Dispose is new Unchecked_Deallocation (LIST_DYNAMIC_TYPE,
      NAME_LINK);

begin
   while LISTER.START /= null loop
      LISTER.CHOICE_POINTER := LISTER.START;
      LISTER.START := LISTER.START.NEXT;
      Dispose (LISTER.CHOICE_POINTER);
   end loop;
   LISTER.MAX_LENGTH := 0;
end Destroy_List;

-- PURPOSE : Draws the static pick_list object into the window.

procedure Draw (LISTER : in LIST_TYPE;
   LIST_DATA : in LIST_ARRAY_TYPE) is

   COUNTER : NATURAL;
   SPACE_LINE, TEMP_LINE : STRING (1..(LISTER.WIDTHS + 5)) :=
      (1..(LISTER.WIDTHS + 5) => ' ');
   TEMP_COLOUR : SCREEN_COLOUR_TYPE;

begin
   for COUNTER in 1..LISTER.LENGTHS loop
      TEMP_LINE := SPACE_LINE;
      if LIST_DATA (COUNTER).STATE then
         if LISTER.EXCLUSIVE then
            TEMP_LINE (1..5) := " (*) ";
         else
            TEMP_LINE (1..5) := " [X] ";
         end if;
      else
         if LISTER.EXCLUSIVE then
            TEMP_LINE (1..5) := " ( ) ";
         else
            TEMP_LINE (1..5) := " [ ] ";
         end if;
      end if;
      TEMP_LINE (6..(LIST_DATA (COUNTER).TEXT_LENGTH + 5))
         := LIST_DATA (COUNTER).TEXT;
      if (COUNTER = LISTER.CHOICE) and LISTER.ACTIVE then
         TEMP_COLOUR := LISTER.BAR_COLOUR;
      else
         TEMP_COLOUR := LISTER.COLOUR;
      end if;
      String_To_Window_Buffer (MESSAGE => TEMP_LINE,
         ROW     => LISTER.TOP_ROW + COUNTER - 1,
         COLUMN  => LISTER.LEFT_SIDE,
         COLOUR1 => TEMP_COLOUR,
         COLOUR2 => TEMP_COLOUR);
   end loop;
end Draw;

-- PURPOSE : Draws the dynamic pick_list object into the window.

procedure Draw (LISTER : in LIST_D_TYPE) is

   CHARACTER_CODE : INTEGER_TYPE;
   COUNTER        : NATURAL;
   TEMP_STRING    : STRING (1..STRING_LENGTH);
   TEMP_COLOUR    : SCREEN_COLOUR_TYPE;
   POINTER        : NAME_LINK;

begin
   POINTER := LISTER.PRESENT_TOP;
   for COUNTER in 1..LISTER.LENGTHS loop
      if COUNTER = LISTER.CHOICE then
         TEMP_COLOUR := LISTER.BAR_COLOUR;
      else
         TEMP_COLOUR := LISTER.COLOUR;
      end if;
      if POINTER /= null then
         TEMP_STRING := POINTER.NAME;
```

```
            POINTER := POINTER.NEXT;
         else
            TEMP_STRING := "                ";
         end if;
         String_To_Window_Buffer (MESSAGE => ' ' & TEMP_STRING & ' ',
            ROW     => LISTER.TOP_ROW + COUNTER - 1,
            COLUMN  => LISTER.LEFT_SIDE,
            COLOUR1 => TEMP_COLOUR,
            COLOUR2 => TEMP_COLOUR);
      end loop;
      if LISTER.MAX_LENGTH > LISTER.LENGTHS then
         Character_To_Window_Buffer (CHARACTER_CODE => 30,
            COUNT   => 1,
            ROW     => LISTER.TOP_ROW,
            COLUMN  => LISTER.LEFT_SIDE + STRING_LENGTH + 3,
            COLOURS => LISTER.SCROLLER_COLOUR);
         for COUNTER in 1..(LISTER.LENGTHS - 2) loop
            if (COUNTER = ((LISTER.PRESENT_LENGTH + 1) * (LISTER.LENGTHS - 2))
               / LISTER.MAX_LENGTH then
               CHARACTER_CODE := 254;
            else
               CHARACTER_CODE := 176;
            end if;
            Character_To_Window_Buffer (CHARACTER_CODE => CHARACTER_CODE,
               COUNT   => 1,
               ROW     => LISTER.TOP_ROW + COUNTER,
               COLUMN  => LISTER.LEFT_SIDE + STRING_LENGTH + 3,
               COLOURS => LISTER.SCROLLER_COLOUR);
         end loop;
         Character_To_Window_Buffer (CHARACTER_CODE => 31,
            COUNT   => 1,
            ROW     => LISTER.TOP_ROW + LISTER.LENGTHS - 1,
            COLUMN  => LISTER.LEFT_SIDE + STRING_LENGTH + 3,
            COLOURS => LISTER.SCROLLER_COLOUR);
      end if;
   end Draw;

   -------------------------------------------------------------------------
   -- PURPOSE : Handles input that is directed to this static pick_list
   --           object, otherwise it ignores it.
   -------------------------------------------------------------------------

   procedure Handle_Input (LISTER : in out LIST_TYPE;
      LIST_DATA : in out LIST_ARRAY_TYPE;
      INPUTS    : in out INPUT_TYPE) is

      COUNTER   : POSITIVE;
      CHANGE    : BOOLEAN := TRUE;
      NEW_STATE : BOOLEAN;

   begin
      if not LISTER.ACTIVE then
         if INPUTS.OPERATION = OP_NON_ACCEPTING then
            LISTER.ACTIVE := TRUE;
            INPUTS.OPERATION := OP_DONE;
         else
            return;
         end if;
      elsif INPUTS.GOT_MOUSE and (INPUTS.BUTTON = LEFT_BUTTON) and
         NEW_PRESS then
         if (INPUTS.Y in LISTER.TOP_ROW..(LISTER.TOP_ROW + LISTER.LENGTHS-1))
            and (INPUTS.X in LISTER.LEFT_SIDE..
            (LISTER.LEFT_SIDE + LISTER.WIDTHS + 4)) then
            LISTER.CHOICE := INPUTS.Y - LISTER.TOP_ROW + 1;
            if LISTER.EXCLUSIVE then
               for COUNTER in 1..LISTER.LENGTHS loop
                  LIST_DATA (COUNTER).STATE := FALSE;
               end loop;
               LIST_DATA (LISTER.CHOICE).STATE := TRUE;
            else
               NEW_STATE := not (LIST_DATA (LISTER.CHOICE).STATE);
               LIST_DATA (LISTER.CHOICE).STATE := NEW_STATE;
            end if;
         else
            CHANGE := FALSE;
         end if;
         NEW_PRESS := FALSE;
      else
         case INPUTS.KEY is
            when KB_TAB =>
               LISTER.ACTIVE := FALSE;
               INPUTS.OPERATION := OP_NON_ACCEPTING;
            when KB_SPACE =>
               if not LISTER.EXCLUSIVE then
                  NEW_STATE := not (LIST_DATA (LISTER.CHOICE).STATE);
                  LIST_DATA (LISTER.CHOICE).STATE := NEW_STATE;
               end if;
            when KB_UP =>
               if LISTER.CHOICE = 1 then
                  LISTER.CHOICE := LISTER.LENGTHS;
               else
                  LISTER.CHOICE := LISTER.CHOICE - 1;
               end if;
            when KB_DOWN =>
               if LISTER.CHOICE = LISTER.LENGTHS then
                  LISTER.CHOICE := 1;
               else
                  LISTER.CHOICE := LISTER.CHOICE + 1;
               end if;
            when KB_HOME =>
               LISTER.CHOICE := 1;
            when KB_END =>
               LISTER.CHOICE := LISTER.LENGTHS;
            when others =>
               CHANGE := FALSE;
         end case;
         if INPUTS.BUTTON /= LEFT_BUTTON then
            NEW_PRESS := TRUE;
         end if;
      end if;
      if CHANGE then
         if LISTER.EXCLUSIVE then
            for COUNTER in 1..LISTER.LENGTHS loop
               LIST_DATA (COUNTER).STATE := FALSE;
            end loop;
            LIST_DATA (LISTER.CHOICE).STATE := TRUE;
         end if;
         Draw (LISTER => LISTER,
            LIST_DATA => LIST_DATA);
         Window_Buffer_To_Screen;
      end if;
      if INPUTS.OPERATION /= OP_NON_ACCEPTING then
         INPUTS.OPERATION := OP_DONE;
      end if;
   end if;
end Handle_Input;

-------------------------------------------------------------------------
-- PURPOSE : Handles input that is directed to this dynamic pick_list
--           object, otherwise it ignores it.
-------------------------------------------------------------------------

procedure Handle_Input (LISTER : in out LIST_D_TYPE;
   INPUTS : in out INPUT_TYPE) is

   CHANGE  : BOOLEAN := TRUE;
   COUNTER : NATURAL;

   ----------------------------------------------------------------------

   procedure Move_Up is

   begin
      if LISTER.CHOICE > 1 then
         LISTER.CHOICE := LISTER.CHOICE - 1;
         LISTER.CHOICE_POINTER := LISTER.CHOICE_POINTER.BACK;
         LISTER.PRESENT_LENGTH := LISTER.PRESENT_LENGTH - 1;
      elsif LISTER.CHOICE_POINTER.BACK /= null then
         LISTER.CHOICE_POINTER := LISTER.CHOICE_POINTER.BACK;
         LISTER.PRESENT_TOP := LISTER.PRESENT_TOP.BACK;
         LISTER.PRESENT_LENGTH := LISTER.PRESENT_LENGTH - 1;
      end if;
   end Move_Up;

   ----------------------------------------------------------------------

   procedure Move_Down is

   begin
      if LISTER.CHOICE_POINTER.NEXT /= null then
         if LISTER.CHOICE < LISTER.LENGTHS then
            LISTER.CHOICE := LISTER.CHOICE + 1;
            LISTER.CHOICE_POINTER := LISTER.CHOICE_POINTER.NEXT;
         else
            LISTER.CHOICE_POINTER := LISTER.CHOICE_POINTER.NEXT;
            LISTER.PRESENT_TOP := LISTER.PRESENT_TOP.NEXT;
         end if;
         LISTER.PRESENT_LENGTH := LISTER.PRESENT_LENGTH + 1;
      end if;
   end Move_Down;

   ----------------------------------------------------------------------

begin
   if INPUTS.GOT_MOUSE and (INPUTS.BUTTON = LEFT_BUTTON) then
      if (INPUTS.Y in LISTER.TOP_ROW..(LISTER.TOP_ROW + LISTER.LENGTHS-1))
         and (INPUTS.X in LISTER.LEFT_SIDE..
         (LISTER.LEFT_SIDE + STRING_LENGTH + 2)) then
         LISTER.PRESENT_LENGTH := LISTER.PRESENT_LENGTH - LISTER.CHOICE;
         LISTER.CHOICE := INPUTS.Y - LISTER.TOP_ROW + 1;
         LISTER.CHOICE_POINTER := LISTER.PRESENT_TOP;
         for COUNTER in 1..(LISTER.CHOICE - 1) loop
            if LISTER.CHOICE_POINTER.NEXT /= null then
               LISTER.CHOICE_POINTER := LISTER.CHOICE_POINTER.NEXT;
            else
               LISTER.CHOICE := COUNTER;
               exit;
            end if;
         end loop;
         LISTER.PRESENT_LENGTH := LISTER.PRESENT_LENGTH + LISTER.CHOICE;
      elsif (INPUTS.Y = LISTER.TOP_ROW) and
         (INPUTS.X = LISTER.LEFT_SIDE + STRING_LENGTH + 3) then
         Move_Up;
      elsif (INPUTS.Y = LISTER.TOP_ROW + LISTER.LENGTHS - 1) and
         (INPUTS.X = LISTER.LEFT_SIDE + STRING_LENGTH + 3) then
         Move_Down;
      else
         CHANGE := FALSE;
      end if;
      delay 0.1;
   else
      case INPUTS.KEY is
         when KB_DOWN  => Move_Down;
         when KB_UP    => Move_Up;
         when KB_ENTER => INPUTS.OPERATION := OP_OK;
         when others   => CHANGE := FALSE;
      end case;
   end if;
   if CHANGE then
      Draw (LISTER => LISTER);
      Window_Buffer_To_Screen;
      INPUTS.OPERATION := OP_DONE;
   end if;
end Handle_Input;

-------------------------------------------------------------------------
-- PURPOSE : Returns the chosen name as well as destroying the list.
-------------------------------------------------------------------------

procedure Get_Name (LISTER : in out LIST_D_TYPE;
   ITEM : out STRING) is

begin
   ITEM (1..STRING_LENGTH) := LISTER.CHOICE_POINTER.NAME;
   Destroy_List (LISTER => LISTER);
end Get_Name;

----------------------------------------------------------------------

end Link_Lists;

----------------------------------------------------------------------

package body Popup is

   OFFSET_Y : ROW_TYPE;
   OFFSET_X : COLUMN_TYPE;

-------------------------------------------------------------------------
-- PURPOSE : Sets all the data needed for the display and behaviour of the
```

```
--            forms object.
-------------------------------------------------------------------------

procedure Create (FORM : in out FORM_TYPE;
   FORM_DATA     : in FIELDS_TYPE;
   OFFSET_ROW    : in ROW_TYPE;
   OFFSET_COLUMN : in COLUMN_TYPE;
   TEXT_COLUMN   : in COLUMN_TYPE;
   VALUE_COLUMN  : in COLUMN_TYPE;
   ACTIVE_COLOUR : in SCREEN_COLOUR_TYPE;
   TEXT_COLOUR   : in SCREEN_COLOUR_TYPE;
   FIELD_COLOUR  : in SCREEN_COLOUR_TYPE) is

begin
   FORM.ACTIVE_FIELD  := 1;
   FORM.ACTIVE        := FALSE;
   FORM.INSERTING     := TRUE;
   FORM.CURSOR_ROW    := FORM_DATA (1).ROW;
   FORM.CURSOR_COLUMN := VALUE_COLUMN;
   FORM.TEXT_COLUMN   := TEXT_COLUMN;
   FORM.VALUE_COLUMN  := VALUE_COLUMN;
   FORM.ACTIVE_COLOUR := ACTIVE_COLOUR;
   FORM.TEXT_COLOUR   := TEXT_COLOUR;
   FORM.FIELD_COLOUR  := FIELD_COLOUR;
   OFFSET_Y           := OFFSET_ROW;
   OFFSET_X           := OFFSET_COLUMN;
   Change_Cursor_Shape (START_LINE => 7,
      ENDING_LINE => 8,
      DISPLAY     => FALSE);
end Create;

-------------------------------------------------------------------------
-- PURPOSE : Draws the forms object into the window.
-------------------------------------------------------------------------

procedure Draw (FORM : in FORM_TYPE;
   FORM_DATA : in FIELDS_TYPE) is

   COUNTER : INTEGER_TYPE;
   COLOUR  : SCREEN_COLOUR_TYPE;

begin
   for COUNTER in 1..FORM_DATA'last loop
      if FORM_DATA (COUNTER).ACCEPTING then
         if (COUNTER = FORM.ACTIVE_FIELD) and FORM.ACTIVE then
            COLOUR := FORM.ACTIVE_COLOUR;
         else
            COLOUR := FORM.FIELD_COLOUR;
         end if;
         String_To_Window_Buffer (MESSAGE => FORM_DATA (COUNTER).TEXT
               (1..FORM_DATA (COUNTER).TEXT_LENGTH),
            ROW       => FORM_DATA (COUNTER).ROW,
            COLUMN    => FORM.TEXT_COLUMN,
            COLOURS1 => FORM.TEXT_COLOUR,
            COLOURS2 => FORM.TEXT_COLOUR);
         String_To_Window_Buffer (MESSAGE => FORM_DATA (COUNTER).VALUE
               (1..FORM_DATA (COUNTER).VALUE_LENGTH),
            ROW       => FORM_DATA (COUNTER).ROW,
            COLUMN    => FORM.VALUE_COLUMN,
            COLOURS1 => COLOUR,
            COLOURS2 => COLOUR);
      end if;
   end loop;
   Set_Cursor (ROW => FORM.CURSOR_ROW + OFFSET_Y,
      COLUMN => FORM.CURSOR_COLUMN + OFFSET_X + 1);
end Draw;

-------------------------------------------------------------------------
-- PURPOSE : Handles input that is directed to this form object, otherwise
--           it ignores it.
-------------------------------------------------------------------------

procedure Handle_Input (FORM : in out FORM_TYPE;
   FORM_DATA : in out FIELDS_TYPE;
   INPUTS    : in out INPUT_TYPE) is

   ACT : INTEGER renames FORM.ACTIVE_FIELD;

   CHARACTER_CODE : CHARACTER;
   CHANGE         : BOOLEAN := TRUE;
   INDEX          : INTEGER := FORM.CURSOR_COLUMN - FORM.VALUE_COLUMN+1;

   KB_LOWER_E     : constant := 16#0045#;
   KB_UPPER_E     : constant := 16#0065#;

   -------------------------------------------------------------------
   -- PURPOSE : This processes the data in a field by converting the input
   --           string to an  integer or  real value.  If  no exception is
   --           raised during the conversion,  the value is checked to see
   --           if it is in a range specified by a minimum and maximum
   --           value.
   -------------------------------------------------------------------

   procedure Process_Field is

   begin
      ERROR_FIELD := ACT;
      Draw (FORM => FORM,
         FORM_DATA => FORM_DATA);
      if FORM_DATA (ACT).IS_INTEGER_TYPE then
         begin
            Texts.Convert_To_Integer (FROM =>
               FORM_DATA (ACT).VALUE (1..FORM_DATA (ACT).VALUE_LENGTH),
               ITEM => FORM_DATA (ACT).INTEGER_VALUE);
         exception
            when others =>
               raise INTEGER_ERROR;
         end;
         if FORM_DATA(ACT).INTEGER_VALUE < FORM_DATA(ACT).MIN_INTEGER then
            raise INTEGER_MIN_ERROR;
         elsif FORM_DATA (ACT).INTEGER_VALUE > FORM_DATA (ACT).MAX_INTEGER
            then
            raise INTEGER_MAX_ERROR;
         elsif FORM_DATA (ACT).TEST_AGAINST > 0 then
            if FORM_DATA (ACT).INTEGER_VALUE =
               FORM_DATA (FORM_DATA (ACT).TEST_AGAINST).INTEGER_VALUE then
               raise INTEGER_EQUAL_ERROR;
            end if;
         end if;
      else
         begin
            Texts.Convert_To_Real (FROM => FORM_DATA (ACT).VALUE,
               ITEM => FORM_DATA (ACT).REAL_VALUE);
         exception
            when others =>
               raise REAL_ERROR;
         end;
         if FORM_DATA (ACT).REAL_VALUE < FORM_DATA (ACT).MIN_REAL then
            raise REAL_MIN_ERROR;
         elsif FORM_DATA (ACT).REAL_VALUE > FORM_DATA (ACT).MAX_REAL then
            raise REAL_MAX_ERROR;
         end if;
      end if;
   end Process_Field;

   -------------------------------------------------------------------
   -- PURPOSE : Finds the  next field  of a  form.  If the  last field  is
   --           reached, the form is complete and the input record is set.
   -------------------------------------------------------------------

   procedure Next_Field (FORWARD : in BOOLEAN) is

   begin
      Process_Field;
      if FORWARD then
         loop
            if ACT < FORM_DATA'last then
               ACT := ACT + 1;
            else
               ACT := 1;
            end if;
            exit when FORM_DATA (ACT).ACCEPTING;
         end loop;
         FORM.CURSOR_COLUMN := FORM.VALUE_COLUMN;
      else
         loop
            if ACT > 1 then
               ACT := ACT - 1;
            else
               ACT := FORM_DATA'last;
            end if;
            exit when FORM_DATA (ACT).ACCEPTING;
         end loop;
         FORM.CURSOR_COLUMN := FORM.VALUE_COLUMN +
            FORM_DATA (ACT).ACTUAL_LENGTH;
         if (FORM_DATA (ACT).ACTUAL_LENGTH =
            FORM_DATA (ACT).VALUE_LENGTH) then
            FORM.CURSOR_COLUMN := FORM.CURSOR_COLUMN - 1;
         end if;
      end if;
      FORM.CURSOR_ROW := FORM_DATA (ACT).ROW;
   end Next_Field;

   -------------------------------------------------------------------
   -- PURPOSE : Moves the cursor to the next character to the right, or if
   --           the end of the field has been reached, then go to the next
   --           field.
   -------------------------------------------------------------------

   procedure Move_Cursor_Right is

   begin
      if (INDEX <= FORM_DATA (ACT).ACTUAL_LENGTH) and
         (INDEX < FORM_DATA (ACT).VALUE_LENGTH) then
         FORM.CURSOR_COLUMN := FORM.CURSOR_COLUMN + 1;
      else
         Next_Field (FORWARD => TRUE);
      end if;
   end Move_Cursor_Right;

   -------------------------------------------------------------------

begin
   if INPUTS.OPERATION = OP_MOVED then
      OFFSET_Y := INPUTS.Y;
      OFFSET_X := INPUTS.X;
      Set_Cursor (ROW => FORM.CURSOR_ROW + OFFSET_Y,
         COLUMN => FORM.CURSOR_COLUMN + OFFSET_X + 1);
   end if;
   if not FORM.ACTIVE then
      if INPUTS.OPERATION = OP_NON_ACCEPTING then
         FORM.ACTIVE := TRUE;
         Show_Cursor;
         INPUTS.OPERATION := OP_DONE;
      else
         return;
      end if;
   else
      case INPUTS.KEY is
         when KB_LEFT =>
            if FORM.CURSOR_COLUMN = FORM.VALUE_COLUMN then
               Next_Field (FORWARD => FALSE);
            else
               FORM.CURSOR_COLUMN := FORM.CURSOR_COLUMN - 1;
            end if;
         when KB_RIGHT =>
            Move_Cursor_Right;
         when KB_DOWN =>
            Next_Field (FORWARD => TRUE);
         when KB_UP =>
            Next_Field (FORWARD => FALSE);
         when KB_TAB =>
            Process_Field;
            Hide_Cursor;
            FORM.ACTIVE := FALSE;
            INPUTS.OPERATION := OP_NON_ACCEPTING;
         when KB_BACKSPACE =>
            if INDEX > 1 then
               FORM_DATA (ACT).VALUE
                  ((INDEX - 1)..(FORM_DATA (ACT).ACTUAL_LENGTH - 1)) :=
               FORM_DATA (ACT).VALUE
                  (INDEX..FORM_DATA(ACT).ACTUAL_LENGTH);
               FORM_DATA (ACT).VALUE(FORM_DATA(ACT).ACTUAL_LENGTH) := ' ';
               FORM_DATA (ACT).ACTUAL_LENGTH :=
                  FORM_DATA (ACT).ACTUAL_LENGTH - 1;
               FORM.CURSOR_COLUMN := FORM.CURSOR_COLUMN - 1;
            end if;
         when KB_DEL =>
            FORM_DATA (ACT).VALUE
               (INDEX..(FORM_DATA (ACT).ACTUAL_LENGTH - 1)) :=
```

```
                FORM_DATA (ACT).VALUE
                ((INDEX + 1)..FORM_DATA (ACT).ACTUAL_LENGTH);
             if INDEX <= FORM_DATA (ACT).ACTUAL_LENGTH then
                FORM_DATA (ACT).VALUE(FORM_DATA(ACT).ACTUAL_LENGTH) := ' ';
                FORM_DATA (ACT).ACTUAL_LENGTH :=
                   FORM_DATA (ACT).ACTUAL_LENGTH - 1;
             end if;
          when KB_INST =>
             FORM.INSERTING := not FORM.INSERTING;
             if FORM.INSERTING then
                Change_Cursor_Shape (START_LINE => 7,
                   ENDING_LINE => 8,
                   DISPLAY     => TRUE);
             else
                Change_Cursor_Shape (START_LINE => 1,
                   ENDING_LINE => 8,
                   DISPLAY     => TRUE);
             end if;
             CHANGE := FALSE;
          when KB_0..KB_9 | KB_UPPER_X | KB_LOWER_E| KB_MINUS| KB_PERIOD =>
             CHARACTER_CODE := CHARACTER'val (INPUTS.KEY);
             if (FORM_DATA (ACT).ACTUAL_LENGTH
                < FORM_DATA (ACT).VALUE_LENGTH) then
                if INDEX = (FORM_DATA (ACT).ACTUAL_LENGTH + 1) then
                   FORM_DATA (ACT).ACTUAL_LENGTH :=
                      FORM_DATA (ACT).ACTUAL_LENGTH + 1;
                elsif FORM.INSERTING then
                   FORM_DATA (ACT).VALUE
                      ((INDEX + 1)..(FORM_DATA (ACT).ACTUAL_LENGTH + 1))
                      := FORM_DATA (ACT).VALUE
                      (INDEX..FORM_DATA (ACT).ACTUAL_LENGTH);
                   FORM_DATA (ACT).ACTUAL_LENGTH
                      := FORM_DATA (ACT).ACTUAL_LENGTH + 1;
                end if;
                FORM_DATA (ACT).VALUE (INDEX) := CHARACTER_CODE;
                Move_Cursor_Right;
             elsif (not FORM.INSERTING) then
                FORM_DATA (ACT).VALUE (INDEX) := CHARACTER_CODE;
                Move_Cursor_Right;
             end if;
          when others =>
             CHANGE := FALSE;
       end case;
    end if;
    if CHANGE then
       Draw (FORM => FORM,
          FORM_DATA => FORM_DATA);
       Window_Buffer_To_Screen;
       if INPUTS.OPERATION /= OP_NON_ACCEPTING then
          INPUTS.OPERATION := OP_DONE;
       end if;
    end if;
end Handle_Input;

-------------------------------------------------------------------------------

end Forms;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package body Windows is

   MAX_SCREEN_LENGTH : constant := 25;
   MAX_SCREEN_WIDTH  : constant := 80;
   NORMAL_FRAME : constant FRAME_TYPE
      := (213, 205, 184, 179, 190, 212, 217, 205, 190, 180);  -- ╒═╕│└═┘┤
   MOVING_FRAME : constant FRAME_TYPE
      := (218, 196, 191, 179, 192, 196, 217, 196, 190, 180);  -- ┌─┐│└─┘┤
   MOVING_COLOUR : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => LIGHT_PURPLE, BACKGROUND => PURPLE, BLINK => FALSE);

   OFFSET,                 OLD_SCREEN_Y,
   OLD_SCREEN_X                         : INTEGER_TYPE;

-------------------------------------------------------------------------------
-- PURPOSE : Removes the window object from the physical screen.
-------------------------------------------------------------------------------

procedure Close is

begin
   Side_Text_Mouse;
   Main_Buffer_To_Screen;
end Close;

-------------------------------------------------------------------------------
-- PURPOSE : Sets all the data needed for the display and behaviour of the
--           window object.
-------------------------------------------------------------------------------

procedure Create (WINDOW : in out WINDOW_TYPE;
   NAME        : in STRING;
   KEYS_TEXT   : in STRING;
   TOP_ROW     : in ROW_TYPE;
   LEFT_SIDE   : in COLUMN_TYPE;
   LENGTHS     : in ROW_TYPE;
   WIDTHS      : in COLUMN_TYPE;
   TEXT_COLOUR : in SCREEN_COLOUR_TYPE;
   FRAME_COLOUR : in SCREEN_COLOUR_TYPE;
   TITLE_COLOUR : in SCREEN_COLOUR_TYPE) is

begin
   if (LENGTHS > 25) or (WIDTHS > 80) then
      raise WINDOW_TOO_LARGE;
   end if;
   WINDOW.NAME_LENGTH  := NAME'length;
   WINDOW.KEYS_LENGTH  := KEYS_TEXT'length;
   WINDOW.NAME (1..NAME'length) := NAME;
   WINDOW.KEYS_TEXT (1..KEYS_TEXT'length) := KEYS_TEXT;
   WINDOW.TOP_ROW      := TOP_ROW;
   WINDOW.LEFT_SIDE    := LEFT_SIDE;
   WINDOW.LENGTHS      := LENGTHS;
   WINDOW.WIDTHS       := WIDTHS;
   WINDOW.TEXT_COLOUR  := TEXT_COLOUR;
   WINDOW.FRAME_COLOUR := FRAME_COLOUR;
   WINDOW.TITLE_COLOUR := TITLE_COLOUR;
end Create;

-------------------------------------------------------------------------------
-- PURPOSE : Displays text  that has been  formatted  so that if the words
--           are not truncated at the right window border if they are less
--           than 10 characters long.
-------------------------------------------------------------------------------

procedure Paragrapher (WINDOW : in WINDOW_TYPE;
   LINE : in STRING;
   ROW  : in ROW_TYPE) is

   WRAP_AROUND : constant := 10;

   LINE_LENGTH         : INTEGER_TYPE := WINDOW.WIDTHS - 6;
   COUNTER1, COUNTER2 : NATURAL       := 0;
   ENDING, START       : NATURAL      := 1;

begin
   loop
      ENDING := LINE_LENGTH + START - 1;
      if ENDING < LINE'last then
         for COUNTER2 in 0..WRAP_AROUND loop
            if LINE (ENDING - COUNTER2) = ' ' then
               ENDING := ENDING - COUNTER2;
               exit;
            end if;
         end loop;
         String_To_Window_Buffer (MESSAGE => LINE (START..ENDING),
            ROW     => ROW + COUNTER1,
            COLUMN  => 2,
            COLOUR1 => WINDOW.TEXT_COLOUR,
            COLOUR2 => WINDOW.TEXT_COLOUR);
         START := ENDING + 1;
         COUNTER1 := COUNTER1 + 1;
      else
         String_To_Window_Buffer (MESSAGE => LINE (START..LINE'last),
            ROW     => ROW + COUNTER1,
            COLUMN  => 2,
            COLOUR1 => WINDOW.TEXT_COLOUR,
            COLOUR2 => WINDOW.TEXT_COLOUR);
         exit;
      end if;
   end loop;
end Paragrapher;

-------------------------------------------------------------------------------
-- PURPOSE : Draws the window object.
-------------------------------------------------------------------------------

procedure Draw (WINDOW : in WINDOW_TYPE;
   TEXT     : in STRING;
   TEXT_ROW : in ROW_TYPE) is

begin
   Window_Buffer_Setup (NUMBER_OF_ROW => WINDOW.LENGTHS,
      NUMBER_OF_COLUMNS => WINDOW.WIDTHS);
   Update (WINDOW => WINDOW,
      TEXT     => TEXT,
      TEXT_ROW => TEXT_ROW);
   New_Window_Location (TOP_ROW => WINDOW.TOP_ROW,
      LEFT_SIDE => WINDOW.LEFT_SIDE);
   Make_Shadow;
end Draw;

-------------------------------------------------------------------------------
-- PURPOSE : Draws the window object.  This is used when the window object
--           has not moved from it's initial position.
-------------------------------------------------------------------------------

procedure Update (WINDOW : in WINDOW_TYPE;
   TEXT     : in STRING;
   TEXT_ROW : in ROW_TYPE) is

begin
   Clear_Window_Buffer (COLOURS => WINDOW.TEXT_COLOUR);
   Frame_Window_Buffer (BORDERS => NORMAL_FRAME,
      COLOURS => WINDOW.FRAME_COLOUR);
   Window_Buffer_Title (TITLE_NAME => WINDOW.NAME (1..WINDOW.NAME_LENGTH),
      COLOURS => WINDOW.TITLE_COLOUR);
   Paragrapher (WINDOW => WINDOW,
      LINE => TEXT,
      ROW  => TEXT_ROW);
   String_To_Window_Buffer (
      MESSAGE  => WINDOW.KEYS_TEXT (1..WINDOW.KEYS_LENGTH),
      ROW      => WINDOW.LENGTHS - 2,
      COLUMN   => 2,
      COLOUR1 => WINDOW.TEXT_COLOUR,
      COLOUR2 => WINDOW.TEXT_COLOUR);
end Update;

-------------------------------------------------------------------------------
-- PURPOSE : Moves the window object  around the physical screen under the
--           control of the cursor keys.
-------------------------------------------------------------------------------

procedure Key_Move (WINDOW : in out WINDOW_TYPE;
   INPUTS : in out INPUT_TYPE) is

   CHANGE : BOOLEAN := FALSE;

begin
   case INPUTS.KEY is
      when KB_UP =>
         if WINDOW.TOP_ROW > 1 then
            WINDOW.TOP_ROW := WINDOW.TOP_ROW - 1;
         end if;
      when KB_DOWN =>
         if (WINDOW.TOP_ROW + WINDOW.LENGTHS) < MAX_SCREEN_LENGTH then
            WINDOW.TOP_ROW := WINDOW.TOP_ROW + 1;
         end if;
      when KB_LEFT =>
         if WINDOW.LEFT_SIDE > 1 then
            WINDOW.LEFT_SIDE := WINDOW.LEFT_SIDE - 1;
         end if;
      when KB_RIGHT =>
         if (WINDOW.LEFT_SIDE + WINDOW.WIDTHS + 1) < MAX_SCREEN_WIDTH then
            WINDOW.LEFT_SIDE := WINDOW.LEFT_SIDE + 1;
         end if;
      when KB_ENTER =>
         WINDOW.KEY_MOVING := FALSE;
         Frame_Window_Buffer (BORDERS => NORMAL_FRAME,
            COLOURS => WINDOW.FRAME_COLOUR);
```

```
            Window_Buffer_Title (
               TITLE_NAME => WINDOW.NAME (1..WINDOW.NAME_LENGTH),
               COLOURS    => WINDOW.TITLE_COLOUR);
            INPUTS.Y := WINDOW.TOP_ROW;
            INPUTS.X := WINDOW.LEFT_SIDE;
            INPUTS.OPERATION := OP_MOVED;
            INPUTS.KEY := KB_NO_KEY;
            CHANGE := TRUE;
         when others =>
            null;
      end case;
      if (WINDOW.TOP_ROW /= OLD_SCREEN_Y) or
         (WINDOW.LEFT_SIDE /= OLD_SCREEN_X) then
         CHANGE := TRUE;
         OLD_SCREEN_Y := WINDOW.TOP_ROW;
         OLD_SCREEN_X := WINDOW.LEFT_SIDE;
      end if;
      if CHANGE then
         Main_Buffer_To_Screen;
         New_Window_Location (TOP_ROW => WINDOW.TOP_ROW,
            LEFT_SIDE => WINDOW.LEFT_SIDE);
         Window_Buffer_To_Screen;
         Make_Shadow;
      end if;
      if INPUTS.OPERATION /= OP_MOVED then
         INPUTS.OPERATION := OP_DONE;
      end if;
   end Key_Move;

   -- ------------------------------------------------------------------
   -- PURPOSE : Moves the window object around the physical screen under the
   --           control of the mouse.
   -- ------------------------------------------------------------------

   procedure Mouse_Move (WINDOW : in out WINDOW_TYPE;
      INPUTS : in out INPUT_TYPE) is

      CHANGE : BOOLEAN := FALSE;

   begin
      if INPUTS.BUTTON = LEFT_BUTTON then
         if INPUTS.Y < (MAX_SCREEN_LENGTH - WINDOW.LENGTHS) then
            WINDOW.TOP_ROW := INPUTS.Y;
         else
            WINDOW.TOP_ROW := MAX_SCREEN_LENGTH - WINDOW.LENGTHS;
         end if;
         if (INPUTS.X - OFFSET) > (MAX_SCREEN_WIDTH - WINDOW.WIDTHS - 2) then
            OFFSET := INPUTS.X - MAX_SCREEN_WIDTH + WINDOW.WIDTHS + 2;
         elsif (INPUTS.X - OFFSET) < 0 then
            WINDOW.LEFT_SIDE := 1;
            OFFSET := INPUTS.X;
         else
            WINDOW.LEFT_SIDE := INPUTS.X - OFFSET + 1;
         end if;
         if (WINDOW.TOP_ROW /= OLD_SCREEN_Y) or
            (WINDOW.LEFT_SIDE /= OLD_SCREEN_X) then
            CHANGE := TRUE;
            OLD_SCREEN_Y := WINDOW.TOP_ROW;
            OLD_SCREEN_X := WINDOW.LEFT_SIDE;
         end if;
      else
         WINDOW.MOUSE_MOVING := FALSE;
         CHANGE := TRUE;
         Frame_Window_Buffer (BORDERS => NORMAL_FRAME,
            COLOURS => WINDOW.FRAME_COLOUR);
         Window_Buffer_Title (
            TITLE_NAME => WINDOW.NAME (1..WINDOW.NAME_LENGTH),
            COLOURS    => WINDOW.TITLE_COLOUR);
         INPUTS.Y := WINDOW.TOP_ROW;
         INPUTS.X := WINDOW.LEFT_SIDE;
         INPUTS.OPERATION := OP_MOVED;
      end if;
      if CHANGE then
         Main_Buffer_To_Screen;
         New_Window_Location (TOP_ROW => WINDOW.TOP_ROW,
            LEFT_SIDE => WINDOW.LEFT_SIDE);
         Window_Buffer_To_Screen;
         Make_Shadow;
      end if;
      if INPUTS.OPERATION /= OP_MOVED then
         INPUTS.OPERATION := OP_DONE;
      end if;
   end Mouse_Move;

   -- ------------------------------------------------------------------
   -- PURPOSE : Handles input that is directed to this window object,
   --           otherwise it ignores it.  Global coordinates are converted to
   --           local coordinates.
   -- ------------------------------------------------------------------

   procedure Handle_Input (WINDOW : in out WINDOW_TYPE;
      INPUTS : in out INPUT_TYPE) is

   begin
      if WINDOW.MOUSE_MOVING then
         Mouse_Move (INPUTS => INPUTS, WINDOW => WINDOW);
      elsif WINDOW.KEY_MOVING then
         Key_Move (INPUTS => INPUTS, WINDOW => WINDOW);
      elsif INPUTS.GOT_MOUSE and (INPUTS.Y = WINDOW.TOP_ROW) and
         (INPUTS.BUTTON = LEFT_BUTTON) and (INPUTS.X in WINDOW.LEFT_SIDE..
         (WINDOW.LEFT_SIDE + WINDOW.WIDTHS - 1)) then
         OFFSET := INPUTS.X - WINDOW.LEFT_SIDE + 1;
         OLD_SCREEN_Y := WINDOW.TOP_ROW;
         OLD_SCREEN_X := WINDOW.LEFT_SIDE;
         WINDOW.MOUSE_MOVING := TRUE;
         Frame_Window_Buffer (BORDERS => MOVING_FRAME,
            COLOURS => MOVING_COLOUR);
         Window_Buffer_To_Screen;
         INPUTS.OPERATION := OP_DONE;
      elsif INPUTS.OPERATION = OP_MOVE then
         WINDOW.KEY_MOVING := TRUE;
         OLD_SCREEN_Y := WINDOW.TOP_ROW;
         OLD_SCREEN_X := WINDOW.LEFT_SIDE;
         Frame_Window_Buffer (BORDERS => MOVING_FRAME,
            COLOURS => MOVING_COLOUR);
         Window_Buffer_To_Screen;
         INPUTS.OPERATION := OP_DONE;
      else
         INPUTS.Y := INPUTS.Y - WINDOW.TOP_ROW;
         INPUTS.X := INPUTS.X - WINDOW.LEFT_SIDE + 1;
      end if;
   end Handle_Input;

   -- ------------------------------------------------------------------

end Windows;

-- ------------------------------------------------------------------

end Screen_Objects;
```

Screens

```
with  Ada_Extensions;
use   Ada_Extensions;

package Screens is

-------------------------------------------------------------------------
-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This contains all the low-level  user interface objects - the
--           subscreens.
-------------------------------------------------------------------------

-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------

   pragma SUPPRESS   (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------

   USER_ESCAPE : exception;

   HP_SUB_ERRORS             : constant := 101;
   HP_SUB_PROMPTS_MULTI      : constant := 102;
   HP_SUB_PROMPTS_FILES      : constant := 103;
   HP_SUB_PROMPTS_BOOLEAN    : constant := 104;
   HP_SUB_PROMPTS_INTEGER    : constant := 105;
   HP_SUB_PROMPTS_REAL       : constant := 106;
   HP_SUB_PROMPTS_STRING     : constant := 107;
   HP_SUB_STATUS             : constant := 108;

-------------------------------------------------------------------------

   procedure Continue_Processing;

   function Halt_Processing return BOOLEAN;

   procedure Initialise_User_Interface (Y_RESOLUTION : out INTEGER_TYPE);

-------------------------------------------------------------------------

   package Errors is

      procedure Write (MESSAGE : in STRING;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_ERRORS);

   end Errors;

-------------------------------------------------------------------------

   package Prompts is

      procedure Add_To_List (FILE_NAME : in STRING);

      procedure Read (FORM : in out FIELDS_TYPE;
         LIST      : in out LIST_ARRAY_TYPE;
         EXCLUSIVE : in BOOLEAN := TRUE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (FORM : in out FIELDS_TYPE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (LIST : in out LIST_ARRAY_TYPE;
         EXCLUSIVE : in BOOLEAN := TRUE;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI);

      procedure Read (FILE_NAME : out STRING;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_FILES);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_BOOLEAN;
         ITEM      : out BOOLEAN);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_INTEGER;
         ITEM      : out INTEGER_TYPE);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_REAL;
         ITEM      : out REAL_TYPE);

      procedure Read (MESSAGE : in STRING;
         HELP_INFO   : in  INTEGER_TYPE := HP_SUB_PROMPTS_STRING;
         ITEM        : out STRING;
         ITEM_LENGTH : out POSITIVE);

   end Prompts;

-------------------------------------------------------------------------

   package Status is

      procedure Write (MESSAGE : in STRING;
         HELP_INFO : in INTEGER_TYPE := HP_SUB_STATUS);

      procedure Wait;

   end Status;

-------------------------------------------------------------------------

   package Main is

      type ATTRIBUTE_TYPE is (BOLD, HEADING, NORMAL, REVERSED,UNDERLINED);

      procedure Clear;

      procedure Cursor_Set  (ROW : in  ROW_TYPE;
         COLUMN : in  COLUMN_TYPE);

      procedure Cursor_Next_Line;

      procedure Open_Screen_Devices;

      procedure Process_Title (TITLE : in STRING);

      procedure Write (ITEM : in INTEGER_TYPE;
         ITEM_WIDTH : in INTEGER_TYPE := INTEGER_TYPE'width);

      procedure Write (ITEM : in REAL_TYPE;
         WHOLE        : in INTEGER_TYPE := 7;
         FRACTION     : in INTEGER_TYPE := REAL_TYPE'digits - 1;
         EXPONENT     : in INTEGER_TYPE := 3;
         ENG_NOTATION : in BOOLEAN      := FALSE);

      procedure Write (ITEM : in STRING;
         FORM : in ATTRIBUTE_TYPE := NORMAL);

   end Main;

-------------------------------------------------------------------------

end Screens;
```

```
with  Dos_Environment,
      Bold,
      Inputter,
      Screen_Objects,
      Text_IO;

use   Dos_Environment,
      Inputter,
      Screen_Objects;

package body Screens is

-------------------------------------------------------------------------
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add  unnecessary  checking  code.  Checks  can be  introduced  again by
-- omitting the SUPPRESS pragma for that check.
-------------------------------------------------------------------------

   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);

-------------------------------------------------------------------------

   INPUT_LINES_TOTAL_LENGTH   : constant := 64;
   PICK_LISTS_DISPLAY_LENGTH  : constant := 10;
   MP_FORM_PROCESSED          : constant := 15;

   BORDER_COLOUR : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => WHITE, BACKGROUND => LIGHT_GRAY, BLINK => FALSE);

   EMPTY_FORM : FIELDS_TYPE (1..1) := (1 =>
      (VALUE_LENGTH     => 1,
       TEXT_LENGTH      => 1,
       IS_INTEGER_TYPE  => TRUE,
       ACCEPTING        => FALSE,
       TEST_AGAINST     => 0,
       ROW              => 1,
       ACTUAL_LENGTH    => 0,
       VALUE            => " ",
       TEXT             => " ",
       MAX_INTEGER      => 0,
       MIN_INTEGER      => 0,
       INTEGER_VALUE    => 0));

   EMPTY_LIST : LIST_ARRAY_TYPE (1..1)
      := (1 => (TEXT_LENGTH => 1,  TEXT => " ", STATE => FALSE));

-------------------------------------------------------------------------

   OBJ_NONE          : constant := 0;
   OBJ_BL_BUTTONS    : constant := 1;
   OBJ_INTEGER_LINER : constant := 2;
   OBJ_REAL_LINER    : constant := 3;
   OBJ_STRING_LINER  : constant := 4;
   OBJ_PICKER        : constant := 5;
   OBJ_MULTI         : constant := 6;

   type STATUS_TYPE is
      record
         OBJECT_ACTIVE  : INTEGER_TYPE;
         CURSOR_ACTIVE  : BOOLEAN;
         NON_ACCEPTANCE : BOOLEAN;
         HELP_INFO      : INTEGER_TYPE;
         TEXT_LENGTH    : NATURAL;
         TEXT           : STRING (1..1_0);
      end record;

   type BUTTON_HOLDER_ARRAY is array (1..4) of Buttons.BUTTON_TYPE;

   type SCREEN_TYPE is
      record
         SCR         : Windows.WINDOW_TYPE;
         BUTTONS     : BUTTON_HOLDER_ARRAY;
         BL_BUTTONS  : Boolean_Buttons.BOOLEAN_BUTTON_TYPE;
         LINER       : Input_Liners.LINE_TYPE;
         PICKER      : Pick_Lists.LIST_TYPE;
         PICKER2     : Pick_Lists.LIST_O_TYPE;
         FORMER      : Forms.FORM_TYPE;
         STATUS      : STATUS_TYPE;
      end record;
```

```
   SCREEN : SCREEN_TYPE;
   INPUTS : Inputter.INPUT_TYPE)

-- PURPOSE : Waits for a key to be pressed on the keyboard.

procedure Continue_Processing is

   KEY : KEY_VALUES_TYPE;

begin
   Get_Keyboard_Character (KEY => KEY);
end Continue_Processing;

-- PURPOSE : Checks to see if a key has been pressed and  if so,  discards
--           the key that was pressed.

function Halt_Processing return BOOLEAN is

   KEY : KEY_VALUES_TYPE;
   HALTING : BOOLEAN;

begin
   HALTING := Keyboard_Character_Ready;
   if HALTING then
      Get_Keyboard_Character (KEY => KEY);
   end if;
   return HALTING;
end Halt_Processing;

-- PURPOSE : Initialises the user interface  by setting initial values for
--           variables in the Dos_Environment package. The inputter object
--           is initialised and the screen is cleared.  Finally  the  Main
--           Title is displayed.

procedure Initialise_User_Interface (Y_RESOLUTION : out INTEGER_TYPE) is

begin
   Screen_Setup (Y_RESOLUTION => Y_RESOLUTION);
   Inputter.Set_Up;
   Hide_Cursor;
   Clear_Main_Buffer (MAIN_COLOURS => (FOREGROUND => LIGHT_GRAY,
         BACKGROUND => BLUE, BLINK => FALSE));
   Main_Buffer_To_Screen;
   Help.Initialise_Help_Path;
   Main_Title;
end Initialise_User_Interface;

-------------------- GENERALISED SUBSCREEN METHODS --------------------

-- PURPOSE : Sets initial values for the  appearance and  behaviour of the
--           subscreen. A subscreen consists of various screen objects.

procedure Create (SCREEN : in out SCREEN_TYPE;
   SUB_WIDTH    : in COLUMN_TYPE          := 44;
   SUB_LENGTH   : in ROW_TYPE             := 12;
   LIST_DATA    : in LIST_ARRAY_TYPE      := EMPTY_LIST;
   EXCLUSIVE    : in BOOLEAN              := TRUE;
   FORM_DATA    : in FIELDS_TYPE          := EMPTY_FORM;
   FRAME_COLOUR : in SCREEN_COLOUR_TYPE := BORDER_COLOUR;
   TITLE_COLOUR : in SCREEN_COLOUR_TYPE := BORDER_COLOUR;
   NAMES        : in STRING) is

   WINDOW_COLOUR        : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => BLACK, BACKGROUND => LIGHT_GRAY, BLINK => FALSE);
   BUTTON_COLOUR        : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => YELLOW, BACKGROUND => GREEN, BLINK => FALSE);
   BUTTON_SELECT_COLOUR : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => WHITE, BACKGROUND => GREEN, BLINK => FALSE);
   BUTTON_UNDER_COLOUR  : constant SCREEN_COLOUR_TYPE :=
      (FOREGROUND => LIGHT_GRAY, BACKGROUND => BLACK, BLINK => FALSE);

   WINDOW_LENGTH : INTEGER_TYPE := SUB_LENGTH;
   WINDOW_WIDTH  : INTEGER_TYPE := SUB_WIDTH;
   KEYS_LENGTH   : INTEGER_TYPE := 0;
   KEYS          : STRING (1..21);

begin
   Buttons.Create (BUTTON => SCREEN.BUTTONS (1),
      NAME          => " =O=k ",
      OPERATION     => OP_OK,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS  => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 2,
      HOT_KEY       => KB_ALT_O);
   Buttons.Create (BUTTON => SCREEN.BUTTONS (2),
      NAME          => " =C=ancel ",
      OPERATION     => OP_CANCEL,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS  => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 9,
      HOT_KEY       => KB_ALT_C);
   Buttons.Create (BUTTON => SCREEN.BUTTONS (4),
      NAME          => " =M=ove ",
      OPERATION     => OP_MOVE,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS  => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 20,
      HOT_KEY       => KB_ALT_M);
   Buttons.Create (BUTTON => SCREEN.BUTTONS (3),
      NAME          => " =H=elp ",
      OPERATION     => OP_HELP,
      COLOURS       => BUTTON_COLOUR,
      SELECT_COLOURS => BUTTON_SELECT_COLOUR,
      UNDER_COLOURS  => BUTTON_UNDER_COLOUR,
      ROW           => 1,
      COLUMN        => 29,
      HOT_KEY       => KB_ALT_H);
   case SCREEN.STATUS.OBJECT_ACTIVE is
      when OBJ_BL_BUTTONS =>
         Boolean_Buttons.Create (BL_BUTTON => SCREEN.BO_BUTTONS,
            COLOURS        => (FOREGROUND => PURPLE, BACKGROUND => GREEN,
               BLINK => FALSE),
            SELECT_COLOURS => (FOREGROUND => LIGHT_BLUE,
               BACKGROUND => GREEN, BLINK => FALSE),
            UNDER_COLOURS  => BUTTON_UNDER_COLOUR,
            TRUE_ROW       => 7,
            TRUE_COLUMN    => 9,
            FALSE_ROW      => 7,
            FALSE_COLUMN   => 29);

      when OBJ_INTEGER_LINER | OBJ_REAL_LINER | OBJ_STRING_LINER =>
         Input_Lines.Create (LINER => SCREEN.LINER,
            ROW            => 7,
            COLUMN         => 4,
            OFFSET_ROW     => (ROW_TYPE'last - WINDOW_LENGTH) / 2 + 1,
            OFFSET_COLUMN  => (COLUMN_TYPE'last - WINDOW_WIDTH) / 2 + 1,
            TOTAL_LENGTH   => INPUT_LINES_TOTAL_LENGTH,
            DISPLAY_LENGTH => 25,
            COLOURS        => (FOREGROUND => BLACK, BACKGROUND => CYAN,
               BLINK => FALSE));

      when OBJ_PICKER =>
         Pick_Lists.Create (LISTER => SCREEN.PICKER2,
            ROW            => 3,
            COLUMN         => 3,
            DISPLAY_LENGTH => PICK_LISTS_DISPLAY_LENGTH,
            COLOURS        => (FOREGROUND => BLACK, BACKGROUND => GREEN,
               BLINK => FALSE),
            BAR_COLOURS    => (FOREGROUND => WHITE, BACKGROUND => PURPLE,
               BLINK => FALSE),
            SCROLLER_COLOURS => (FOREGROUND => WHITE, BACKGROUND => BLACK,
               BLINK => FALSE));

      when OBJ_MULTI =>
         declare
            COUNTER                  : INTEGER_TYPE;
            LEAST_WIDTH              : INTEGER_TYPE := SUB_WIDTH;
            FORM_TEXT,     FORM_VALUE,
            FORM_ROW,      LIST_TEXT   : INTEGER_TYPE := 0;
         begin
            if (FORM_DATA/= EMPTY_FORM) and (LIST_DATA /= EMPTY_LIST) then
               KEYS_LENGTH := 15;
               KEYS := "Tab=List<->Form";
               LEAST_WIDTH := 36;
            end if;
            if LIST_DATA /= EMPTY_LIST then
               for COUNTER in 1..LIST_DATA'last loop
                  if LIST_DATA (COUNTER).TEXT_LENGTH > LIST_TEXT then
                     LIST_TEXT := LIST_DATA (COUNTER).TEXT_LENGTH;
                  end if;
               end loop;
               Pick_Lists.Create (LISTER => SCREEN.PICKER,
                  ROW            => 4,
                  COLUMN         => 3,
                  LIST_DATA      => LIST_DATA,
                  EXCLUSIVE      => EXCLUSIVE,
                  WIDTHS         => LIST_TEXT + 1,
                  COLOURS        => (FOREGROUND => BLACK,
                     BACKGROUND => GREEN, BLINK => FALSE),
                  BAR_COLOURS    => (FOREGROUND => WHITE,
                     BACKGROUND => PURPLE, BLINK => FALSE));
               LIST_TEXT := LIST_TEXT + 1;
            else
               LIST_TEXT := 3;
            end if;
            if FORM_DATA /= EMPTY_FORM then
               for COUNTER in 1..FORM_DATA'last loop
                  if FORM_DATA (COUNTER).ACCEPTING then
                     if FORM_DATA (COUNTER).VALUE_LENGTH > FORM_VALUE then
                        FORM_VALUE := FORM_DATA (COUNTER).VALUE_LENGTH;
                     end if;
                     if FORM_DATA (COUNTER).TEXT_LENGTH > FORM_TEXT then
                        FORM_TEXT := FORM_DATA (COUNTER).TEXT_LENGTH;
                     end if;
                     if FORM_DATA (COUNTER).ROW > FORM_ROW then
                        FORM_ROW := FORM_DATA (COUNTER).ROW;
                     end if;
                  end if;
               end loop;
               FORM_TEXT  := FORM_TEXT + 2;
               FORM_VALUE := FORM_VALUE + 2;
            end if;
            WINDOW_WIDTH := LIST_TEXT + FORM_TEXT + FORM_VALUE + 3;
            if WINDOW_WIDTH < LEAST_WIDTH then
               WINDOW_WIDTH := LEAST_WIDTH;
            end if;
            WINDOW_LENGTH := FORM_ROW + 4;
            if WINDOW_LENGTH < (LIST_DATA'length + 6) then
               WINDOW_LENGTH := LIST_DATA'length + 8;
            end if;
            Forms.Create (FORM => SCREEN.FORMER,
               FORM_DATA      => FORM_DATA,
               OFFSET_ROW     => (ROW_TYPE'last - WINDOW_LENGTH) / 2 + 1,
               OFFSET_COLUMN  => (COLUMN_TYPE'last - WINDOW_WIDTH) / 2 + 1,
               TEXT_COLUMN    => LIST_TEXT,
               VALUE_COLUMN   => LIST_TEXT + FORM_TEXT,
               ACTIVE_COLOUR  => (FOREGROUND => WHITE,
                  BACKGROUND => PURPLE, BLINK => FALSE),
               TEXT_COLOUR    => (FOREGROUND => BLACK,
                  BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
               FIELD_COLOUR   => (FOREGROUND => BLACK, BACKGROUND => CYAN,
                  BLINK => FALSE));
         end;

      when others => null;
   end case;
   Windows.Create (WINDOW => SCREEN.SCR,
      NAME          => NAMES,
      KEYS_TEXT     => "Esc=Escape  F1=Help  Enter=Ok  "
                       & KEYS (1..KEYS_LENGTH),
      TOP_ROW       => (ROW_TYPE'last - WINDOW_LENGTH) / 2 + 1,
      LEFT_SIDE     => (COLUMN_TYPE'last - WINDOW_WIDTH) / 2 + 1,
```

```
      LENGTHS      => WINDOW_LENGTH,
      WIDTHS       => WINDOW_WIDTH,
      TEXT_COLOUR  => WINDOW_COLOUR,
      FRAME_COLOUR => FRAME_COLOUR,
      TITLE_COLOUR => TITLE_COLOUR);
end Create;

-- ------------------------------------------------------------------
-- PURPOSE : Draws the subscreen in the window buffer which is then copied
--           to the physical screen.
-- ------------------------------------------------------------------

procedure Draw (SCREEN : in out SCREEN_TYPE;
   LIST_DATA : in LIST_ARRAY_TYPE := EMPTY_LIST;
   FORM_DATA : in FIELDS_TYPE     := EMPTY_FORM;
   UPDATING  : in BOOLEAN         := FALSE) is

   COUNTER : INTEGER_TYPE;

begin
   if UPDATING then
      Windows.Update (WINDOW => SCREEN.SCR,
         TEXT     => SCREEN.STATUS.TEXT (1..SCREEN.STATUS.TEXT_LENGTH),
         TEXT_ROW => 3);
   else
      if SCREEN.STATUS.CURSOR_ACTIVE then
         Show_Cursor;
      end if;
      Windows.Draw (WINDOW => SCREEN.SCR,
         TEXT     => SCREEN.STATUS.TEXT (1..SCREEN.STATUS.TEXT_LENGTH),
         TEXT_ROW => 3);
   end if;
   for COUNTER in 1..SCREEN.BUTTONS'last loop
      Buttons.Draw (BUTTON => SCREEN.BUTTONS (COUNTER));
   end loop;

   case SCREEN.STATUS.OBJECT_ACTIVE is
      when OBJ_BL_BUTTONS =>
         Boolean_Buttons.Draw (BL_BUTTON => SCREEN.BL_BUTTONS);

      when OBJ_INTEGER_LINER | OBJ_REAL_LINER | OBJ_STRING_LINER =>
         Input_Lines.Draw (LINER => SCREEN.LINER);

      when OBJ_PICKER =>
         Pick_Lists.Draw (LISTER => SCREEN.PICKER2);

      when OBJ_MULTI =>
         if LIST_DATA /= EMPTY_LIST then
            Pick_Lists.Draw (LISTER => SCREEN.PICKER,
               LIST_DATA => LIST_DATA);
         end if;
         if FORM_DATA /= EMPTY_FORM then
            Forms.Draw (FORM => SCREEN.FORMER,
               FORM_DATA => FORM_DATA);
         end if;

      when others => null;
   end case;
   Window_Buffer_To_Screen;
   Show_Text_Mouse;
end Draw;

-- ------------------------------------------------------------------
-- PURPOSE : Converts the data in a field of a form to an integer or real
--           number. If there is an error, an error message will be
--           displayed.
-- ------------------------------------------------------------------

procedure Process_Form (SCREEN : in out SCREEN_TYPE;
   FORM_DATA : in out FIELDS_TYPE;
   LIST_DATA : in LIST_ARRAY_TYPE;
   INPUTS    : in out Inputter.INPUT_TYPE) is

   DISPLAY_LINE : Texts.TEXT_TYPE;

   -- ---------------------------------------------------------------
   -- PURPOSE : Displays an appropriate error message for a predefined
   --           time and then removes it.
   -- ---------------------------------------------------------------

   procedure Field_Error (MESSAGE : in STRING) is

   begin
      String_To_Window_Buffer (MESSAGE => "The value " & MESSAGE,
         ROW      => 3,
         COLUMN   => 2,
         COLOURS1 => (FOREGROUND => LIGHT_RED, BACKGROUND => LIGHT_GRAY,
            BLINK => FALSE),
         COLOURS2 => (FOREGROUND => LIGHT_RED, BACKGROUND => LIGHT_GRAY,
            BLINK => FALSE));
      Window_Buffer_To_Screen;
      delay 1.5;
      Draw (SCREEN => SCREEN,
         FORM_DATA => FORM_DATA,
         LIST_DATA => LIST_DATA,
         UPDATING  => TRUE);
   end Field_Error;

   -- ---------------------------------------------------------------

begin
   Forms.Handle_Input (FORM => SCREEN.FORMER,
      FORM_DATA => FORM_DATA,
      INPUTS    => INPUTS);
   if INPUTS.OPERATION = OP_NON_ACCEPTING then
      SCREEN.STATUS.NON_ACCEPTANCE := TRUE;
   end if;
   INPUTS.OPERATION := OP_FORM_PROCESSED;
exception
   when Forms.INTEGER_ERROR =>
      Texts.Combine (ITEM => "is not a recognised integer.",
         TEXT_LINE => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
   when Forms.INTEGER_MIN_ERROR =>
      Texts.Combine (ITEM => "must be >=",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => FORM_DATA (Forms.ERROR_FIELD).MIN_INTEGER,
         ITEM_WIDTH => 4,
         TEXT_LINE  => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
   when Forms.INTEGER_MAX_ERROR =>
      Texts.Combine (ITEM => "must be <=",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => FORM_DATA (Forms.ERROR_FIELD).MAX_INTEGER,
         ITEM_WIDTH => 4,
         TEXT_LINE  => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
   when Forms.INTEGER_EQUAL_ERROR =>
      Field_Error (MESSAGE => "cannot equal other group values.");
   when Forms.REAL_ERROR =>
      Texts.Combine (ITEM => "is not a recognised real.",
         TEXT_LINE => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
   when Forms.REAL_MIN_ERROR =>
      Texts.Combine (ITEM => "must be >=",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => FORM_DATA (Forms.ERROR_FIELD).MIN_REAL,
         FRACTION  => 5,
         TEXT_LINE => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
   when Forms.REAL_MAX_ERROR =>
      Texts.Combine (ITEM => "must be <=",
         TEXT_LINE => DISPLAY_LINE);
      Texts.Combine (ITEM => FORM_DATA (Forms.ERROR_FIELD).MAX_REAL,
         FRACTION  => 5,
         TEXT_LINE => DISPLAY_LINE);
      Field_Error (MESSAGE =>
            DISPLAY_LINE.TEXT (1..DISPLAY_LINE.TEXT_INDEX));
end Process_Form;

-- ------------------------------------------------------------------
-- PURPOSE : Handles the input to the subscreen.  Any input that does not
--           involve the subscreen is ignored.
-- ------------------------------------------------------------------

procedure Handle_Input (SCREEN : in out SCREEN_TYPE;
   FORM_DATA : in out FIELDS_TYPE;
   LIST_DATA : in out LIST_ARRAY_TYPE;
   INPUTS    : in out Inputter.INPUT_TYPE) is

   KB_UPPER_Y  : constant := 16#0059#;
   KB_LOWER_Y  : constant := 16#0079#;
   KB_UPPER_N  : constant := 16#004E#;
   KB_LOWER_N  : constant := 16#006E#;
   KB_UPPER_E  : constant := 16#0045#;
   KB_LOWER_E  : constant := 16#0065#;

   COUNTER : INTEGER_TYPE;

begin
   Windows.Handle_Input (WINDOW => SCREEN.SCR, INPUTS => INPUTS);
   case INPUTS.KEY is
      when KB_ENTER => INPUTS.KEY := KB_ALT_O;
      when KB_F1    => INPUTS.KEY := KB_ALT_H;
      when KB_ESC   => INPUTS.KEY := KB_ALT_C;
      when others   => null;
   end case;
   COUNTER := 1;
   loop
      Buttons.Handle_Input (BUTTON => SCREEN.BUTTONS (COUNTER),
         INPUTS => INPUTS);
      exit when (INPUTS.OPERATION /= OP_DO) or
         (COUNTER = SCREEN.BUTTONS'last);
      COUNTER := COUNTER + 1;
   end loop;
   case INPUTS.OPERATION is
      when OP_CANCEL =>
         if SCREEN.STATUS.CURSOR_ACTIVE then
            Hide_Cursor;
         end if;
         Windows.Close;
         raise USER_ESCAPE;

      when OP_HELP =>
         Windows.Close;
         if SCREEN.STATUS.CURSOR_ACTIVE then
            Hide_Cursor;
         end if;
         Help.Show (HELP_REQUIRED => SCREEN.ST.T.*.HELP_INFO);
         if SCREEN.STATUS.CURSOR_ACTIVE then
            Show_Cursor;
         end if;
         Show_Text_Mouse;
         Draw (SCREEN => SCREEN,
            LIST_DATA => LIST_DATA,
            FORM_DATA => FORM_DATA);

      when OP_MOVE =>
         Windows.Handle_Input (WINDOW => SCREEN.SCR,
            INPUTS => INPUTS);

      when OP_MOVED =>
         case SCREEN.STATUS.OBJECT_ACTIVE is
            when OBJ_INTEGER_LINER | OBJ_REAL_LINER | OBJ_STRING_LINER =>
               Input_Lines.Handle_Input (LINER => SCREEN.LINER,
                  INPUTS => INPUTS);
            when OBJ_MULTI =>
               Forms.Handle_Input (FORM => SCREEN.FORMER,
                  FORM_DATA => FORM_DATA,
                  INPUTS    => INPUTS);
            when others => null;
         end case;

      when OP_DO =>
         case SCREEN.STATUS.OBJECT_ACTIVE is
            when OBJ_BL_BUTTONS =>
               case INPUTS.KEY is
                  when KB_UPPER_Y | KB_LOWER_Y => INPUTS.KEY := KB_ALT_Y;
                  when KB_UPPER_N | KB_LOWER_N => INPUTS.KEY := KB_ALT_N;
                  when others => null;
               end case;
               Boolean_Buttons.Handle_Input (BL_BUTTON=>SCREEN.BL_BUTTONS,
                  INPUTS => INPUTS);

            when OBJ_INTEGER_LINER =>
```

```
            if (INPUTS.KEY in KB_0..KB_9) or (INPUTS.KEY = KB_MINUS)
               or (INPUTS.KEY=KB_BACKSPACE) or (INPUTS.KEY > 255) then
               Input_Lines.Handle_Input (LINER => SCREEN.LINER,
                  INPUTS => INPUTS);
            end if;

         when OBJ_REAL_LINER =>
            if (INPUTS.KEY in KB_0..KB_9) or (INPUTS.KEY = KB_MINUS)
               or (INPUTS.KEY = KB_PERIOD) or (INPUTS.KEY = KB_UPPER_E)
               or (INPUTS.KEY=KB_LOWER_E) or (INPUTS.KEY=KB_BACKSPACE)
               or (INPUTS.KEY > 255) then
               Input_Lines.Handle_Input (LINER => SCREEN.LINER,
                  INPUTS => INPUTS);
            end if;

         when OBJ_STRING_LINER =>
            Input_Lines.Handle_Input (LINER => SCREEN.LINER,
               INPUTS => INPUTS);

         when OBJ_PICKER =>
            Pick_Lists.Handle_Input (LISTER => SCREEN.PICKER2,
               INPUTS => INPUTS);

         when OBJ_MULTI =>
            if SCREEN.STATUS.NON_ACCEPTANCE then
               SCREEN.STATUS.NON_ACCEPTANCE := FALSE;
               SCREEN.STATUS.CURSOR_ACTIVE := FALSE;
               INPUTS.OPERATION := OP_NON_ACCEPTING;
            end if;
            if LIST_DATA /= EMPTY_LIST then
               Pick_Lists.Handle_Input (LISTER => SCREEN.PICKER,
                  LIST_DATA => LIST_DATA,
                  INPUTS    => INPUTS);
            end if;
            if INPUTS.OPERATION = OP_NON_ACCEPTING then
               SCREEN.STATUS.CURSOR_ACTIVE := TRUE;
            end if;
            if (FORM_DATA /= EMPTY_FORM) and
               (INPUTS.OPERATION /= OP_DONE) then
               Process_Form (SCREEN => SCREEN,
                  LIST_DATA => LIST_DATA,
                  FORM_DATA => FORM_DATA,
                  INPUTS    => INPUTS);
            end if;

         when others => null;
      end case;
   when others => null;
   end case;
end Handle_Input;

-------------------- SPECIALISED SUBSCREEN METHODS --------------------

package body Errors is

-- PURPOSE : Displays an error message on the physical screen.

procedure Write (MESSAGE : in STRING;
   HELP_INFO : in INTEGER_TYPE := HP_SUB_ERRORS) is

begin
   SCREEN.STATUS.OBJECT_ACTIVE := OBJ_NONE;
   SCREEN.STATUS.CURSOR_ACTIVE := FALSE;
   SCREEN.STATUS.HELP_INFO     := HELP_INFO;
   SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
   SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
   Create (SCREEN => SCREEN,
      FRAME_COLOUR => (FORGROUND => RED, BACKGROUND => DIGHT_GRAY,
         BLINK => FALSE),
      TITLE_COLOUR => (FORGROUND => LIGHT_RED, BACKGROUND => LIGHT_GRAY,
         BLINK => TRUE),
      NAMES        => " ERROR ");
   Draw (SCREEN => SCREEN);
   loop
      Inputter.Get_Input (INPUTS => INPUTS);
      Handle_Input (SCREEN => SCREEN,
         FORM_DATA => EMPTY_FORM,
         LIST_DATA => EMPTY_LIST,
         INPUTS    => INPUTS);
      if INPUTS.OPERATION = OP_OK then
         exit;
      end if;
   end loop;
   Windows.Close;
end Write;

end Errors;

package body Prompts is

-- PURPOSE : Asks the user to choose/set an option on a list as well as
--           filling in a form.

procedure Read (FORM : in out FIELDS_TYPE;
   LIST      : in out LIST_ARRAY_TYPE;
   EXCLUSIVE : in BOOLEAN := TRUE;
   HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI) is

   WINDOW_LENGTH, WINDOW_WIDTH : POSITIVE;

begin
   SCREEN.STATUS.OBJECT_ACTIVE  := OBJ_MULTI;
   SCREEN.STATUS.NON_ACCEPTANCE := TRUE;
   SCREEN.STATUS.CURSOR_ACTIVE  := FALSE;
   SCREEN.STATUS.HELP_INFO      := HELP_INFO;
   SCREEN.STATUS.TEXT_LENGTH    := 1;
   SCREEN.STATUS.TEXT (1)       := ' ';
   Create (SCREEN => SCREEN,
      FORM_DATA  => FORM,
      LIST_DATA  => LIST,
      EXCLUSIVE  => EXCLUSIVE,
      NAMES      => " PROMPT ");
   Draw (SCREEN => SCREEN,
      LIST_DATA => LIST,
      FORM_DATA => FORM);
   loop
      Inputter.Get_Input (INPUTS => INPUTS);
      Screens.Handle_Input (SCREEN => SCREEN,
         LIST_DATA => LIST,
         FORM_DATA => FORM,
         INPUTS    => INPUTS);
      if INPUTS.OPERATION = OP_OK then
         if FORM /= EMPTY_FORM then
            INPUTS.KEY := KB_TAB;
            Process_Form (SCREEN => SCREEN,
               LIST_DATA => LIST,
               FORM_DATA => FORM,
               INPUTS    => INPUTS);
            if INPUTS.OPERATION = OP_FORM_PROCESSED then
               exit;
            end if;
         else
            exit;
         end if;
      end if;
   end loop;
   Hide_Cursor;
   Windows.Close;
end Read;

-- PURPOSE : Asks the user to  fill in the form.

procedure Read (FORM : in out FIELDS_TYPE;
   HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI) is

begin
   Read (FORM => FORM,
      LIST      => EMPTY_LIST,
      HELP_INFO => HELP_INFO);
end Read;

-- PURPOSE : Asks the user to choose/set an option on a list.

procedure Read (LIST : in out LIST_ARRAY_TYPE;
   EXCLUSIVE : in BOOLEAN := TRUE;
   HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_MULTI) is

begin
   Read (FORM => EMPTY_FORM,
      LIST      => LIST,
      EXCLUSIVE => EXCLUSIVE,
      HELP_INFO => HELP_INFO);
end Read;

-- PURPOSE : Adds  another option  to  the list  from which  the user must
--           choose.

procedure Add_To_List (FILE_NAME : in STRING) is

begin
   Pick_Lists.Create_List (LISTER => SCREEN.PICKER2,
      NAME => FILE_NAME);
end Add_To_List;

-- PURPOSE : Asks the user to choose a filename from a list.

procedure Read (FILE_NAME : out STRING;
   HELP_INFO : in INTEGER_TYPE := HP_SUB_PROMPTS_FILES) is

begin
   SCREEN.STATUS.OBJECT_ACTIVE  := OBJ_PICKER;
   SCREEN.STATUS.CURSOR_ACTIVE  := FALSE;
   SCREEN.STATUS.HELP_INFO      := HELP_INFO;
   SCREEN.STATUS.TEXT_LENGTH    := 1;
   SCREEN.STATUS.TEXT (1)       := ' ';
   Create (SCREEN => SCREEN,
      SUB_LENGTH => PICK_LISTS_DISPLAY_LENGTH + 6,
      NAMES      => " PROMPT ");
   Draw (SCREEN => SCREEN);
   loop
      Inputter.Get_Input (INPUTS => INPUTS);
      Handle_Input (SCREEN => SCREEN,
         FORM_DATA => EMPTY_FORM,
         LIST_DATA => EMPTY_LIST,
         INPUTS    => INPUTS);
      if INPUTS.OPERATION = OP_OK then
         Pick_Lists.Get_Name (LISTER => SCREEN.PICKER2,
            ITEM => FILE_NAME);
         exit;
      end if;
   end loop;
   Windows.Close;
exception
   when USER_ESCAPE =>
      Pick_Lists.Destroy_List (LISTER => SCREEN.PICKER2);
      raise USER_ESCAPE;
end Read;

-- PURPOSE : Asks the user for a boolean input (press either Yes or No).

procedure Read (MESSAGE : in STRING;
   HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_BOOLEAN;
   ITEM      : out BOOLEAN) is

begin
   SCREEN.STATUS.OBJECT_ACTIVE := OBJ_RB_BUTTONS;
   SCREEN.STATUS.CURSOR_ACTIVE := FALSE;
```

```
      SCREEN.STATUS.HELP_INFO     := HELP_INFO;
      SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
      SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
      Create (SCREEN => SCREEN,
         NAMES => " PROMPT ");
      Screens.Draw (SCREEN => SCREEN);
      loop
         Inputter.Get_Input (INPUTS => INPUTS);
         Screens.Handle_Input (SCREEN => SCREEN,
            FORM_DATA => EMPTY_FORM,
            LIST_DATA => EMPTY_LIST,
            INPUTS    => INPUTS);
         case INPUTS.OPERATION is
            when OP_OK | OP_TRUE =>
               ITEM := TRUE;
               exit;
            when OP_FALSE =>
               ITEM := FALSE;
               exit;
            when others => null;
         end case;
      end loop;
      Windows.Close;
   end Read;

   ---------------------------------------------------------------------
   -- PURPOSE : Displays a message informing the user  of the error  that has
   --           occured in the Input_Line object.
   ---------------------------------------------------------------------

   procedure Display_Input_Line_Error (MESSAGE : in STRING) is

   begin
      String_To_Window_Buffer (MESSAGE => MESSAGE,
         ROW    => 3,
         COLUMN => 2 ,
         COLOURS1 => (FOREGROUND => LIGHT_RED,
            BACKGROUND => LIGHT_GRAY, BLINK => FALSE),
         COLOURS2 => (FOREGROUND => LIGHT_RED,
            BACKGROUND => LIGHT_GRAY, BLINK => FALSE));
      Window_Buffer_To_Scre*'
      delay 1.0;
      Draw (SCREEN => SCREEN
         UPDATING => TRUE)
   end Display_Input_Line_Error;

   ---------------------------------------------------------------------
   -- PURPOSE : Asks the user for an integer number which it then gets.
   ---------------------------------------------------------------------

   procedure Read (MESSAGE : in STRING;
      HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_INTEGER;
      ITEM      : out INTEGER_TYPE) is

      TEMP_STRING : STRING (1..INPUT_LINES_TOTAL_LENGTH);
      LENGTHS     : NATURAL;

   begin
      SCREEN.STATUS.OBJECT_ACTIVE := OBJ_INTEGER_LINER;
      SCREEN.STATUS.CURSOR_ACTIVE := TRUE;
      SCREEN.STATUS.HELP_INFO     := HELP_INFO;
      SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
      SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
      Create (SCREEN => SCREEN,
         NAMES => " PROMPT ");
      Screens.Draw (SCREEN => SCREEN);
      loop
         Inputter.Get_Input (INPUTS => INPUTS);
         Screens.Handle_Input (SCREEN => SCREEN,
            FORM_DATA => EMPTY_FORM,
            LIST_DATA => EMPTY_LIST,
            INPUTS    => INPUTS);
         if INPUTS.OPERATION = OP_OK then
            begin
               Input_Lines.Get_Line (LINER => SCREEN.LINER,
                  LINE    => TEMP_STRING,
                  LENGTHS => LENGTHS);
               Texts.Convert_To_Integer (FROM => TEMP_STRING (1..LENGTHS),
                  ITEM => ITEM);
               exit;
            exception
               when Text_IO.DATA_ERROR | Text_IO.END_ERROR =>
                  TEMP_STRING (1..LENGTHS) := (1..LENGTHS => ' ');
                  Display_Input_Line_Error (MESSAGE =>
                     "The value is not a recognised integer.");
            end;
         end if;
      end loop;
      Hide_Cursor;
      Windows.Close;
   end Read;

   ---------------------------------------------------------------------
   -- PURPOSE : Asks the user for a real number which it then gets.
   ---------------------------------------------------------------------

   procedure Read (MESSAGE : in STRING;
      HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_REAL;
      ITEM      : out REAL_TYPE) is

      TEMP_STRING : STRING (1..INPUT_LINES_TOTAL_LENGTH);
      LENGTHS     : NATURAL;

   begin
      SCREEN.STATUS.OBJECT_ACTIVE := OBJ_REAL_LINER;
      SCREEN.STATUS.CURSOR_ACTIVE := TRUE;
      SCREEN.STATUS.HELP_INFO     := HELP_INFO;
      SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
      SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
      Create (SCREEN => SCREEN,
         NAMES => " PROMPT ");
      Screens.Draw (SCREEN => SCREEN);
      loop
         Inputter.Get_Input (INPUTS => INPUTS);
         Screens.Handle_Input (SCREEN => SCREEN,
            FORM_DATA => EMPTY_FORM,
            LIST_DATA => EMPTY_LIST,
            INPUTS    => INPUTS);
         if INPUTS.OPERATION = OP_OK then
            begin
               Input_Lines.Get_Line (LINER => SCREEN.LINER,
                  LINE    => TEMP_STRING,
                  LENGTHS => LENGTHS);
               Texts.Convert_To_Real (FROM => TEMP_STRING (1..LENGTHS),
                  ITEM => ITEM);
               exit;
            exception
               when Text_IO.DATA_ERROR | Text_IO.END_ERROR =>
                  TEMP_STRING (1..LENGTHS) := (1..LENGTHS => ' ');
                  Display_Input_Line_Error (MESSAGE =>
                     "The value is not a recognised real.");
            end;
         end if;
      end loop;
      Hide_Cursor;
      Windows.Close;
   end Read;

   ---------------------------------------------------------------------
   -- PURPOSE : Asks the user for a string which it then gets.
   ---------------------------------------------------------------------

   procedure Read (MESSAGE : in STRING;
      HELP_INFO : in  INTEGER_TYPE := HP_SUB_PROMPTS_STRING;
      ITEM      : out STRING;
      ITEM_LENGTH : out POSITIVE) is

      TEMP_STRING : STRING (1..INPUT_LINES_TOTAL_LENGTH);
      LENGTHS     : NATURAL;

   begin
      SCREEN.STATUS.OBJECT_ACTIVE := OBJ_STRING_LINER;
      SCREEN.STATUS.CURSOR_ACTIVE := TRUE;
      SCREEN.STATUS.HELP_INFO     := HELP_INFO;
      SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
      SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
      Create (SCREEN => SCREEN,
         NAMES => " PROMPT ");
      Screens.Draw (SCREEN => SCREEN);
      loop
         Inputter.Get_Input (INPUTS => INPUTS);
         Screens.Handle_Input (SCREEN => SCREEN,
            FORM_DATA => EMPTY_FORM,
            LIST_DATA => EMPTY_LIST,
            INPUTS    => INPUTS);
         if INPUTS.OPERATION = OP_OK then
            Input_Lines.Get_Line (LINER => SCREEN.LINER,
               LINE    => TEMP_STRING,
               LENGTHS => LENGTHS);
            if LENGTHS >= 1 then
               ITEM (1..LENGTHS) := TEMP_STRING (1..LENGTHS);
               ITEM_LENGTH := LENGTHS;
               exit;
            else
               Display_Input_Line_Error (MESSAGE =>
                  "At least one character must be entered.");
            end if;
         end if;
      end loop;
      Hide_Cursor;
      Windows.Close;
   end Read;

   ---------------------------------------------------------------------

end Prompts;

---------------------------------------------------------------------
---------------------------------------------------------------------

package body Status is

   ---------------------------------------------------------------------
   -- PURPOSE : Displays a status message about the status of some process in
   --           the program.
   ---------------------------------------------------------------------

   procedure Write (MESSAGE : in STRING;
      HELP_INFO : in INTEGER_TYPE := HP_SUB_STATUS) is

   begin
      SCREEN.STATUS.OBJECT_ACTIVE := OBJ_NONE;
      SCREEN.STATUS.CURSOR_ACTIVE := FALSE;
      SCREEN.STATUS.HELP_INFO     := HELP_INFO;
      SCREEN.STATUS.TEXT_LENGTH   := MESSAGE'length;
      SCREEN.STATUS.TEXT (1..MESSAGE'length) := MESSAGE;
      Create (SCREEN => SCREEN,
         NAMES => " STATUS ");
      Screens.Draw (SCREEN => SCREEN);
      loop
         Inputter.Get_Input (INPUTS => INPUTS);
         Screens.Handle_Input (SCREEN => SCREEN,
            FORM_DATA => EMPTY_FORM,
            LIST_DATA => EMPTY_LIST,
            INPUTS    => INPUTS);
         if INPUTS.OPERATION = OP_OK then
            exit;
         end if;
      end loop;
      Windows.Close;
   end Write;

   ---------------------------------------------------------------------
   -- PURPOSE : Displays a wait message  which is used when the program needs
   --           to wait until the user has finished reading results, etc.
   ---------------------------------------------------------------------

   procedure Wait is

      MESSAGE : constant STRING := "Press the OK button to continue.";

   begin
      Write (MESSAGE => MESSAGE);
   end Wait;

   ---------------------------------------------------------------------

end Status;

---------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

package body Main is

   DISK_FILE                          : Text_IO.FILE_TYPE;
   OUTPUT_TO_DISK                     : BOOLEAN      := FALSE;
   COLUMNS                            : COLUMN_TYPE := 1;
   ROWS                               : ROW_TYPE    := 3;

-------------------------------------------------------------------------------
-- PURPOSE : Calculates the new position of the cursor once some text has
--           been written to the main screen.
-------------------------------------------------------------------------------

procedure New_Cursor_Position (LENGTHS : in NATURAL) is

   ADD_POSITION,          ADD_ROWS           : INTEGER_TYPE;

begin
   ADD_POSITION := COLUMNS + LENGTHS;
   if ADD_POSITION > COLUMN_TYPE'last then
      ADD_ROWS := ADD_POSITION / COLUMN_TYPE'last;
      COLUMNS := ADD_POSITION - (ADD_ROWS * COLUMN_TYPE'last);
      ROWS := ROWS + ADD_ROWS;
   else
      COLUMNS := ADD_POSITION;
   end if;
exception
   when CONSTRAINT_ERROR =>
      ROWS := ROW_TYPE'last;
      COLUMNS := COLUMN_TYPE'last;
end New_Cursor_Position;

-------------------------------------------------------------------------------
-- PURPOSE : Clears the main screen and sets the cursor to the start of
--           the second line.
-------------------------------------------------------------------------------

procedure Clear is

begin
   Clear_Main_Buffer (MAIN_COLOURS => (FOREGROUND => LIGHT_GRAY,
      BACKGROUND => BLUE, BLINK => FALSE));
   Main_Buffer_To_Screen;
   ROWS    := 2;
   COLUMNS := 1;
end Clear;

-------------------------------------------------------------------------------
-- PURPOSE : Sets the cursor of the main screen.
-------------------------------------------------------------------------------

procedure Cursor_Set (ROW : in ROW_TYPE; COLUMN : in COLUMN_TYPE) is

begin
   ROWS := ROW;
   COLUMNS := COLUMN;
end Cursor_Set;

-------------------------------------------------------------------------------
-- PURPOSE : Moves the cursor to the start of the next line.  If there are
--           no more lines, the main screen is scrolled up one line.
-------------------------------------------------------------------------------

procedure Cursor_Next_Line is

begin
   if ROWS >= ROW_TYPE'last then
      ROWS := ROW_TYPE'last;
   else
      ROWS := ROWS + 1;
   end if;
   COLUMNS := 1;
   if OUTPUT_TO_DISK then
      Text_IO.New_Line (FILE => DISK_FILE);
   end if;
end Cursor_Next_Line;

-------------------------------------------------------------------------------
-- PURPOSE : Open device files for the screen output if such device output
--           is specified.
-------------------------------------------------------------------------------

procedure Open_Screen_Devices is

   CIRCUIT_LINE                       : STRING (1..64);
   LAST_CHARACTER                     : POSITIVE;
   OVERWRITE                          : BOOLEAN;

begin
   if OUTPUT_TO_DISK then
      Text_IO.Close (FILE => DISK_FILE);
      Status.Write (MESSAGE => "Screen output disk file closed.");
      OUTPUT_TO_DISK := FALSE;
   else
      begin
         Prompts.Read (
            MESSAGE     => "Enter the name of the screen output file.",
            ITEM        => CIRCUIT_LINE,
            ITEM_LENGTH => LAST_CHARACTER);
         Text_IO.Open (FILE => DISK_FILE,
            MODE => Text_IO.IN_FILE,
            NAME => CIRCUIT_LINE (1..LAST_CHARACTER));
         Text_IO.Close (FILE => DISK_FILE);
         Prompts.Read (MESSAGE => "The file already exists. Do you"
            & " want to overwrite it ?",
            ITEM => OVERWRITE);
         if OVERWRITE then
            Text_IO.Open (FILE => DISK_FILE,
               MODE => Text_IO.OUT_FILE,
               NAME => CIRCUIT_LINE (1..LAST_CHARACTER));
            OUTPUT_TO_DISK := TRUE;
         end if;
      exception
         when Text_IO.NAME_ERROR =>
            Text_IO.Create (FILE => DISK_FILE,
               NAME => CIRCUIT_LINE (1..LAST_CHARACTER));
            OUTPUT_TO_DISK := TRUE;
      end;
   end if;

exception
   when Text_IO.DATA_ERROR | Text_IO.DEVICE_ERROR | Text_IO.USE_ERROR =>
      Errors.Write (MESSAGE => "Device Error");
   when Text_IO.NAME_ERROR =>
      Errors.Write (MESSAGE => "Name Error");
end Open_Screen_Devices;

-------------------------------------------------------------------------------
-- PURPOSE : Draws a solid bar on which the title of the process is
--           centred.
-------------------------------------------------------------------------------

procedure Process_Title (TITLE : in STRING) is

begin
   Clear_Main_Buffer (MAIN_COLOURS => (FOREGROUND => LIGHT_GRAY,
      BACKGROUND => BLUE, BLINK => FALSE));
   Main_Buffer_Line (LINE_COLOURS => (FOREGROUND => BLACK,
      BACKGROUND => LIGHT_GRAY, BLINK => FALSE));
   ROWS    := 1;
   COLUMNS := (80 - TITLE'length) / 2;
   Write (ITEM => TITLE,
      FORM => BLADING);
   Cursor_Next_Line;
end Process_Title;

-------------------------------------------------------------------------------
-- PURPOSE : Writes an integer number to the main screen.
-------------------------------------------------------------------------------

procedure Write (ITEM : in INTEGER_TYPE;
   ITEM_WIDTH : in INTEGER_TYPE := INTEGER_TYPE'width) is

   TEMP_LINE : Texts.TEXT_TYPE;

begin
   Texts.Combine (ITEM => ITEM,
      ITEM_WIDTH => ITEM_WIDTH,
      TEXT_LINE  => TEMP_LINE);
   String_To_Main_Buffer (
      MESSAGE => TEMP_LINE.TEXT (1..TEMP_LINE.TEXT_INDEX),
      ROW     => ROWS,
      COLUMN  => COLUMNS,
      COLOURS1 => (FOREGROUND => YELLOW, BACKGROUND => BLUE,
         BLINK => FALSE),
      COLOURS2 => (FOREGROUND => YELLOW, BACKGROUND => BLUE,
         BLINK => FALSE));
   New_Cursor_Position (LENGTHS => TEMP_LINE.TEXT_INDEX);
   Main_Buffer_To_Screen;
   if OUTPUT_TO_DISK then
      Text_IO.Put (FILE => DISK_FILE,
         ITEM => TEMP_LINE.TEXT (1..TEMP_LINE.TEXT_INDEX));
   end if;
end Write;

-------------------------------------------------------------------------------
-- PURPOSE : Writes a real number on the main screen.  Engineering units
--           can be calculated and the number adjusted accordingly.
-------------------------------------------------------------------------------

procedure Write (ITEM : in REAL_TYPE;
   WHOLE        : in INTEGER_TYPE := 3;
   FRACTION     : in INTEGER_TYPE := REAL_TYPE'digits - 1;
   EXPONENT     : in INTEGER_TYPE := 3;
   ENG_NOTATION : in BOOLEAN      := FALSE) is

   TEMP_LINE    : Texts.TEXT_TYPE;

begin
   Texts.Combine (ITEM => ITEM,
      WHOLE      => WHOLE,
      FRACTION   => FRACTION,
      EXPONENT   => EXPONENT,
      ENG_NOTATION => ENG_NOTATION,
      TEXT_LINE  => TEMP_LINE);
   String_To_Main_Buffer (
      MESSAGE  => TEMP_LINE.TEXT (1..TEMP_LINE.TEXT_INDEX),
      ROW      => ROWS,
      COLUMN   => COLUMNS,
      COLOURS1 => (FOREGROUND => LIGHT_GREEN, BACKGROUND => BLUE,
         BLINK => FALSE),
      COLOURS2 => (FOREGROUND => YELLOW, BACKGROUND => BLUE,
         BLINK => FALSE));
   New_Cursor_Position (LENGTHS => TEMP_LINE.TEXT_INDEX);
   Main_Buffer_To_Screen;
   if OUTPUT_TO_DISK then
      Text_IO.Put (FILE => DISK_FILE,
         ITEM => TEMP_LINE.TEXT (1..TEMP_LINE.TEXT_INDEX));
   end if;
end Write;

-------------------------------------------------------------------------------
-- PURPOSE : Writes text on the main screen in a particular attribute.
-------------------------------------------------------------------------------

procedure Write (ITEM : in STRING;
   FORM : in ATTRIBUTE_TYPE := NORMAL) is

   COLOURS : SCREEN_COLOUR_TYPE;

begin
   case FORM is
      when BOLD =>
         COLOURS := (FOREGROUND => WHITE, BACKGROUND => BLUE,
            BLINK => FALSE);
      when HEADING =>
         COLOURS := (FOREGROUND => PURPLE, BACKGROUND => LIGHT_GRAY,
            BLINK => FALSE);
      when NORMAL =>
         COLOURS := (FOREGROUND => LIGHT_CYAN, BACKGROUND => BLUE,
            BLINK => FALSE);
      when REVERSED =>
         COLOURS := (FOREGROUND => BLUE, BACKGROUND => LIGHT_GRAY,
            BLINK => FALSE);
      when UNDERLINED =>
         COLOURS := (FOREGROUND => LIGHT_GREEN, BACKGROUND => CYAN,
            BLINK => FALSE);
   end case;
   String_To_Main_Buffer (MESSAGE => ITEM,
      ROW     => ROWS,
```

```
      COLUMN    => COLUMNS,
      COLOUR1 => COLOURS,
      COLOUR2 => COLOURS);
   Now_Cursor_Position (LENGTHS => ITEM'length);
   Main_Buffer_To_Screen;
   if OUTPUT_TO_DISK then
      Text_IO.Put (FILE => DISK_FILE,
         ITEM => ITEM);
   end if;
end Write;
```

```
end Main;
```

Tutorial

```
package Tutorial is
```

```
-- Author  : Warren Buckle
-- Date    : June 1992
-- Purpose : This contains the tutorial methods which find and display the
--           tutorial screens on the Main screen.  A generic package is
--           instantiated for the reading of the tutorials image files.
--           The data structure holds the image for the tutorial screen as
--           well as the index which is used to locate each image.  The
--           image consists of 3404 locations - 23 rows * 74 columns of 1
--           character and 1 attribute.
```

```
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
```

```
   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);
```

```
   procedure Initialize_Tutorial_Path;

   procedure Show;
```

```
end Tutorial;
```

```
with  Ada_Extensions,
      Dos_Environment,
      Screens,
      Sequential_IO;

use   Ada_Extensions,
      Screens;

package body Tutorial is
```

```
-- Suppression of these checks are done because the assignment is known to
-- be correct and it will not cause an error.  Thus these checks will only
-- add unnecessary checking code.  Checks can be introduced again by
-- omitting the SUPPRESS pragma for that check.
```

```
   pragma SUPPRESS (ACCESS_CHECK);
   pragma SUPPRESS (DISCRIMINANT_CHECK);
   pragma SUPPRESS (ELABORATION_CHECK);
   pragma SUPPRESS (INDEX_CHECK);
   pragma SUPPRESS (LENGTH_CHECK);
   pragma SUPPRESS (RANGE_CHECK);
```

```
   HP_GET_TUTORIAL  : constant := 200;
   HP_GET_TUTORIAL2 : constant := 201;
   HP_GET_TUTORIAL3 : constant := 202;
   HP_GET_TUTORIAL4 : constant := 203;
   HP_GET_TUTORIAL5 : constant := 204;
   HP_FIND_TUTORIAL : constant := 207;
```

```
   MAIN_TUTORIAL_LIST : LIST_ARRAY_TYPE (1..4) := (
   (TEXT_LENGTH => 19, TEXT => "Elector Information", STATE => TRUE),
   (TEXT_LENGTH => 17, TEXT => "Circuit Modelling",   STATE => FALSE),
   (TEXT_LENGTH => 14, TEXT => "Electrical Eng",      STATE => FALSE),
   (TEXT_LENGTH => 19, TEXT => "Equivalent Circuits", STATE => FALSE));

   TUTORIAL_1_LIST : LIST_ARRAY_TYPE (1..8) := (
   (TEXT_LENGTH => 14, TEXT => "What is it for", STATE => TRUE),
   (TEXT_LENGTH => 14, TEXT => "User Interface", STATE => FALSE),
   (TEXT_LENGTH => 10, TEXT => "Subscreens",     STATE => FALSE),
   (TEXT_LENGTH => 7,  TEXT => "Windows",        STATE => FALSE),
   (TEXT_LENGTH => 7,  TEXT => "Buttons",        STATE => FALSE),
   (TEXT_LENGTH => 10, TEXT => "Input Line",     STATE => FALSE),
   (TEXT_LENGTH => 9,  TEXT => "Pick List",      STATE => FALSE),
   (TEXT_LENGTH => 4,  TEXT => "Help",           STATE => FALSE));

   TUTORIAL_2_LIST : LIST_ARRAY_TYPE (1..5) := (
   (TEXT_LENGTH => 7,  TEXT => "Rules 1",  STATE => TRUE),
   (TEXT_LENGTH => 7,  TEXT => "Rules 2",  STATE => FALSE),
   (TEXT_LENGTH => 7,  TEXT => "Rules 3",  STATE => FALSE),
   (TEXT_LENGTH => 8,  TEXT => "Design 1", STATE => FALSE),
   (TEXT_LENGTH => 8,  TEXT => "Design 2", STATE => FALSE));

   TUTORIAL_3_LIST : LIST_ARRAY_TYPE (1..13) := (
   (TEXT_LENGTH => 11, TEXT => "Definitions",          STATE => TRUE),
   (TEXT_LENGTH => 18, TEXT => "Passive Components",   STATE => FALSE),
   (TEXT_LENGTH => 13, TEXT => "Ideal Sources",        STATE => FALSE),
   (TEXT_LENGTH => 12, TEXT => "Real Sources",         STATE => FALSE),
   (TEXT_LENGTH => 16, TEXT => "Electrical Power",     STATE => FALSE),
   (TEXT_LENGTH => 15, TEXT => "Kirchoff's Laws",      STATE => FALSE),
   (TEXT_LENGTH => 8,  TEXT => "Thevenin",             STATE => FALSE),
   (TEXT_LENGTH => 16, TEXT => "Frequency Domain",     STATE => FALSE),
   (TEXT_LENGTH => 9,  TEXT => "Impedance",            STATE => FALSE),
   (TEXT_LENGTH => 8,  TEXT => "AC Power",             STATE => FALSE),
   (TEXT_LENGTH => 12, TEXT => "Power Factor",         STATE => FALSE),
```

```
   (TEXT_LENGTH => 11, TEXT => "Three Phase",          STATE => FALSE),
   (TEXT_LENGTH =>     TEXT => "Resonance",            STATE => FALSE));

   TUTORIAL_4_LIST : LAST_ARRAY_TYPE (1..9) := (
   (TEXT_LENGTH => 10, TEXT => "Transistor",           STATE => TRUE),
   (TEXT_LENGTH => 12, TEXT => "Op Amplifier",         STATE => FALSE),
   (TEXT_LENGTH => 11, TEXT => "Transformer",          STATE => FALSE),
   (TEXT_LENGTH => 15, TEXT => "Induction Motor",      STATE => FALSE),
   (TEXT_LENGTH => 17, TEXT => "Synchronous Motor",    STATE => FALSE),
   (TEXT_LENGTH => 5,  TEXT => "Fluid",                STATE => FALSE),
   (TEXT_LENGTH => 12, TEXT => "Mechanical 1",         STATE => FALSE),
   (TEXT_LENGTH => 12, TEXT => "Mechanical 2",         STATE => FALSE),
   (TEXT_LENGTH => 7,  TEXT => "Thermal",              STATE => FALSE));

   LEVEL1         : BOOLEAN;
   TUTORIAL_PATH  : Dos_Environment.PATH_STRING_TYPE;
   PATH_LENGTH    : INTEGER_TYPE;

-------------------------------------------------------------------------------
-- PURPOSE : Gets the initial default path.
-------------------------------------------------------------------------------

procedure Initialize_Tutorial_Path is

   COUNTER : INTEGER_TYPE;

begin
   Dos_Environment.Get_Drive_Directory (PATH_STRING => TUTORIAL_PATH);
   for COUNTER in 1..TUTORIAL_PATH'length loop
      if TUTORIAL_PATH (COUNTER) = CHARACTER'Val (0) then
         PATH_LENGTH := COUNTER - 1;
         exit;
      end if;
   end loop;
   if TUTORIAL_PATH (PATH_LENGTH) = '\' then
      PATH_LENGTH := PATH_LENGTH - 1;
   end if;
end Initialize_Tutorial_Path;

-------------------------------------------------------------------------------
-- PURPOSE : Gets the tutorial option from the user.
-------------------------------------------------------------------------------

procedure Get_Tutorial_Option (OPTION : out INTEGER_TYPE) is

   COUNTER1,              COUNTER2,
   TUTORIAL_NUMBER                        : INTEGER_TYPE;
   GET_OPTION                             : BOOLEAN := TRUE;

begin
   while GET_OPTION loop
      if LEVEL1 then
         Prompts.Read (LIST => MAIN_TUTORIAL_LIST,
            HELP_INFO => HP_GET_TUTORIAL);
      end if;
      for COUNTER1 in 1..4 loop
         if MAIN_TUTORIAL_LIST (COUNTER1).STATE then
            begin
               TUTORIAL_NUMBER := (COUNTER1 - 1) * 20;
               LEVEL1 := FALSE;
               case COUNTER1 is
                  when 1 =>
                     Prompts.Read (LIST => TUTORIAL_1_LIST,
                        HELP_INFO => HP_GET_TUTORIAL2);
                     for COUNTER2 in 1..8 loop
                        if TUTORIAL_1_LIST (COUNTER2).STATE then
                           TUTORIAL_NUMBER := TUTORIAL_NUMBER + COUNTER2;
                        end if;
                     end loop;
                  when 2 =>
                     Prompts.Read (LIST => TUTORIAL_2_LIST,
                        HELP_INFO => HP_GET_TUTORIAL3);
                     for COUNTER2 in 1..5 loop
                        if TUTORIAL_2_LIST (COUNTER2).STATE then
                           TUTORIAL_NUMBER := TUTORIAL_NUMBER + COUNTER2;
                        end if;
                     end loop;
                  when 3 =>
                     Prompts.Read (LIST => TUTORIAL_3_LIST,
                        HELP_INFO => HP_GET_TUTORIAL4);
                     for COUNTER2 in 1..13 loop
                        if TUTORIAL_3_LIST (COUNTER2).STATE then
                           TUTORIAL_NUMBER := TUTORIAL_NUMBER + COUNTER2;
                        end if;
                     end loop;
                  when 4 =>
                     Prompts.Read (LIST => TUTORIAL_4_LIST,
                        HELP_INFO => HP_GET_TUTORIAL5);
                     for COUNTER2 in 1..9 loop
                        if TUTORIAL_4_LIST (COUNTER2).STATE then
                           TUTORIAL_NUMBER := TUTORIAL_NUMBER + COUNTER2;
                        end if;
                     end loop;
               end case;
               GET_OPTION := FALSE;
               OPTION := TUTORIAL_NUMBER + 1000;
            exception
               when USER_ESCAPE =>
                  LEVEL1 := TRUE;
                  exit;
            end;
         end if;
      end loop;
   end loop;
end Get_Tutorial_Option;

-------------------------------------------------------------------------------
-- PURPOSE : Searches a  disk file  for the tutorial item.  If the item or
--           the disk file containing the tutorial images is not found, an
--           appropriate error message appears.
-------------------------------------------------------------------------------

procedure Find_Tutorial (OPTION : INTEGER_TYPE) is

      subtype ASCII_CHARACTERS is INTEGER_TYPE range 0..255;

   -- Value 3404 = 23 rows * 74 columns of 1 character and 1 attribute
   type ASCII_ARRAY_TYPE is array (1..3404) of ASCII_CHARACTERS;

      pragma pack (ASCII_ARRAY_TYPE);

   type TUTORIAL_ITEM_TYPE is
      record
         INDEX : INTEGER_TYPE;
         IMAGE : ASCII_ARRAY_TYPE;
      end record;

   package Tutorial_IO is new Sequential_IO (TUTORIAL_ITEM_TYPE);

   procedure Tutorial_To_Screen (TUTORIAL_IMAGE : in ASCII_ARRAY_TYPE);

      pragma interface (assembly, Tutorial_To_Screen);


   TUTORIAL_ITEM  : TUTORIAL_ITEM_TYPE;
   TUTORIAL_FILE  : Tutorial_IO.FILE_TYPE;
   FOUND_TUTORIAL : BOOLEAN := TRUE;

begin
   begin
      Tutorial_IO.Open (FILE => TUTORIAL_FILE,
         MODE => Tutorial_IO.IN_FILE,
         NAME => TUTORIAL_PATH (1..PATH_LENGTH) & "\ELECTOR.TUT");
   exception
      when Tutorial_IO.NAME_ERROR =>
         FOUND_TUTORIAL := FALSE;
         Errors.Write (MESSAGE => "The tutorial file cannot be found.",
            HELP_INFO => HP_FIND_TUTORIAL);
   end;
   if FOUND_TUTORIAL then
      FOUND_TUTORIAL := FALSE;
      while not Tutorial_IO.End_Of_File (FILE => TUTORIAL_FILE) loop
         Tutorial_IO.Read (FILE => TUTORIAL_FILE,
            ITEM => TUTORIAL_ITEM);
         if OPTION = TUTORIAL_ITEM.INDEX then
            FOUND_TUTORIAL := TRUE;
            exit;
         end if;
      end loop;
      if FOUND_TUTORIAL then
         Tutorial_To_Screen (TUTORIAL_IMAGE => TUTORIAL_ITEM.IMAGE);
         Continue_Processing;
      else
         Errors.Write (MESSAGE => "There is no Tutorial available for" &
            " this item.");
      end if;
      Tutorial_IO.Close (FILE => TUTORIAL_FILE);
   end if;
end Find_Tutorial;

-------------------------------------------------------------------------------
-- PURPOSE : Displays the tutorial image on the main screen.
-------------------------------------------------------------------------------

procedure Show is

   OPTION : INTEGER_TYPE;

begin
   LEVEL1 := TRUE;
   begin
      loop
         Main.Process_Title (TITLE => "TUTORIAL");
         Get_Tutorial_Option (OPTION => OPTION);
         Find_Tutorial (OPTION => OPTION);
      end loop;
   exception
      when USER_ESCAPE => null;
   end;
   Main.Clear;
end Show;

-------------------------------------------------------------------------------

end Tutorial;
```

ELECTOR ASSEMBLER SOURCE CODE

APPENDIX C

```
                TITLE   Dos and Language Extensions for Ada Elector

;------------------------------------------------------------------------
;-- Author  : Warren Buckle
;-- Date    : June 1992
;-- Purpose : This provides access to the machine resources in a
;--           controlled manner.
;------------------------------------------------------------------------

;------------------------------------------------------------------------

COLUMN_MAXIMUM          equ     80              ; Number of text columns
ROW_MAXIMUM             equ     25              ; Number of text rows
SCREEN_MEMORY           equ     0B800H          ; Start screen base memory
SPACE_CHAR              equ     32              ; ASCII space character
TEXT_MODE3              equ     3               ; Screen mode for text
CGA_MODE6               equ     6               ; Screen mode for graphics
EGAVGA_MODE16           equ     16              ; Screen mode for graphics
MAIN_CHARACTERS         equ     25*80           ; No of main characters
MAIN_END                equ     2*25*80         ; End of main buffer
TITLE_LINE_CHARACTERS   equ     80              ; No of line characters

;------------------------------------------------------------------------

_DATA       segment     public  'DATA'

; Variable for the start of screen memory
SCREEN_BASE_MEMORY      dw      SCREEN_MEMORY

; Variable for mouse status
MOUSE_STATUS            dw      0       ; Bit 0 = Mouse present
                                        ; Bit 1 = Mouse visible
                                        ; Bit 3 = Mouse not visible due to
                                        ;   screen modifying procedure

; Variables for drawing line
DIAGONAL_X_INCREMENT    dw      ?       ; Graphical X diagonal line incr
DIAGONAL_Y_INCREMENT    dw      ?       ; Graphical Y diagonal line incr
DIAGONAL_COUNT          dw      ?       ; Graphical diagonal line count
STRAIGHT_X_INCREMENT    dw      ?       ; Graphical X straight line incr
STRAIGHT_Y_INCREMENT    dw      ?       ; Graphical Y straight line incr
STRAIGHT_COUNT          dw      ?       ; Graphical straight line count

; Variables for line on text screen
LINE_START              dw      0       ; Start of line on screen
LINE_LENGTH             db      0       ; Length of line
ATTRIBUTE               db      0       ; Attributes when writing strings

; Variables for pop-up screen
BUFFER_BOTTOM_RIGHT     dw      0       ; End of text display of pop-up
SCREEN_LEFT_COLUMN      db      0       ; Left column of pop-up screen
SCREEN_TOP_ROW          db      0       ; Top row of pop-up screen
SIZE_COLUMNS            db      0       ; Column width of pop-up screen
SIZE_ROWS               db      0       ; Row height of pop-up screen

; Variables for the graphic screen
GRAPHIC_SCREEN          db      EGAVGA_MODE16
SCREEN_MODE             db      TEXT_MODE3
X_MINIMUM               dw      1               ; Minimum X length
Y_MINIMUM               dw      1               ; Minimum Y length
X_MAXIMUM               dw      640             ; Maximum X length
Y_MAXIMUM               dw      350             ; Maximum Y length
PLOT_COLOUR             db      0

; Varaibles for the disk functions
PATH1                   db      "C:*.MTL",0
PATH2                   db      "C:*.MNT",0

; This buffer holds the overwritten part of the screen. However, the main
; title is also stored in this buffer. This does not cause a problem as
; the title is displayed before any subscreen or menu is drawn, so it can
; be displayed before being written over. This method saves quite a bit of
; memory.

; Buffer for the main screen
MAIN_BUFFER     dw 80*25        dup (?)

; Buffer for the pop-up window
WINDOW_BUFFER   db 2*80*25-(56+56*7+19+31+37+23+12+19+25+16*2) dup(?)

ERROR_MESSAGE   db "ELECTOR only works with CGA and higher screen
adapters.$"

TITLE1  db '[illegible]'
TITLE2  db '[illegible]'
TITLE3  db '[illegible]'
TITLE4  db '[illegible]'
TITLE5  db '[illegible]'
TITLE6  db '[illegible]'
TITLE7  db '[illegible]'

TITLE11 db 'Warren Dean Buckle1'
TITLE12 db 'Software Engineering Application Laboratory (SEAL)1'
TITLE13 db 'Department of Electr. Engineering1'
TITLE14 db 'Faculty of Engineering1'
TITLE15 db 'University of the Witwatersrand1'
TITLE16 db '22 September 19921'
TITLE17 db 'Elector Ada Version 2.011'

DISPLAY1        label   word
                dw      offset  TITLE1
                dw      offset  TITLE2
                dw      offset  TITLE3
                dw      offset  TITLE4
                dw      offset  TITLE5
                dw      offset  TITLE6
                dw      offset  TITLE7

DISPLAY2        label   word
                dw      offset  TITLE17
                dw      offset  TITLE16
                dw      offset  TITLE15
                dw      offset  TITLE14
```

```
                dw      offset  TITLE13
                dw      offset  TITLE12
                dw      offset  TITLE11

LOGO_COUNT      dw      0
WAIT_COUNTER    dw      0

; Buffer for the line on the screen
LINE_BUFFER             db      80 dup (?)

_DATA       ends

;------------------------------------------------------------------------

DOS_CODE        segment public  'CODE'
                assume  cs:DOS_CODE, ds:_DATA, es:NOTHING, ss:NOTHING

;------------------------------------------------------------------------
;------------------------ Cursor Functions ------------------------------
;------------------------------------------------------------------------

;------------------------------------------------------------------------
;-- PURPOSE  : Sets the cursor shape. Gets the new cursor shape from
;--            parameters on the stack and then calls BIOS. The cursor
;--            lines are numbered starting at with 0 at the top of the
;--            character block. If the starting line is below the ending
;--            line, the cursor will wrap around to the top of the
;--            character block. Do not use numbers over 15. There are 8
;--            lines for CGA and 14 for monochrome and EGA. The cursor is
;--            displayed, or not displayed, depending on the value of the
;--            parametr on the stack.
;-- BIOS FUNCTION (Set cursor shape)
;--    INTERRUPT : 10h
;--    FUNCTION : 1
;--    CALLED WITH  : AH = 1
;--                   CH = starting cursor line  (bits 3 - 0)
;--                   CL = ending cursor line    (bits 3 - 0)
;--                   CH = display cursor        (bit  5 = 0)
;--                   CH = hide cursor           (bit  5 = 1)
;--    RETURNS WITH : AX changed
;------------------------------------------------------------------------

                public  Change_Cursor_Shape
Change_Cursor_Shape     proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get new cursor shape and if displayed from parameters
                mov     ch, 6 [bp]      ; Get start line of cursor
                mov     cl, 8 [bp]      ; Get cursor end line from stack
                and     cx, 0000111100001111b; Mask out invalid bits
                mov     dx, 10 [bp]     ; Get value for cursor visibility
                cmp     dx, 1           ; Is it display the cursor ?
                je      display_cursor  ; If so, skip hide instructions
                or      cx, 0010000000000000b; Set bit 5 to hide cursor

; Set cursor shape and visibility
display_cursor: mov     ah, 1           ; Set ah - setting of cursor shape
                int     10h             ; Execute BIOS

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     6               ; Return and deallocate 3 near
                                        ;   parameters
Change_Cursor_Shape     endp

;------------------------------------------------------------------------
;-- PURPOSE  : Hides the text cursor. No parameters are passed by the ADA
;--            subprogram. The shape of the cursor is preserved.
;-- BIOS FUNCTION (Read cursor position)
;--    INTERRUPT : 10h
;--    FUNCTION : 3
;--    CALLED WITH  : AH = 3           BH = page
;--    RETURNS WITH : AX changed       CH = starting cursor line
;--                   DH = row         CL = ending cursor line
;--                   DL = column
;-- BIOS FUNCTION (Set cursor shape)
;--    INTERRUPT : 10h
;--    FUNCTION : 1
;--    CALLED WITH  : AH = 1
;--                   CH = starting cursor line  (bits 3 - 0)
;--                   CL = ending cursor line    (bits 3 - 0)
;--                   CH = display cursor        (bit  5 = 0)
;--                   CH = hide cursor           (bit  5 = 1)
;--    RETURNS WITH : AX changed
;------------------------------------------------------------------------

                public  Hide_Cursor
Hide_Cursor     proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Read cursor position and shape
                xor     bh, bh                  ; Set page number
                mov     ah, 3                   ; Set ah - reading cursor values
                int     10h                     ; Execute BIOS

; Mask out the necessary bits and call BIOS to display cursor
                and     cx, 0000111100001111b; Mask out invalid bits
                or      cx, 0010000000000000b; Set bit 5 to hide cursor
                mov     ah, 1                   ; Set ah for cursor shape
                int     10h                     ; Execute BIOS function

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret                             ; Return
Hide_Cursor     endp

;------------------------------------------------------------------------
;-- PURPOSE  : Sets the cursor position. Gets the new cursor position from
;--            parameters on the stack, decrements both the row and column
;--            to convert from the ADA screen coordintes to those used by
;--            BIOS, and then calls BIOS.
;--            Column screen coordinates : ADA (1 - 80)   BIOS (0 - 79)
;--            Row    screen coordinates : ADA (1 - 25)   BIOS (0 - 24)
;-- BIOS FUNCTION (Set cursor position)
;--    INTERRUPT : 10h
;--    FUNCTION : 2
;--    CALLED WITH  : AH = 2          DH = row         DL = column
;--                   BH = page
```

```
;--    RETURNS WITH : AX changed
;---------------------------------------------------------------------------

                public  Set_Cursor
Set_Cursor      proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get new cursor position from parameters
                mov     dx, 6 [bp]              ; Row    (integer format)
                dec     dx                      ; Adjust row for BIOS
                mov     dh, dl                  ; Get row into byte form
                mov     ax, 8 [bp]              ; Column  (integer format)
                dec     ax                      ; Adjust column for BIOS
                mov     dl, al                  ; Get column into byte form

; Set cursor position
                xor     bh, bh                  ; Set page number
                mov     ah, 2                   ; Set ah - setting cursor position
                int     10h                     ; Execute BIOS

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret     4                       ; Return & deallocate 2 parameters
Set_Cursor      endp

;---------------------------------------------------------------------------
;-- PURPOSE  : Shows the text cursor.  No parameters are passed by the ADA
;--             subprogram. The shape of the cursor is preserved.
;-- BIOS FUNCTION (Read cursor position)
;--     INTERRUPT : 10h
;--     FUNCTION : 3
;--     CALLED WITH  : AH = 3              BH = page
;--     RETURNS WITH : AX changed          CH = starting cursor line
;--                    DH = row            CL = ending cursor line
;--                    DL = column
;-- BIOS FUNCTION (Set cursor shape)
;--     INTERRUPT : 10h
;--     FUNCTION : 1
;--     CALLED WITH  : AH = 1
;--                    CH = starting cursor line  (bits 3 - 0)
;--                    CL = ending cursor line    (bits 3 - 0)
;--                    CH = display cursor        (bit  5 = 0)
;--                    CH = hide cursor           (bit  5 = 1)
;--     RETURNS WITH : AX changed
;---------------------------------------------------------------------------

                public  Show_Cursor
Show_Cursor     proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Read cursor position and shape
                xor     bh, bh                  ; Set page number
                mov     ah, 3                   ; Set ah - reading cursor value
                int     10h                     ; Execute BIOS

; Mask out the necessary bits and call BIOS to display cursor
                and     cx, 0000111100001111b; Mask out invalid bits
                mov     ah, 1                   ; Set ah for cursor shape
                int     10h                     ; Execute BIOS function

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret                             ; Return
Show_Cursor     endp

;---------------------------------------------------------------------------
;------------------------------ Graphic Functions ---------------------------
;---------------------------------------------------------------------------

;---------------------------------------------------------------------------
;-- PURPOSE  : Changes  the screen  from text  to  graphics mode  or  back
;--             again.  It also clears the screen  to a blue background and
;--             hides the cursor for text screens.
;-- BIOS FUNCTION (Change mode)
;--     INTERRUPT : 10h
;--     FUNCTION : 00h
;--     CALLED WITH  : AH = 00h
;--                    AL = screen mode
;--
;-- BIOS FUNCTION (Set colour palette)
;--     INTERRUPT : 10h
;--     FUNCTION : 0Bh
;--     CALLED WITH  : AH = 0Bh
;--                    BH = 00 for background colour     = 01 for palette
;--                    BL = palette for BH = 01
;---------------------------------------------------------------------------

                public  Change_Screen
Change_Screen   proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame
                xor     ax, ax                  ; Clear ax
                cmp     word ptr 6 [bp], 1; Get column of pixel to plot
                jz      graphic_mode            ; If graphic, set graphic mode
                mov     al, TEXT_MODE3          ; Set text mode
                jmp     short set_screen_mode; Jump to BIOS call
graphic_mode:   mov     al, GRAPHIC_SCREEN; Set graphic mode
set_screen_mode:mov     SCREEN_MODE, al ; Save screen mode
                int     10h                     ; Execute BIOS

; Clear the screen with a blue background and hide the cursor
                cmp     SCREEN_MODE, TEXT_MODE3; Is screen in text mode ?
                jne     change_for_graph; If so, skip graphic instructions
                call    Hide_Cursor             ; Hide the cursor on a text screen
                mov     bh, 1Eh                 ; Set attribute for yellow on blue
                jmp     short skip_graphic; Skip over graphic attribute
change_for_graph:cmp    SCREEN_MODE, CGA_MODE4; Is the screen a CGA ?
                jne     not_cga                 ; If not skip CGA colour setting
                mov     ah, 0Bh                 ; Set for colour palette
                mov     bx, 0001h               ; Set for blue background
                int     10h                     ; Execute BIOS function
                mov     ah, 0Bh                 ; Set for colour palette
                mov     bx, 0100h               ; Set for green foreground (1 pal)
                int     10h                     ; Execute BIOS function
                jmp     short cga1              ; Skip screen clearing procedure
not_cga:        mov     bh, 01                  ; Set attribute for blue
skip_graphic:   xor     cx, cx                  ; Set top row and left side
```

```
                mov     dx, 184Fh               ; Set bottom row and right side
                mov     ax, 0600h               ; Set ax to clear screen
                int     10h                     ; Execute BIOS

; Return from procedure
cga1:           pop     bp                      ; Restore the frame pointer
                ret     2                       ; Return and deallocate 1 near
                                                ;    parameter
Change_Screen   endp

;---------------------------------------------------------------------------
;-- PURPOSE : Copies a character to the graphic screen. The colour  of the
;--            character can be set. The location to which the character is
;--            written is determined from coordinates passed as parameters.
;--            These Ada screen coordinates are converted  to those used by
;--            BIOS.
;--             Column screen coordinates : ADA (1 - 80)   BIOS (0 - 79)
;--             Row    screen coordinates : ADA (1 - 25)   BIOS (0 - 24)
;-- BIOS FUNCTION (Set cursor position)
;--     INTERRUPT : 10h
;--     FUNCTION : 2
;--     CALLED WITH  : AH = 2              DH = row          DL = column
;--                    BH = page
;--     RETURNS WITH : AX changed
;--
;-- BIOS FUNCTION (Write Teletype)
;--     INTERRUPT : 10h
;--     FUNCTION : 14
;--     CALLED WITH  : AH = 14             AL = character  BH = page
;--                    BL = colour
;---------------------------------------------------------------------------

                public  Character_To_Graphic
Character_To_Graphic    proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get new cursor position from parameters
                mov     dx, 8 [bp]              ; Row    (integer format)
                dec     dx                      ; Adjust row for BIOS
                jl      invalid_coord4          ; If Y < 0, then exit procedure
                cmp     dx, ROW_MAXIMUM         ; Is Y > maximum Y coordinate
                jge     invalid_coord4          ; If so, then exit procedure
                mov     dh, dl                  ; Get row into byte form
                mov     ax, 10 [bp]             ; Column  (integer format)
                dec     ax                      ; Adjust row value
                jl      invalid_coord4          ; If X < 0, then exit procedure
                cmp     ax, COLUMN_MAXIMUM; Is X > maximum X coordinate
                jge     invalid_coord4          ; If so, then exit procedure
                mov     dl, al                  ; Get column into byte form

; Set cursor position
                xor     bh, bh                  ; Set page number
                mov     ah, 2                   ; Set ah - setting cursor position
                int     10h                     ; Execute BIOS

; Set up and call BIOS character write function
                mov     bx, 12 [bp]             ; Get foreground colour from stack
                xor     bl, 01                  ; XOR colour - screen attributes
                or      bl, 80h                 ; Set high bit for XORing - screen
                cmp     SCREEN_MODE, CGA_MODE4; Is the screen a CGA ?
                jne     not_cga4                ; If not skip CGA colour setting
                mov     bl, 01h                 ; Set for foreground colour
not_cga4:       mov     ax, 6 [bp]              ; Get character from stack
                mov     ah, 0Eh                 ; Set for character write function
                int     10h                     ; Execute BIOS function

; Return from procedure
invalid_coord4: pop     bp                      ; Restore the frame pointer
                ret     8                       ; Return and deallocate 4 near
                                                ;    parameters
Character_To_Graphic    endp

;---------------------------------------------------------------------------
;-- PURPOSE  : Scrolls  a  specified  part  of the  screen  up.  It  calls
;--             GetScreenParameters to get the coordinates of the specified
;--             part from the ADA record data structure which is passed.
;-- BIOS FUNCTION (Scroll up)
;--     INTERRUPT : 10h
;--     FUNCTION : 6
;--     CALLED WITH  : AH = 6
;--                  : AL = number of lines. Input lines are blanked at the
;--                         bottom  of the  scroll_window.  With  (AL = 0),
;--                         entire scroll_window is cleared.
;--                    BH = Attribute for added blank line
;--                    CH = upper row of scroll_window
;--                    CL = left column of scroll_window
;--                    DH = lower row of scroll_window
;--                    DL = right column of scroll_window
;---------------------------------------------------------------------------

                public  Clear_Screen
Clear_Screen    proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame
                call    MouseVisible            ; Hide mouse if active and visible

; Clear specified part of the screen
                xor     cx, cx                  ; Set upper left corner
                mov     dx, 184Fh               ; Set bottom right corner
                mov     bh, 6 [bp]              ; Get attributes from stack
                cmp     SCREEN_MODE, TEXT_MODE3; Is screen in text mode ?
                je      leave_for_text          ; If so, skip graphic instructions
                cmp     SCREEN_MODE, CGA_MODE4; Is the screen a CGA ?
                jne     not_cga5                ; If not skip CGA colour setting
                xor     bh, bh
not_cga5:       and     bh, 00001111b           ; Only foreground attributes used
leave_for_text: mov     ax, 0600h               ; Set ax to clear screen
                int     10h                     ; Execute BIOS

; Return from procedure
cga_finish:     call    MouseVisibleAgain; Mouse active and was visible?
                pop     bp                      ; Restore the frame pointer
                ret     2                       ; Return and deallocate 1 near
                                                ;    parameters
Clear_Screen    endp

;---------------------------------------------------------------------------
;-- PURPOSE  : Draws a  dotted  horizontal  line starting at  (X1, Y1)  to
;--             (X2, Y1) in the present foreground colour.
```

```
;--- BIOS FUNCTION (Plot pixel)
;---     INTERRUPT : 10h
;---     FUNCTION : 0Ch
;---     CALLED WITH  : AH = 0Ch
;---                  : AL = colour of the pixel
;---                    CX = column where pixel is to be plotted
;---                    DX = row where pixel is to be plotted
;-------------------------------------------------------------------

                public  Dot_Horizontal_Line
Dot_Horizontal_Line     proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Plot a horizontal line
                mov     cx, 8 [bp]      ; Get X1 value from the stack
                cmp     cx, 10 [bp]     ; If X2 < X1, then decrement
                jl      next_horiz_dot  ; Skip next instruction if X2 < X1
                xchg    cx, 10 [bp]     ; Swap end with start of line
next_horiz_dot: mov     dx, 12 [bp]     ; Get Y value from the stack
                call    Basic_Plot      ; Plot point
                inc     cx              ; Adjust X for plot adjusting
                add     cx, 6 [bp]      ; Add dot offset to X value
                cmp     cx, 10 [bp]     ; Is the line finished ?
                jl      next_horiz_dot  ; If not do another dot

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 4 near
                                        ;    parameter
Dot_Horizontal_Line     endp

;-------------------------------------------------------------------
;--- PURPOSE  : Draws a dotted vertical line starting at (X1, Y1) to
;---          (X1, Y2) in the present foreground colour.
;--- BIOS FUNCTION (Plot pixel)
;---     INTERRUPT : 10h
;---     FUNCTION : 0Ch
;---     CALLED WITH  : AH = 0Ch
;---                  : AL = colour of the pixel
;---                    CX = column where pixel is to be plotted
;---                    DX = row where pixel is to be plotted
;-------------------------------------------------------------------

                public  Dot_Vertical_Line
Dot_Vertical_Line       proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Plot a vertical line
                mov     dx, 8 [bp]      ; Get Y1 value from the stack
                cmp     dx, 10 [bp]     ; If Y2 < Y1, then decrement
                jl      next_vert_dot   ; Skip next instruction if Y2 < Y1
                xchg    dx, 10 [bp]     ; Swap end with start of line
next_vert_dot:  mov     cx, 12 [bp]     ; Get X value from the stack
                call    Basic_Plot      ; Plot point
                inc     dx              ; Adjust Y for plot adjusting
                add     dx, 6 [bp]      ; Add dot offset to Y value
                cmp     dx, 10 [bp]     ; Is the line finished ?
                jl      next_vert_dot   ; If not do another dot

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 4 near
                                        ;    parameter
Dot_Vertical_Line       endp

;-------------------------------------------------------------------
;--- PURPOSE  : Draws a line starting at (X1, Y1) to (X2, Y2) in the
;---          present foreground colour.
;--- BIOS FUNCTION (Plot pixel)
;---     INTERRUPT : 10h
;---     FUNCTION : 0Ch
;---     CALLED WITH  : AH = 0Ch
;---                  : AL = colour of the pixel
;---                    CX = column where pixel is to be plotted
;---                    DX = row where pixel is to be plotted
;-------------------------------------------------------------------

                public  Line
Line            proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Check if the pair of coordinates are the same
                mov     ax, 8 [bp]      ; Get X1 from stack
                mov     bx, 6 [bp]      ; Get Y1 from stack
                mov     cx, 10 [bp]     ; Get X2 from stack
                mov     dx, 12 [bp]     ; Get Y2 from star
                cmp     cx, ax          ; Are the X coords  as the same ?
                jne     different_coord1 ; If not, line is not horizontal
                cmp     dx, bx          ; Are the Y coordinates the same ?
                jne     different_coord2 ; If not, then a line is drawn
                call    Basic_Plot      ; Plot point
                jmp     line_drawn      ; Exit from procedure

different_coord1:cmp    dx, bx          ; Are the Y coordinates the same ?
                jne     different_coord ; If not, diag line is drawn

; Plot a horizontal line
                cmp     cx, ax          ; If end > start, skip instruct
                jl      next_horiz_plot ; If the start is after the end
                xchg    ax, cx          ; Swap start with end of line
next_horiz_plot:mov     dx, 6 [bp]      ; Get Y value from stack
                push    ax              ; Save the end of the line value
                call    Basic_Plot      ; Plot point
                pop     ax              ; Restore end of the line value
                inc     cx              ; Adjust X for plot adjusting
                inc     cx              ; Increment X value
                cmp     cx, ax          ; Is X > end of the line ?
                jle     next_horiz_plot ; If not, then plot another point
                jmp     line_drawn; Exit from procedure

; Plot a vertical line
different_coord2:jl     next_vert_plot  ; If end > start, skip instruct
                xchg    dx, bx          ; Swop start with end of line
next_vert_plot: mov     cx, 8 [bp]      ; Get X value from the stack
                push    bx              ; Save the end of the line value
                call    Basic_Plot      ; Plot point
                pop     bx              ; Restore end of the line value
                inc     dx              ; Adjust Y for plot adjusting
                inc     dx              ; Increment Y value
                cmp     dx, bx          ; If Y > end of the line ?
                jle     next_vert_plot  ; If not, then plot another point
                jmp     short line_drawn; Exit from procedure

; Calculate values needed in drawing the line
different_coord:mov     si, dx          ; Get X end of line
                mov     cx, 1           ; Set initial X increment
                sub     si, ax          ; Subtract X start of line
                jge     line_goes_right ; If line goes down, don't change
                neg     cx              ; Negate X increment
                neg     si              ; Negate diagonal x increment
line_goes_right:mov     DIAGONAL_X_INCREMENT, cx; Set diagonal X increment
                mov     di, dx          ; Get Y end of line
                mov     dx, 1           ; Set initial Y increment
                sub     di, bx          ; Subtract Y start of line
                jge     line_goes_down  ; If line goes down, don't change
                neg     dx              ; Negate Y increment
                neg     di              ; Negate diagonal Y increment
line_goes_down: mov     DIAGONAL_Y_INCREMENT, dx; Set diagonal Y increment
                cmp     si, di          ; Is X length > than Y length
                jge     horizontal      ; Skip vertical line code
                xchg    si, di          ; Swop X length with Y length
                xor     cx, cx          ; Clear X increment - vertcal line
                jmp     vertical        ; Skip horizontal line code
horizontal:     xor     dx, dx          ; Clear Y increment - hrntl line
vertical:       mov     STRAIGHT_X_INCREMENT, cx; X horizontal increment
                mov     STRAIGHT_Y_INCREMENT, dx; Y vertical increment
                mov     ax, di          ; Get longest component
                shl     ax, 1           ; Multiply by 2
                mov     STRAIGHT_COUNT, ax; Set straight count
                sub     ax, si          ; Subtract short length
                mov     bx, ax          ; Set count
                sub     ax, si          ; Subtract short length
                mov     DIAGONAL_COUNT, ax; Set diagonal count
                mov     cx, 6 [bp]      ; Set to the X start of the line
                mov     dx, 8 [bp]      ; Set to the Y start of the line
                inc     si              ; Adjust short length for loop
main_line_loop: dec     si              ; Decrement short length
                jz      line_drawn      ; Exit if line has been drawn
                push    bx              ; Save count
                call    Basic_Plot      ; Plot point
                inc     cx              ; Adjust X for plot adjusting
                inc     dx              ; Adjust Y for plot adjusting
                pop     bx              ; Restore count
                cmp     bx, 0           ; Is the diagonal line drawn ?
                jge     diagonal_line   ; If not, more of diagonal line
                add     cx, STRAIGHT_X_INCREMENT; Add straight incr to x
                add     dx, STRAIGHT_Y_INCREMENT; Add straight incr to y
                add     bx, STRAIGHT_COUNT; Add straight count to count
                jmp     short main_line_loop; Go back - plot another point
diagonal_line:  add     cx, DIAGONAL_X_INCREMENT; Add diagonal count to x
                add     dx, DIAGONAL_Y_INCREMENT; Add diagonal count to y
                add     bx, DIAGONAL_COUNT; Add diagonal count to count
                jmp     short main_line_loop; Go back - plot another point

; Return from procedure
line_drawn:     pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 4 near
                                        ;    parameter
Line            endp

;-------------------------------------------------------------------
;--- PURPOSE  : Plots a pixel in the present plot colour on the screen at
;---          the position specified. The column must be given in the CX
;---          register and the row in the DX register.
;--- BIOS FUNCTION (Plot pixel)
;---     INTERRUPT : 10h
;---     FUNCTION : 0Ch
;---     CALLED WITH  : AH = 0Ch
;---                  : AL = colour of the pixel
;---                    CX = column where pixel is to be plotted
;---                    DX = row where pixel is to be plotted
;-------------------------------------------------------------------

Basic_Plot      proc
                cmp     dx, Y_MINIMUM   ; Is Y < minimum Y coordinate
                jl      invalid_coord1  ; If so, then exit procedure
                cmp     dx, Y_MAXIMUM   ; Is Y > maximum Y coordinate
                jg      invalid_coord2  ; If so, then exit procedure
                cmp     cx, X_MINIMUM   ; Is X < minimum X coordinate
                jl      invalid_coord1  ; If so, then exit procedure
                cmp     cx, X_MAXIMUM   ; Is X > maximum X coordinate
                jg      invalid_coord1  ; If so, then exit procedure
                dec     dx              ; Adjust row value
                dec     cx              ; Adjust column value
                xor     bx, bx          ; Set for page 0
                mov     al, PLOT_COLOUR ; Get colour of the pixel
                mov     ah, 0Ch         ; Set ah to plot pixel
                int     10h             ; Execute BIOS
invalid_coord1: ret                     ; Return
Basic_Plot      endp

;-------------------------------------------------------------------
;--- PURPOSE  : Plots a pixel in the present plot colour on the screen at
;---          the position specified. It calls Basic_Plot to actually do
;---          the plotting.
;-------------------------------------------------------------------

                public  Plot
Plot            proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                mov     cx, 6 [bp]      ; Get column of pixel to plot
                mov     dx, 8 [bp]      ; Get row of pixel to plot
                call    Basic_Plot      ; Plot point

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 2 near
                                        ;    parameters
Plot            endp

;-------------------------------------------------------------------
;--- PURPOSE  : Plots four pixels about a centre point in the present fore-
;---          ground colour using a specified position.
;--- BIOS FUNCTION (Plot pixel)
;---     INTERRUPT : 10h
;---     FUNCTION : 0Ch
```

```
;---    CALLED WITH  : AH = 0Ch
;---                 : AL = colour of the pixel
;---                   CX = column where pixel is to be plotted
;---                   DX = row where pixel is to be plotted
;-------------------------------------------------------------------

                public  Plot_Four_Pixels
Plot_Four_Pixels        proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Plot first quadrant dot
                mov     cx, 6 [bp]      ; Get x centre of object
                add     cx, 10 [bp]     ; Add x coordinate
                mov     dx, 8 [bp]      ; Get y centre of object
                sub     dx, 12 [bp]     ; Subtract y coordinate
                call    Basic_Plot      ; Plot point

; Plot second quadrant dot
                mov     cx, 6 [bp]      ; Get x centre of object
                add     cx, 10 [bp]     ; Add x coordinate
                mov     dx, 8 [bp]      ; Get y centre of object
                add     dx, 12 [bp]     ; Add y coordinate
                call    Basic_Plot      ; Plot point

; Plot third quadrant dot
                mov     cx, 6 [bp]      ; Get x centre of object
                sub     cx, 10 [bp]     ; Subtract x coordinate
                mov     dx, 8 [bp]      ; Get y centre of object
                add     dx, 12 [bp]     ; Add y coordinate
                call    Basic_Plot      ; Plot point

; Plot forth quadrant dot
                mov     cx, 6 [bp]      ; Get x centre of object
                sub     cx, 10 [bp]     ; Subtract x coordinate
                mov     dx, 8 [bp]      ; Get y centre of object
                sub     dx, 12 [bp]     ; Subtract y coordinate
                call    Basic_Plot      ; Plot point

; Return from procedure
no_more_points: pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 4 near
                                        ;    parameter
Plot_Four_Pixels        endp

;-------------------------------------------------------------------
;--- PURPOSE  : Sets  the  plot colour used when drawing lines,  plotting 4
;---            points, etc.
;-------------------------------------------------------------------

                public  Set_Plot_Colour
Set_Plot_Colour proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get new plot colour from parameter
                mov     ax, 6 [bp]      ; Get plot colour from stack
                cmp     SCREEN_MODE, CGA_MODE; Is the screen a CGA ?
                jne     not_cga2        ; If not skip CGA colour setting
                mov     al, 01h         ; Set for foreground colour
not_cga2:       mov     PLOT_COLOUR, al ; Set plot colour

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     2               ; Return & deallocate 1 parameters
Set_Plot_Colour endp

;-------------------------------------------------------------------
;--- PURPOSE  : Sets the window where a dot can be plotted.
;-------------------------------------------------------------------

                public  Set_Plot_Window
Set_Plot_Window proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get new plot colour from parameter
                mov     ax, 6 [bp]      ; Get minimum x value from stack
                mov     X_MINIMUM, ax   ; Set minimum x value
                mov     ax, 8 [bp]      ; Get minimum y value from stack
                mov     Y_MINIMUM, ax   ; Set minimum y value
                mov     ax, 10 [bp]     ; Get maximum x value from stack
                mov     X_MAXIMUM, ax   ; Set maximum x value
                mov     ax, 12 [bp]     ; Get maximum y value from stack
                mov     Y_MAXIMUM, ax   ; Set maximum y value

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     8               ; Return & deallocate 4 parameters
Set_Plot_Window endp

;-------------------------------------------------------------------
;--- PURPOSE : Copies a string to the  graphic screen.  The  colour  of the
;---           characters can be set.  The location  to which the string is
;---           written is determined from coordinates passed as parameters.
;---           These Ada screen coordinates are converted  to those used by
;---           BIOS. If the string is longer  than what can be displayed on
;---           the screen, the screen is scrolled.
;---            Column screen coordinates : ADA (1 - 80)   BIOS (0 - 79)
;---            Row    screen coordinates : ADA (1 - 25)   BIOS (0 - 24)
;--- BIOS FUNCTION (Set cursor position)
;---     INTERRUPT : 10h
;---     FUNCTION : 2
;---     CALLED WITH  : AH = 2               DH = row        DL = column
;---                    BH = page
;---     RETURNS WITH : AX changed
;---
;--- BIOS FUNCTION (Write teletype)
;---     INTERRUPT : 10h
;---     FUNCTION : 14
;---     CALLED WITH  : AH = 14              AL = character  BH = page
;---                    BL = colour
;-------------------------------------------------------------------

                public  String_To_Graphic
String_To_Graphic       proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment

; Get new cursor position from parameters
                mov     dx, 10 [bp]     ; Row    (integer format)
                dec     dx              ; Adjust row for BIOS
                jl      invalid_coord3  ; If Y < 0, then exit procedure
                cmp     dx, ROW_MAXIMUM ; Is Y > maximum Y coordinate
                jge     invalid_coord3  ; If so, then exit procedure
                mov     dh, dl          ; Get row into byte form
                mov     ax, 12 [bp]     ; Column  (integer format)
                dec     ax              ; Adjust row value
                jl      invalid_coord3  ; If X < 0, then exit procedure
                cmp     ax, COLUMN_MAXIMUM; Is X > maximum X coordinate
                jge     invalid_coord3  ; If so, then exit procedure
                mov     dl, al          ; Get column into byte form

; Set cursor position
                xor     bh, bh          ; Set page number
                mov     ah, 2           ; Set ah = setting cursor position
                int     10h             ; Execute BIOS

; Get pointer to string from stack
                push    ds
                lds     si, 6 [bp]      ; Get value of pointer from stack
                les     di, [si]        ; Get address of string pointer

; Determine string length
                mov     cx, 6 [si]      ; Get upper bound of string
                sub     cx, 4 [si]      ; Subtract lower bound of string
                inc     cx              ; Adjust string length
                pop     ds              ; Restore seg pointer to Ada data

; Set up And call BIOS character write function
next_graph_char:mov     bx, 14 [bp]     ; Get foreground colour from stack
                xor     bl, 01          ; XOR colour = screen attributes
                or      bl, 80h         ; Set high bit for XORing = screen
                cmp     SCREEN_MODE, CGA_MODE; Is the screen a CGA ?
                jne     not_cga3        ; If not skip CGA colour setting
                mov     bl, 01h         ; Set for foreground colour
not_cga3:       mov     ah, 0Eh         ; Set for character write function
                mov     al, es: [di]    ; Get string character
                inc     di              ; Adjust to true length of string
                push    cx              ; Save string count
                int     10h             ; Execute BIOS function
                pop     cx              ; Restore count
                loop    next_graph_char

; Return from procedure
invalid_coord3: pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret     10              ; Return and deallocate 5 near
                                        ;    1 far parameters
String_To_Graphic       endp

;-------------------------------------------------------------------
;                          Mouse Functions
;-------------------------------------------------------------------

;-------------------------------------------------------------------
;--- PURPOSE  : Checks to if mouse is installed, and if so, gets the number
;---            of buttons,  sets the  mouse cursor to  the center of  the
;---            screen, using the default mouse cursor and movement ratios.
;---            The mouse text cursor  is not displayed.  The  MOUSE_STATUS
;---            flag is set to 1 if a mouse is present, 0 if it is not.
;--- MOUSE FUNCTION (Check if mouse installed)
;---                (If mouse present, determines number of buttons)
;---                (If mouse present, resets mouse position to the center)
;---     INTERRUPT : 33h
;---     FUNCTION : 0
;---     CALLED WITH  : AX = 0
;---     RETURNS WITH : AX = -1 if mouse installed
;---                    =  0 if mouse not installed
;---                    BX = number of buttons
;-------------------------------------------------------------------

                public  Check_For_Mouse
Check_For_Mouse proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Read mouse text cursor position
                xor     ax, ax          ; Set ax = reading cursor position
                int     33h             ; Call mouse driver

; Return mouse text cursor position as parameters
                push    ds              ; Save data segment pointer
                mov     cx, bx          ; Move number of buttons into cx
                lds     bx, 4 [bp]      ; Get address of return variable
                cmp     ax, -1          ; Is ax equal to -1
                jne     no_correction
                add     ax, 2           ; Adjust for Ada enumeration type
no_correction:  mov     [bx], ax        ; If mouse installed 1, else  0
                lds     bx, 10 [bp]     ; Get address of return variable
                mov     [bx], cx        ; Put number of buttons onto stack
                pop     ds              ; Restore value of data segment
                mov     MOUSE_STATUS, ax; Set flag if the mouse is present

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 2 far
                                        ;    parameter
Check_For_Mouse endp

;-------------------------------------------------------------------
;--- PURPOSE  : Hides the mouse text cursor.  No  parameters are  passed by
;---            the ADA subprogram.  If there is no mouse present,  (checks
;---            if MOUSE_STATUS not equal to 1), the procedure exits.
;--- MOUSE FUNCTION (Hide mouse text cursor)
;---     INTERRUPT : 33h
;---     FUNCTION : 2
;---     CALLED WITH  : AX = 2
;-------------------------------------------------------------------

                public  Hide_Text_Mouse
Hide_Text_Mouse proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Check if a mouse is present and visible
                cmp     MOUSE_STATUS, 1 ; Is a mouse present and visible ?
                jne     no_mouse4       ; If not, exit
```

```
; Hide the mouse text cursor
                mov     ax, 2           ; Set ax to hide cursor
                int     33h             ; Call mouse driver
                and     MOUse_STATUS, 0ffF0h; Clear flag for mouse visible

; Return from procedure
no_mouse4:      pop     bp              ; Restore the frame pointer
                ret                     ; Return
Hide_Text_Mouse endp

;-------------------------------------------------------------------------
;-- PURPOSE  : Reads the mouse text cursor position.  Gets the  mouse text
;--            cursor position from the MOUSE driver. The coordinates  are
;--            adjusted to the ADA coordinates.  The  coordinates  are put
;--            onto the stack as return parameters. The buttons pressed on
;--            the mouse are also returned.  If there is no mouse present,
;--            (checks if MOUSE_STATUS not = 1), the procedure exits.
;--            Column screen coordinates : ADA (1 - 80) MOUSE (0 - 79))
;--            Row    screen coordinates : ADA (1 - 25) MOUSE (0 - 24))
;-- MOUSE FUNCTION (Read mouse text cursor position and buttons pressed)
;--    INTERRUPT : 33h
;--    FUNCTION : 3
;--    CALLED WITH  : AX = 3
;--    RETURNS WITH : BX = button status
;--                       bit 0 = left   button depressed
;--                       bit 1 = right  button depressed
;--                       bit 2 = middle button depressed
;--                   CX = x coordinate          DX = y coordinate
;-------------------------------------------------------------------------

                public  Read_Text_Mouse
Read_Text_Mouse proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Check if a mouse is present
                cmp     MOUSE_STATUS, 0 ; Is a mouse present and visible ?
                jne     no_mouse        ; If not, exit

; Read mouse text cursor position
                mov     ax, 3           ; Set ax = reading cursor position
                int     33h             ; Call mouse driver

; Return mouse text cursor position as parameters
                mov     ax, bx          ; Move buttons depressed into ax
                push    ds              ; Save data segment pointer
                lds     bx, 14 [bp]     ; Get address of return variable
                mov     [bx], ax        ; Put buttons depressed onto stack
                mov     ax, cx          ; Move cursor column into ax
                mov     cx, 3           ; Load shift count
                shr     ax, cl          ; Divide by 8
                inc     ax              ; Adjust column in ax
                lds     bx, 10 [bp]     ; Get address of return variable
                mov     [bx], ax        ; Put cursor column onto stack
                mov     ax, dx          ; Move cursor row int, ax
                mov     cx, 3           ; Load shift count
                shr     ax, cl          ; Divide by 8
                inc     ax              ; Adjust row in ax
                lds     bx, 6 [bp]      ; Get address of return Variable
                mov     [bx], ax        ; Put cursor row onto stack
                pop     ds              ; Restore value of data segment

; Return from procedure
no_mouse:       pop     bp              ; Restore the frame pointer
                ret     12              ; Return and deallocate 3 far
                                        ;    parameter
Read_Text_Mouse endp

;-------------------------------------------------------------------------
;-- PURPOSE  : Shows the mouse text cursor.  No parameters  are  passed by
;--            the ADA subprogram.
;-- MOUSE FUNCTION (Show mouse text cursor)
;--    INTERRUPT : 33h
;--    FUNCTION : 1
;--    CALLED WITH  : AX = 1
;-------------------------------------------------------------------------

                public  Show_Text_Mouse
Show_Text_Mouse proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Check if a mouse is present and visible
                cmp     MOUSE_STATUS, 1 ; Mouse present but not visible?
                jne     no_mouse3       ; If not, exit

; Show the mouse text cursor
                mov     ax, 1           ; Set ax to show cursor
                int     33h             ; Call mouse driver
                or      MOUSE_STATUS, 0003h; Set flag for Mouse visible

; Return from procedure
no_mouse3:      pop     bp              ; Restore the frame pointer
                ret                     ; Return
Show_Text_Mouse endp

;-------------------------------------------------------------------------
;-- PURPOSE : All subroutines that write  to  the screen must check to see
;--           if the mouse pointer is active.  If the mouse is active,  it
;--           is turned off before writing to the screen.
;-------------------------------------------------------------------------

MouseVisible    proc
                cmp     MOUSE_STATUS, 3 ; Is mouse active and visible ?
                jne     visible_check   ; If so, make mouse not visible
                call    Hide_Text_Mouse ; Hide the mouse pointer
                or      MOUSE_STATUS, 0004h; Set flag - mouse was visible
visible_check:  ret
MouseVisible    endp

;-------------------------------------------------------------------------
;-- PURPOSE : All subroutines that write  to  the screen must check to see
;--           if the mouse pointer is active.  If the mouse was active, but
;--           MouseVisible turned it off, it is turned on again.
;-------------------------------------------------------------------------

MouseVisibleAgain       proc
                cmp     MOUSE_STATUS, 5 ; Mouse active and Was visible ?
                jne     visible_reset   ; If so, make mouse visible
                and     MOUSE_STATUS, 0fFF0h;Clear flag-mouse was visible
                call    Show_Text_Mouse ; Show mouse pointer
visible_reset:  ret
MouseVisibleAgain       endp

;=========================================================================
;=========================== Keyboard Functions ==========================
;=========================================================================

;-------------------------------------------------------------------------
;-- PURPOSE  : Gets a keystroke from the keyboard,  and  waits until a key
;--            is  pressed  if  a  keystroke  is  not  available when  first
;--            called. If the key is an  extended key,  then the high byte
;--            contains the keycode while the low byte is 0. If the key is
;--            a normal key,  then the  high byte is  0  and the  low byte
;--            contains the keycode.
;-- BIOS FUNCTION (Read character from keyboard and removes it  from the)
;--                (  keyboard buffer.  It waits for a  keystroke  if none)
;--                (  initially available.  It does not process Ctrl-Break)
;--                (  and does not echo.)
;--    INTERRUPT : 16h
;--    FUNCTION : 0
;--    CALLED WITH  : AH = 0
;--    RETURNS WITH : AL = normal code if character ready
;--                   AH = extended code if character ready
;-------------------------------------------------------------------------

                public  Get_Keyboard_Character
Get_Keyboard_Character  proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Read the keyboard character
                xor     ax, ax          ; Set ah to read character which
                                        ;   waits for a keystroke
                int     16h             ; Execute BIOS
                cmp     al, 0           ; Is key an extended key
                je      extended_key    ; If so, skip next instruction
                mov     ah, 0           ; Clear ah for normal key pressed

; Put keyboard value onto stack for return parameter
extended_key:   push    ds              ; Save data segment pointer
                lds     bx, 6 [bp]      ; Get address of return variable
                mov     [bx], ax        ; Put keystroke onto stack
                pop     ds              ; Restore value of data segment

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Get_Keyboard_Character  endp

;-------------------------------------------------------------------------
;-- PURPOSE  : Determines if there is a character in the keyboard buffer.
;--            In other words, has a key been pressed.
;-- BIOS FUNCTION (Read character from keyboard. If a character is ready,)
;--                (  its code is returned, but it is not removed from the)
;--                (  keyboard buffer.  It does not wait  for a keystroke,)
;--                (  does not process Ctrl-Break and does not echo.)
;--    INTERRUPT : 16h
;--    FUNCTION : 1
;--    CALLED WITH  : AH = 1
;--    RETURNS WITH : Zero flag set   = no character ready
;--                   Zero flag clear = character ready
;--                   AL = normal code if character ready
;--                   AH = extended code if character ready
;-------------------------------------------------------------------------

                public  KeyBoard_Character_Ready
KeyBoard_Character_Ready        proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Ask BIOS if a character is ready and report result
                mov     ah, 1           ; Set ah to      character but
                                        ;   does wai     a keystroke
                int     16h             ; Execute BI
                jnz     character_ready ; Jump if cha        e ready
                xor     cx, cx          ; Set cx for c...  ...ter not ready
                jmp     short exit_keyboard; Report result
character_ready:
                mov     cx, 1           ; Set cx for character ready

; Return from procedure
exit_keyboard:  pop     bp              ; Restore the frame pointer
                ret                     ; Return
KeyBoard_Character_Ready        endp

;=========================================================================
;========== Screen and Window Buffer Initialisation Function ============
;=========================================================================

;-------------------------------------------------------------------------
;-- PURPOSE  : Determines if a VGA, EGA,  CGA or MDA is installed.  If MDA
;--            is installed, an error message is displayed and the program
;--            is terminated. If any of the other 3 adaptors are installed
;--            the resolution for  X  and  Y as well as the screen mode is
;--            determined.
;-- BIOS FUNCTION (Select alternate screen routine)
;--    INTERRUPT : 10h
;--    FUNCTION : 12h
;--    CALLED WITH  : AH = 12h              BL = 10h (EGA information)
;--    RETURNS WITH : BH = 00h for colour, 01h monochrome
;--                   BL = 00h for 64K, 01h for 128K, 02h for 192K,
;--                        03h for 256K
;--                   CH = Adapter bits
;--                   CL = Switch setting
;--
;-- BIOS FUNCTION (Read/write display combination code - for PS/2)
;--    INTERRUPT : 10h
;--    FUNCTION : 1Ah
;--    CALLED WITH  : AH = 1Ah
;--    RETURNS WITH : ?
;--
;-- DOS FUNCTION (Write string to standard output - delimiter $)
;--    INTERRUPT : 21h
;--    FUNCTION : 9
;--    CALLED WITH  : AH = 9            DX = points to  tart of screen
;--    RETURNS WITH : -----
```

```
;---
;--- DOS FUNCTION [Terminate program]
;---     INTERRUPT : 21h
;---     FUNCTION  : 4Ch
;---     CALLED WITH  : AH = 4Ch
;---     RETURNS WITH : -----
;-------------------------------------------------------------------------

                public  Screen_Setup
Screen_Setup    proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Determine if the system contains a VGA
                mov     ax, 1A00h       ; Request VGA service, function 0
                int     10h             ; Execute BIOS
                cmp     al, 1Ah         ; If al contains 1Ah on return
                je      vga_found       ;    system contains a VGA

; Determine if the system contains a EGA
                mov     ah, 12h         ; Request EGA service
                mov     bl, 10h         ; Get amount of EGA memory
                int     10h             ; Execute BIOS
                cmp     bl, 10h         ; If bl no longer contains 10h,
                jne     ega_found       ;    system contains an EGA

; Determine if the system contains a CGA or MDA
                push    es              ; Save es segment value
                mov     ax, 0040h       ; Set ax the BIOS data segment
                mov     es, ax          ; Set es and bx to the BIOS data
                mov     bx, 0063h       ;   area where screen address is
                mov     ax, es:[bx]     ; Get screen memory base address
                pop     es              ; Restore es segment value
                cmp     ax, 3D0h        ; Is the address for CGA ?
                jge     cga_found       ; If so, skip monochrome code

; Give error message that the program cannot be used monochrome and exit
                mov     ah, 09          ; Set for DOS display
                mov     dx, offset ERROR_MESSAGE; Set pointer to string
                int     21h             ; Execute DOS
                mov     ah, 4Ch         ; Set for program termination
                int     21h             ; Execute DOS

vga_found:
ega_found:      mov     ax, 350         ; Set Y resolution for mode 16
                mov     GRAPHIC_SCREEN, EGAVGA_MODE16; Set graphic mode
                jmp     short set_screen

cga_found:      mov     ax, 200         ; Set Y resolution for mode 4
                mov     GRAPHIC_SCREEN, CGA_MODE4; Set graphic mode

; Return screen resolution, set internal resolution, and set text screen
set_screen:     push    es              ; Save es segment pointer
                les     bx, 6 [bp]      ; Get address of return variable
                mov     es:[bx], ax     ; Put Y resolution onto stack
                mov     Y_MAXIMUM, ax   ; Set internal Y resolution
                pop     es              ; Restore value of es segment
                mov     ax, 0003        ; Set Text screen mode 3
                int     10h             ; Execute BIOS

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Screen_Setup    endp

;=========================================================================
;-------------------------  Main Buffer Functions  -----------------------
;=========================================================================

;-------------------------------------------------------------------------
;--- PUR     : Clears the buffer for the main screen. The attributes of
;---           the characters of the buffer can be specified.
;-------------------------------------------------------------------------

                public  Clear_Main_Buffer
Clear_Main_Buffer       proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                cld

; Get attributes from stack and set character for clearing buffer to space
                lea     di, 6 [bp]      ; Address of attribute variable
                mov     ah, es:[di]     ; Get attribute value
                mov     al, SPACE_CHAR  ; Get ASCII space character

; Set pointer to main buffer
                push    ds              ; Put data segment onto stack
                pop     es              ; Point segment to defined data
                mov     di, offset MAIN_BUFFER + 2 * TITLE_LINE_CHARACTERS
                mov     cx, MAIN_CHARACTERS - TITLE_LINE_CHARACTERS
                rep     stosw           ; Clear whole buffer

; Return from procedure
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Clear_Main_Buffer       endp

;-------------------------------------------------------------------------
;--- PURPOSE : Clears the buffer for the main screen. The attributes of
;---           the characters of the buffer can be specified.
;-------------------------------------------------------------------------

                public  Main_Buffer_Line
Main_Buffer_Line        proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                cld

; Get attributes from stack and set character for clearing buffer to space
                lea     di, 6 [bp]      ; Address of attribute variable
                mov     ah, es:[di]     ; Get attribute value
                mov     al, SPACE_CHAR  ; Get ASCII space character

; Set pointer to main buffer
                push    ds              ; Put data segment onto stack
                pop     es              ; Point segment to defined data
                mov     di, offset MAIN_BUFFER
                mov     cx, TITLE_LINE_CHARACTERS;No of line characters
                rep     stosw           ; Clear whole buffer

; Return from procedure
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Main_Buffer_Line        endp

;-------------------------------------------------------------------------
;--- PURPOSE : Copies the characters and their attributes from the main
;---           screen buffer to the screen. In other words, anything on top
;---           of the main screen disappears.
;-------------------------------------------------------------------------

                public  Main_Buffer_To_Screen
Main_Buffer_To_Screen   proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                mov     ax, SCREEN_BASE_MEMORY; Point seg to screen area
                call    MouseVisible    ; Hide mouse if active and visible
                cld
                mov     si, offset MAIN_BUFFER; Point SI to buffer
                xor     di, di          ; Point DI at start of screen
                mov     cx, MAIN_CHARACTERS; Count for no of characters
                rep     movsw           ; Copy buffer to screen

; Return from procedure
                call    MouseVisibleAgain; Mouse active and was visible?
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret                     ; Return
Main_Buffer_To_Screen   endp

;-------------------------------------------------------------------------
;--- PURPOSE : Copies a string to the buffer of the main screen. The
;---           attributes of the string characters can be set to either of
;---           two specified sets. The character ~ is used as a special
;---           selector character (this can be changed very easily) which
;---           will switch the attributes to the second set if the first
;---           are active, or back to the first if the second are active.
;---           The location for the string in the buffer is set by passing
;---           coordinates. If the coordinates are out of range, the string
;---           is not written to the buffer. If the string is longer than
;---           the text line of the main screen, it is wrapped around.
;---           If the string is longer than what can be accomodated, even
;---           after wrapping, it is truncated at the end of the main
;---           buffer.
;-------------------------------------------------------------------------

                public  String_To_Main_Buffer
String_To_Main_Buffer   proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    ds              ; Save segment pointer to Ada data
                push    es              ; Save extra segment
                push    ds              ; Save segment pointer to Ada data

; Determine pointer to main buffer
                mov     di, offset MAIN_BUFFER
                mov     ax, 80 * 2      ; Load ax with size of each row
                mov     dl, 10 [bp]     ; Get row from stack
                dec     dl              ; Adjust rows from Ada coordinates
                mul     dl              ; Calculate row offset
                xor     dx, dx          ; Clear dx
                mov     dl, 12 [bp]     ; Get column from stack
                dec     dl              ; Adjust columns from Ada coord
                shl     dx, 1           ; Multiply 2 (include attributes)
                add     ax, dx          ; Add column to row offset
                add     di, ax          ; Add offset for screen pointer
                push    di              ; Save screen pointer

; Get attributes from stack
                lea     di, 16 [bp]     ; Address of attribute variable
                mov     ah, es:[di]     ; Get value of attribute
                mov     ATTRIBUTE, ah   ; Save 2nd type of attributes
                les     di, 14 [bp]     ; Address of attribute variable
                mov     ah, es:[di]     ; Get value of attribute

; Get pointer to string from stack
                les     di, 6 [bp]      ; Get address of string pointer
                lds     si, es: [di]    ; Get value of pointer from stack

; Determine string length
                mov     cx, es: 8 [di]  ; Get upper bound of string
                sub     cx, es: 4 [di]  ; Subtract lower bound of string
                inc     cx              ; Adjust string length

; Write line to buffer
                pop     di              ; Restore pointer to main buffer
                pop     es              ; Point segment to defined data

next_main_character:
                lodsb                   ; Character from string into al
                cmp     al, '~'         ; Is character selector character
                jne     not_special     ; If not, store only
                dec     cx              ; Actual string 1 character
shorter
                mov     al, ah          ; Move attribute to al
                mov     ah, es:ATTRIBUTE; Get 2nd type of attribute
                mov     es:ATTRIBUTE, al; Save 1st type of attribute
                lodsb                   ; Move next character from string
not_special:    stosw                   ; Attribute, character into buffer
                cmp     di, offset MAIN_BUFFER + MAIN_END
                                        ; Pointer at end of main buffer ?
                jge     ending          ; If so, exit
                loop    next_main_character; If more characters, get them

; Return from procedure
ending:         pop     es              ; Restore extra segment
                pop     ds              ; Restore seg pointer to Ada data
                pop     bp              ; Restore the frame pointer
                ret     16              ; Return and deallocate 2 near
                                        ;    3 far parameters
String_To_Main_Buffer   endp
```

```
;-----------------------------------------------------------------------------
;---------------------------- Window Buffer Functions ------------------------
;-----------------------------------------------------------------------------

;-----------------------------------------------------------------------------
;-- PURPOSE : Calculates the top left corner  where the pop-up screen will
;--           be displayed.  The  calculated value is returned  in bx  and
;--           is the offset that is added to the base  of screen memory to
;--           get the required location on the screen.
;-- RETURNS WIDTH : BX = screen offset
;-----------------------------------------------------------------------------

Determine_Left_Corner   proc
            mov     bl, SCREEN_TOP_ROW; Get screen top row
            mov     ax, 2 * 80      ; Get number of bytes for each row
            mul     bl              ; Calculate top row offset
            xor     bx, bx          ; Clear bx
            mov     bl, SCREEN_LEFT_COLUMN; Get screen left column
            shl     bx, 1           ; Multiply 2 (include attributes)
            add     bx, ax          ; Add column offset
            ret
Determine_Left_Corner   endp

;-----------------------------------------------------------------------------
;-- PURPOSE : Copies a character with it's  attribute a number of times to
;--           the Window buffer.  The  actual location  for the string  in
;--           the buffer is set by passing coordinates. If the coordinates
;--           are out of range,  the  characters  are  not written  to the
;--           buffer.  If the number of characters is longer than the text
;--           line  of  the  pop-up  screen,  it is  wrapped  around.  The
;--           characters are kept within the  text area.  If the number of
;--           characters is more than what can be accomodated  by the text
;--           area, even after wrapping, it is truncated at the or' of the
;--           text area.
;-----------------------------------------------------------------------------

            public  Character_To_Window_Buffer
Character_To_Window_Buffer      proc    far
            push    bp              ; Save old stack frame
            mov     bp, sp          ; Set up stack f
            push    es              ; Save extra seg

; Check if coordinates are within acceptable range
            mov     ax, 10 [bp]     ; Get row from stack
            add     ax, 2           ; Convert inner display to buffer
            cmp     al, SIZE_ROWS   ; If Y > Ymaximum
            jg      ending3         ;   then exit
            mov     ax, 12 [bp]     ; Get column from stack
            add     ax, 4           ; Convert inner display to buffer
            cmp     al, SIZE_COLUMNS; If X > Xmaximum
            jg      ending3         ;   then exit

; Determine pointers for start and last possible character for buffer
            mov     di, offset WINDOW_BUFFER
            mov     bx, di          ; Copy offset to WINDOW_BUFFER
            add     bx, BUFFER_BOTTOM_RIGHT
            xor     ax, ax          ; Clear ax
            mov     al, SIZE_COLUMNS; Load al with size of each row
            shl     ax, 1           ; Multiply 2 (include attributes)
            mov     dl, 10 [bp]     ; Get row from stack
            mul     dl              ; Calculate offset
            xor     dx, dx          ; Clear dx
            mov     dl, 12 [bp]     ; Get column from
            inc     dl              ; Adjust column f..       ord
            shl     dx, 1           ; Multiply 2 (incl)       butes;
            add     ax, dx          ; Add column to r
            add     di, ax          ; Add offset for buffer pointer
            push    ds              ; Save segment pointer to Ada data
            push    di              ; Save buffer pointer

; Get the number of times to repeat the character from stack
            mov     cx, 8 [bp]      ; Get count from stack

; Determine the display length for the first line
            mov     dl, SIZE_COLUMNS; Get length of total display line
            sub     dl, 4           ; Adjust length for inside 'box'
            mov     dh, dl          ; Length of total display line
            sub     dl, 12 [bp]     ; Adjust length of first line
            inc     dl              ; Adjust length for Ada coord

; Get attributes from stack
            les     di, 14 [bp]     ; Address of attribute variable
            mov     ah, es:[di]     ; Get value of attribute

; Get the ASCII character code from stack
            mov     al, 6 [bp]      ; ASCII character code from stack

; Write line to buffer
            pop     di              ; Restore buffer pointer
            pop     es              ; Point segment to defined data

next_character5:stosw               ; Attribute, character into buffer
            dec     dl              ; Decrement remain c splay length
            jnz     same_line3      ; If more characters, get them
            mov     dl, dh          ; Get length of total display line
            add     di, 4 * 2       ; Skip of borders of 'box'
same_line3: cmp     di, bx          ; Is pointer at end of box ?
            jge     ending3         ; If so, exit
            loop    next_character5 ; If more characters, get them

; Return from procedure
ending3:    pop     es              ; Restore extra segment
            pop     bp              ; Restore the frame pointer
            ret     12              ; Return and deallocate 4 near
                                    ;   1 far parameters
Character_To_Window_Buffer      endp

;-----------------------------------------------------------------------------
;-- PURPOSE : Clears the buffer for the  pop-up screen.  The attributes of
;--           the characters  of the buffer can be specified.  The size of
;--           the buffer was  set by  Buffersetup,  and only  this area is
;--           cleared.
;-----------------------------------------------------------------------------

            public  Clear_Window_Buffer
Clear_Window_Buffer     proc    far
            push    bp              ; Save old stack frame
            mov     bp, sp          ; Set up stack frame
            push    es              ; Save extra segment
            cld
```

```
; Get attributes from stack and set character for clearing buffer to space
            les     di, 6 [bp]      ; Address of attribute variable
            mov     ah, es:[di]     ; Get attribute value
            mov     al, SPACE_CHAR  ; Get ASCII space character

; Set pointer to Window buffer
            push    ds              ; Put data segment onto stack
            pop     es              ; Point segment to defined data
            mov     di, offset WINDOW_BUFFER

            xor     cx, cx          ; Set counter to zero
            mov     dh, SIZE_ROWS   ; Get number of rows of buffer
next_row:   mov     cl, SIZE_COLUMNS; Get number of columns of buffer
            rep     stosw           ; Clear line
            dec     dh              ; Has the last line been reached ?
            jns     next_row        ; If not, get next line

; Return from procedure
            pop     es              ; Restore extra segment
            pop     bp              ; Restore the frame pointer
            ret     4               ; Return and deallocate 1 far
                                    ;   parameter
Clear_Window_Buffer     endp

;-----------------------------------------------------------------------------
;-- PURPOSE : Frames the pop-up screen. A string of 7 characters is passed
;--           to the procedure  which are the  graphic characters  used to
;--           border the pop-up screen. They are :
;--               Top left corner
;--               Top side
;--               Top right corner
;--               Left and right side
;--               Bottom left corner
;--               Bottom side
;--               Bottom right side
;--           The size of the frame  is  determined  by  the  size  of the
;--           buffer.  The top  and botto  of the frame is next to the top
;--           and bottom of the buffer.  The left  and right  sides of the
;--           frame is one column spaced from the left  and right sides of
;--           the buffer.  The attributes of the frame characters can also
;--           be set.
;-----------------------------------------------------------------------------

            public  Frame_Window_Buffer
Frame_Window_Buffer     proc    far
            push    bp              ; Save old stack frame
            mov     bp, sp          ; Set up stack frame
            push    ds              ; Save segment pointer to Ada data
            push    es              ; Save extra segment

; Adjust the rows and columns for a frame which is smaller than the buffer
            mov     dh, SIZE_ROWS   ; Get number of rows
            sub     dh, 2           ; Adjust rows
            mov     dl, SIZE_COLUMNS; Get number of columns
            sub     dl, 4           ; Adjust columns

; Get attributes from stack
            les     di, 10 [bp]     ; Address of attribute variable
            mov     ah, es:[di]     ; Get value of attribute

; Get pointer to string from stack
            push    ds              ; Save segment pointer to Ada data
            lds     si, 6 [bp]      ; Get pointer of string from stack

; Get pointer to Window buffer
            pop     es              ; Point segment to defined data
            mov     di, offset WINDOW_BUFFER + 2

; Draw frame in buffer
            xor     cx, cx          ; Set counter to zero
            lodsb                   ; Move left top graphic into al
            stosw                   ; Write graphic to buffer
            lodsb                   ; Move horizontal graphic into al
            mov     cl, dl          ; Move number of columns
            rep     stosw           ; Write graphic to buffer
            lodsb                   ; Move right top graphic into al
            stosw                   ; Write graphic to buffer
            add     di, 2 * 2       ; Add in offset for left side
            push    di              ; Save for top division
            xor     bx, bx          ; Clear bx
            mov     bl, dl          ; Move number of columns
            shl     bx, 1           ; Calculate offset between insides
            lodsb                   ; Move vertical graphic into al
            mov     cl, dh          ; Move number of rows
next_side:  stosw                   ; Write graphic to buffer
            add     di, bx          ; Add in offset for right side
            stosw                   ; Write graphic to buffer
            add     di, 2 * 2       ; Add in offset for left side
            loop    next_side       ; Loop if vertcl side not complete
            push    di              ; Save position - bottom division
            lodsb                   ; Move left bottom graphic into al
            stosw                   ; Write graphic to buffer
            lodsb                   ; Move horizontal graphic into al
            mov     cl, dl          ; Move number of columns
            rep     stosw           ; Write graphic to buffer
            lodsb                   ; Right bottom graphic into al
            stosw                   ; Write graphic to buffer
            pop     di              ; Restore pointer to second row
            sub     di, bx          ; Subtract offset for bottom part
            sub     di, bx          ; Subtract offset for bottom part
            sub     di, 8 * 2       ; Subtract offset for bottom part
            lodsb                   ; Move left start division graphic
            stosw                   ; Write graphic to buffer
            lodsb                   ; Move horizontal graphic into al
            mov     cl, dl          ; Move number of columns
            rep     stosw           ; Write graphic to buffer
            lodsb                   ; Move right end division graphic
            stosw                   ; Write graphic to buffer
            sub     si, 3           ; Reset start of division graphics
            pop     di              ; Restore pointer to second row
            add     di, bx          ; Add in offset for top part
            add     di, 4 * 2       ; Add in offset for top part
            lodsb                   ; Move left start division graphic
            stosw                   ; Write graphic to buffer
            lodsb                   ; Move horizontal graphic into al
            mov     cl, dl          ; Move number of columns
            rep     stosw           ; Write graphic to buffer
            lodsb                   ; Move right end division graphic
            stosw                   ; Write graphic to buffer
```

```
; Return from procedure
                pop     es              ; Restore extra segment
                pop     ds              ; Restore seg pointer to Ada data
                pop     bp              ; Restore the frame pointer
                ret     8               ; Return and deallocate 2 far
                                        ;    parameters
Frame_Window_Buffer     endp

;-------------------------------------------------------------------------
;-- PURPOSE : Draws a line on the screen by changing the attributes of
;--           characters to the specified attribute value.  The attributes
;--           The attributes of the characters that are changed are first
;--           stored in a line buffer.
;-------------------------------------------------------------------------

                public  Make_Line_Bar
Make_Line_Bar   proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                call    MouseVisible    ; Hide mouse if active and visible
                cld

; Get the start row, column and length for the screen line from stack
                mov     cx, 10 [bp]     ; Get length of line from stack
                mov     LINE_LENGTH, cl ; Store length of line
                mov     ax, 80 * 2      ; Load ax with size of each row
                mov     dx, 6 [bp]      ; Get start row of line from stack
                dec     dx              ; Adjust row from Ada coordinates
                mul     dl              ; Calculate row offset
                xor     dx, dx          ; Clear dx
                mov     dx, 8 [bp]      ; Start column of line from stack
                dec     dx              ; Adjust column from Ada coord
                shl     dx, 1           ; Multiply 2 (include attributes)
                add     ax, dx          ; Add column to row offset
                mov     si, ax          ; Add offset for screen pointer
                inc     si              ; Add offset for attributes
                mov     LINE_START, si  ; Save start pointer of line

; Get attributes from stack
                les     di, 12 [bp]     ; Address of attribute variable
                mov     al, es:[di]     ; Get value of attribute

; Set up ds and es registers
                push    ds              ; Save segment pointer to data
                push    ds              ; Save segment pointer to data
                pop     es              ; Point segment to defined data
                mov     ds, SCREEN_BASE_MEMORY; Point to screen area
                mov     di, offset LINE_BUFFER

; Store attributes which the line will overwrite
more_lines:     movsb                   ; Copy attribute to line buffer
                inc     si              ; Skip over character location
                loop    more_lines      ; If more attributes, get them
                pop     ds              ; Point segment to defined data
                mov     es, SCREEN_BASE_MEMORY; Point to screen area

; Draw the line
                mov     cx, 10 [bp]     ; Get length of line from stack
                mov     di, LINE_START  ; Get start pointer of line
more_line2:     stosb                   ; Write line attribute to screen
                inc     di              ; Skip screen character location
                loop    more_line2      ; Loop if line not complete

; Return from procedure
                call    MouseVisibleAgain; Mouse active and was visible?
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret     10              ; Return and deallocate 3 wwer and
                                        ;    1 far parameter
Make_Line_Bar   endp

;-------------------------------------------------------------------------
;-- PURPOSE : Draws a shadow for the pop-up screen by changing the
;--           attributes of characters to the right (2 columns) and bottom
;--           (1 row) to light gray on black.  The size of the shadows is
;--           determined from the size of the buffer for the pop-up
;--           screen.  The attributes of the characters that are changed
;--           are first stored in a shadow buffer.
;-------------------------------------------------------------------------

SHADOW_ATT      equ     07H             ; Light gray on black

                public  Make_Shadow
Make_Shadow     proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                call    MouseVisible    ; Hide mouse if active and visible
                cld

; Determine start and length of right side shadow
                call    Determine_Left_Corner
                mov     di, bx
                add     di, 80 * 2 + 1  ; Adjust for attribute 1 row down
                xor     cx, cx          ; Clear cx
                mov     cl, SIZE_ROWS   ; Get number of rows for shadow
                xor     dx, dx          ; Clear dx
                mov     dl, SIZE_COLUMNS; Get number of columns
                mov     bl, dl          ; Save number of columns
                shl     dx, 1           ; Multiply 2 (include attributes)
                add     dl, dx          ; Add column offset
                mov     es, SCREEN_BASE_MEMORY; Segment to screen area

; Draw the right side shadow
                mov     al, SHADOW_ATT  ; Get shadow attribute
right_shadow2:  stosb                   ; Write shadow to screen
                inc     di              ; Skip screen character location
                stosb                   ; Write shadow to screen
                add     di, 78*2+1      ; Add offset for vertical shadow
                loop    right_shadow2   ; Loop if more right shadow

; Draw the bottom shadow
                sub     dl, dx          ; Subtract overshoot value
                sub     dl, 78 * 2      ; Subtract (1 row + 2 columns)
                mov     cl, bl          ; Move number of columns
                sub     cl, 2           ; Adjust 2 columns already stored
bottom_shadow2: stosb                   ; Write shadow to screen
                inc     di              ; Skip screen character location
                loop    bottom_shadow2  ; Loop if bottom not complete

; Return from procedure
                call    MouseVisibleAgain; Mouse active and was visible?
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret                     ; Return
Make_Shadow     endp

;-------------------------------------------------------------------------
;-- PURPOSE : Gets the new location where the window is going to be drawn.
;-------------------------------------------------------------------------

                public  New_Window_Location
New_Window_Location     proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get top left corner where window buffer will be copied
                mov     ax, 6 [bp]      ; Get number of rows from stack
                dec     al              ; Adjust rows from Ada coordinates
                mov     SCREEN_TOP_ROW, al; Store number of rows
                mov     ax, 8 [bp]      ; Get number of columns from stack
                dec     al              ; Adjust columns from Ada coord
                mov     SCREEN_LEFT_COLUMN, al; Store number of columns

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 2 near
                                        ;    parameters
New_Window_Location     endp

;-------------------------------------------------------------------------
;-- PURPOSE : Clears the line from the screen by copying the attributes
;--           from the line buffer back to the screen characters affected
;--           by the line.
;-------------------------------------------------------------------------

                public  Remove_Line_Bar
Remove_Line_Bar proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                mov     es, SCREEN_BASE_MEMORY; Segment to screen area
                call    MouseVisible    ; Hide mouse if active and visible
                cld

; Determine start and length of line
                mov     di, LINE_START  ; Get start pointer of line
                xor     cx, cx          ; Clear cx
                mov     al, LINE_LENGTH ; Get length of line
                mov     si, offset LINE_BUFFER

; Clear the line from the screen by restoring attributes
more_lines3:    movsb                   ; Restore screen attribute
                inc     di              ; Skip screen character location
                loop    more_lines3     ; Loop if more right shadow

; Return from procedure
                call    MouseVisibleAgain; Mouse active and was visible?
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret                     ; Return
Remove_Line_Bar endp

;-------------------------------------------------------------------------
;-- PURPOSE : Copies a string to the buffer of the pop-up screen.  The
;--           attributes of the string characters can be set to either of
;--           two specified sets.  The character ~ is used as a special
;--           selector character (this can be changed very easily) which
;--           will switch the attributes to the second set if the first
;--           are active, or back to the first if the second are active.
;--           The location for the string in the buffer is set by passing
;--           coordinates.  If the coordinates are out of range, the string
;--           is not written to the buffer.  If the string is longer than
;--           the text line of the pop-up screen, it is wrapped around.
;--           String characters are kept within the text area.  If the
;--           string is longer than what can be accomodated by the text
;--           area, even after wrapping, it is truncated at the end of the
;--           text area.
;-------------------------------------------------------------------------

                public  String_To_Window_Buffer
String_To_Window_Buffer proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    ds              ; Save segment pointer to Ada data
                push    es              ; Save extra segment

; Check if coordinates are within acceptable range
                mov     ax, 10 [bp]     ; Get row from stack
                add     ax, 2           ; Convert inner display to buffer
                cmp     al, SIZE_ROWS   ; If Y <= Ymaximum
                jle     second_check    ;    then check second parameter
                jmp     ending2         ; Otherwise, exit
second_check:   mov     ax, 12 [bp]     ; Get column from stack
                add     ax, 4           ; Convert inner display to buffer
                cmp     al, SIZE_COLUMNS; If X > Xmaximum
                jg      ending2         ;    then exit

; Determine pointers for start and last possible character for buffer
                mov     di, offset WINDOW_BUFFER
                mov     bx, di          ; Copy offset to WINDOW_BUFFER
                add     bx, BUFFER_BOTTOM_RIGHT
                xor     ax, ax          ; Clear ax
                mov     al, SIZE_COLUMNS; Load al with size of each row
                shl     ax, 1           ; Multiply 2 (include attributes)
                mov     dl, 10 [bp]     ; Get row from stack
                mul     dl              ; Calculate offset of rows
                xor     dx, dx          ; Clear dx
                mov     dl, 12 [bp]     ; Get column from stack
                inc     dl              ; Adjust column from Ada coord
                shl     dx, 1           ; Multiply 2 (include attributes)
                add     ax, dx          ; Add column to row offset
                add     di, ax          ; Add offset for buffer pointer
                push    ds              ; Save segment pointer to Ada data
                push    di              ; Save buffer pointer

; Determine the display length for the first line
                mov     dl, SIZE_COLUMNS; Get length of total display line
                sub     dl, 4           ; Adjust length for inside 'box'
                mov     dh, 11          ; Length of total display line
```

```
                sub     dl, 12 [bp]     ; Adjust length of first line
                inc     dl              ; Adjust length for Ada count

; Get attributes from stack
                lea     di, 18 [bp]     ; Address of attribute variable
                mov     ah, es:[di]     ; Get value of attribute
                mov     ATTRIBUTE, ah   ; Save 2nd type of attributes
                lea     di, 14 [bp]     ; Address of attribute variable
                mov     ah, es:[di]     ; Get value of attribute

; Get pointer to string from stack
                lea     di, 6 [bp]      ; Get address of string pointer
                lds     si, es:[di]     ; Get value of pointer from stack

; Determine string length
                mov     cx, es: 6 [di]  ; Get upper bound of string
                sub     cx, es: 4 [di]  ; Subtract lower bound of string
                inc     cx              ; Adjust string length

; Write line to buffer
                pop     di              ; Restore buffer pointer
                pop     es              ; Point segment to defined data

next_character: lodsb                   ; Character from string into al
                cmp     al, '^'         ; Is character selector character
                jnz     not_special2    ; If not, store only
                dec     cx              ; Actual string 1 character less
                mov     al, ah          ; Move attribute to al
                mov     ah, es:ATTRIBUTE; Get 2nd type of Attribute
                mov     es:ATTRIBUTE, al; Save 1st type of attribute
                lodsb                   ; Move next character from string
not_special1:   stosw                   ; Attribute, character into buffer
                dec     dl              ; Decrement remain display length
                jnz     same_line       ; If more characters, get them
                mov     dl, dh          ; Get length of total display line
                add     di, 4 * 2       ; Skip of borders of 'box'
same_line:      cmp     di, bx          ; Is pointer at end of box ?
                jge     ending2         ; If so, exit
                loop    next_character  ; If more characters, get them

; Return from procedure
ending2:        pop     es              ; Restore extra segment
                pop     ds              ; Restore seg pointer to Ada data
                pop     bp              ; Restore the frame pointer
                ret     14              ; Return and deallocate 2 near
                                        ;    7 for parameters
String_To_Window_Buffer endp

;-----------------------------------------------------------------------
;-- PURPOSE : Sets up the size of the pop-up screen that will be displayed
;--           on the screen. The width is in columns and  the height is in
;--           rows.  Note that this is the  full size that  will appear on
;--           the screen. The size of the area for text display within the
;--           pop-up screen is also calculated.
;-----------------------------------------------------------------------

                public  Window_Buffer_Setup
Window_Buffer_Setup     proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get the number of rows and columns for the buffered screen from stack
                mov     ax, 6 [bp]      ; Get number of rows from stack
                mov     SIZE_ROWS, al   ; Store number of rows
                mov     ax, 8 [bp]      ; Get number of columns from stack
                mov     SIZE_COLUMNS, al; Store number of columns

; Determine value of pointer for bottom right corner of display in buffer
                mov     bl, SIZE_ROWS   ; Get number of rows
                sub     bl, 2           ; Adjust for inner display area
                xor     ax, ax          ; Clear ax
                mov     al, SIZE_COLUMNS; Get number of columns
                shl     ax, 1           ; Multiply 2 (include attributes)
                mul     bl              ; Top row offset for buffer
                xor     bx, bx          ; Clear bx
                mov     bl, SIZE_COLUMNS; Get number of columns
                sub     bx, 2           ; Adjust for inner display area
                shl     bx, 1           ; Multiply 2 (include attributes)
                add     ax, bx          ; Add column offset-inner display
                mov     BUFFER_BOTTOM_RIGHT, ax; Store offset for buffer

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 2 near
                                        ;    parameters
Window_Buffer_Setup     endp

;-----------------------------------------------------------------------
;-- PURPOSE : Copies a string  to the  buffer  of the  pop-up screen.  The
;--           attributes of the  string characters can be set.
;-----------------------------------------------------------------------

                public  Window_Buffer_Title
Window_Buffer_Title     proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    ds              ; Save segment pointer to Ada data
                push    es              ; Save extra segment
                push    ds              ; Save segment pointer to Ada data

; Get attributes from stack
                lea     di, 10 [bp]     ; Address of attribute variable
                mov     ah, es: [di]    ; Get value of attribute

; Get pointer to string from stack
                lea     di, 6 [bp]      ; Get address of string pointer
                lds     si, es: [di]    ; Get value of pointer from stack

; Determine string length
                mov     cx, es: 6 [di]  ; Get upper bound of string
                sub     cx, es: 4 [di]  ; Subtract lower bound of string
                inc     cx              ; Adjust string length

; Get width of buffer
                pop     es              ; Point segment to defined data
                mov     dl, es: SIZE_COLUMNS; Get number of columns
                sub     dl, 4           ; Subtract side border columns
                sub     dl, cl          ; Subtract length of string
                cmp     dl, 2           ; String longer than title area ?
                jl      no_title        ; If so, exit
                and     dl, 0feh        ; Mask out lsb
                xor     dh, dh          ; Set dl to 0
                mov     di, offset Window_Buffer + 2
                add     di, dx          ; Determine start of title

; Write left holder of title to screen
                mov     al, '['         ; Move left title holder into al
                stosb                   ; Left title holder into buffer
                inc     di              ; Skip over attribute

next_buffl_char:lodsb                   ; Character from string into al
                stosw                   ; Attribute, character into buffer
                loop    next_buffl_char ; If more characters, get them

; Write right holder of title to screen
                mov     al, ']'         ; Move right title holder into al
                stosb                   ; Right title holder into buffer

; Return from procedure
no_title:       pop     es              ; Restore extra segment
                pop     ds              ; Restore seg pointer to Ada data
                pop     bp              ; Restore the frame pointer
                ret     7               ; Return and deallocate 2 for
                                        ;    parameters
Window_Buffer_Title     endp

;-----------------------------------------------------------------------
;-- PURPOSE : Copies  the characters  and their attributes from the pop-up
;--           screen buffer  to the  screen.  In other words,  the  pop-up
;--           screen pops up.  The  number of characters copied depends on
;--           the size of the pop-up screen buffer set by BufferSetup.
;-----------------------------------------------------------------------

                public  Window_Buffer_To_Screen
Window_Buffer_To_Screen proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                mov     es, SCREEN_BASE_MEMORY; Point to screen area
                call    MouseVisible    ; Hide mouse if active and visible
                cld

; Determine the left corner and size of screen area to copy from buffer
                call    Determine_Left_Corner
                mov     dh, SIZE_ROWS   ; Get number of rows
                mov     dl, SIZE_COLUMNS; Get number of columns

; Characters from screen moved to buffer
                xor     ch, ch          ; set high byte of counter to zero
                mov     si, offset WINDOW_BUFFER; Point SI to buffer
Next_Line1:     mov     di, bx          ; Base address of screen block
                mov     cl, dl          ; Move number of columns into cl
                rep     movsw           ; Copy line from screen to buffer
                add     bx, 80 * 2      ; Update screen offset
                dec     dh              ; Has the last row been reached ?
                jnz     Next_Line1      ; If not, get next line

; Return from procedure
                call    MouseVisibleAgain; Mouse active and was visible?
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret                     ; Return
Window_Buffer_To_Screen endp

;-----------------------------------------------------------------------
;-----------------------  Direct Screen Functions  ---------------------
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
;-- PURPOSE : Displays ELECTOR and a copyright message on the screen. Each
;--           line of the display is written  to the screen.  The top part
;--           is scrolled down, the bottom part is scrolled up. There is a
;--           short  delay  before  the  next  two lines of the display are
;--           written.
;-----------------------------------------------------------------------

DELIMITER       equ     14d             ; End of string character (1)
BACK_ATTRIBUTE  equ     07h             ; Light gray on black colour
LOGO_ATTRIBUTE  equ     1Ah             ; Light green on cyan colour
LONGEST_STRING  equ     66              ; Length of longest logo string

                public  Main_Title
Main_Title      proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                mov     es, SCREEN_BASE_MEMORY; Point to screen area
                mov     LOGO_COUNT, 7   ; Set the number of lines for logo

; Make Window on screen
                mov     cx, 0407h       ; Set top left corner
                mov     dx, 164Ch       ; Set bottom right corner
                mov     bh, BACK_ATTRIBUTE; Set shadow attribute
                mov     ax, 0600h       ; Set ax to clear screen
                int     10h             ; Execute BIOS
                mov     cx, 0305h       ; Set top left corner
                mov     dx, 184Ah       ; Set bottom right corner
                mov     bh, LOGO_ATTRIBUTE; Set shadow attribute
                mov     ax, 0600h       ; Set ax to clear screen
                int     10h             ; Execute BIOS

; Scroll top part of logo down
next_logo_line: mov     cx, 0507h       ; Set top left corner
                mov     dx, 0B4Ah       ; Set bottom right corner
                mov     bh, LOGO_ATTRIBUTE; Set logo attribute
                mov     ax, 0701h       ; Set ax to scroll 1 line down
                int     10h             ; execute BIOS video function

; Determine string data for top title
                mov     cx, LONGEST_STRING; Length of longest string
                dec     LOGO_COUNT      ; Decrement number of logo lines
                mov     di, LOGO_COUNT  ; Count of lines of logo screen
                add     di, di          ; Calculate offset to logo string
                mov     si, DISPLAY1[di]; Get pointer to logo string
                mov     di, (5*80 +7)*2 ; Pointer to screen
                mov     ah, LOGO_ATTRIBUTE; Set ah to the title attribute

; Write string to screen
next_title_char:lodsb                   ; Character from string into al
```

```
                cmp     al, DELIMITER   ; Is character a delimiter
                je      no_more_char    ; If so, exit
                stosw                   ; Attribute, character into screen
                loop    next_title_char

; Scroll bottom part of logo up
                mov     cx, 0B0Dh       ; Set top left corner
                mov     dx, 143Dh       ; Set bottom right corner
                mov     bh, LOGO_ATTRIBUTE; Get logo attribute
                mov     ax, 0601h       ; Set ax to scroll 1 line up
                int     10h             ; Execute Bios video function

; Determine string data for bottom title
no_more_char:   mov     cx, LONGEST_STRING; Length of longest string
                mov     di, LOGO_COUNT  ; Count of lines of logo screen
                add     di, di          ; Calculate offset to logo string
                mov     si, DISPLAY2[di]; Get pointer to logo string
                mov     di, (20*80+12)*2; Pointer to screen
                mov     ah, LOGO_ATTRIBUTE; Set ah to the title attribute

; Write string to screen
next_title_char2:lodsb                  ; Character from string into al
                cmp     al, DELIMITER   ; Is character a delimiter?
                je      no_more_char2   ; If so, exit
                stosw                   ; Attribute, character into screen
                loop    next_title_char2

no_more_char2:  call    WaitSomeTime    ; Wait some time
                cmp     LOGO_COUNT, 0   ; Are all logo lines on screen ?
                jne     next_logo_line  ; If not, get another line

; Get a key from the keyboard before continuing
                xor     dx, ax          ; Set ah to read character which
                                        ;   waits for a keystroke
                int     16h             ; Execute BIOS
                mov     bh, 1Eh         ; Set attribute for yelow on blue
                xor     cx, cx          ; Set top row and left side
                mov     dx, 144Fh       ; Set bottom row and right side
                mov     ax, 0600h       ; Set ax to clear screen
                int     10h             ; Execute BIOS

; Return from procedure
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret                     ; Return
Main_Title      endp

;-----------------------------------------------------------------------
;-- PURPOSE : Waits for a certain amount of time.
;-----------------------------------------------------------------------

WaitSomeTime    proc
                mov     WAIT_COUNTER, 3 ; Set high number wait counter
                push    ax              ; Save value of ax
waiting1:       xor     ax, ax          ; Clear low number wait counter
waiting2:       dec     ax              ; Decrement low wait counter
                jne     waiting2        ; If low counter > 0, go back
                dec     WAIT_COUNTER    ; Decrement high wait counter
                jne     waiting1        ; If high counter > 0, go back
                pop     ax              ; Restore value of ax
                ret
WaitSomeTime    endp

;=======================================================================
;=============================  Tutorial Functions  ====================
;=======================================================================

;-----------------------------------------------------------------------
;-- PURPOSE : The part of the screen that was copied to the buffer is
;--           copied back to the screen.  The size of the buffer and area
;--           of screen to be written to is calculated from values set by
;--           the BufferSetup and ScreenToBuffer procedures.
;-----------------------------------------------------------------------

                public  Tutorial_To_Screen
Tutorial_To_Screen      proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame
                push    es              ; Save extra segment
                push    ds              ; Save segment pointer to Ada data

; Get pointer to string from stack
                mov     es, SCREEN_BASE_MEMORY; Point to screen area
                mov     di, (80 + 3)*2  ; Start of tut area on screen
                lds     si, 6 [bp]      ; Get value of pointer from stack
                cld

; Characters from tutorial array moved to screen
next_tut_line:  mov     cx, 74          ; Width of tutorial screen
                rep     movsw           ; Copy line from screen to buffer
                add     di, 6 * 2       ; Add offset for next line
                cmp     di, (24*80+3)*2 ; Whole tutorial screen done ?
                jl      next_tut_line   ; If not, Write another line

; Return from procedure
                pop     ds              ; Restore seg pointer to Ada data
                pop     es              ; Restore extra segment
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Tutorial_To_Screen      endp


;=======================================================================
;=============================  Disk Functions  ========================
;=======================================================================

;-----------------------------------------------------------------------
;-- PURPOSE  : Gets the default drive and directory. It updates the search
;--            strings for listing the files and returns the  default path
;--            in the passed string parameter.
;-- DOS FUNCTION (Get default drive)
;--     INTERRUPT : 21h
;--     FUNCTION : 19h
;--     CALLED WITH  : AH = 19h
;--     RETURNS WITH : AL containing logical drive 0=A, 1=B, etc
;--
;-- DOS FUNCTION (Get current directory)
;--     INTERRUPT : 21h
;--     FUNCTION : 47h
;--     CALLED WITH  : AH = 47h         DX = 0 (for default directory)
;--                    SI = Pointer to string for path
;--     RETURNS WITH : Path string filled.
;-----------------------------------------------------------------------

Get_Defaults    proc
                mov     ah, 19h         ; Set for default drive
                int     21h             ; Execute DOS function
                add     al, 41h         ; Convert number to character
                mov     PATH1, al       ; Change drive for path 1
                mov     PATH2, al       ; Change drive for path 2
                push    es              ; Save the extra segment value
                push    ds              ; Save data area pointer
                lds     si, 6 [bp]      ; Get address of string
                mov     [si], al        ; Store drive letter
                inc     si              ; Point to the next space
                mov     [si], ":\"      ; Store colon and slash
                inc     si              ; Point to the next space
                inc     si              ; Point to the next space

; Clear the string with spaces
                push    ds              ; Set es to point to the same area
                pop     es              ;   as ds
                mov     di, si          ; Move pointer to string
                mov     ax, 2020h       ; Set with spaces
                mov     cx, 32          ; Set length of string
                rep     stosw           ; Clear string with spaces

; Fill in the new default path
                mov     ah, 47h         ; Set for current directory
                xor     dx, dx          ; Set for default drive
                int     21h             ; Execute DOS function
                pop     ds              ; Restore data area pointer
                pop     es              ; Restore the extra segment value
                ret
Get_Defaults    endp

;-----------------------------------------------------------------------
;-- PURPOSE  : Sets a new drive and then returns the present defaults.
;-- DOS FUNCTION (Set new drive)
;--     INTERRUPT : 21h
;--     FUNCTION : 0Eh
;--     CALLED WITH  : AH = 0Eh         DL = drive value
;--     RETURNS WITH : -----
;-----------------------------------------------------------------------

                public  Change_Drive
Change_Drive    proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get new drive from parameters
                mov     dl, 10 [bp]     ; Get drive from stack
                and     dl, 00001111b   ; Mask out high bits
                dec     dl              ; Adjust drive number
                mov     ah, 0Eh         ; Set for changing drive
                int     21h             ; Execute DOS function
                call    Get_Defaults    ; Get present defaults

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     6               ; Return and deallocate 1 far
                                        ;    and 1 word parameters
Change_Drive    endp

;-----------------------------------------------------------------------
;-- PURPOSE  : Sets a new path and then returns the present defaults.
;-- DOS FUNCTION (Set new path)
;--     INTERRUPT : 21h
;--     FUNCTION : 3Bh
;--     CALLED WITH  : AH = 3Bh         DX = Points to string for new path
;--     RETURNS WITH : -----
;-----------------------------------------------------------------------

                public  Change_Directory
Change_Directory        proc    far
                push    bp              ; Save old stack frame
                mov     bp, sp          ; Set up stack frame

; Get new path from parameters
                push    ds              ; Save pointer to data area
                lds     dx, 6 [bp]      ; Get address of string from stack
                mov     ah, 3Bh         ; Set for changing path
                int     21h             ; Execute DOS function
                pop     ds              ; Restore pointer to data area
                call    Get_Defaults    ; Get present defaults

; Return from procedure
                pop     bp              ; Restore the frame pointer
                ret     4               ; Return and deallocate 1 far
                                        ;    parameter
Change_Directory        endp


;-----------------------------------------------------------------------
;-- PURPOSE  : Returns the filename found in the  PSP.  An error value is
;--            passed back which corresponds to the success of the
;--            previous search.
;-- DOS FUNCTION (Get address of PSP)
;--     INTERRUPT : 21h
;--     FUNCTION : 62h
;--     CALLED WITH  : AH = 62h
;--     RETURNS WITH : BX = address of PSP
;-----------------------------------------------------------------------

Get_Filename    proc
                push    es              ; Save extra segment value
                les     di, 6 [bp]      ; Get address of error variable
                mov     es:[di], ax     ; Return error value parameter
                mov     ah, 62h         ; Get address of PSP
                int     21h             ; Execute DOS function
                push    ds              ; Save pointer to data area
                mov     ds, bx          ; Move address into ds

; Clear the string with spaces
                les     di, 10 [bp]     ; Get address of filename variable
                push    di              ; Save offset pointer to string
                mov     ax, 2020h       ; Set with spaces
                mov     cx, 6           ; Set length of string
                rep     stosw           ; Clear string with spaces
```

```
                pop     di                      ; Restore offset pointer to string

; Copy name from PSP to string
                mov     si, 158                 ; Filename area of current DTA
                mov     cx, 12                  ; Set number of characters in name
next_filename:  lodsb                           ; Get filename character from PSP
                cmp     al, 0                   ; Has the end of the name being
                je      no_more                 ;   reached, if so exit
                stosb                           ; Store character in string
                loop    next_filename           ; Get another filename character
no_more:        pop     ds                      ; Restore pointer to data area
                pop     es                      ; Restore extra segment value
                ret
Get_Filename    endp

;-----------------------------------------------------------------------
;-- PURPOSE  : Finds the first matching file to  PATH1  or  PATH2 and then
;--            returns the filename. An error value is passed back also.
;-- DOS FUNCTION (Find first matching file)
;--    INTERRUPT : 21h
;--    FUNCTION : 4Eh
;--    CALLED WITH  : AH = 4Eh              DX = Points to string for matching
;--                   CX = Attributes (0 = Normal)
;--    RETURNS WITH : AX = error value (0 = no error)
;--                   Current DTA contains file info if successful
;-----------------------------------------------------------------------

                public  Find_First_File
Find_First_File proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get first matching file and return it
                mov     dx, offset path1;       Set path for Netlists
                cmp     14 [bp], word ptr 01; Is it for Monte ?
                jne     no_monte                ; If not, then it must be Netlist
                mov     dx, offset path2;       Set path for Monte
no_monte:       mov     ah, 4Eh                 ; Set for find first matching file
                xor     cx, cx                  ; Set for normal attributes
                int     21h                     ; Execute DOS function
                call    Get_Filename

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret     10                      ; Return and deallocate 2 far
                                                ;    and 1 near parameters
Find_First_File endp

;-----------------------------------------------------------------------
;-- PURPOSE  : Finds  the  next  matching  file after  first calling
;--            Find_First_File procedure. The filename and  an error value
;--            is passed back.
;-- DOS FUNCTION (Find next matching file)
;--    INTERRUPT : 21h
;--    FUNCTION : 4Fh
;--    CALLED WITH  : AH = 4Fh
;--    RETURNS WITH : AX = error value (0 = no error)
;--                   Current DTA contains file info if successful
;-----------------------------------------------------------------------

                public  Find_Next_File
Find_Next_File  proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get first matching file and return it
                mov     ah, 4Fh                 ; Set for find next matching file
                int     21h                     ; Execute DOS function
                call    Get_Filename

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret     0                       ; Return and deallocate 2 far
                                                ;    parameters
Find_Next_File  endp

;-----------------------------------------------------------------------
;-- PURPOSE  : Gets and returns the present drive and directory.
;-----------------------------------------------------------------------

                public  Get_Drive_Directory
Get_Drive_Directory     proc    far
                push    bp                      ; Save old stack frame
                mov     bp, sp                  ; Set up stack frame

; Get new drive from parameters
                call    Get_Defaults            ; Get present defaults

; Return from procedure
                pop     bp                      ; Restore the frame pointer
                ret     4                       ; Return and deallocate 1 far
                                                ;    parameter
Get_Drive_Directory     endp

;-----------------------------------------------------------------------

DOS_CODE        ENDS

END
```

User Reference Manual

Author : **Warren Buckle**
Revision : **1.00**
Date : **21 / 05 /93**

VOLUME 11

CONTENTS

CONTENTS ii

LIST OF FIGURES vi

LIST OF TABLES xiii

LIST OF EQUATIONS ix

CHAPTER 1 : INTRODUCTION 1

1.1 What is Elector Used For ? 1
1.2 History of Elector 1

CHAPTER 2 : CIRCUIT MODELLING 3

2.1 Introduction 3
2.2 Branches and Nodes 3
2.3 Branch Components 4
2.4 Mutual Inductance Circuits 4
2.5 Voltage Controlled Current Sources 5
2.6 Semiconductor Circuits 5
2.7 Method for Designing Circuits 6

CHAPTER 3 : THE USER INTERFACE 7

3.1 Introduction 7
3.2 Subscreens 7
3.3 Buttons 8
3.3.1 Purpose 8
3.3.2 Appearance and Behaviour 8
3.4 Boolean_Buttons 9
3.4.1 Purpose 9
3.4.2 Appearance and Behaviour 9
3.5 Input_Lines 9
3.5.1 Purpose 9
3.5.2 Appearance and Behaviour 9
3.6 Pick_Lists (Static) 10
3.6.1 Purpose 10

3.6.2 Appearance and Behaviour 11
3.7 Pick_Lists (Dynamic) 12
3.7.1 Purpose 12
3.7.2 Appearance and Behaviour 12
3.8 Forms 13
3.8.1 Purpose 13
3.8.2 Appearance and Behaviour 13
3.9 Windows 13
3.9.1 Purpose 13
3.9.2 Appearance and Behaviour 14
3.10 Spread Menu 14
3.10.1 Purpose 14
3.10.2 Appearance and Behaviour 15
3.11 Stacked Menu 15
3.11.1 Purpose 16
3.11.2 Appearance and Behaviour 16

CHAPTER 4 : MENU OPERATIONS 17

4.1 Menu Hierarchy 17
4.2 Main Menu 17
4.2.1 File 17
4.2.2 Edit 17
4.2.3 Nodes 17
4.2.4 Analyse 17
4.2.5 Tutorials 19
4.3 File Menu 19
4.3.1 Load 19
4.3.2 Save 19
4.3.3 Drive 20
4.3.4 Directory 20
4.3.5 Results 20
4.3.6 Quit 20
4.4 Edit 20
4.4.1 New Circuit 20
4.4.2 Title 21
4.4.3 Insert 21
4.4.4 Change Magnitude / Phase 22
4.4.5 List Components 22
4.4.6 Show Branch 22
4.4.7 Show Component 22

4.4.8 Draw Circuit 22
4.5 **Nodes** 23
4.5.1 Convert to Admittance 23
4.5.2 Convert to RLC 24
4.5.3 Renumber Node 24
4.5.4 Delete Node 24
4.6 **Analyse** 24
4.6.1 Define Nodes 24
4.6.2 Show Nodes 25
4.6.3 Calculate Admittance 25
4.6.4 Node Voltages 25
4.6.5 Branch Currents 26
4.6.6 Power 26
4.6.7 Circuit Response 26
4.6.8 Fourier 27
4.6.9 Phasor 27
4.6.10 Polar 27
4.6.11 Sensitivity 27
4.6.12 Monte-Carlo 28
4.6.13 Optimisation 28
4.7 **Monte Carlo** 28
4.7.1 Recover 28
4.7.2 Save Monte 29
4.7.3 Clear Monte 29
4.7.4 Insert Monte 29
4.7.5 Show Components 30
4.7.6 Perform 30
4.7.7 Test Random 31
4.7.8 Worst Case 31
4.7.9 Worst Case 2 31
4.8 **Optimisation** 32
4.8.1 Clear Variables 32
4.8.2 Insert Variables 33
4.8.3 Show Components 33
4.8.4 Bandwidth 33
4.8.5 Maximise 34
4.8.6 Minimise 34
4.8.7 Set 34

CHAPTER 5 : TUTORIAL EXAMPLE 35

5.1 Introduction 35
5.2 Starting Elector 35
5.3 Loading a Circuit File 35
5.4 Listing the Circuit Components 36
5.5 Viewing the Circuit 38
5.6 Viewing the Nodes to be Analyzed 38
5.7 Calculation of the Node Voltages 38
5.8 Calculation of Branch Currents 39
5.9 Calculation of Absorbed Power 40
5.10 Changing the Nodes to be Analyzed 41
5.11 Time Domain Circuit Response 41
5.12 Frequency Domain Circuit Response 42
5.13 Effect of Component Sensitivities 43
5.14 Monte Carlo Analysis 43
5.15 Finding the Bandwidth 45
5.16 Help 46
5.17 Tutorials 47

REFERENCES 48

LIST OF FIGURES

Figure 2-1 Branch Construction 3
Figure 2-2 Two Mutually Coupled Inductors 4
Figure 2-3 Hybrid Pi Model of a Transistor 5
Figure 2-4 Equivalent Model for an Op-Amp 6
Figure 3-1 The Subscreen Object 7
Figure 3-2 The Button Object 8
Figure 3-3 Boolean_Button Object 9
Figure 3-4 The Input_Line Object 9
Figure 3-5 The Static Pick_List Object 11
Figure 3-6 The Dynamic Pick_List Object 12
Figure 3-7 The Form Object 13
Figure 3-8 The Window Object 14
Figure 3-9 The Spread Menu Object 14
Figure 3-10 The Stacked Menu Object 15
Figure 4-1 Main Menu 17
Figure 4-2 Menu Hierarchy 18
Figure 4-3 File Menu 19
Figure 4-4 Edit Menu 21
Figure 4-5 Nodes Menu 23
Figure 4-6 Analyse Menu 25
Figure 4-7 Monte-Carlo Menu 29
Figure 4-8 Optimise Menu 32
Figure 5-1 Single Stage Common Emitter Amplifier 35
Figure 5-2 Opening Screen for Elector 36
Figure 5-3 Loading a Circuit 36
Figure 5-4 Listing the Circuit Components 37
Figure 5-5 List of Components in a Circuit Branch 37
Figure 5-6 Viewing the Circuit 38
Figure 5-7 Viewing the Nodes to be Analyzed 39
Figure 5-8 Calculation of the Node Voltages 39
Figure 5-9 Calculation of Branch Currents 40
Figure 5-10 Calculation of Absorbed Power 40
Figure 5-11 Changing the Nodes to be Analyzed 41
Figure 5-12 Square Wave Time Domain Circuit Response 42
Figure 5-13 Frequency Domain Circuit Response 42
Figure 5-14 Effect of Component Sensitivities 43
Figure 5-15 Monte-Carlo Analysis 44
Figure 5-16 Monte-Carlo Analysis 44
Figure 5-17 Monte-Carlo Bar Plot 45
Figure 5-18 Monte-Carlo Worst Case Plot 45
Figure 5-19 Finding the Bandwidth 46

Figure 5-20 Help Screen 46
Figure 5-21 Tutorial Screen 47

LIST OF TABLES

Table 3-1	Purpose of the Subscreen Buttons	8
Table 3-2	Input_Line Key Definition	10
Table 3-3	Pick_List Characteristics	11
Table 3-4	Pick_List (Static) Key Definition	11
Table 3-5	Form Key Definition	13
Table 3-6	Spread Menu Key Definition	15
Table 3-7	Stack Menu Key Definition	16
Table 4-1	Circuit Drawing Key Definition	23

LIST OF EQUATIONS

Equation 2-1	Mutual Inductance	4
Equation 2-2	Transconductance	5
Equation 4-1	Sensitivity	28

CHAPTER 1

INTRODUCTION

1.1 What is Elector Used For ?

Elector is designed to be a teaching aid in the area of linear circuit analysis. It allows you a visual means to analyse, simulate and optimise a linear circuit. Approximate models of non-linear components can be used for the entry of components such as transistors, op-amps etc. The simulator allows you to ask "What if...?" and to experiment with your design. Some of the features include :

- Creating, editing and drawing the schematic diagram.
- Calculating the node voltages and the branch currents.
- Calculating the power absorbed by passive components.
- Measuring the sensitivity of node voltages to perturbations in a component value.
- Plotting the frequency response of the circuit.
- Analyzing the steady state time domain using Fourier theory.
- Plotting polar and phasor plots.
- Monte-Carlo statistical analysis.

Elecsim (the pascal version) was given to second and third year undergraduate electrical engineering students at the University of the Witwatersrand [SEAL, 1990].

1.2 History of Elector

The development history of Elector can be broken down into four main phases.

Phase 1

- LINCAD was first released in 1974 by Prof W.H. Cornetet and Prof F.E. Battocletti.
- LINCAD was used on an IBM 370 mainframe computer in a time sharing environment and had no graphics.
- It consisted of 936 lines of FORTRAN source code in 7 linkable modules.

Phase 2

- LINSIM was build around LINCAD, supporting mutual inductances, Monte-Carlo analysis, bandwidth determination and power computation.

- It was build by Dr L.N.M Edward over a 14 year period.
- It was used on a DEC VAX 11-70 using the VMS operating system and also had no graphics.
- It consisted of 7986 lines of FORTRAN.

Phase 3

- ELECSIM resulted from renewing LINSIM. The renewal process needed a change of implementation language (FORTRAN to Pascal) and restructuring along the lines of object oriented design.
- It consisted of 11054 lines of Turbo Pascal code and supported graphics.
- It was used on a PC computer running DOS 3.x

Phase 4

- ELECTOR resulted from renewing ELECSIM. The renewal process needed a change of implementation language (Pascal to Ada), restructuring to enforce objects with abstract data types and greater use of object-oriented design.
- The user interface was redesigned along Common User Access lines.

CHAPTER 2

CIRCUIT MODELLING

2.1 Introduction

Circuits that are entered into Elector need to be modelled in a certain manner. The following points will help and guide the modelling process.

2.2 Branches and Nodes

- A circuit consists of a series of interconnected branches.
- Each branch is associated with a single node pair.
- The number of branches is limited to 2850 or the maximum available memory, which ever runs out first. However, there is a limit of 36 branches in circuits with mutually coupled inductors.
- Node numbers start from zero (reference node) and must increase consecutively.
- At the moment, Elector can only analyse circuits of up to 75 nodes. This does not restrict the usefulness of Elector. Decomposing complex systems into well defined modules enables large systems (consisting of more than 75 nodes) to be broken into simpler modules (each containing less than 75 nodes).

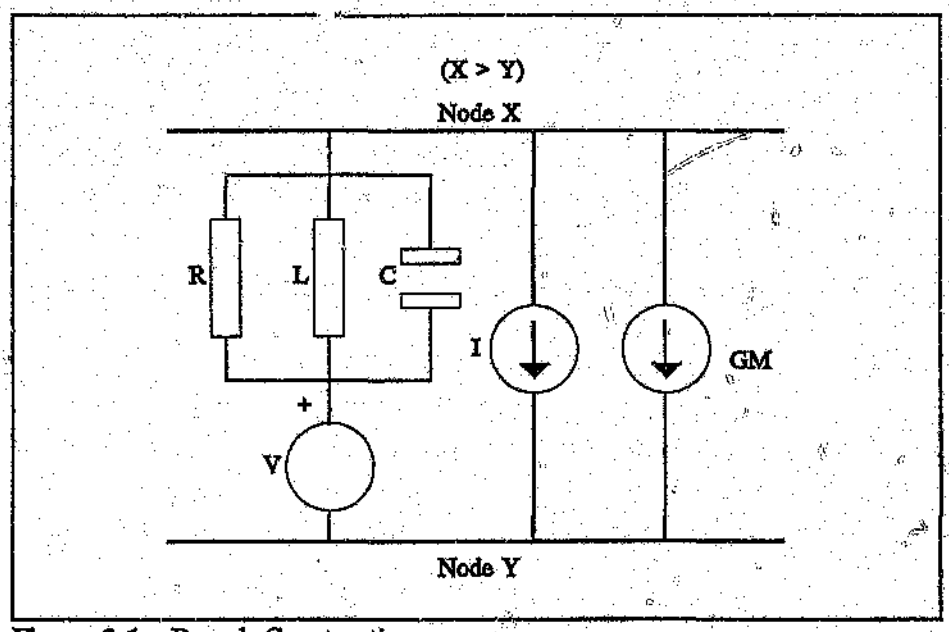


Figure 2-1 Branch Construction

2.3 Branch Components

- One component of each type (except for voltage controlled current sources) can be entered into a branch.
- A voltage source must have at least one capacitor (C), one inductor (L) or one resistor (R) in series with it.
- A current source defines independent current flow from node J to K.
- Components are assumed to be ideal. For simulations at very high frequencies (above 1 Mhz), components can become non-linear. Real resistances contain shunting capacitances and become inductive due to leads.

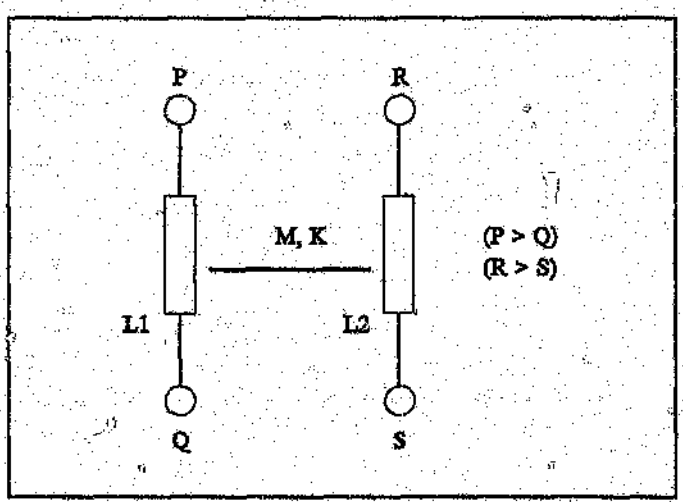


Figure 2-2 Two Mutually Coupled Inductors

2.4 Mutual Inductance Circuits

- Loose-coupled air-core and close-coupled iron-core transformers, auto-transformers and tapped coils can be modelled using mutual inductance networks.
- Individual inductors must be inserted into the circuit before the coupling between them is entered.
- Coupling is specified in Equation 2.1. The sign of the coupling (M or F) is positive if the induced emf in the second inductor is positive at the higher numbered node. L1 and L2 are the self inductances.

$$M = K \times \sqrt{L1 \times L2}$$

K – Coupling Factor (0..1)
M – Mutual Inductance

Equation 2-1 Mutual Inductance

2.5 Voltage Controlled Current Sources

- Any number of voltage controlled current sources can be inserted between a particular node pair, though each must have a different pair of controlling nodes.
- The frequency response can have a first order pole in it. This helps in the linear modelling of semiconductor devices like operational amplifiers and bipolar junction transistors.

$$Gm = \frac{Gmo}{1 + j\frac{F}{Fo}}$$

Gmo – *Nominal Transconductance*
F – *Current Operating Frequency*
Fco – *Pole Frequency*

Equation 2-2 Transconductance

2.6 Semiconductor Circuits

Circuits that contain semiconductor devices can have small signal analysis (about the bias or operating point) performed on them if :

- An appropriate linear circuit is developed which is equivalent to the semiconductor device in the region of interest.
- This equivalent circuit replaces the semiconductor device in the circuit.
- The internal nodes of the equivalent circuit are renumbered.

To model bipolar junction transistors, use h - parameter or hybrid - pi models.

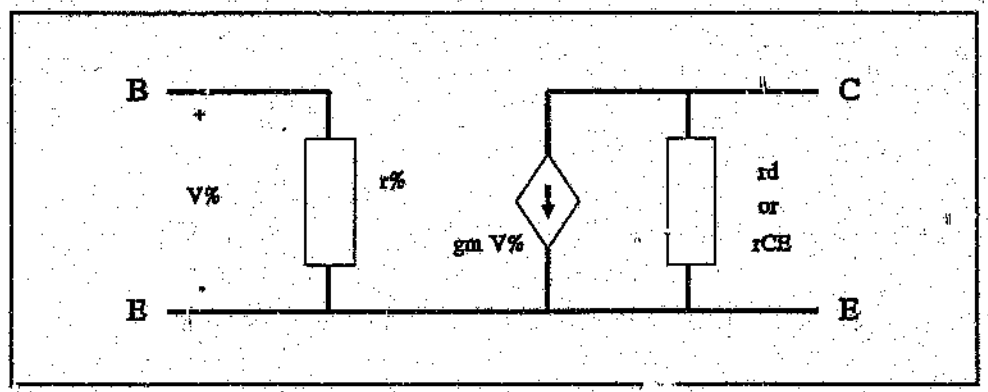


Figure 2-3 Hybrid Pi Model of a Transistor

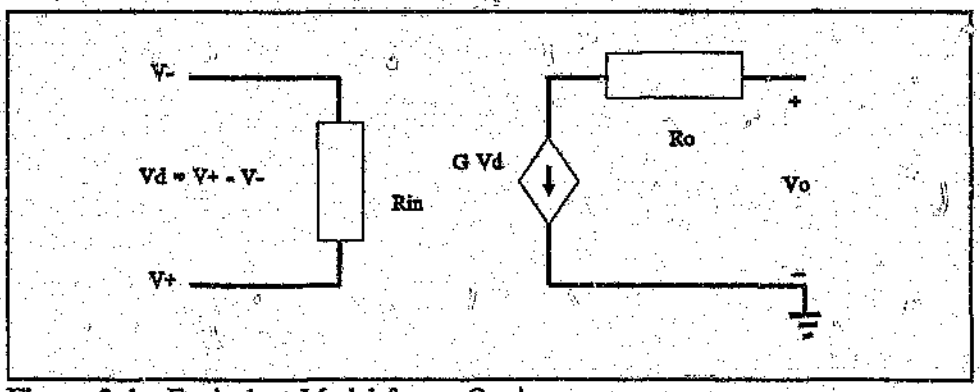


Figure 2-4 Equivalent Model for an Op-Amp

2.7 Method for Designing Circuits

Lipworth has proposed a method for designing circuits on Elecsim. The same approach can be used for Elector [Lipworth, 1989]:

(1) Understand the problem and what type of circuit will solve it.
(2) Sketch the circuit.
(3) Semiconductors (transistors and operation amplifiers) in the circuit must be replaced by an appropriate linear circuit which is equivalent in the region of interest.
(4) Redraw the circuit with the equivalent circuits replacing the semiconductors.
(5) Label all the nodes in the circuit, starting at the reference node (0). Coupled inductors and voltage controlled current sources must be numbered as discussed previously.
(6) Determine approximate component values.
(7) Enter the circuit into Elector.
(8) List the components and draw the schematic diagram to verify that no typographical errors have occurred.
(9) Give the circuit a short descriptive title and save the circuit to a disk file.
(10) Node voltages can then be determined at special nodes (input and output) at key frequencies.
(11) The frequency and phase response can be plotted to determine if the circuit is operating correctly.
(12) The circuit's performance can also be optimised.
(13) If the desired performance cannot be obtained, more components might have to be added. Repeat steps (10-13) on the modified circuit.
(14) The allowed tolerances of each component may be determined by Monte-Carlo analysis.
(15) Save the final circuit to a disk file.
(16) Construct the physical circuit.

CHAPTER 3

THE USER INTERFACE

3.1 Introduction

The *User Interface* is that part of the program you use to communicate with the rest of the program. It is divided into three main parts :

- *Main Screen* : This is the large blue background that is always visible. Results and data are written to it, as well as graphs and circuit schematics.
- *Subscreens* or *Dialogue Boxes* : These appear when certain input is needed by the program, or when you need to be informed about something.
- *Menus* : Different options can be chosen from the menus. The options that can be selected will be in a different colour to those that cannot be selected.

3.2 Subscreens

Subscreens get input from the you and give output to you. It can be moved around the screen using the mouse, or by pressing the *Move* button and then using the cursor keys. Each subscreen consists of a Window object, 4 *Buttons* and possibly an *Input_Line*, *Pick_List*, *Forms* or *Boolean_Buttons* object. There are three types of subscreen objects.

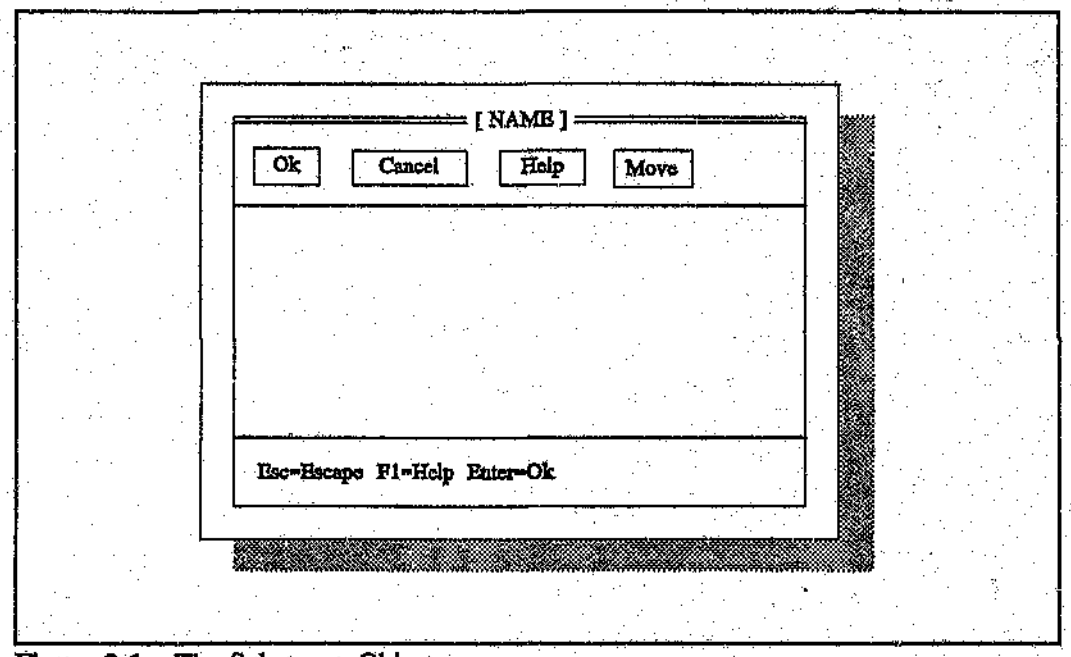


Figure 3-1 The Subscreen Object

- *Errors* : Displays error messages that the program needs to inform you about. The frame will also be red.
- *Prompts* : Gets some input from you and possibly displays a message. Different screen objects get different types of input :
  - A number of related values - Forms
  - Lists of options - Pick_List
  - Integer value, real value, string - Input_Line
  - Boolean response (Yes/No) - Boolean_Buttons
- *Status* : Displays a status message about the progress of some process.

Button	Purpose of the Button
Ok	This tells the subscreen object that the input entered is correct (if input was requested) or that the message has been read.
Cancel	This cancels the present input as well as the present process. Control is given back to the scheduler. This appears to the user as though the process was cancelled, another process can then be chosen from the menus.
Help	This calls the help object to display on-line help about the present subscreen or a part of the application program.
Move	This allows the subscreen to be moved around the physical screen by the cursor keys. The frame will change colour when this mode is active. Once a suitable position is found for the subscreen, press the Enter key. The frame will change back to it's normal colour.

Table 3-1 Purpose of the Subscreen Buttons

3.3 Buttons

3.3.1 Purpose

It initiates some action. The action can be considered to be connected to the pressing of the button.

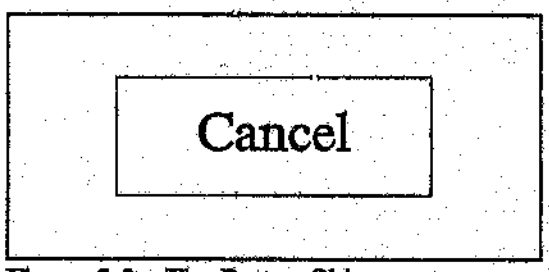


Figure 3-2 The Button Object

3.3.2 Appearance and Behaviour

A button is a bar that appears on the screen and does not have a shadow. When the button is *pressed*, the button changes colour. The button is *released* about half a second later and appears like it did before being pressed - in it's original colour.

The button can be *pressed* in two ways :

- Pressing the Alt key with the character that is highlighted on the button's surface. In the above example, this would be Alt - P.
- Pressing the left button of the mouse when the mouse pointer is on top of the button. This means that the mouse pointer can be anywhere of the button's surface, but not on the shadow, for the button to be pressed.

3.4 Boolean_Buttons

3.4.1 Purpose

To get a Boolean input response from the user. In other words, a yes/no, true/false type response.

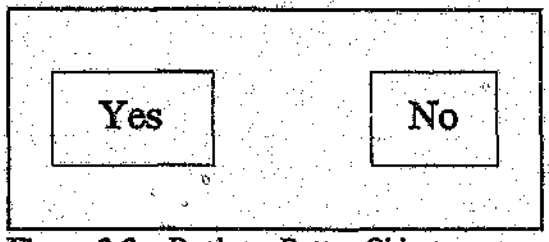


Figure 3-3 Boolean_Button Object

3.4.2 Appearance and Behaviour

This object is made up from two button objects - a *True* button and a *False* button. The description and behaviour of the two buttons is discussed in the Buttons section.

3.5 Input_Lines

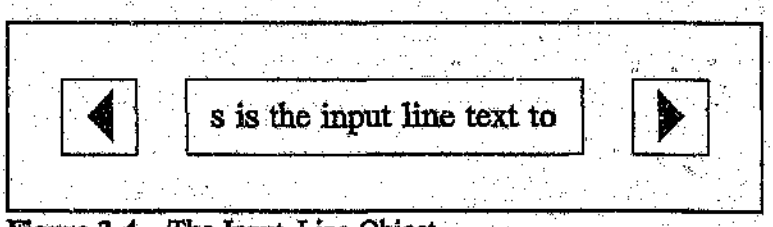


Figure 3-4 The Input_Line Object

3.5.1 Purpose

It gets string input. String input includes integer and real numbers which are entered as strings and then converted.

3.5.2 Appearance and Behaviour

The Input_Line object appears on the screen as three bars in a line. The centre bar is the longest

and this is where characters of the string are entered. However, this centre bar can only hold a certain number of characters. If more characters are entered than can be displayed, they will be scrolled off to the left. A left arrow will appear in the first bar to show that there are characters to the left of those displayed, but because of the limited length of the display area, they cannot be displayed. The cursor can be moved to the edges of the display bar to scroll a string that is longer than the display bar to the left or right. If the string is being scrolled to the right and the right most characters disappear, a right arrow will appear to show that there are characters to the right of those displayed. There are two modes of character entry :

- *Insert* : Characters are inserted into the string at the position of the cursor. The characters to the right of the cursor are moved one place to the right before the character is inserted. The cursor appears as a flashing thin line.
- *Typeover* : Characters are inserted into the string at the position of the cursor. The character on which the cursor is flashing is overwritten by the new character. Characters to the right of the cursor remain in their present position. The cursor appears as a flashing block.

Key	Function of the key
A - Z	Only for string input (E for real input as well).
a - z	Only for string input (e for real input as well).
\/_['^]	Only for string input.
0 - 9	For integer, real and string input.
-	For integer, real and string input.
.eE	For real and string input.
Cursor Left	Move the cursor to the left, or if the cursor is at the left most position, scroll the text to the right if there is a left arrow.
Cursor Right	Move the cursor to the right, or if the cursor is at the right most position, scroll the text to the left if there is a right arrow.
Insert	Toggle between insert and typeover mode. The shape of the cursor will change to a thin line in insert mode, and to block in typeover mode.
Delete	Deletes the character the cursor is on.
Backspace	Deletes the character to the left of the cursor.

Table 3-2 Input_Line Key Definition

3.6 Pick_Lists (Static)

3.6.1 Purpose

The Pick_List object is used when list-type input is needed. The list can be either exclusive (only one option can be chosen) or inclusive (many options may be chosen simultaneously).

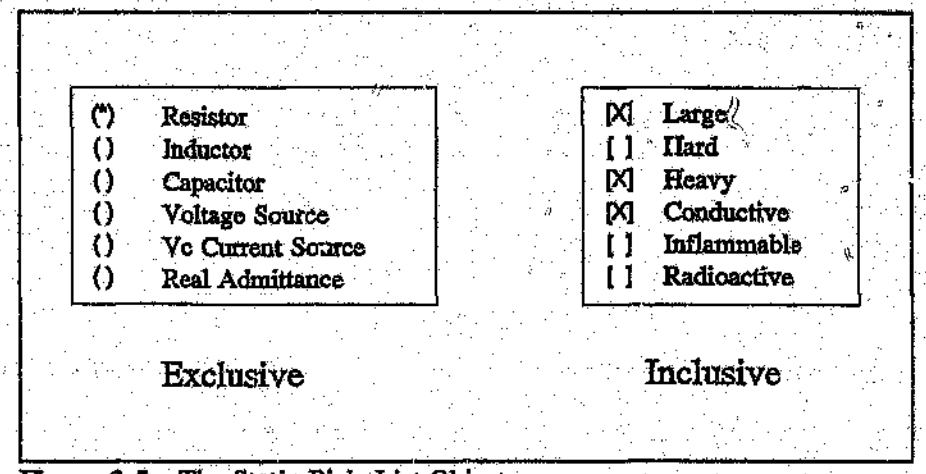


Figure 3-5 The Static Pick_List Object

3.6.2 Appearance and Behaviour

The Pick_List appears as a large flat area on the screen. On the left of this area are radio buttons if there is an exclusive list, or switches if there is an inclusive list. Text for each option of the list is written in the remaining area.

Characteristics of Pick_Lists	Exclusive	Inclusive
Option is not chosen	( )	[ ]
Option has been chosen	(*)	[X]

Table 3-3 Pick_List Characteristics

Key pressed	Affect on the Pick_List object
Cursor up	Moves the selection bar up to the previous option.
Cursor down	Moves the selection bar down to the next option.
Home	Moves the selection bar to the first option.
End	Moves the selection bar to the last option.
Space	Selects the option that the selection bar is highlighting for inclusive lists.

Table 3-4 Pick_List (Static) Key Definition

Options can be chosen by the mouse or the keyboard. To use the mouse, just move the mouse pointer to the desired option and press the left button. To use the keyboard, move the selection bar until it highlights the desired option. If the list is inclusive, select or de-select the highlighted

option by pressing the space bar. When choosing an option, the behaviour of the Pick_List object will depend on the type of list presented.

- If the list is exclusive, choosing a different option to one already set, will deselect the previously option and select the new option.
- If the list is inclusive, choosing an option will toggle it's state. That means that if the option is selected, it will be deselected, and vice versa.

3.7 Pick_Lists (Dynamic)

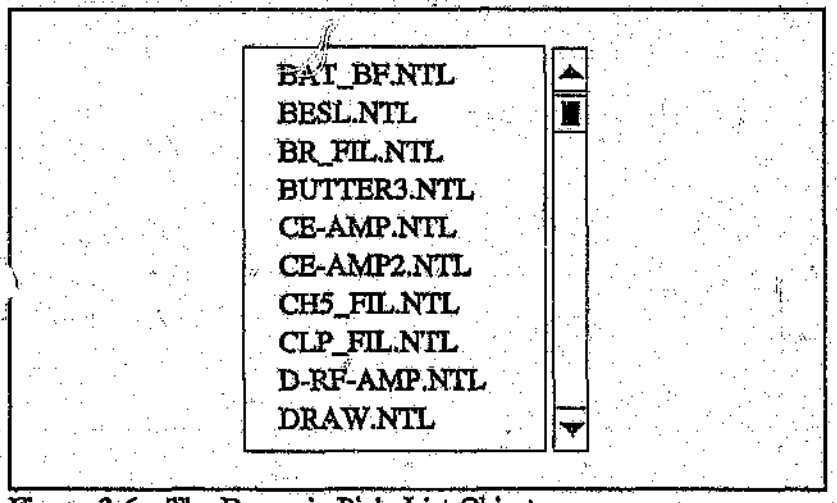


Figure 3-6 The Dynamic Pick_List Object

3.7.1 Purpose

The dynamic Pick_List object is used when lists of different lengths (dynamic) need to be displayed from which one option must be chosen. This object is usually used for choosing a filename from a list.

3.7.2 Appearance and Behaviour

The dynamic Pick_List appears as a large flat area with a possible scroll bar to the right. On the flat area is a text list of available options. If there are more options than can be displayed, a scroll bar appears. There is an up and a down arrow. An indicator shows what part of the total list is visible. The highlighting bar can be moved up or down to the desired choice by pressing the cursor up or down keys respectively.

The mouse can also be used to choose an option. By placing the mouse pointer on the list and pressing the left mouse button, the option that the mouse pointer is on will be highlighted and selected. Placing the mouse pointer on the up arrow of the scroll bar and pressing the left mouse

button moves the highlighting bar to the previous option if there is one. Placing the mouse pointer on the down arrow of the scroll bar and pressing the left mouse button moves the highlighting bar to the next option if there is one.

3.8 Forms

3.8.1 Purpose

The form object is used when many related numerical values need to be entered.

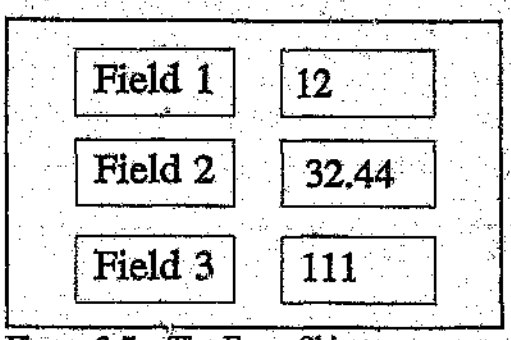


Figure 3-7 The Form Object

3.8.2 Appearance and Behaviour

A text string is displayed to the left of each input field. The size of the input field shows the number of characters that can be entered. Real values and integer values can be entered, but only one type for each input field. Real and integer values that are entered are checked to see that they are not greater than a maximum value, and not less than a minimum value. An integer value can also be checked against another integer value from another input field to see if they are equal.

Key Pressed	Affect on the Form object
Cursor up	Moves the cursor to the previous field. If the cursor is already in the first field, it moves to the last field.
Cursor down	Moves the cursor to the next field. If the cursor is already in the last field, it moves to the first field.
Cursor left	Moves the cursor to the left by one column. If the cursor is already at the start of the field, the cursor moves to the previous field.
Cursor right	Moves the cursor to the right by one column. If the cursor is already at the of the text in the field, it moves to the next field.
Tab	Makes the form inactive. This prevents the form from accepting any more input. This is used when switching between the form and static Pick_List objects.

Table 3-5 Form Key Definition

3.9 Windows

3.9.1 Purpose

The Window display object can be moved around the physical screen and contain other objects.

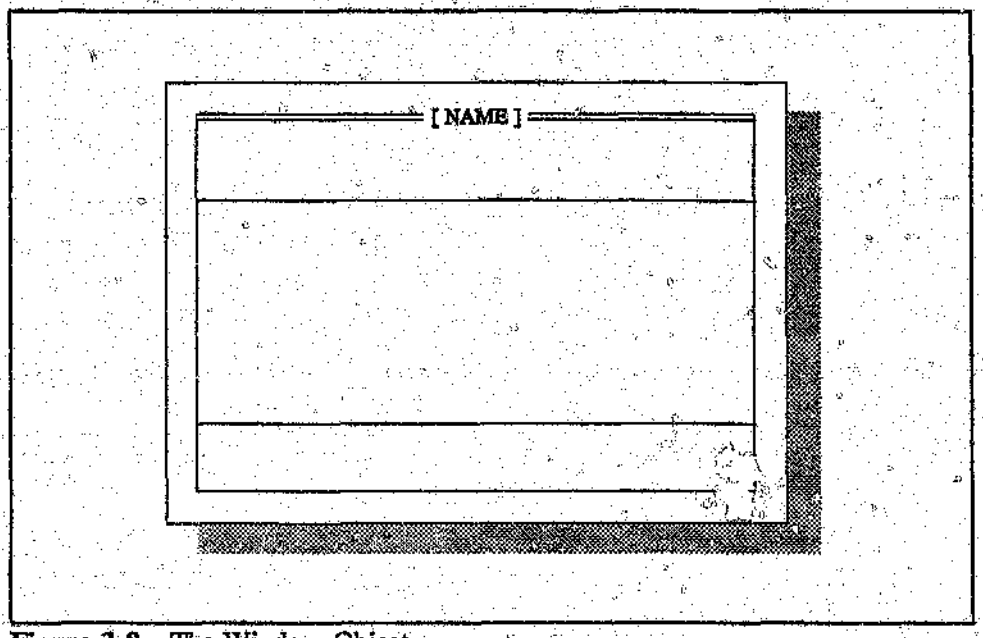


Figure 3-8 The Window Object

3.9.2 Appearance and Behaviour

The window appears on the physical screen as a window display with a shadow to the bottom right of it. It is framed with a single border. The window usually has a name and can have text in it. The window object also converts global coordinates to local coordinates for those objects that appear inside it.

The window can be moved around the screen using the mouse or the keyboard cursor keys. When the window is in a moving mode, it's frame will change to a different colour. To move the window with the mouse, press the left mouse button when the mouse pointer is on the top frame of the window. Keep the left button down while dragging the window to the desired position. The keyboard mode cannot be invoked directly, but by another object, usually a button.

3.10 Spread Menu

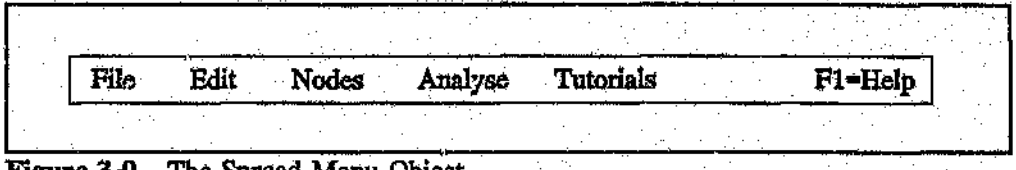


Figure 3-9 The Spread Menu Object

3.10.1 Purpose

The spread menu present the user with a list of options from which one is selected.

3.10.2 Appearance and Behaviour

The menu options are spread out one next to another. An option on the menu can be selected in three different ways :

- *Menu Selection Bar* : The menu selection bar is a highlighted bar that appears on the menu. This bar highlights one of the options on the menu. This option will be selected if the *Enter* key is pressed. Certain cursor keys can move the selection bar.
- *Menu Option Shortcut Key* : On the menu, one of the characters for each option will be highlighted. The character shows which key must be pressed on the keyboard to select that option. Pressing the highlighted character key is equivalent to moving the selection bar to that option and pressing *Enter*.
- *Mouse Pointer* : The mouse can be used to select a menu option. Move the mouse pointer to the desired option and press the left mouse button. The mouse pointer can be anywhere on the menu option to select it.

Key pressed	Affect on the Spread Menu
Cursor left	Moves the selection bar left to the previous option.
Cursor right	Moves the selection bar right to the next option.
Home	Moves the selection bar to the first option.
End	Moves the selection bar to the last option.
Enter	Selects the option that the selection bar is highlighting.

Table 3-6 Spread Menu Key Definition

3.11 Stacked Menu

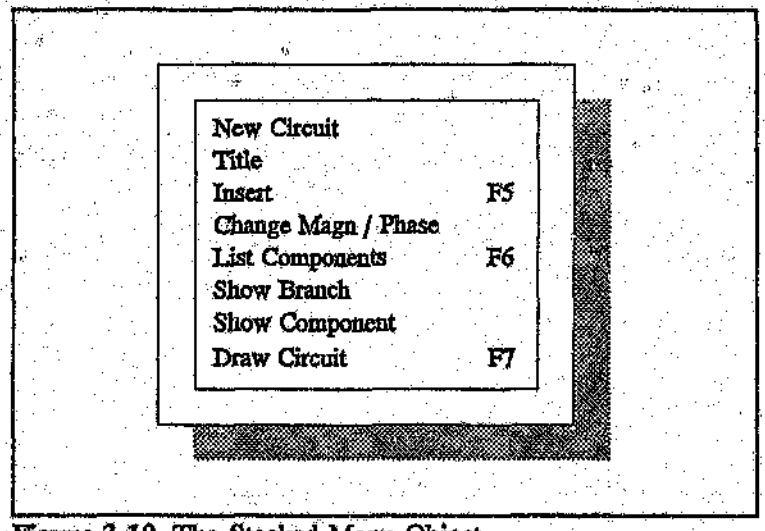


Figure 3-10 The Stacked Menu Object

3.11.1 Purpose

The stack menu presents the user with a list of options from which one is selected.

3.11.2 Appearance and Behaviour

The menu options are displayed one under another in the form of a stack. An option on the menu can be selected in four different ways :

- *Menu Selection Bar* : The menu selection bar is a highlighted bar that appears on the menu. This bar highlights one of the options on the menu. This option will be selected if the *Enter* key is pressed. Certain cursor keys can move the selection bar.
- *Menu Option Shortcut Key* : On the menu, one of the characters for each option will be highlighted. The character shows which key must be pressed on the keyboard to select that option. Pressing the highlighted character key is equivalent to moving the selection bar to that option and pressing *Enter*.
- *Menu Option Hotkey* : Next to some of the menu items, there is a key description eg. *F2*. This method differs from the shortcut-key in that the option will be selected even if a different menu is on the screen when this key combination is pressed.
- *Mouse Pointer* : The mouse can be used to select a menu option. Move the mouse pointer to the desired option and press the left mouse button. The mouse pointer can be anywhere on the menu row of the desired option to select it.

Key pressed	Affect on the Stack Menu
Cursor up	Moves the selection bar up to the previous option.
Cursor down	Moves the selection bar down to the next option.
Cursor left	Moves to the previous menu on the same level.
Cursor right	Moves to the next menu on the same level.
Home	Moves the selection bar to the first option.
End	Moves the selection bar to the last option.
Enter	Selects the option that the selection bar is highlighting.

Table 3-7 Stack Menu Key Definition

CHAPTER 4

MENU OPERATIONS

4.1 Menu Hierarchy

All the major functions can be invoked using the menus. The functions are grouped into four units. See Figure 4.2 for a diagram of the menu hierarchy. Some of the options on the menus will not be selectable. These options will be in a lighter colour.

4.2 Main Menu

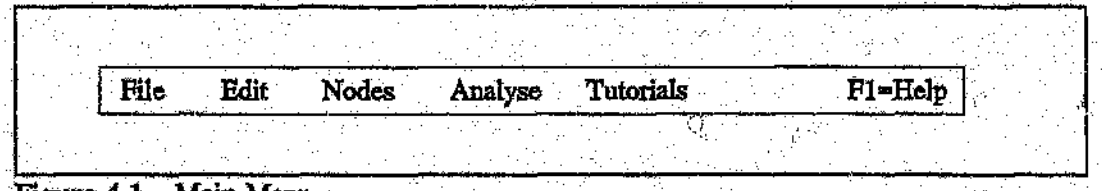


Figure 4-1 Main Menu

4.2.1 File

It calls the file submenu on which there are options for recovering and saving circuit files.

4.2.2 Edit

It calls the edit submenu on which there are options for entering components and a title for the circuit; listing the components in the circuit or a branch; and drawing the circuit.

4.2.3 Nodes

It calls the nodes submenu on which there are options for deleting and renumbering the nodes.

4.2.4 Analyse

It calls the analyse submenu on which there are options for calculating the node voltages and branch currents; the power used in passive components; the sensitivity of the circuit; Monte-Carlo analysis and optimisation. The circuit response, Fourier analysis and polar and phasor plotting are also available.

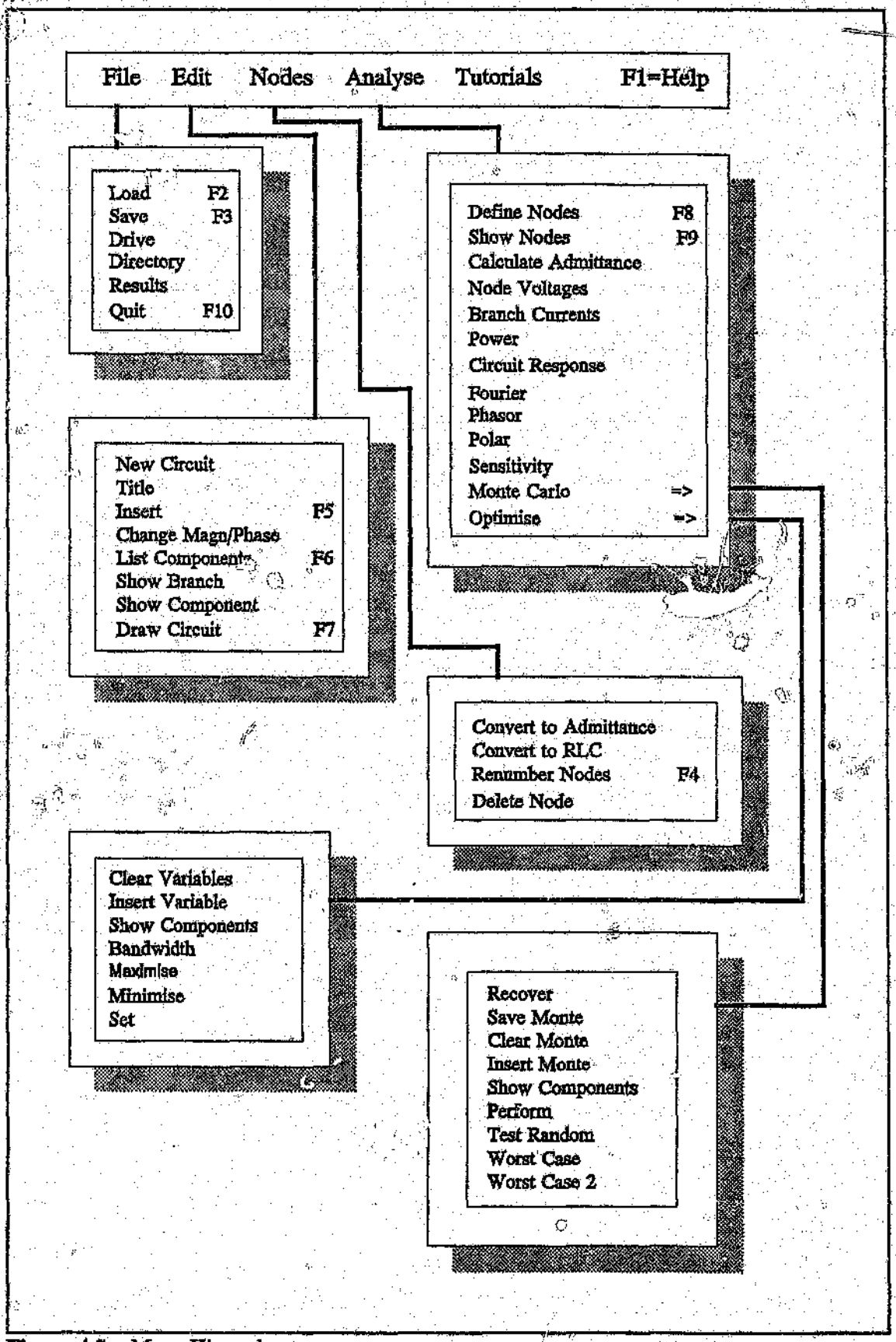


Figure 4-2 Menu Hierarchy

4.2.5 Tutorials

It calls the tutorial. There are a number of different options to choose from, including how the user interface works and Electrical Engineering circuit theory.

4.3 File Menu

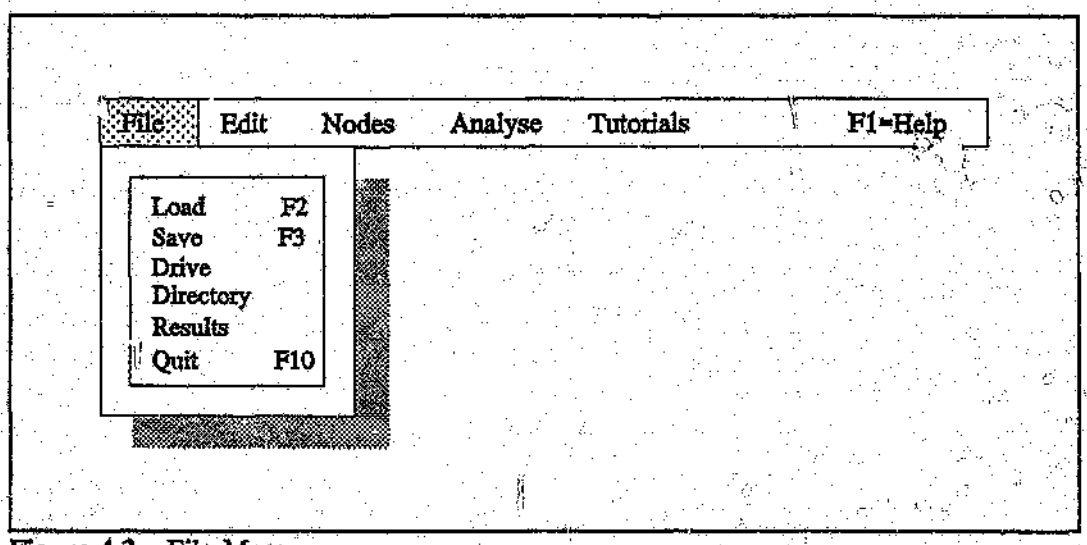


Figure 4-3 File Menu

4.3.1 Load

It recovers a circuit that has been saved to disk. A list of circuit files in the default directory is displayed on a PROMPT subscreen. If the circuit file is in the default directory, it will be displayed. However, if it is in another directory, the path must be changed using the *Drive* and *Directory* commands. The circuit file is chosen by moving the highlighting bar to highlight the circuit filename and pressing the *Ok* button.

If there is already a circuit presently in use, the option of saving the present circuit will be given. This circuit will be saved to the default directory if only a filename is given. However, by specifying a path, it can be saved in another directory.

4.3.2 Save

It saves a circuit to a disk file. The size of the file depends on the size of the circuit. Only the circuit components and their node numbers are stored. A list of circuit files in the default directory is displayed on the main screen. If the file is to be saved in the default directory, only the name of the file need be entered. However, if it is to be saved in another directory, the path

must also be specified.

4.3.3 Drive

It displays the default drive and path on the main screen. A prompt appears asking for the new default drive. Only the first character that is typed is used. If you do not want to change the default drive, press the *Cancel* button.

4.3.4 Directory

It displays the present path showing the default directory on the main screen. A prompt appears asking for the new path. If you do not want to change the present path, press the *Cancel* button. After the new path has been entered, the new path will be displayed on the main screen. If the new path is the same as the old, the path that was entered could not be set.

4.3.5 Results

It saves the non-graphic results of a circuit simulation, design or optimisation to a text file. If the file is to be saved in the default directory, only the name of the file need be entered. However, if it is to be saved in another directory, the path must also be specified. If this function is selected a second time, it will close the results file if it was open.

4.3.6 Quit

If there is a circuit, a prompt will appear asking if it should be saved. If the response is *Yes*, a filename for the file is requested. A list of circuit files in the default directory is displayed on the main screen. If the file is to be saved in the default directory, only the name of the file need be entered. However, if it is to be saved in another directory, the path must also be specified. The file is then saved. The program is exited and any open device files are closed.

4.4 Edit

4.4.1 New Circuit

It removes the current circuit so that a new one can be entered. This is done by clearing the program memory and initializing program variables. This option is not needed on first entry into Elector.

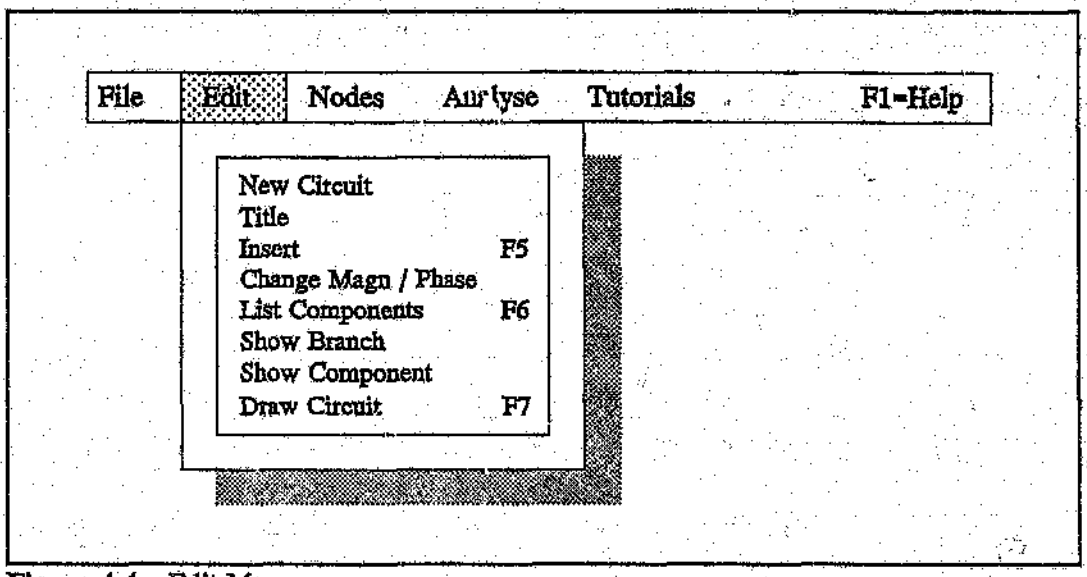


Figure 4-4 Edit Menu

Because a circuit that is present could be lost, a prompt asking if you want to save the circuit is displayed. If you decide to save the circuit, you will be asked for filename. A list of circuit files in the default directory is displayed on the main screen. If the circuit file is to be saved in the default directory, only the name of the file need be entered. However, if it is to be saved in another directory, the path must also be specified.

4.4.2 Title

It gives the circuit a title, or overwrites the present title if one is set. The present title is displayed on the main screen. The title can have up to 64 characters.

4.4.3 Insert

It inserts a component between two nodes and is used when creating and extending circuits.

- Two components of the same type cannot be entered between the same two nodes because the value of the second component will overwrite the value of the first. If two such components need to be entered in parallel, their equivalent resistance must be calculated, and that value entered. Components of different types can be entered in parallel.
- Two components cannot be entered in series between the same two nodes and each component must have it's own node pair when inserted in series with other components.
- A component is deleted by entering a value of 0.0 for that component followed by the nodes where it is found. A value of zero means that the component is not there. In the case of a resistance, there is no short-circuit. To enter a short-circuit between any two nodes, make the value of the resistor insignificant when compared to other resistances in the circuit.

4.4.4 Change Magnitude / Phase

It changes the magnitude or phase of a component in the current circuit. This option cannot be used for initial entry of a component.

4.4.5 List Components

It lists all the components that have been inserted into the current circuit. Each component's type, value, units and the nodes to which it is connected is displayed. If the component has a phase, such as voltage sources, it is also displayed. The listing can take three forms:

- *Impedances* : Resistance in Ohms, inductance in Henry and capacitance in Farads.
- *Admittances* : Circuit components will be displayed as admittances.
- *Equivalent Inductances* : Lists the equivalent inductors, including those derived from mutual inductances.

4.4.6 Show Branch

It lists all the components in a specified circuit branch (between a specific node pair). The higher and lower nodes of the branch are required. The node values must be entered as integer values.

4.4.7 Show Component

It lists all the components of a selected type in the current circuit. The type of component is selected from a list of components on a PROMPT subscreen. If there are no components of the selected type in the circuit, the main screen will be blank.

4.4.8 Draw Circuit

It draws the schematic diagram of the circuit on the main screen. A PROMPT subscreen with a list appears on which :

- The default node placement can be selected. If a different node placement is required, enter the nodes in the order they must be drawn. When you have specified the last node number, enter 0.
- The fast drawing mode can be selected. In fast mode, only that part of the circuit that fits onto the main screen is drawn. In slow mode, the entire circuit is drawn - even that part that does not fit onto the main screen. For a large circuit, this can take a long time.

An ERROR subscreen appears if the node order would result in a messy and distorted schematic.

Key	Drawing Function
Cursor up	Scroll the circuit up.
Cursor down	Scroll the circuit down.
Cursor left	Scroll the circuit to the left.
Cursor right	Scroll the circuit to the right.
Page Up	Zoom into the circuit.
Page Down	Zoom out of the circuit.
Home	Display the start of the circuit.
End	Display the end of the circuit.

Table 4-1 Circuit Drawing Key Definition

4.5 Nodes

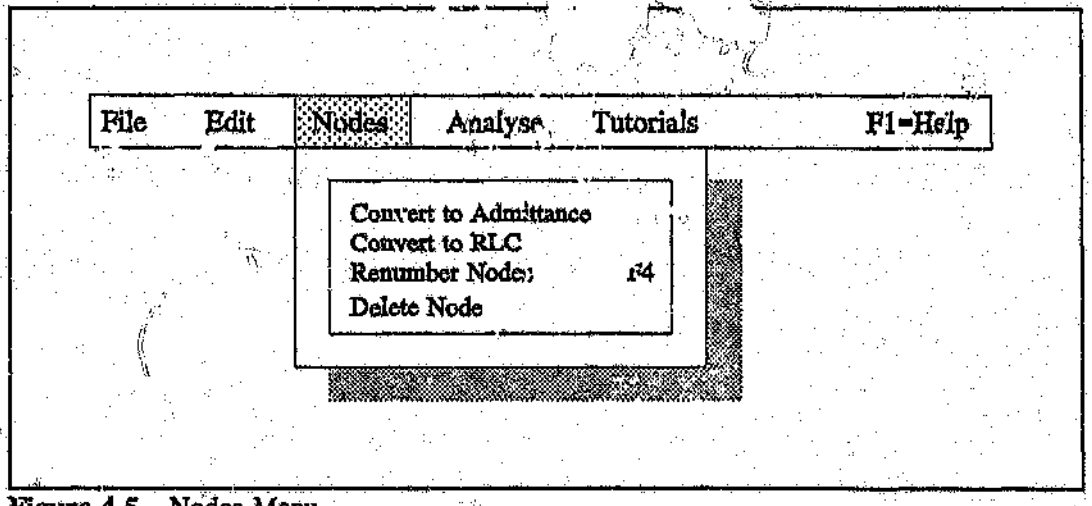


Figure 4-5 Nodes Menu

4.5.1 Convert to Admittance

It converts a circuit from resistors, capacitors and inductances to an equivalent set of admittances at a specified frequency. The inductors will be converted to equivalent imaginary admittances using the formula : $Y = 1/jwL$. The capacitors will be converted to equivalent imaginary admittances using the formula $Y = jwC$. Note ; $w = 2\pi f$, f = frequency, L = inductance and C = capacitance. The voltage controlled current sources will be converted to the form of *Real_Vccs* + *j.Imag_Vccs*.

The default frequency which will be used in the conversion is displayed. This value can be changed by answering *Yes* to the prompt asking if a new frequency is required.

4.5.2 Convert to RLC

It converts a circuit from admittances at a specified frequency to an equivalent set of resistors, capacitors and inductances. This is the inverse of *Convert to Admittance*. The default frequency which will be used in the conversion is displayed. This value can be changed by answering *Yes* to the prompt that a new frequency is required.

4.5.3 Renumber Node

It renumbers any node in the circuit to a new value. The old node number is entered first, followed by the new node number. An ERROR subscreen will appear if the new node already exists and / or if the old node does not exist.

4.5.4 Delete Node

It deletes a circuit node. The node number which must be an integer is required. All components that are connected to the chosen node will be deleted and will appear on the main screen. An ERROR subscreen will appear if the node to delete does not exist. Note : The reference node (0) cannot be deleted. This node is fixed as the common earth.

4.6 Analyse

4.6.1 Define Nodes

It defines which circuit node pairs will be analysed. To define a node with respect to the common earth, just enter the node number. To define a node with respect to another node (not the reference node 0), use the following sequence : node number <ENTER> - node number <ENTER>. When the last node number has been entered, enter <0> to exit and return to the menus.

Note : There is a specific order in which the required nodes are entered. Defining node 2 with respect to node 3 is different to defining node 3 with respect to node 2 - there is a 180 degree phase shift between the two specifications.

Example : Voltage differences between nodes are specified by a pair of integers :

6 <ENTER> -4 <ENTER> 0 <ENTER>

This specifies that voltage between node 6 and node 4 is to be output.

6 <ENTER> -4 <ENTER> 1 <ENTER> 2 <ENTER> 0 <ENTER>

This specifies the voltage between node 6 and node 4, and node 1 and 2, are to be output.

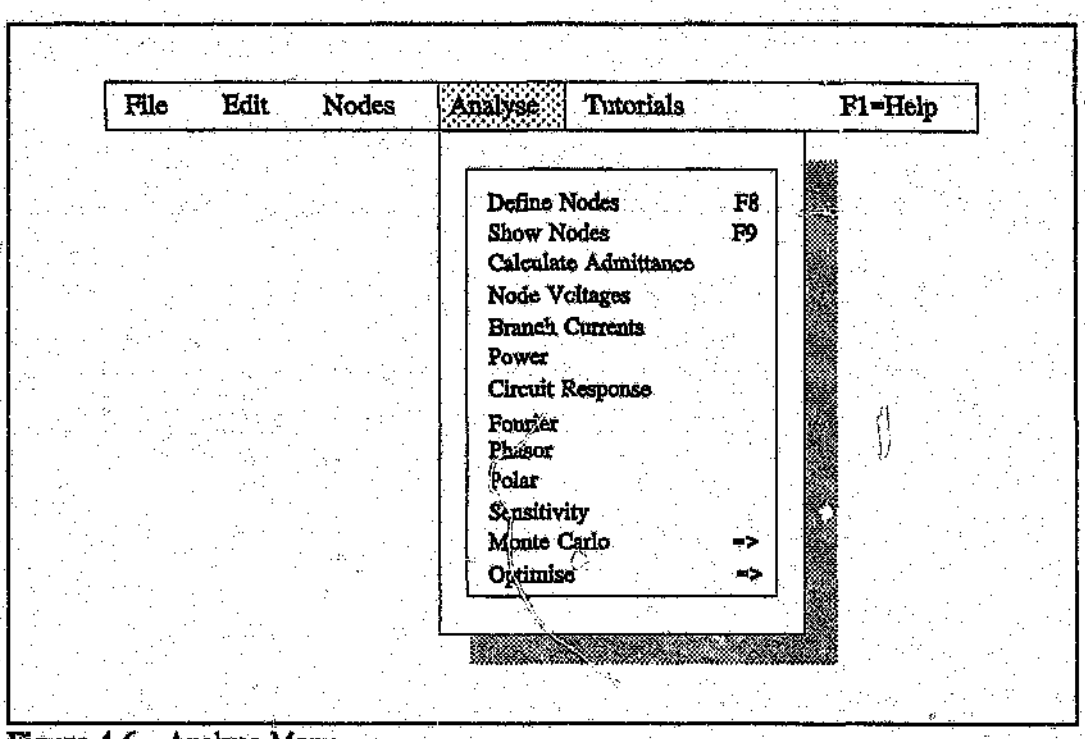


Figure 4-6 Analyse Menu

4.6.2 Show Nodes

It displays the circuit nodes which are to be analysed on the main screen.

4.6.3 Calculate Admittance

It calculates the admittance matrix used to analyse the circuit. The matrices used to calculate the node voltages are displayed on the main screen. A prediction can be made as to whether the results will contain rounding and/or other numerical errors.

4.6.4 Node Voltages

It calculates the node voltages for the defined node pairs. The default frequency which will be used in the calculation is displayed. This value can be changed by answering *Yes* to the prompt that a new frequency is required. Another prompt will then ask for this new frequency.

A list is displayed on the screen which gives each node voltage in rectangular coordinates (real and imaginary), polar coordinates (magnitude and phase) and logarithmic magnitude in decibels. For a large circuit, all the information might not fit onto a single screen. A prompt to press any key will appear. Once the display has been examined, press any key and more values will appear.

4.6.5 Branch Currents

It calculates the currents in the branches between the defined node pairs. The default frequency which will be used in the calculation is displayed. This value can be changed by answering *Yes* to the prompt that a new frequency is required. Another prompt will then ask for this new frequency.

A list is displayed on the main screen which gives each branch current in rectangular coordinates (real and imaginary), polar coordinates (magnitude and phase) and logarithmic magnitude in decibels. For a large circuit, all the information might not fit onto a single screen. A prompt to press any key will appear. Once the display has been examined, press any key and more values will appear.

4.6.6 Power

It calculates the power absorbed (in watts/vars) by passive components. On the main screen the following are displayed :

- The power factor.
- The real power absorbed.
- The reactive power, as well as a breakdown of the reactive power into inductive and capacitive power.
- The apparent power.
- The magnitude and phase of the impedance in that branch.

If there is mutual inductance present, the energy stored in the magnetic field is shared between the coupled inductors. The mutually stored energy is not calculated. Only those parts of inductive energy associated with each inductor are calculated.

4.6.7 Circuit Response

There are five different options to choose from :

- Plot the frequency response of a circuit on a linear frequency axis.
- Plot the frequency response of a circuit on a logarithmic frequency axis.
- Generate a parametric plot on linear axis showing how the defined node pairs respond to the value of a circuit component being varied at a single frequency.
- Generate a parametric plot on logarithmic axis showing how the defined node pairs respond to the value of a circuit component being varied at a single frequency.
- Generate a 3-dimensional plot on linear axis showing how the defined node pair responds to the value of a circuit component and the frequency being varied.

4.6.8 Fourier

It plots a steady-state, time domain graph of node voltage against time for a specified node pair. A PROMPT subscreen with a list and a form appears. The list shows a number of different waves. The wave is chosen by moving the highlighting bar to that option. To fill in the form, press the tab key. The form has the following fields :

- The number of harmonics (1 - 1000) that are used in the calculation of the Fourier series. The more harmonics, the greater the accuracy, but the longer it takes to calculate.
- The node pair for which the plot is to be generated.
- The start and finishing times of the graph. It is desirable to have the time window show one or two periods of the wave form. Period $T = 1/f$, where f is the fundamental frequency.
- The plot increments (1 - 50).
- The impedance frequency which will be used in the calculation.

4.6.9 Phasor

It plots a phasor diagram for a defined node pairs. The phasor diagram is a plot of node voltages at only one frequency. The phasors are plotted to scale.

4.6.10 Polar

It plots a polar diagram for the defined node pair. A polar plot is generated for only one node pair at a time, instead of doing an analysis for all the defined node pairs. A PROMPT subscreen with a list and a form will appear. The increments in frequency (linear or logarithmic) is selected from the list. The form has the following fields :

- The higher and lower nodes for the polar plot.
- The starting and ending frequencies.
- The number of points (1 - 2500). This determines the resolution of the plot. The more points, the better the resolution, but the longer it takes to generate them.

4.6.11 Sensitivity

It calculates the sensitivity of the node voltages of defined node pairs to perturbations in component values. Sensitivity is defined as the fractional change in node voltage divided by the fractional change in a particular component value.

The default frequency which will be used in the calculation is displayed. This number can be changed by answering *Yes* to the prompt that a new frequency is required. Another prompt will then ask for this new frequency. Next, enter the sensitivity factor as a real number.

$$Sensitivity = \frac{\frac{dV}{V}}{\frac{dRes}{Res}}$$

dV = *fractional change in node voltage*
V = *nominal value of node voltage*
dRes = *fractional change in component value*
Res = *nominal component value*

Equation 4-1 Sensitivity

A list of components (resistors, inductors, capacitors and voltage controlled current sources) with their nominal values (inserted values) and the sensitivity of each defined node pair with respect to the perturbation, will be displayed on the main screen. For a large circuit, all the information might not fit onto a single screen. A prompt to press any key will appear. Once the display has been examined, press any key and more values will appear.

4.6.12 Monte-Carlo

It calls the Monte-Carlo submenu on which there are options for loading, saving and performing Monte-Carlo analysis on a circuit.

4.6.13 Optimisation

It calls the Optimise submenu on which there are options for maximising, minimising and setting the response of a circuit.

4.7 Monte Carlo

4.7.1 Recover

It recovers previously saved files of components and their tolerances used for Monte-Carlo analysis. A list of Monte-Carlo files in the default directory is displayed on a PROMPT subscreen. If the file is in another directory, the path must be changed using the *Drive* and/or *Directory* command. There must already be a circuit present before this function can be used.

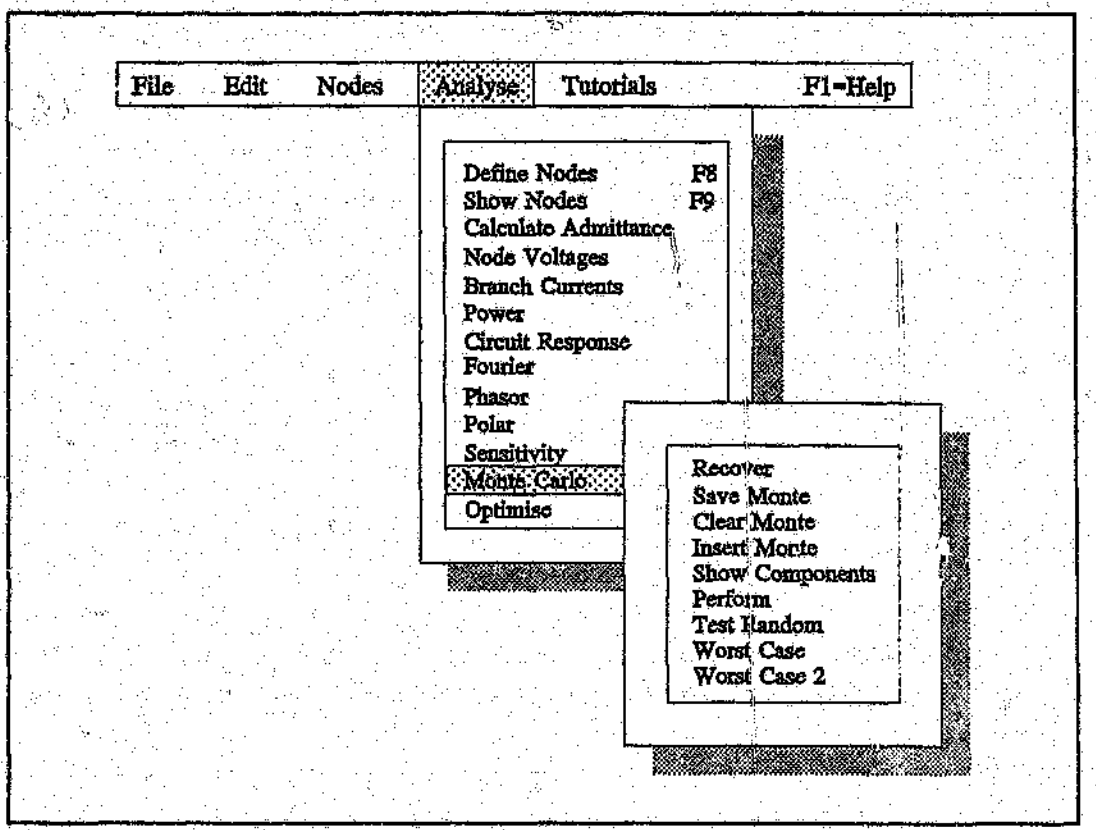


Figure 4-7 Monte-Carlo Menu

4.7.2 Save Monte

It saves the current list of Monte-Carlo components with their tolerances to a file on disk. A list of Monte-Carlo files in the default directory is displayed on the main screen. If the Monte-Carlo file exists, an option of overwriting the file is given. If the *Yes* option is chosen, the file is overwritten.

4.7.3 Clear Monte

It clears the current Monte-Carlo components and their tolerances. The first prompt asks if the components must really be cleared. If *Yes* is chosen, all Monte-Carlo variables are cleared.

4.7.4 Insert Monte

It defines the components and tolerances that are to be used for Monte-Carlo analysis. The component(s) to be inserted must be chosen from a component list. At least one component must be defined before any Monte-Carlo simulation can be done. More than one component can be used in a simulation.

A PROMPT subscreen with a list and a form appears. The type of distribution (Gaussian or Uniform) is specified using the list. The form has fields for :

- The higher and lower nodes of node pair 1.
- A second pair of nodes (if the selected component needs them, for example, the voltage controlled current source.
- The percentage standard deviation. The absolute limit of component tolerances is usually found on component data sheets. Since the Gaussian distribution has about 99% of values in $\pm 3\sigma$, the standard deviation entered will be the data sheet tolerance divided by 3.

The selected component, it's nominal value, standard deviation, and type of distribution will be displayed on the main screen.

4.7.5 Show Components

It shows the list of components selected for the Monte-Carlo simulations on the main screen. The list also shows the associated nodes, the nominal value, the type of distribution (Gaussian or Uniform) and the standard deviation.

4.7.6 Perform

It performs a statistical analysis using the Monte-Carlo simulation with the selected components. A PROMPT subscreen with a list and a form appears. Interim results and a bar graph of the results can be selected from the list. The form has fields for :

- The node pair which will be used in the analysis.
- The number of simulations. The maximum number of simulations is 500.
- The seed for the random number generator. It should be a positive integer less than 32767.
- The impedance frequency which will be used in the calculation.

If the bar graph of the results is selected, another PROMPT subscreen will appear with a list from which the type of vertical scale is selected. The result[illegible]h simulation will be displayed on the main screen. Once the simulations are complete, the following will be displayed on the main screen :

- Nominal value
- Mean value
- Minimum value
- Maximum value
- Standard deviation
- Variance
- Three times the standard deviation (3 σ) of the node voltages.
- The quadrants in which the phase angles lie.

4.7.7 Test Random

It tests the random number generator. A PROMPT subscreen with a list appears. A Uniform or a Gaussian (normal) distribution can be selected from the list. Another PROMPT subscreen with a form appears which has fields for:

- The number of simulations.
- The seed for the random number generator.
- The standard deviation (0.0 - 290.0).

A bar graph is then drawn from the results.

4.7.8 Worst Case

It performs a worst case analysis using the Monte-Carlo simulation for selected components within a specified frequency range. This function is similar to that of the Perform function. It repeats the analysis at specific intervals along the defined frequency range and plots the maximum and minimum response. The Perform function does the analysis at one frequency only.

A PROMPT subscreen appears with a list and a form. The type of graph (vertical scale) is selected from the list. The form has fields for :

- The high and low node numbers of the node pair to be analysed.
- The number of simulations.
- The seed for the random number generator.
- The plot increment. The increment determines how often the Monte-Carlo simulation will be repeated along the horizontal frequency axis. The smaller the increment, the greater the accuracy, but the longer it takes to calculate.

Another PROMPT subscreen appears with a form which has fields for the start and final frequencies for the graph.

4.7.9 Worst Case 2

It performs an absolute worst case analysis within a specified frequency range. This function is similar to that of the Perform function. It alternately sets the component's value to its minimum and maximum value with respect to the standard deviation and type of distribution. For n components, 2^n simulations will be done. This can be a large number. For 10 components, this is 1024 simulations. The Perform function randomly varies the component's value.

A PROMPT subscreen appears with a list and a form. The type of graph (vertical scale) is selected using the list. The form has fields for :

- The high and low node numbers of the node pair to be analysed.
- The plot increment. The increment determines how often the Monte-Carlo simulation will be repeated along the horizontal frequency axis. The smaller the increment, the greater the accuracy, but the longer it takes to calculate.

Another PROMPT subscreen appears with a form which has fields for the start and final frequencies for the graph. The graph will display two plots that represent the upper and lower limits of the node voltage over the frequency range.

4.8 Optimisation

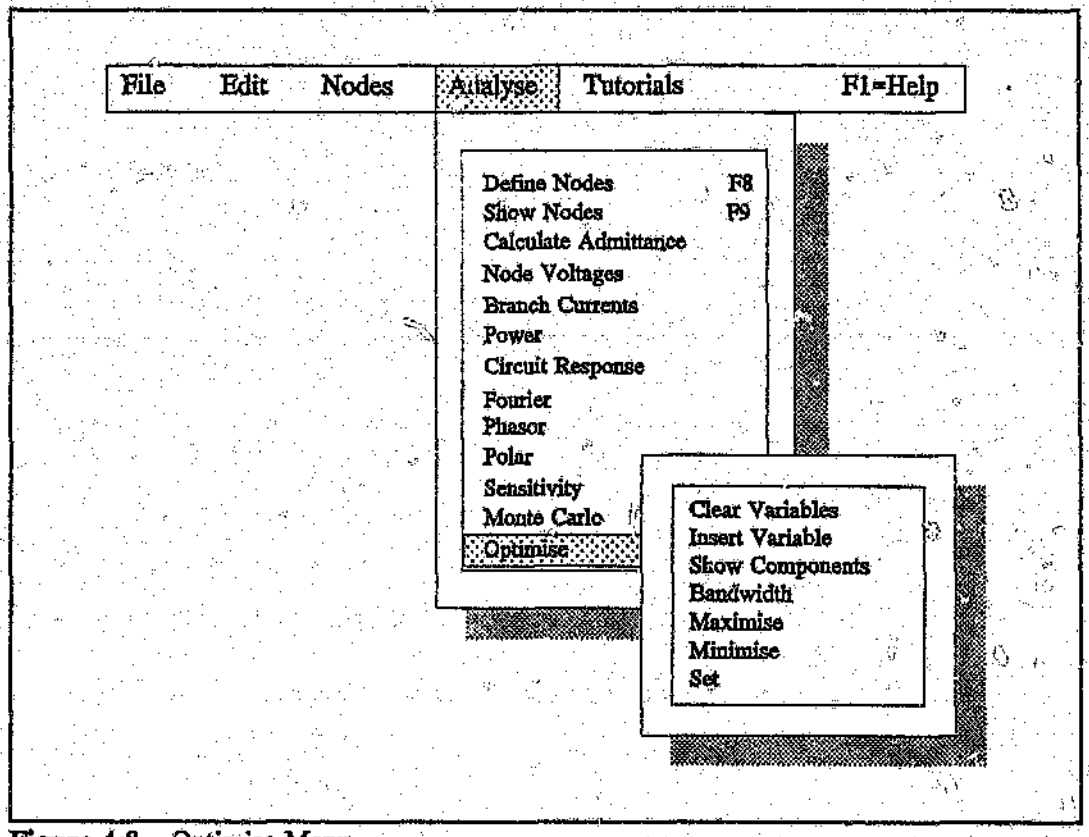


Figure 4-8 Optimise Menu

4.8.1 Clear Variables

It clears all the defined optimisation variables. A prompt asks if these variables must really be cleared. If *Yes* is chosen, all optimisation variables are cleared.

4.8.2 Insert Variables

It defines the components that are to be used as optimisation variables. A prompt appears with a list asking if the component's value, a branch's magnitude or it's phase, or the operating frequency must be optimised. At least one variable must be defined before any optimising can take place. More than one variable can be tuned at the same time. The variables that are defined can only be used once.

If the component's value is to be optimised, a PROMPT subscreen with the component list appears from which a component is selected. Another PROMPT subscreen appears with a form which has fields for:

- The higher and lower node numbers for node pair 1. This is the position of the component in the circuit.
- The higher and lower node numbers for node pair 2 will appear for components that need second node pair to be defined.

4.8.3 Show Components

It displays a list of the variables selected for optimisation. The list appears on the main screen.

4.8.4 Bandwidth

It automatically finds the bandwidth, the upper and lower corner frequencies, the centre frequency, the maximum gain and the Q-factor of bandpass response circuits. A PROMPT subscreen appears with a list and a form. Interim results are specified using the list. The form has fields for :

- The higher and lower node numbers of the node pair to be analysed.
- The impedance frequency.
- The percentage accuracy required.

When all the calculations are complete, the following results are displayed on the main screen :

- The centre frequency in Hertz.
- The maximum gain in volts and decibels.
- The upper and lower corner frequencies given in Hertz.
- The bandwidth.
- The Q factor.

4.8.5 Maximise

It maximises the response of a specified node pair by tuning selected component values and/or frequency. The local maximum (near the specified starting frequency) of the response is calculated. A PROMPT subscreen appears with a list and a form. Interim results are specified using the list. The form has fields for :

- The higher and lower node numbers of the node pair to be analysed.
- The impedance frequency.
- The percentage accuracy required.

Another PROMPT subscreen with a list appears. A parameter of the defined node pair's voltage response to be optimised is selected. When all the calculations are complete, the main screen will display the tuned component values and/or frequency. If optimisation is not achieved in 100 iterations, a prompt will appear asking if the analysis should be aborted.

4.8.6 Minimise

It minimises the response of a specified node pair by tuning component values and/or frequency. This function works the same way as in the Maximum case, except that the absolute error is required instead of the percentage accuracy.

4.8.7 Set

It tunes component values and/or frequencies so that the response of the selected node pair will be set at a specified value. A PROMPT subscreen appears with a list and a form. Interim results are specified using the list. The form has fields for :

- The higher and lower node numbers of the node pair to be analysed.
- The impedance frequency.
- The percentage accuracy required.

Another PROMPT subscreen with a list appears. A parameter of the defined node pair's voltage response to be optimised is selected. The value to set the defined node pair's response is prompted for.

CHAPTER 5

TUTORIAL EXAMPLE

5.1 Introduction

This simple example illustrates some of the features of Elector. The example chosen is a single stage common emitter amplifier.

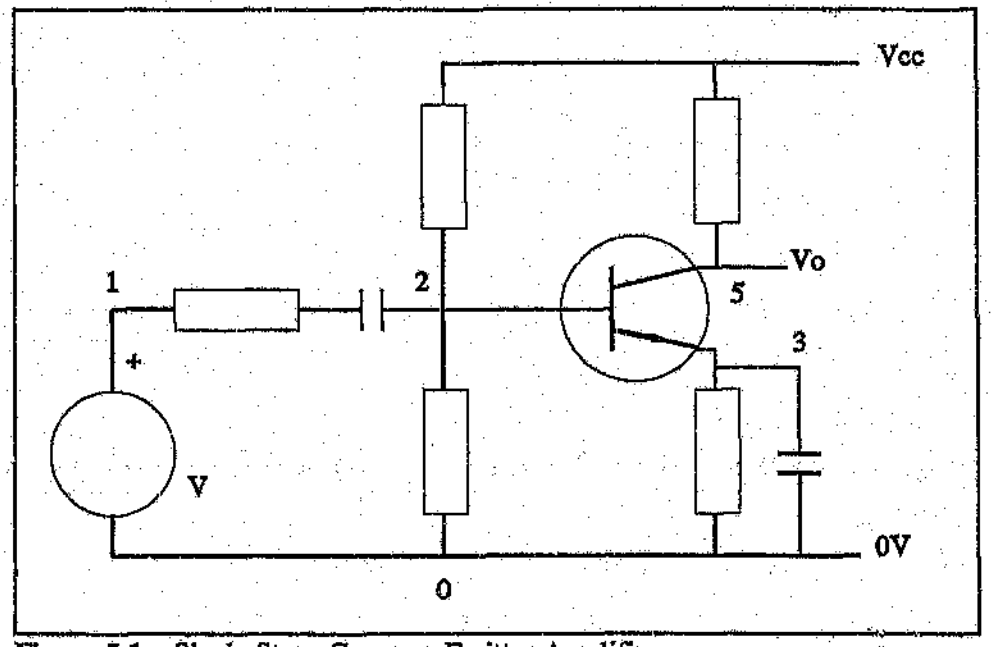


Figure 5-1 Single Stage Common Emitter Amplifier

5.2 Starting Elector

Type ELECTOR and press <ENTER> at the DOS prompt. You will see the Title scrolled onto the screen. Press any key to continue. The screen will be cleared and the Main menu will appear.

5.3 Loading a Circuit File

You will notice that most of the menu options cannot be selected. Only once a circuit has been loaded or entered, these options will become available. So, lets get started and load a circuit. Select the *File* option on the *Main* menu and then select the *Load* option on the *File* menu. Pressing the hot key *F2* will also select the *Load* option. A PROMPT subscreen will pop up displaying a list of circuit files in the default directory. This assumes that the file is in the default directory. If it is not, change the path using the *Drive* and/or *Directory* options on the *File* menu.

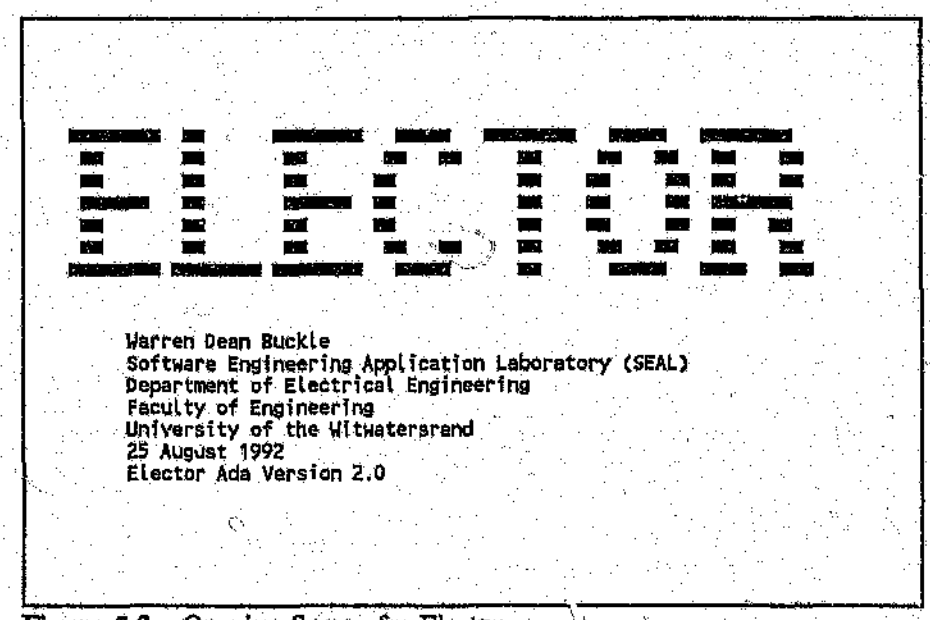


Figure 5-2 Opening Screen for Elector

Choose CE-AMP.NTL and press the *Ok* button. The <ENTER> key will also press the Ok button. The circuit is loaded and the program returns to the Main menu. You will notice that many more of the menu options can be selected.

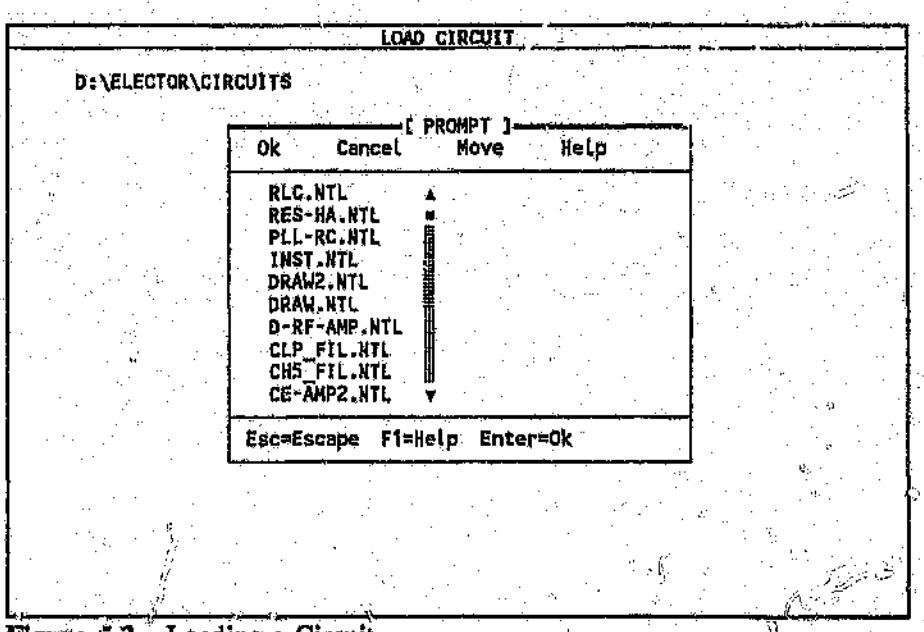


Figure 5-3 Loading a Circuit

5.4 Listing the Circuit Components

Now, lets see what this circuit consists of. To see what components this circuit consists of, select the *List Components* option on the *Edit* menu. To get there, Press *E* followed by *L*, or press the hotkey *F6*. A PROMPT subscreen with a list will appear on the main screen. There are three

different forms that the listing can take. Let's choose the *RLC* option. This is done by moving the highlighting bar to that option. Now press the *Ok* button. The list of components is displayed on the main screen.

To display the components in a particular circuit branch, select *Show Branch* on the *Edit* menu. A PROMPT subscreen will appear asking for the upper and lower nodes of the branch. Let's choose the branch between nodes 3 and 4. The components in the branch will be displayed on the main screen.

File Edit Nodes Analyse Tutorials F1=Help

Component	Nodes	Value	Units	Additional Data
R	1 0	1.0000 K	Ohms	
V	1 0	1.0000	V	Phase = 0.00000 Degrees
C	2 1	2.0000 u	F	
R	2 0	21.2600 K	Ohms	
R	4 2	50.0000	Ohms	
C	4 3	4.0400 p	F	
R	4 3	574.0000	Ohms	
C	3 0	100.0000 u	F	
R	3 0	100.0000	Ohms	
C	5 4	882.0000 f	F	
R	5 3	22.2000 K	Ohms	
C	5 0	2.0000 p	F	
R	5 0	1.0000 K	Ohms	
GM	5 3	181.0000 m	MHos	Controlling nodes 4 3

Figure 5-4 Listing the Circuit Components

File Edit Nodes Analyse Tutorials F1=Help

Component	Nodes	Value	Units	Additional Data
C	4 3	4.0400 p	F	
R	4 3	574.0000	Ohms	

Figure 5-5 List of Components in a Circuit Branch

5.5 Viewing the Circuit

Let's see what the circuit looks like. Select the *Draw Circuit* on the *Edit* menu. A PROMPT subscreen with a list will appear on which the default node placement can be selected. The fast drawing mode can also be selected.

If you are unsure about a choice on the menus or a part of the program, press the *Help* button. The help is context sensitive and usually deals with that part of the program. If there is not specific help, general help on that particular subscreen will appear.

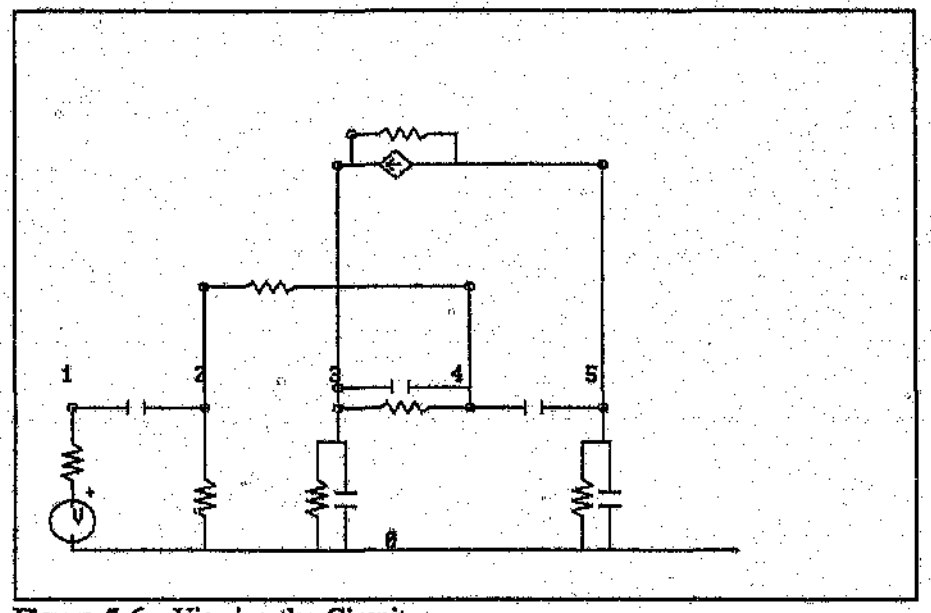


Figure 5-6 Viewing the Circuit

5.6 Viewing the Nodes to be Analyzed

The nodes that are to be analyzed can be seen by choosing *Show Nodes* on the *Analyse* menu. Pressing the hotkey *F9* will also show the nodes.

5.7 Calculation of the Node Voltages

To calculate the node voltages, choose *Node Voltages* on the *Analyse* menu. A PROMPT subscreen displaying the default frequency will appear and ask you if it is acceptable. Let's assume that you want 2.0 kHz. Press the No button. Another PROMPT subscreen will appear. Enter the frequency 2000.0. The node voltages will now appear on the main screen.

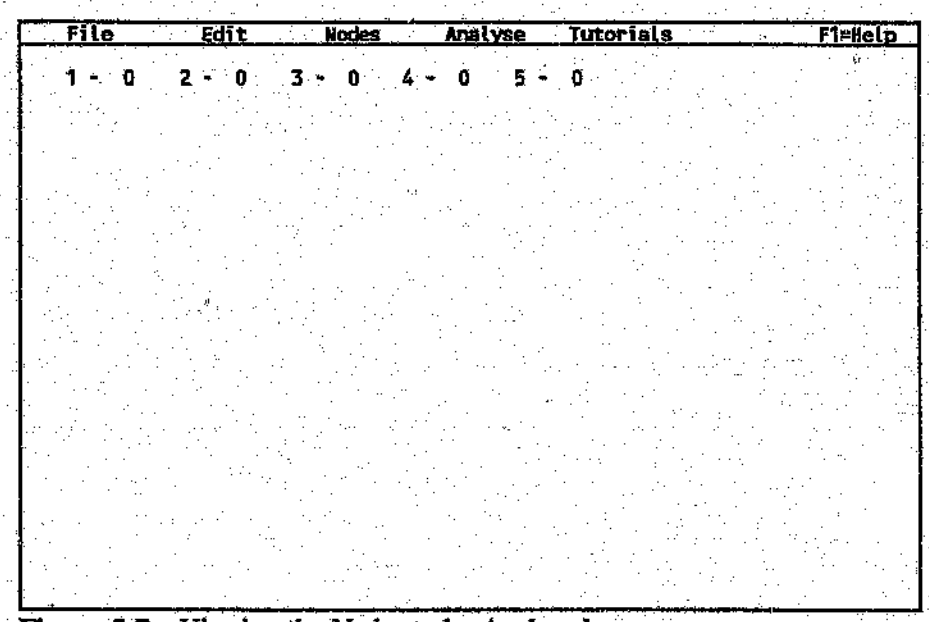


Figure 5-7 Viewing the Nodes to be Analyzed

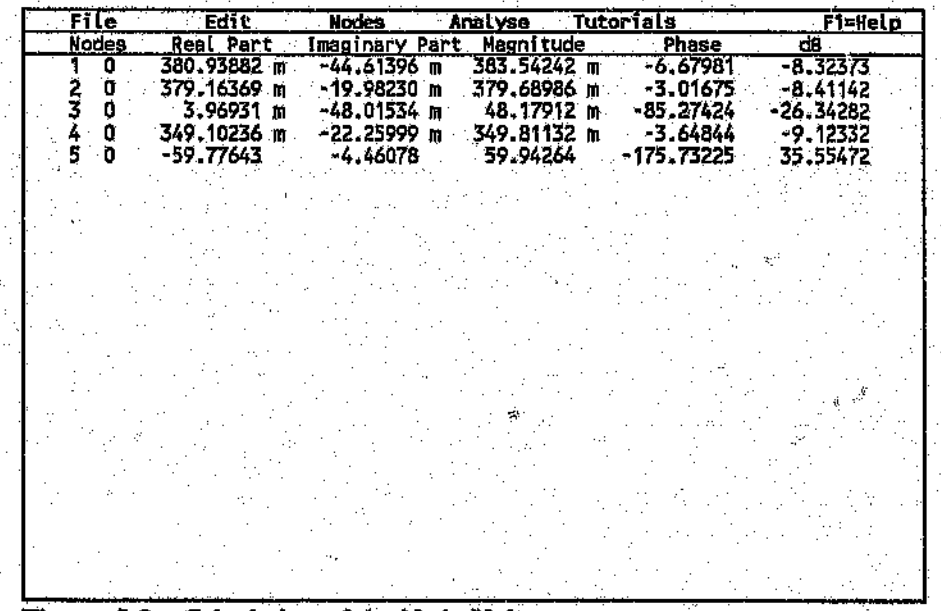


Figure 5-8 Calculation of the Node Voltages

5.8 Calculation of Branch Currents

Now, let's calculate the current in the different branches of the circuit. Select the *Branch Currents* on the *Analyse* menu. You will be again be asked about the default frequency. Let's leave it at 2.0 kHz by pressing the *Yes* or *Ok* button, or pressing the <ENTER> key. The branch currents appear on the main screen.

File Edit Nodes Analyse Tutorials F1=Help

Nodes	Real Part	Imaginary Part	Magnitude	Phase	dB
1 0	-619.06118 u	-44.61396 u	620.66670 u	-175.87798	-64.14283
2 0	17.83460 u	-939.90123 n	17.85935 u	-3.01675	-94.96269
3 0	60.37755 m	4.50783 m	60.54559 m	4.26982	-24.35835
4 0	No branch exists				
5 0	-59.77632 m	-4.46228 m	59.94264 m	-175.73081	-24.44528

Figure 5-9 Calculation of Branch Currents

5.9 Calculation of Absorbed Power

The absorbed power in the circuit can be calculated by selecting *Power* on the *Analyse* menu. You will be again be asked about the default frequency. Let's leave it at 2.0 kHz by pressing the *Yes* or *Ok* button, or pressing the <ENTER> key. A PROMPT subscreen appears with a list with options for *Peak* or *RMS* values. Choose *RMS*. The branch currents appear on the main screen.

File Edit Nodes Analyse Tutorials F1=Help

Nodes	Real	Reactive	Capactive	Inductive	Apparent
1 0	192.61358 u	0.00000	0.00000	0.00000	192.61358 u
2 0	3.39051 u	0.00000	0.00000	0.00000	3.39051 u
3 0	11.60614 u	1.45847 m	1.45847 m	0.00000	1.45852 m
4 0	Branch does not exist				
5 0	1.79656	45.15248 u	45.15248 u	0.00000	1.79656

Nodes	Impedance	Phase	Power Factor
1 0	1.00000 K	0.00000	1.00000
2 0	21.26000 K	0.00000	1.00000
3 0	795.74952 m	-89.54406	7.95750 m
4 0	Branch does not exist		
5 0	1.00000 K	-0.00144	1.00000

Figure 5-10 Calculation of Absorbed Power

5.10 Changing the Nodes to be Analyzed

To change the nodes to be analyzed, select *Define Nodes* on the *Analyse* menu. Lets choose only one node pair to analyse, say node 5-0. Enter 5 and then press the *Ok* button. Next enter 0 followed by pressing the *Ok* button.

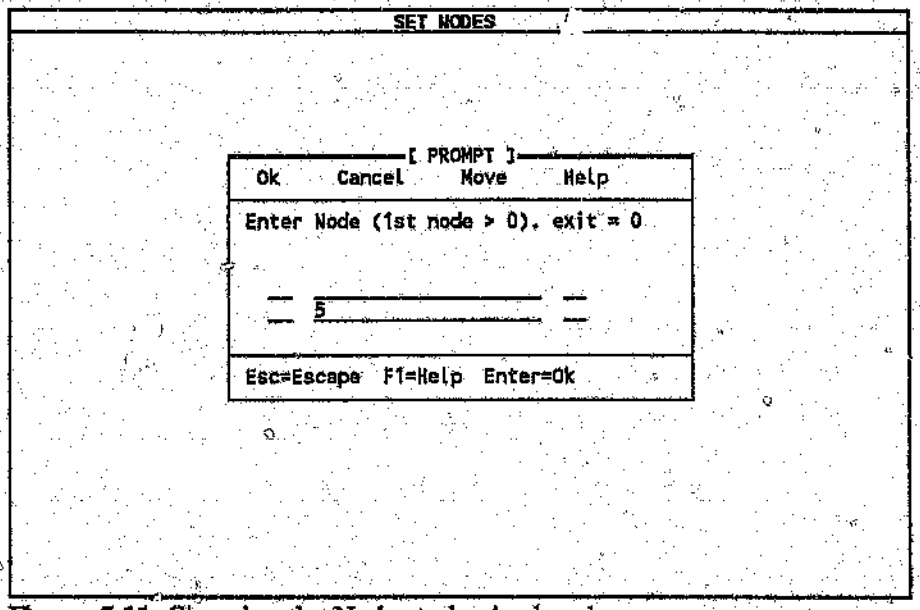


Figure 5-11 Changing the Nodes to be Analyzed

5.11 Time Domain Circuit Response

To see how the circuit responds in the time domain, select *Fourier* on the *Analyse* menu. A PROMPT subscreen with a list and a form appears. The list shows a number of different waveforms. Lets choose the *Square* waveform by moving the highlighting bar to that option. Now lets fill in the form. Press the tab key and enter the following :

- The number of harmonics. Remember, the more harmonics, the greater the accuracy, but the longer it takes to calculate. Choosing 200 harmonics will give a fairly accurate response.
- The node pair for which the plot is to be generated. We'll choose nodes 5-0.
- The start and finishing times of the graph. Choose 0.0 and 2.0e-6 respectively.
- The impedance frequency. Let's assume that you want 1.0 MHz.

This will take some time to generate, so a little patience is required.

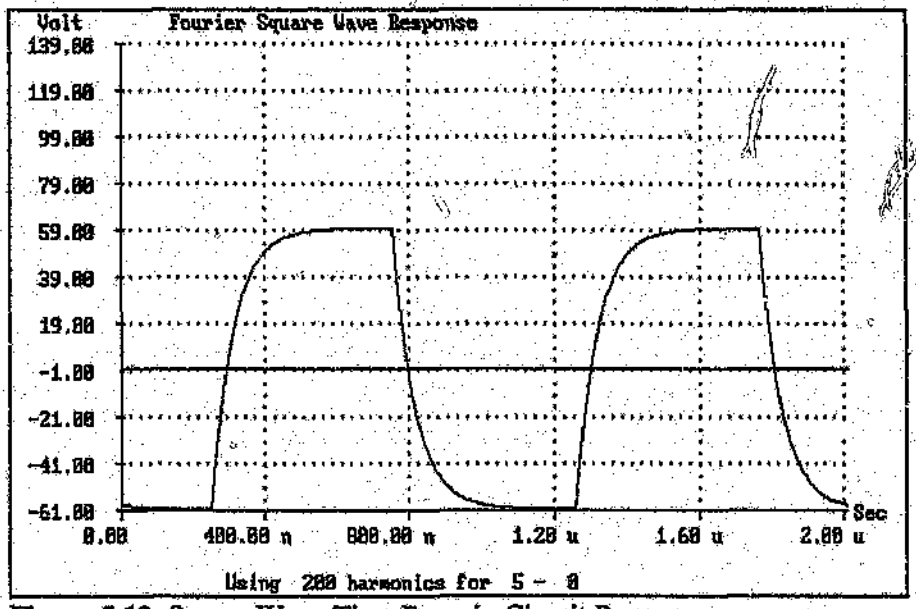


Figure 5-12 Square Wave Time Domain Circuit Response

5.12 Frequency Domain Circuit Response

To see how the circuit responds in the frequency domain, select *Circuit Response* on the *Analyse* menu. Next, choose the *Log Frequency* option from the list that is displayed on a PROMPT subscreen. Another PROMPT subscreen with a form appears. The form requests the start and ending frequency of the graph. Choose 10.0 Hz and 10.0 Mhz respectively. Press the *Yes* button to the question of autoscaling. Finally, choose the *Bode* option from the list of possible graphs and fill in the field for plot increment to 5. This will take some time to generate, so a little patience is required.

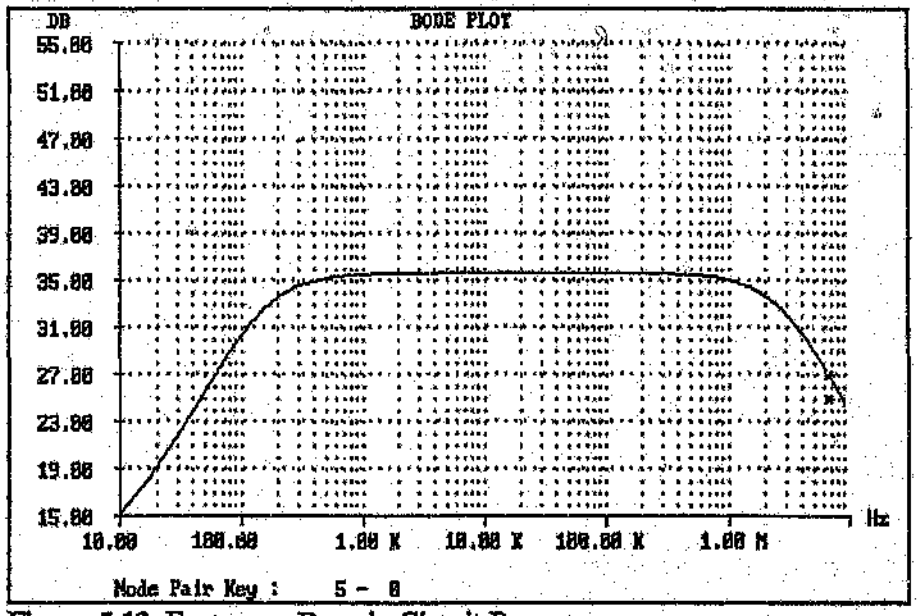


Figure 5-13 Frequency Domain Circuit Response

5.13 Effect of Component Sensitivities

What happens if our circuit has components whose values vary by some amount. To see the effect on the circuit's performance, use the *Sensitivity* option on the *Analyse* menu. Lets assume that the resistor, capacitor and inductor values vary 10.0 % about their nominal values. When asked for a perturbation step size, enter 10.0 followed by pressing the *Ok* button.

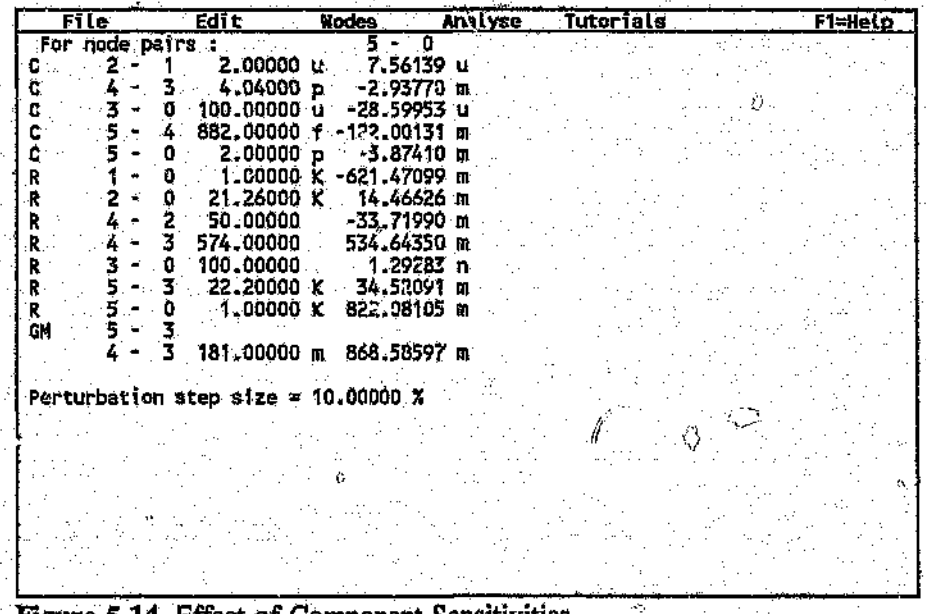


Figure 5-14 Effect of Component Sensitivities

5.14 Monte Carlo Analysis

The Monte-Carlo Analysis is an advanced statistical method that helps simulate the effect of a component that has a certain tolerance on the performance of the circuit. To see how this works, lets assume we have a capacitor between nodes 3 and 0 that has a tolerance of 10.0 %, and that we want to see what the effect would be on the node pair 5-0. First select *Monte-Carlo* on the *Analysis* menu followed by *Insert Monte* on the *Monte-Carlo* menu. Select the capacitor on the list of components that is displayed. Next, select the Gaussian distribution on the list, and fill in the form - 10.0 for the percentage standard deviation, 3 for the higher node and 0 for the lower node. To verify this information, use the *Show Components* on the *Monte-Carlo* menu.

To actually perform the simulation, select *Perform*. Enter 5-0 for the node pair, 50 for the number of simulations, 123 for the seed for the Random Number Generator and 100.0 Hz for the impedance frequency. Interim values can be displayed and a bar graph can be obtained from the results. After the iterations, the results will appear on the main screen.

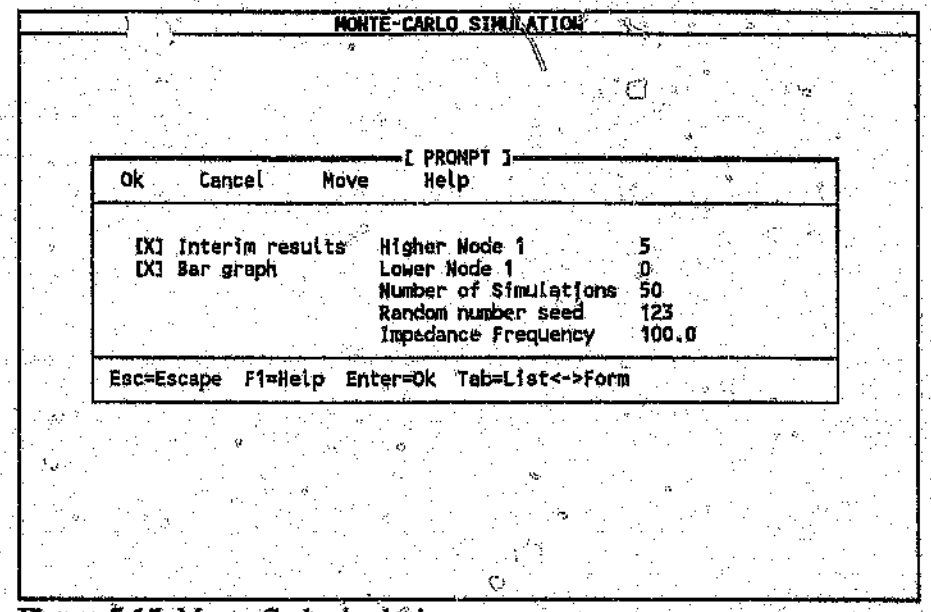


Figure 5-15 Monte-Carlo Analysis

A worst case frequency plot can be obtained for the node pair 5-0 by selecting *Worst Case* on the *Monte-Carlo* menu. Lets choose 10 simulations with 123 for the seed and 5 for the plot increments, and a frequency range from 1.0 Hz to 10.0 kHz on a Bode plot.

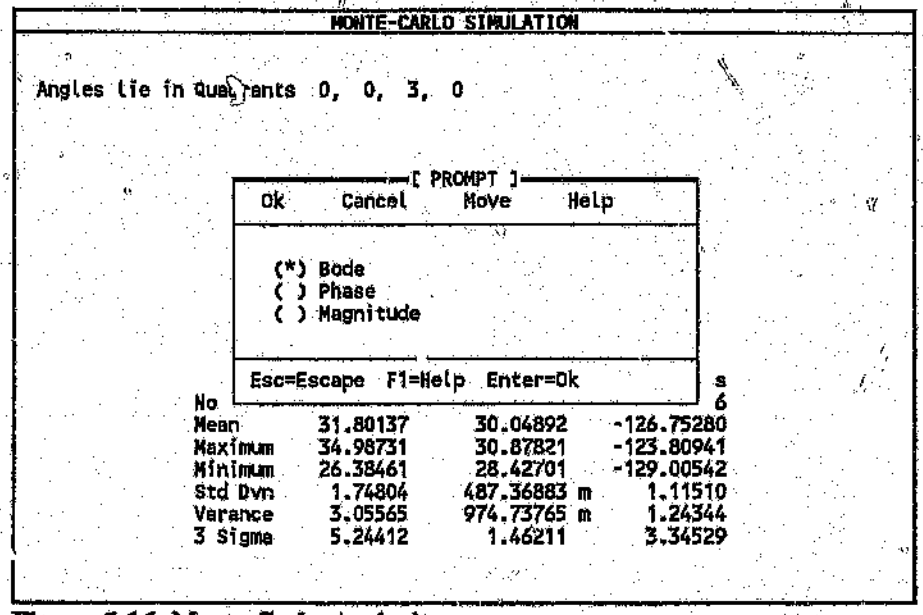


Figure 5-16 Monte-Carlo Analysis

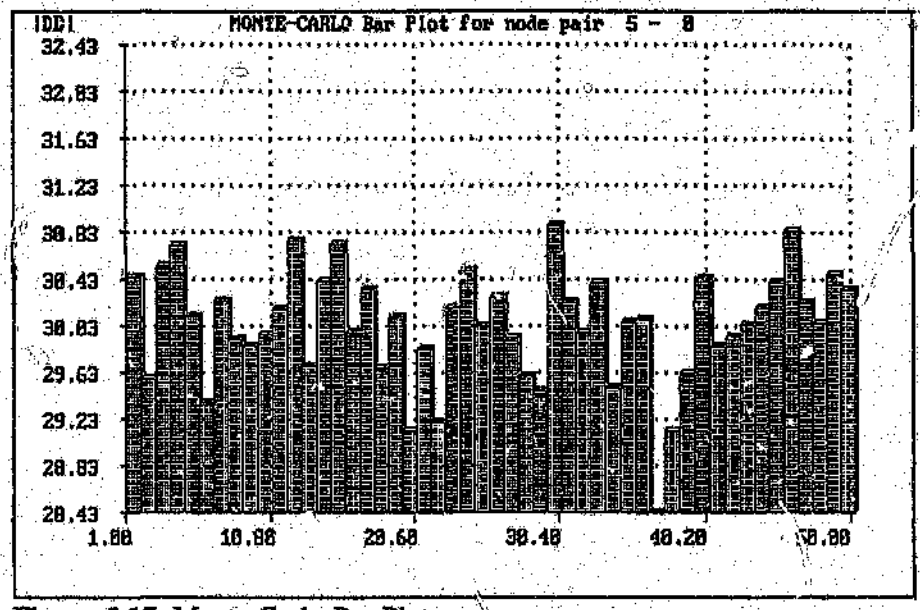


Figure 5-17 Monte-Carlo Bar Plot

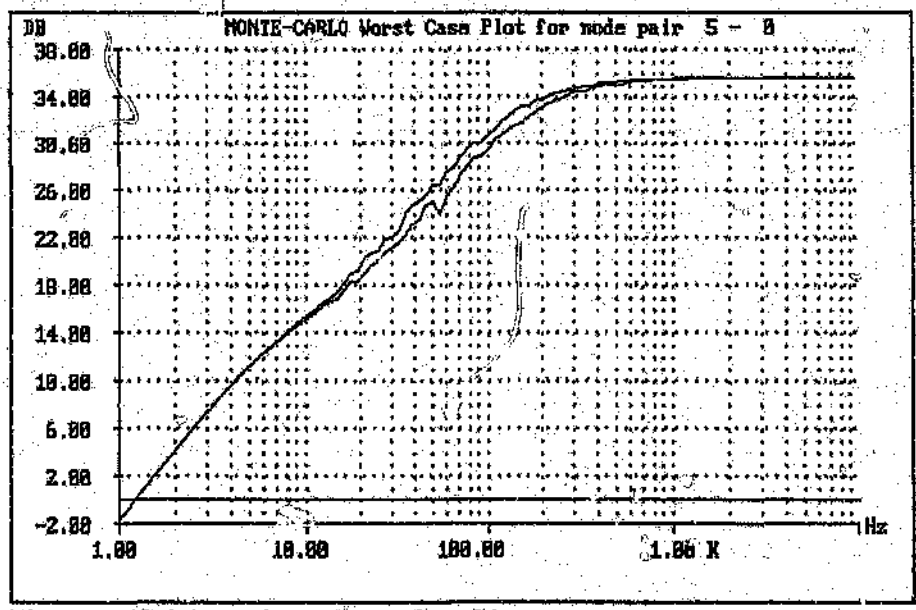


Figure 5-18 Monte-Carlo Worst Case Plot

5.15 Finding the Bandwidth

To find the bandwidth, Q factor, etc. select *Optimise* on the *Analyse* menu. Next, select *Bandwidth* on the *Optimise* menu. Lets choose an accuracy of 99.99 % and a default impedance frequency of 1.0 Khz at the node pair 5-0.

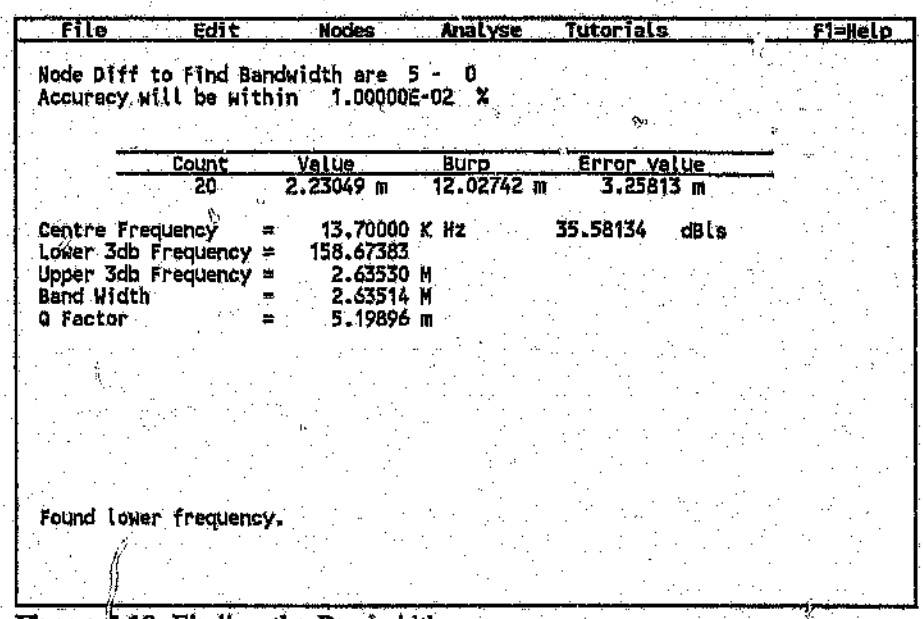


Figure 5-19 Finding the Bandwidth

5.16 Help

There is on-line help for Elector. It is context-sensitive, ie. help information relevant to that particular part of the program is displayed. When a subscreen appears, help is obtained by pressing the *Help* button. If there is no specific help when the subscreen's help button is pressed, help information on that type of subscreen will appear. When selecting an item on the menu, pressing the *F1* key will display help information for the highlighted menu item.

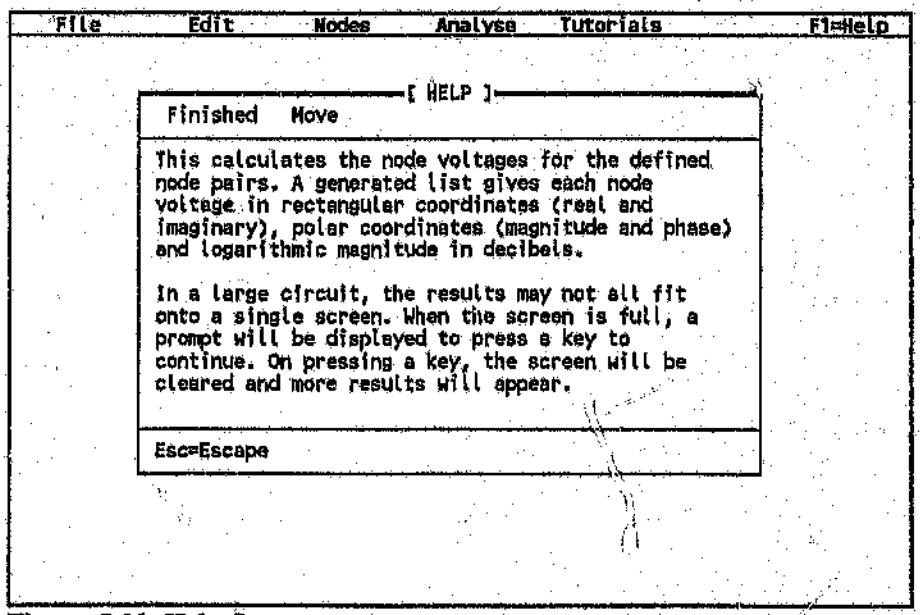


Figure 5-20 Help Screen

5.17 Tutorials

The tutorials cover important aspects of the program - the user interface, etc. Also, because Elector is meant to be a learning and teaching aid to electrical engineering students, it has information on basic electrical engineering equations and circuits. It can also be used by other branches of engineering as well. Many of the equations and models used in these branches of engineering have electrical circuit analogues. Thus, some information is also given for these branches.

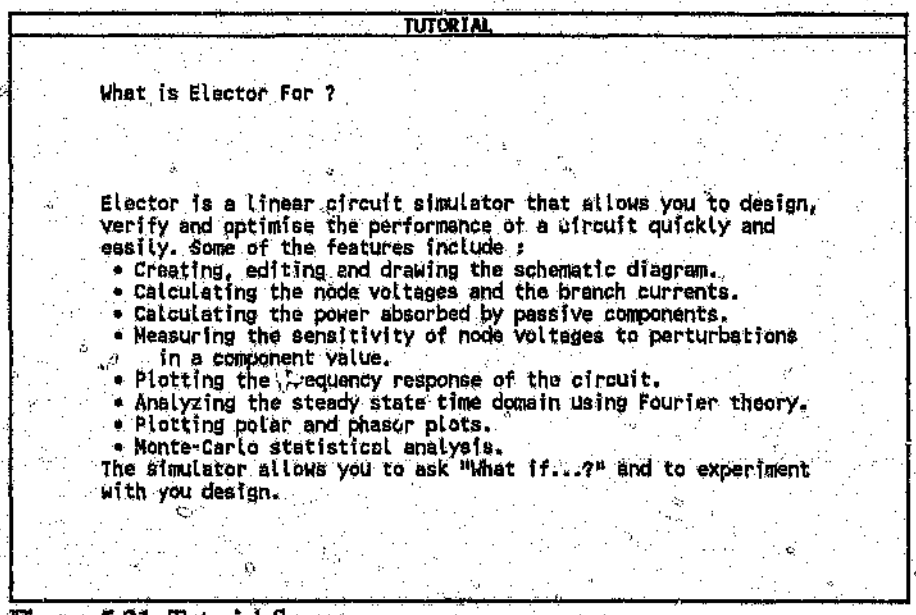


Figure 5-21 Tutorial Screen

REFERENCES

Lipworth, A.D. (1989) *Elecsim - An Integrated Interactive Linear Electric Network Simulation And Optimisation Package*, MSc Dissertation, University of the Witwatersrand, Johannesburg, South Africa, 1989

SEAL, (1990) *Elecsim : User and Reference Manual*, Version 1.0, SEAL, University of the Witwatersrand, Johannesburg, South Africa, 1990

Technical Reference Manual

Author : **Warren Buckle**
Revision : **1.00**
Date : **21 / 05 /93**

CONTENTS

CONTENTS ii

LIST OF FIGURES x

LIST OF TABLES xii

LIST OF EQUATIONS xiii

CHAPTER 1 : INTRODUCTION 1

1.1 Chapter Summaries 1

CHAPTER 2 : DESIGN ISSUES 3

2.1 Ada as a PDL 3
2.2 Elector's User Interface 3
2.3 Turbo Vision and CXL C Library 3
2.4 Partitioning 4
2.5 Meridian Ada Memory Organisation 4
2.6 Initialising Variables 5
2.7 Named Parameter Association 5
2.8 Parameter Modes 5
2.9 Overloading Subprograms 6
2.10 Packages 6
2.11 Exceptions 6
2.12 Dynamic Strings 7
2.12.1 Behaviour 8
2.12.2 Components 8
2.12.3 Methods 8
2.13 Maths 10
2.13.1 Arctan Function 10
2.13.2 Cos Function 10
2.13.3 Exponent Function 10
2.13.4 Logarithm to Base 10 10
2.13.5 Sine Function 11
2.14 Code Layout 11
2.14.1 Upper and Lower Case 11
2.14.2 Indenting 11
2.14.3 Parameter Lists 11

2.14.4 Extended Names 12
2.14.5 Comments 12
2.15 Managing Memory 12
2.15.1 Pascal Overlays 12
2.15.2 Suppressing Checks 12
2.15.3 Removing Source Code 14

CHAPTER 3 : SOFTWARE TOOLS 15

3.1 Introduction 15
3.2 Editors 15
3.2.1 Turbo Pascal 15
3.2.2 Brief V3 15
3.3 Compilers and Assemblers 18
3.3.1 Assembler : MASM 5 18
3.3.2 Ada : Meridian AdaVantage 2.1 18
3.3.3 Ada : Meridian Open Ada 18
3.3.4 Linkers : TLINK V2 (Turbo Linker V2) 19
3.4 Debugging 20
3.4.1 Turbo Debugger 2.5 20
3.5 Documentation 20
3.5.1 Text and Final Formatting : WordPerfect 5.1 20
3.5.2 Graphics : Harvard Graphics 2.3 and DrawPerfect 20

CHAPTER 4 : ELECTOR OBJECTS 21

4.1 Introduction 21
4.2 Size of the Elector Packages 21
4.3 Ada_Extensions 24
4.3.1 Purpose 24
4.3.2 Construction 24
4.3.3 Data Types and Structures 24
4.3.4 Methods 25
4.4 Analysers 26
4.4.1 Purpose 26
4.4.2 Methods 26
4.5 Central_Solver 27
4.5.1 Purpose 27
4.5.2 Data Types and Structures 27
4.5.3 Methods 28

4.6 Dos_Environment 28
4.6.1 Purpose 28
4.6.2 Construction 28
4.6.3 Data Types and Structures 29
4.7 Drawer 29
4.7.1 Purpose 29
4.7.2 Data Types and Structures 29
4.7.3 Methods 29
4.8 Graphs 32
4.8.1 Purpose 32
4.8.2 Methods 32
4.9 Help 33
4.9.1 Purpose 33
4.9.2 Construction 33
4.10 Inputter 33
4.10.1 Purpose 34
4.11 Menus 34
4.11.1 Purpose 34
4.12 Monte 34
4.12.1 Purpose 34
4.12.2 Data Types and Structures 35
4.12.3 Methods 35
4.13 Netlist_Suppor... 36
4.13.1 Purpose 36
4.13.2 Methods 36
4.14 Netlists 36
4.14.1 Purpose 36
4.14.2 Construction 37
4.15 Optimiser 37
4.15.1 Purpose 37
4.15.2 Data Types and Structures 37
4.15.3 Methods 37
4.16 Plotter 38
4.16.1 Purpose 38
4.16.2 Data Types and Structures 38
4.16.3 Methods 39
4.17 Scheduler 40
4.17.1 Purpose 40
4.17.2 Methods 41
4.18 Screen_Objects 41
4.18.1 Purpose 41
4.19 Screens 41

4.19.1 Purpose 42
4.20 Tutorials 42
4.20.1 Purpose 42
4.20.2 Construction 42
4.20.3 Data Types and Structures 42
4.20.4 Methods 43

CHAPTER 5 : DATAFLOW DIAGRAMS AND PDL 44

5.1 Introduction 44
5.2 The First Level of the Elector Design 44
5.3 The Second Level of the Elector Design 45
5.4 The Third Level - Editing Processes 47
5.5 The Third Level - Analysis Processes 53
5.6 The Third Level - Optimisation Processes 62

CHAPTER 6 : THE NETLISTS ABSTRACT DATA TYPE 67

6.1 Introduction 67
6.2 Behaviour 67
6.2.1 Introduction 67
6.2.2 Exceptions 68
6.3 Components 69
6.3.1 Circuit Components 69
6.3.2 Linked List Structure 70
6.3.3 Base Structure 71
6.3.4 Branch Data Structure 71
6.3.5 General Component Branch Structure 72
6.3.6 Inductive Component Branch Structure 72
6.3.7 Mutual Inductive Component Branch Structure 72
6.3.8 Dependent Source Component Branch Structure 73
6.4 Methods 73
6.4.1 Services Provided 73
6.4.2 Screen_Supporter Methods 75
6.4.3 Inner Methods 75
6.4.4 Outer Methods 88

CHAPTER 7 : DOS ENVIRONMENT PACKAGE 97

7.1 Introduction 97
7.2 Interfacing to Other Languages 97
7.2.1 Pragma Interface 97
7.2.2 Meridian Ada Interface Implementation 97
7.2.3 Interface to Assembly Language 98
7.2.4 Stack Frames 98
7.3 Representations of Various Types 99
7.3.1 Discrete Types 100
7.3.2 Real Types 100
7.3.3 Array Types 100
7.3.4 Record Types 101
7.3.5 Access Types 101
7.4 System Features 101
7.4.1 BIOS, DOS and Mouse Interrupts 101
7.5 Buffering Concepts 104
7.5.1 Method 1 104
7.5.2 Method 2 105
7.5.3 Buffer Implementation 108
7.5.4 Shadows 109
7.5.5 Highlighting Bar 109
7.5.6 Screen Setup 110
7.6 Mouse Pointer 110
7.7 The Structure of Assembler Subroutines 111
7.7.1 Frame Buffer Setup 111
7.7.2 Subprogram Parameters 111
7.8 Assembly Output from the Ada Compiler 112
7.9 Library Management (Version 2.1) 113
7.10 Dos_Environment Subprograms 113

CHAPTER 8 : SCREEN OBJECTS 117

8.1 Introduction 117
8.2 Coordinates 117
8.3 Inputter Object 118
8.3.1 Purpose 118
8.3.2 Appearance and Behaviour 118
8.3.3 Components 118
8.3.4 Methods 118
8.4 Buttons 119

8.4.1 Purpose . . . 119
8.4.2 Appearance and Behaviour . . . 120
8.4.3 Components . . . 120
8.4.4 Methods . . . 121
8.5 Boolean_Buttons . . . 122
8.5.1 Purpose . . . 122
8.5.2 Appearance and Behaviour . . . 122
8.5.3 Components . . . 123
8.5.4 Methods . . . 123
8.6 Input_Lines . . . 124
8.6.1 Purpose . . . 124
8.6.2 Appearance and Behaviour . . . 124
8.6.3 Components . . . 125
8.6.4 Methods . . . 126
8.7 Pick_Lists (Static) . . . 129
8.7.1 Purpose . . . 129
8.7.2 Appearance and Behaviour . . . 129
8.7.3 Components . . . 130
8.7.4 Methods . . . 130
8.8 Pick_Lists (Dynamic) . . . 133
8.8.1 Purpose . . . 133
8.8.2 Appearance and Behaviour . . . 133
8.8.3 Components . . . 134
8.8.4 Methods . . . 135
8.9 Forms . . . 138
8.9.1 Purpose . . . 138
8.9.2 Appearance and Behaviour . . . 138
8.9.4 Components . . . 139
8.9.5 Methods . . . 140
8.10 Windows . . . 143
8.10.1 Purpose . . . 143
8.10.2 Appearance and Behaviour . . . 144
8.10.3 Components . . . 144
8.10.4 Methods . . . 145

CHAPTER 9 : THE HELP OBJECT . . . 150

9.0.1 Purpose . . . 150
9.0.2 Appearance and Behaviour . . . 150
9.0.3 Components . . . 151
9.0.4 Methods . . . 152

CHAPTER 10 : THE SCREENS OBJECT 154

10.1 Introduction 154
10.2 Subscreen 154
10.2.1 Purpose 154
10.2.2 Appearance and Behaviour 154
10.2.3 Components 156
10.2.4 Methods 157
10.3 Error Subscreen 162
10.3.1 Purpose 162
10.3.2 Methods 163
10.4 Prompt Subscreen 163
10.4.1 Purpose 163
10.4.2 Methods 164
10.5 Status 169
10.5.1 Purpose 169
10.5.2 Methods 169
10.6 Main Screen 170
10.6.1 Purpose 170
10.6.2 Appearance and Behaviour 170
10.6.3 Methods 171

CHAPTER 11 : THE MENUS OBJECT 177

11.1 Introduction 177
11.1.1 Purpose 177
11.1.2 Appearance and Behaviour 177
11.1.3 Components 177
11.1.4 Methods 179
11.2 Spread Menu 183
11.2.1 Purpose 183
11.2.2 Appearance and Behaviour 183
11.2.3 Methods 184
11.3 Stacked Menu 186
11.3.1 Purpose 186
11.3.2 Appearance and Behaviour 186
11.3.3 Methods 187

CHAPTER 12 : TUTORIAL SUPPORT PROGRAM 190

12.1 Purpose 190
12.2 Behaviour 190
12.3 Tutorial Index 191
12.4 Creating Tutorial Screens 192

CHAPTER 13 : TESTING AND VALIDATING ELECTOR 193

REFERENCES 195

LIST OF FIGURES

Figure 2-1 The Main Objects of Elector 4
Figure 5-1 The First Level of the Elector Simulation and Optimisation Program 44
Figure 5-2 The Editing Process 45
Figure 5-3 The Analysis Process 46
Figure 5-4 The Optimisation Process 47
Figure 5-5 The Change Magnitude and Phase Process 48
Figure 5-6 The Insert Process 49
Figure 5-7 The List Process 50
Figure 5-8 The Schematic Drawing Process 51
Figure 5-9 The Show Component Process 52
Figure 5-10 The Show Circuit Branch Process 52
Figure 5-11 The Admittance Matrix Process 54
Figure 5-12 The Circuit Response Process 54
Figure 5-13 The Fourier Analysis Process 55
Figure 5-14 The Monte-Carlo Statistical Analysis Process 57
Figure 5-15 The Node Voltage Process 58
Figure 5-16 The Phasor Diagram Process 59
Figure 5-17 The Polar Plotting Process 59
Figure 5-18 The Power Process 60
Figure 5-19 The Sensitivity Process 61
Figure 5-20 The Optimisation Clear Process 62
Figure 5-21 The Optimisation Find Process 63
Figure 5-22 The Optimisation Insert Process 64
Figure 5-23 The Maximisation, Minimisation and Set Process 65
Figure 5-24 The Show Optimisation Variables Process 66
Figure 6-1 The Netlist_Type Linked List Structure 70
Figure 6-2 Netlist_Type Base Structure 71
Figure 6-3 Branch Data Structure 71
Figure 6-4 General Branch Components 72
Figure 6-5 Inductive Branch Components 72
Figure 6-6 Mutual Inductive Component Branch Structure 73
Figure 6-7 Dependent Source (Voltage Controlled Current Source) 73
Figure 7-1 The Stack Frame 99
Figure 7-2 Array Object Implementation 100
Figure 7-3 Unconstrained Array Object Implementation 101
Figure 7-4 Screen Buffer (Method 1) 105
Figure 7-5 Buffer Coping Process (Method 1) 106
Figure 7-6 Buffer Coping Process (Method 2) 107
Figure 7-7 Matrix and String Representation 108
Figure 8-1 Button Object (1st) 119
Figure 8-2 Button Object (2nd) 119

Figure 8-3 Boolean_Button Object (1st) 122
Figure 8-4 Boolean_Button Object (2nd) 122
Figure 8-5 The Input_Line Object 124
Figure 8-6 The Static Pick_List Object 129
Figure 8-7 The Dynamic Pick_List Object 133
Figure 8-8 The Form Object 138
Figure 8-9 The Window Object 144
Figure 9-1 The Help Object 150
Figure 10-1 The Subscreen Object 154
Figure 10-2 A Subscreen Object consists of several Screen Objects. 155
Figure 10-3 The Error Subscreen Object 162
Figure 10-4 The Prompt Subscreen Object (Input_Line) 164
Figure 10-5 The Prompt Subscreen Object (Pick_List & Form) 164
Figure 10-6 The Status Subscreen Object 169
Figure 11-1 The Spread Menu Object 183
Figure 11-2 The Stacked Menu Object 186

LIST OF TABLES

Table 2-1	Error Codes in Elecsim and Exceptions in Elector	7
Table 2-2	Runtime Checks	13
Table 2-3	Removed Source Code	14
Table 3-1	Brief Ada Keyword Elaboration	17
Table 4-1	Smaller Packages within the Major Packages of Elector	21
Table 4-2	Line, Statement and Comment count for Elector Packages	22
Table 6-1	Exceptions raised by the Netlists Package	69
Table 6-2	Circuit Components and Elector's Internal Representation	69
Table 6-3	Services provided by the Netlists Package	74
Table 7-1	Discrete Type Representation	100
Table 7-2	Real Type Representation	100
Table 7-3	System Features	102
Table 7-4	BIOS Functions	103
Table 7-5	DOS Functions	103
Table 7-6	Mouse Functions	104
Table 7-7	Buffer Operations	109
Table 8-1	Input_Line Key Definition	124
Table 8-2	Pick_List Characteristics	129
Table 8-3	Pick_List (Static) Key Definition	130
Table 8-4	Form Key Definition	139
Table 10-1	Purpose of the Subscreen Buttons	156
Table 10-2	Different types of Prompt Subscreen Inputs	163
Table 11-1	Spread Menu Key Definition	184
Table 11-2	Stack Menu Key Definition	187
Table 12-1	Tutorial Support Program Key Definition	190
Table 12-2	Tutorial Options used in Elector	191
Table 13-1	Test Subprograms	193
Table 13-2	Test Circuits	194

LIST OF EQUATIONS

Equation 2-1	Taylor Series for Arctan Function	10
Equation 2-2	Equation for Cosine Function	10
Equation 2-3	Taylor Series for Exponent Function	10
Equation 2-4	Logarithm to base 10 Formula	10
Equation 2-5	Taylor Series for Sine Function	11

CHAPTER 1

INTRODUCTION

This document describes the construction and behaviour of *Elector* from a programmer's perspective. Each chapter covers a certain aspect of *Elector*. The user interface is build as a set of abstraction layers. Each *layer* is discussed in a separate chapter.

1.1 Chapter Summaries

- *Design Issues* : Discusses the basic design approach and ideas. Topics include the development of dynamic strings, exceptions, code layout and memory solutions.

- *Software Tools* : Discusses the different tools used in the development of Elector. This includes compilers and editors for the source code and wordprocessors and graphic drawing programs for the documentation.

- *Elector Objects* : Describes the main objects from which Elector is constructed. These objects are implemented in separate compilation modules. Each module is discussed in terms of :
  - The inter-relationship and visibility between the different modules.
  - Components (Large data types and data structures).
  - Methods (Procedures and functions).

- *Data Flow Diagrams and PDL* : The behaviour of the major objects in Elector are described in terms of dat.. flow diagrams and program description language.

- *The Netlists Abstract Data Type* : The Netlists abstract data type is discussed in detail. This is the foundation ADT of the application part of Elector and holds all the information about the components in the circuit being analyzed.

- *Dos_Environment Package* : Discusses the foundation processes which are implemented in assembler and which provide access to the underlying computer in a controlled manner. Two different methods of using buffers for the user interface are described.

- *Screen Objects* : Describes the smallest objects which appear on the screen and from which all the other user interface objects are build. It also describes the Inputter object which gets input from the keyboard and the mouse.

- *The Help Object* : Describes the on-line help information object.

- *The Screens Object* : Describes the subscreen objects which get and give information from the user to application processes of Elector.

- *The Menus Object* : Describes the menus from which the user chooses the next process to execute.

- *Tutorial Support Program* : Describes the Tutorial system used by Elector and the program used to create these tutorial *images*.

- *Testing and Validating Elector* : Discusses the test programs and test circuits used by Elector.

CHAPTER 2

DESIGN ISSUES

2.1 Ada as a PDL

All the PDL designs in this dissertation are done in an Ada variant. There are many views for and against such a strategy [Sommerville and Morrison, 1987], but the following points were considered important :

- Ada has sufficient expressive power to act as a starting point for an effective PDL.
- The control and structuring facilities are used directly, but a more relaxed approach is taken to implementation detail.
- English language comments are used in places where they would be illegal in normal code.
- All unit dependencies and subprogram parameters are left out. This can be filled in later when the design needs more detail.
- An exception handler which raises the exception STORAGE_ERROR appears at the end of subprograms which could run out of memory. This makes the exception which is usually raised by the run-time environment of Ada, more noticeable.

2.2 Elector's User Interface

Elector is divided into two main parts :

- *Application Code* : It forms the functionality of the program. It consists of the NETLIST_TYPE ADT (which holds the values of the components in the circuit) as well as all the calculation and analyzing objects.
- *User Interface* : It can be broken down into two further parts :
  - Subscreens
  - Menus

2.3 Turbo Vision and CXL C Library

Many of the concepts used in the user interface were extracted from the Turbo Vision user interface for Turbo Pascal version 6 and the CXL C library. Both these user interface libraries had definitions of the data structures. There was also a guide to Turbo Vision [Borland, 1990].

The Turbo Vision library is more advanced than the CXL library and so more attention was given to it. This library uses the advanced features found in Turbo Pascal version 6 such as classes, inheritance and polymorphism which form the backbone of Turbo Vision. These features are not found in Ada, and so a different approach was adopted.

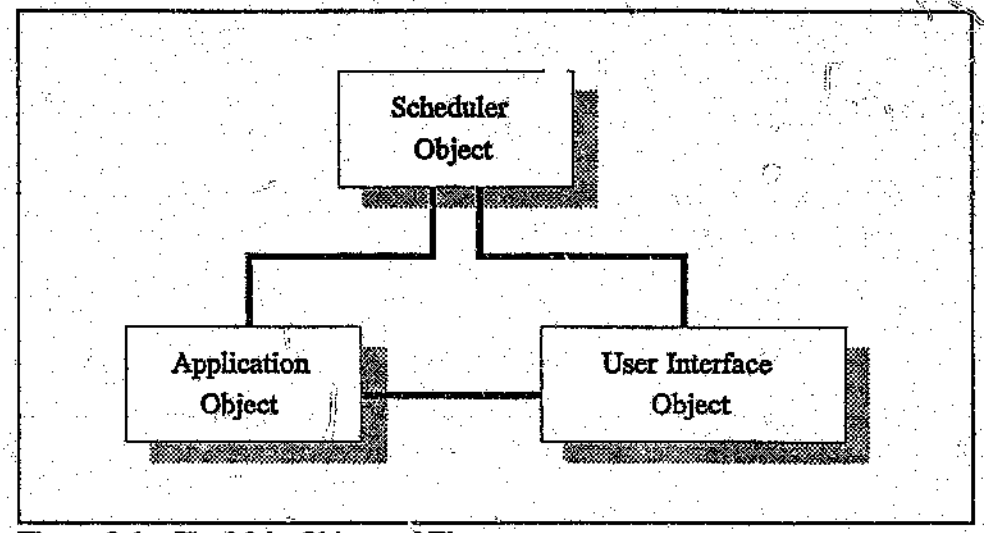


Figure 2-1 The Main Objects of Elector

2.4 Partitioning

Elecsim was divided into a number of different modules. Many of the modules corresponded to the major objects in the program, but some did not. Modules were restructured so that each one represented an object. Where an object consisted of smaller objects that were building blocks of it, these smaller objects were grouped inside the larger object. For the actual partitioning of Elector, see Chapter 4.

2.5 Meridian Ada Memory Organisation

The architecture of the Intel 80x86 microprocessor family on which the IBM PC, XT, AT and compatibles are build has certain characteristics which influence the Ada source code. The addressing of the microprocessor is segmented. A segment and an offset value are required to determine an actual physical address in memory. As a result, there are many different memory models for the 80x86 family, but Meridian Ada supports only one - a medium large model.

- Each compilation unit < 64K
- The total amount of global data in an entire program < 64K
- The stack size < 64K. The stack is used to store the values of local variables, the return addresses and parameters of subprograms as well as for tasking purposes.
- The heap is an area of memory where dynamically instantiated objects are created. The heap area is not limited to 64K, but an individual dynamic object < 64K.

In Elector, there are many large data structures - mainly arrays. Two of these arrays are 60K each. Declaring two of these as local static objects simultaneously would use more memory than

could be allocated from the stack. Therefore, many of these objects that could have been declared as local static objects in the design have to be declared as local dynamic objects so that they can be implemented.

2.6 Initialising Variables

In many of the type definitions in Elector, default values are specified. When a variable is created as an instantiation of such a type, it's value will be the default value. For data structures, some components have defaults while others do not. For the NETLIST_TYPE, all the components in the branch are set to zero when the branch is created. This helps prevent cases where a variable is used without being initialised.

2.7 Named Parameter Association

All subprograms are called with named parameter associations. The connection between the variables and/or values being passed to formal parameters is made explicit. This makes the code easier to understand. Otherwise, the subprogram's declaration has to be constantly referenced to see which variables and/or values are being passed to which formal parameters using positional parameter associations.

```
procedure Procedure_Name (PARAMETER_1, PARAMETER_2 : in NUMBER_TYPE);

•  Positional Parameter Association

        Procedure_Name (VARIABLE_1, VALUE1);

•  Named Parameter Association

        Procedure_Name (PARAMETER_1 => VARIABLE_1,
           PARAMETER_2 => VALUE1);
```

2.8 Parameter Modes

Parameters that are passed to a subprogram with the *in* mode are considered constants. This means that if a parameter needs to be changed within the subprogram, another variable must be created. The value of the parameter should be copied to the variable when the variable is declared. This makes the purpose of the variable easier to understand.

Parameters with *out* mode allow assignments to them only once in a subprogram. This is a problem if values need to be checked, and maybe even modified before being returned by the

subprogram. This can be solved using a local variable that can be read and modified. When the value is satisfactory, it can be copied from the local variable to the *out* mode parameter of the subprogram.

```
procedure Procedure_Name (PARAMETER_1 : in NUMBER_TYPE) is
   VARIABLE_1 : NUMBER_TYPE := PARAMETER_1;
begin
   -- statements
end Procedure_Name;
```

2.9 Overloading Subprograms

Overloading a subprogram is when more than one subprogram has the same name, but different parameters. This allows the same algorithm to be specified with the same name, but for different data types. Overloaded subprograms in Elector are :

- Write (Screen_Objects.Main package)
- Write (Screens.Prompts package)
- Combine (Ada_Extensions.Texts package)

2.10 Packages

Objects are realised using packages. Packages allow encapsulation of data structures and even the building of abstract data types. The NETLIST_TYPE is an abstract data type. All the packages in the Help, Inputter, Menus, Screen_Objects and Screens packages represent objects that are used in the user interface.

2.11 Exceptions

When a process that could fail due to some circumstance beyond it's control is invoked , it must have some way to inform the calling process if it was unable to complete it's activity. Usually, a flag or error code is passed as an out parameter. In Elecsim, an error code parameter system was used.

In many languages, this is the only possible way to deal with abnormal situations. However, Ada has a built-in mechanism for such situations - exceptions. This is a major improvement and has many benefits :

- Exceptions are a higher abstraction when compared to error codes and error flags.
- Execution of a subprogram is immediately abandoned on the raising of an exception. This is critical in real-time applications.

- Exception handlers (code that deals with a particular exception that has been raised) can be placed in the subprogram where the exception was raised, or in a higher level subprogram.
- There is no need for additional code (that would be necessary) to check if an error has occurred. Thus, the resulting Ada code is not *cluttered* with this additional error detection code which confuses the maintenance programmer to the actual purpose of the subprogram.

The screen objects have a USER_ESCAPE exception. This is raised if the *Cancel* button on any subscreen is pressed. This cancels the process currently being undertaken and that was selected on the main menu. Control is passed back to the Scheduler object. The Scheduler then calls the menus for the next menu option. To the user, the present process is aborted and the menu reappears waiting for a menu option to be chosen.

In Table 2.1, the error codes found in Elecsim are given, as well as the equivalent exceptions in Elector. Most of the exceptions occur in the Netlists package. (See Chapter 6 for an explanation of each exception generated in the Netlists)

Error Codes	Equivalent Exception
6	Attempted_Reference_Node_Removal
2, 22	Branch_Out_Of_Range
666	Cannot_Convert_With_Mutual_Inductances
7	Delete_Node_Does_Not_Exist
8, 44, 3	Meaningless_Component
33	Meaningless_Component_Connection
666	No_Circuit
2	No_More_Branches
999	No_Vccs_Controlling_Node
111, 777	Node_Pair_Does_Not_Exist
99	Reference_Node_Missing
88	Specific_Node_Missing

Table 2-1 Error Codes in Elecsim and Exceptions in Elector

2.12 Dynamic Strings

The Ada language does not provide dynamic strings as in Pascal. For input and output, this can be very limiting. To overcome this, the Texts package was developed. A logical dynamic string is implemented using a static *base* string. A string is declared with the longest useful length. A string index pointer points to the next free location in the declared string. A logical string's length

will be the value held in the index pointer.

2.12.1 Behaviour

A text type is instantiated with the maximum length that will be needed. Then, as more characters are added, the string grows. The string can have a maximum length of 255 characters. Integer and real values are converted to strings before being added. The representation of the value can be largely controlled. Engineering units can also be determined for real values within a certain range.

2.12.2 Components

```
type TEXT_TYPE (TEXT_LENGTH : POSITIVE := 70) is
   record
      TEXT       : STRING (1..TEXT_LENGTH);
      TEXT_INDEX : NATURAL := 1;
   end record;
```

2.12.3 Methods

The text combination methods are all overloaded.

Combine (String)

Combines a string with another string to form a larger string.

begin
  if (the present length of the main string = maximum length) **then**
    Return from this subprogram.
  elsif (the possible new length > maximum length) **then**
    Set the length of the string to be added only as long as what can be added to the main string.
    Set the possible new length to the maximum length.
  end if;
  Add the new string to the main string.
end Combine;

Combine (Integer value)

Combines a string and an integer value into a larger string.

```
begin
  Convert the integer value to a string.
  Add this string to the main string.
end Combine;
```

Combine (Real value)

Combines a string and a real value into a larger string. The real number can be converted to an engineering value with an engineering symbol if desired.

```
begin
  if (find engineering symbol) then
    if (VALUE is positive) then
      Add in offset value for preempt positive rounding.
    else
      Add in offset value for preempt negative rounding.
    end if;
    if (there is an engineering symbol for this number) then
      while (VALUE < ENG_EXPONENT) loop
        Increment counter.
        Multiply ENG_EXPONENT with engineering constant.
      end loop;
      Adjust VALUE for conversion.
      Determine engineering symbol and set string generation values.
    elsif (VALUE = 0.0) then
      Set string generation values for a 0.0000 value.
    end if;
  end if;
  if (the exponent of the value has 3 characters) then
    Adjust string generation values.
  end if;
  Convert the real value to a string and add this string to the main string.
  if (an engineering symbol has been found) then
    Add the engineering constant to the main string.
  end if;
end Combine;
```

Reset

Clears the string so that it is effectively empty.

```
begin
  Set the string length to 0.
end Reset;
```

2.13 Maths

The Meridian Ada compiler v2.1 does not provide mathematical functions such as Sin, Arctan etc. which are needed in Elector. In the Meridian Ada compiler v4.1, most of the required functions are implemented.

The initial implementation of these functions for the Ada compiler v2.1 consisted of truncating the Taylor infinite series after a few terms for most of the functions. Functions that dealt with angles had to have the angles specified in radians in the range between -2π and 2π. The formula for the following functions were :

2.13.1 Arctan Function

$$Arctan\, x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots$$

Equation 2-1 Taylor Series for Arctan Function

2.13.2 Cos Function

$$\cos x = \sin\left(x + \frac{\Pi}{2}\right)$$

Equation 2-2 Equation for Cosine Function

2.13.3 Exponent Function

$$\exp x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Equation 2-3 Taylor Series for Exponent Function

2.13.4 Logarithm to Base 10

$$\log x = \frac{\ln x}{\ln 10} = \frac{\ln x}{2.302585093}$$

Equation 2-4 Logarithm to base 10 Formula

2.13.5 Sine Function

$$Sin\,x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Equation 2-5 Taylor Series for Sine Function

2.14 Code Layout

A particular style has been used in the layout of Ada source code to enhance readability. (See Appendix B for the Ada code of Elector)

2.14.1 Upper and Lower Case

- All reserved words, including attributes, are lowercase.
- All variables, constants and types are uppercase.
- All packages, tasks and subprograms (procedures and functions) have the first letter of each word in the name in uppercase, while the remaining letters are in lowercase.

2.14.2 Indenting

- Indenting is set at 3 spaces.
- For source lines that do not fit on a single line (usually equations), the code is carried on to the next line, but indented a further 3 spaces to the right.

2.14.3 Parameter Lists

- Named parameter association is used whenever possible.
- For parameter lists that have many parameters, each parameter is placed on a separate line, with only one indentation to the right from the subprogram name.

```
Subprogram_Name (FORMAL_PAR1 => ACTUAL_PAR1,
      FORMAL_PAR2 => ACTUAL_PAR2,
      FORMAL_PAR3 => ACTUAL_PAR3,
      FORMAL_PAR4 => ACTUAL_PAR4);
```

2.14.4 Extended Names

- To limit the names directly visible in any particular package, the *use* clause has only been used with selected packages. The objects in other packages can still be accessed by using extended names instead of simple names.
- By using the extended names, readability is enhanced in certain circumstances. This applies to Elector, where the package names reflect the objects they represent. The subprograms from these packages are the methods associated with these objects. eg Analysers.Calculate_Power

2.14.5 Comments

- Before each subprogram, there is a prologue comment that explains the purpose of that subprogram.

2.15 Managing Memory

Elector is a large program. Not only does it need a lot of memory to hold the program code, but it also uses large arrays in the solution and analysis of circuits. Because memory is severely limited, the approaches discussed here were used.

2.15.1 Pascal Overlays

Elecsim uses overlays for about one third of the program. It consists of 200K of resident code and another 100K of overlaid sections at the Turbo Pascal unit level. Overlays slow down the execution of the program because the relevant section of code has to be loaded from disk before being executed. However, they enable the program to reduce it's memory requirements.

Meridian Ada 4.1 does not support overlays. This was a problem because Elector needs at least 130K of memory in addition to that needed to load the program. Elector had to be reduced in some way. Two things that achieved this were :

- Suppressing a number of run-time checks.
- Physically removing unused source code.

2.15.2 Suppressing Checks

The Ada language allows certain run-time checks to be suppressed. This makes the program smaller and faster to execute. However, when initially developing the program, or debugging it, it is best to leave the checks in because they raise exceptions if an abnormal condition arises (for

which there are checks). Checks can be suppressed by :

- Using the *-fs* flag on the command line to the Ada compiler. This suppresses all checks.
- Using the pragma *suppress* with the relevant check. The numeric checks DIVISION_CHECK and OVERFLOW_CHECK can only be suppressed by using the *-fN* flag on the DOS command line to the Ada compiler.

Check Name	Description of the Checks Undertaken
Access	• Accessing a selected component (record). • Accessing an indexed component (array). • Accessing a slice. • Accessing an attribute.
Discrimination	• Discriminant of a composite value has the value imposed by a discriminant constraint. • A record component exists for the current discriminant value when accessed.
Division	• Second operant is not zero for operators /, *rem* or *mod*.
Elaboration	• The body of a subprogram, task or generic instantiation has already been elaborated.
Index	• Bounds of an array = corresponding bounds of an index constraint. • When accessing array components, check for each dimension. • When accessing a slice, check that the given discrete range is compatible with the defined range bounds.
Length	• For array assignments, type conversions, and logical operators, check there is a matching component for each component of the array.
Overflow	• Results of a numeric operation do not overflow.
Range	• Value satisfies a range constraint. • Subtype indication - constraint (if present) is compatible with type mark. • For aggregate, check that index or discriminant value belongs to the corresponding subtype. • Constraint checks performed by a generic instantiation.
Storage	• Allocator does not require more space than is available. • Space available for a subprogram or task has been exceeded.

Table 2-2 Runtime Checks

Both numeric checks were left in for Elector. The STORAGE_CHECK was also left in because the environment in which Elector operates has limited memory and Elector uses large arrays to solve circuits. The program must be alerted as soon as any array cannot be created due to a shortage of memory. In the drawer package, the RANGE_CHECK was left in.

2.15.3 Removing Source Code

A number of subprograms that were not used, but which were initially developed as methods for the different objects were removed. These subprograms were put into the file SPARE.ADA for a maintenance programmer to use at a later date if the need arose. The subprograms that were removed are listed in Table 2.3.

Subprogram	Found in Package
Read_Cursor Scroll_Down Scroll_Up	Dos_Environment
Cursor_Down Cursor_Left Cursor_Right Cursor_Up	Screens.Main

Table 2-3 Removed Source Code

CHAPTER 3

SOFTWARE TOOLS

3.1 Introduction

In general, whenever human skills were supplemented by automated tools, there has been drastic increases in productivity, especially in manufacturing. To produce a software system, certain tools [Schach, 1990] are required. There are some tools which are considered as essential.

- Tools can help the software engineer in every phase of the software lifecycle.
- Tools automate activities, they do not do them. The software engineers creativity and judgment are needed. However, time and effort are reduced through the use of tools. Other options which would have been unfeasible can now be evaluated.

3.2 Editors

3.2.1 Turbo Pascal

This editor was used mainly in the examination of pascal source code. Some attempt was made to use it for coding of Ada source code, but it was abandoned when the capabilities of Brief were discovered. (See Brief V3)

3.2.2 Brief V3

In the documentation to Brief, it is said to be a Programmer's editor. Brief is a powerful, general-purpose editor that can be modified to meet certain requirements. Minor modifications can be done using the Setup program. Major modifications use a C type programming language to build macros which are compiled with a special compiler and then used by Brief to implement the desired function. Environmental variables are set in the AUTOEXEC.BAT file.

Brief is a full-screen, modeless (commands have the same meaning almost all the time) editor with all of it's commands and editing capabilities available from the editing environment. It allows multiple windows to be open simultaneously, each of which can be sized and moved around. Blocks of text can be copied, moved, pasted, etc. using a paste buffer. Brief uses buffers to store text while editing. The buffer is a temporary copy of a file (usually has the same name) and all changes made to this buffer are stored in memory. The file is only updated when the buffer is written to disk.

What sets this editor apart from other editors is that it has some special capabilities for the

programmer. The first is that a programming language compiler can be directly invoked from within the brief editing environment. Once the compilation is complete, editing can continue as before. Also, because of Brief's ability to swap itself out of memory before invoking a compiler, compilers that require large amounts of memory (eg. Meridian Ada compiler) can be invoked.

Using the Turbo Pascal editor requires a protracted process :

- Create the Ada source code using the Turbo editor.
- Save the Ada source file and Exit the Turbo editor.
- Invoke the Ada compiler with the output being redirected to an error file which can be examined later. If the output is not redirected, it appears on the screen. If there are more than 24 errors, the top error messages scroll off the screen. Also, any other DOS command that is issued will cause the display to scroll further up.
- Invoke the Turbo editor again with the error file.
- Once the error messages have been examined, the line number would have to be remembered. The Ada source file would be loaded (replacing the error file), and the cursor moved to the remembered line.
- The correct would then be undertaken.

This process of first loading the error file and then the source would have to be done for each error, unless many different errors and the corresponding line numbers could be remembered in one go. To some extent, Turbo Pascal Version 6 did alleviate this problem of switching between the error and the source files because it supports multiple windows in which different files can be open simultaneously. However, the problem of leaving the editor to compile still remained.

Using the Brief editor required a shorter editing and compiling process :

- Create the Ada source file using Brief.
- Press F10. The editors swaps itself out of memory and then invokes the Ada compiler. The error output is redirected into an error file. The editor swaps itself back into memory when the compiler has finished, placing the cursor on the line of source that caused the error, while displaying the corresponding error message on the status line.
- To go to the next error, simply press Ctrl - N. The cursor will move to the line of source that caused the error while a corresponding error message is displayed on the status line.
- If there were errors in the compilation, an error window can be called (Ctrl - P). A line bar can be moved up or down highlighting a particular error. Pressing Enter places the cursor on the line that caused the highlighted error.

The second special capability is that of language-construct template editing and smart indenting.

- *Template Editing* : A keyword (or abbreviation thereof) is entered and the space bar is pressed. The template of the Ada construct corresponding to the keyword is elaborated (expanded).
- *Smart Indenting* : The cursor is indented on the next line (to the left or right of where the present construct started) depending on the type of Ada construct on the present line.

```
p <space>
procedure _ is                  The cursor is waiting for the name
begin                           of the procedure to be entered.
end ;

if (condition) then             The cursor is indented one tab position
   _                            to the right.


end loop;                       The cursor is not indented and follows
_                               directly underneath the previous statement.
```

Abbreviation	Keyword	Abbreviation	Keyword
a	accept	pa	package
ab	abort	pb package b	package body is
b	begin	pn package i	package is new
c	case	r	return
e	else	rec	record
ei elsi	elsif	s	select
en	entry	su	subtype
ex	exception	t	type
exi	exit	ta	task
ew exit w	exit when	tb task b	task body
f	function	te	terminate
fo	for	u	use
i	if	w	when
l	loop	whi	while
o	or	wi	with
p	procedure	wo when o	when others

Table 3-1 Brief Ada Keyword Elaboration

3.3 Compilers and Assemblers

3.3.1 Assembler : MASM 5

It was used to assemble the Dos_Environment assembly code to produce an object file. Although the assembler supports macros, they were not needed.

3.3.2 Ada : Meridian AdaVantage 2.1

The Compiler : It was used in the development of Elector. Although the compiler was not validated by the AJPO and did not implement some features, the most important features as far as general program construction were implemented. This meant that most of the *application* part of the code could be compiled.

Problems did arise when the Halo Professional Graphical Kernel Library was compiled. The compilations were unsuccessful because the Halo code depended on a feature of the Ada language that was not implemented in this compiler. Thus, a character based user interface had to be developed.

Other problems encountered were :

- The compiler generates 8087 source code when real type numbers are used. Thus, an 8087 coprocessor, or an 8087 emulation program has to be present when executing such a program.
- Compilation that halted midway without any indication resulting in a *hanging* computer with a locked up the keyboard.
- Generation of a PROGRAM_ERROR exception even though the packages in question were elaborated by the time their subprograms were called.
- Overlooking non-declared variables that were only discovered when the subprograms did not execute as expected.

BAMP (Build Ada Main Program) : The high level linker which produced the executable program. It is a high-level linker that sorts out all the necessary context clauses before going into a low-level linking phase which links all the object files. This worked well for all programs that consisted of only Ada code. Problems arose for programs that made calls to assembler subroutines which had their own separately declared data areas.

3.3.3 Ada : Meridian Open Ada

The Meridian Open Ada package became available at the start of December 1991. This package consists of the following software tools.

Meridian Ada Compiler 4.1 : It is a production quality Ada compiler validated by AJPO in accordance with current testing procedures. Pre-emptive scheduling for tasking is supported. C and assembly language interfacing is also supported.

Ada Compiler Environment (ACE) : ACE is a menu-driven, multi-window interface with tightly coupled integration to the Meridian Ada compilation system. Additional features include :

- A powerful text editor.
- Extensive hypertext online help for the editor and compiler.
- An online Reference Manual for the Ada Programming Language (LRM).
- Ada syntax driven editing capability.
- Customizing of features.

BAMP High Level Linker : Links compiled Ada modules into an executable or object program.

Source Level Debugger : It is an interactive source-level debugger which is linked into the executable program if the debugging option is set when the Ada source modules are compiled. The execution can be traced and variables can be assigned values. It has a number of other features, including :

- Source line breakpoints.
- Single stepping.
- Variable watches.

Optimiser : It analysers the compiler's internal representation of a program and performs many code-improving transformations from local optimisations to global subprogram removal.

Dos Environment Library : It gives programs access to DOS and BIOS system facilities including screen management.

Utility Library : It has facilities for accessing command line arguments, efficient bit manipulations, transcendental math functions, variable length strings and large arrays.

3.3.4 Linkers : TLINK V2 (Turbo Linker V2)

It was used to link object files that were produced with the *-r* switch to BAMP.

3.4 Debugging

3.4.1 Turbo Debugger 2.5

It allows source level debugging for Turbo C, Turbo Pascal and assembler languages. For other languages, only assembly code disassembly of the executable program can be used for debugging. This was adequate for testing the Dos_Environment package body which was written in assembler code. This program helped extensively in the tracking and examining of variables and the program stack.

3.5 Documentation

3.5.1 Text and Final Formatting : WordPerfect 5.1

The documentation for Elector, including this dissertation, were developed using *WordPerfect 5.1*. *WordPerfect* was chosen for it's ease of use, as well as it's many capabilities including :

- Full wordprocessing features : formatting, copying and moving of text blocks, etc.
- Dictionaries and Thesaurus.
- Pull down menus and mouse support.
- Tables
- Adjustable font selection, importation of graphics and even basic desktop publishing features.
- End-notes, foot-notes, contents and index generation
- Basic maths facilities.
- Context sensitive help features for all the features of the program.

3.5.2 Graphics : Harvard Graphics 2.3 and DrawPerfect

Initially, *DrawPartner* from *Harvard Graphics 2.3* was used to create the drawing. Later, *DrawPerfect* became available. The drawings were converted to WPG format and slightly modified. The WPG is the same format used by *WordPerfect*. This permits the direct importation of pictures into the document without any transformations or conversions.

CHAPTER 4

ELECTOR OBJECTS

4.1 Introduction

Elector consists of many modules which are compiled separately. These modules are the following major packages :

- Ada_Extensions
- Analysers
- Central_Solver
- Dos_Environment
- Drawer
- Graphs
- Help
- Inputter
- Menus
- Monte
- Netlist_Supporter
- Netlists
- Optimiser
- Scheduler
- Screen_Objects
- Screens
- Tutorials

The *Help_Text* subprogram and *Help_Text1* package generate *ELECTOR.HLP* - the help file used by the *Help* object.

Package Name	Found in Package			
Ada Extensions	Maths	Texts		
Analysers	Plotter	Analysers		
Menus	Spread	Stacked		
Netlists	Screen_Supporter	Inner	Outer	
Screens	Errors	Prompts	Main	Status
Screen_Objects	Boolean_Buttons Character_Lines	Buttons Forms	Input_Lines Pick_Lists	Windows

Table 4-1 Smaller Packages within the Major Packages of Elector

4.2 Size of the Elector Packages

Three size metrics are used to determine the size of Elector packages.

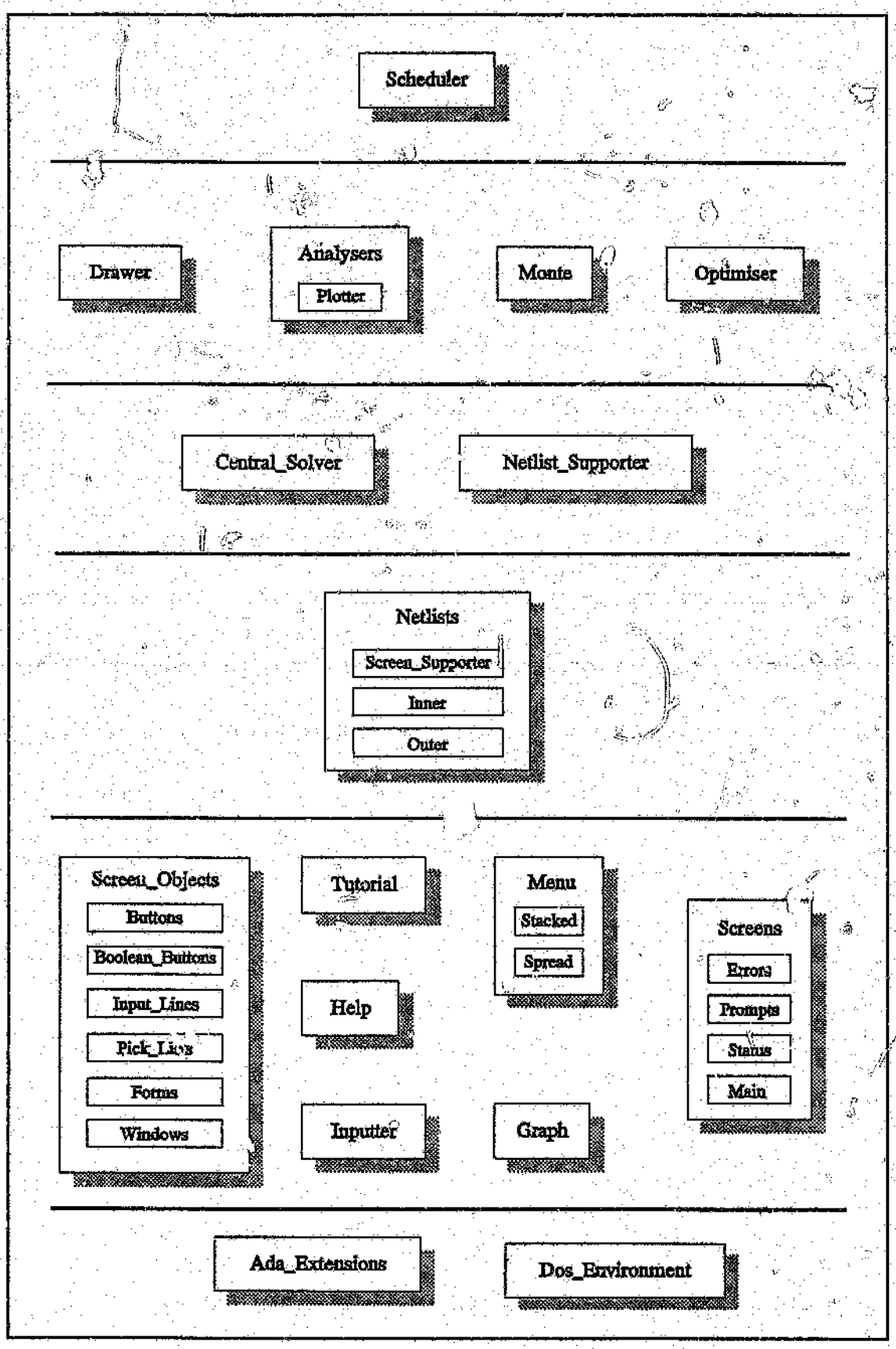
- *Line Count* : The number of physical lines in a file. It includes the comment count.
- *Comment Count* : The number of comment lines in a file, including those lines used for separating certain sections of the code.
- *Statement Count* : The number of lines that end with a ; which makes it more accurate than the line count. This is illustrate in the code fragment below. A single statement could take many lines, or one line.

```
Inputter.Read_Netlist (COMPONENT => GR_ADMIT,
    VALUE          => RESISTORS,
    HIGH_NODE1     => HIGHER_NODES (COUNTER),
    LOW_NODE1      => LOWER_NODES (COUNTER),
    HIGH_NODE2     => HIGH_NODE,
    LOW_NODE2      => LOW_NODE,
    NETLIST_OBJECT => NETLIST_OBJECT);

LOW_NODE := 0;
```

Elector Package	Line Count		Statement Count		Comment Count	
	Spec	Body	Spec	Body	Spec	Body
Ada_Extensions	153	309	48	101	25	72
Analysers	76	2443	21	952	18	120
Central_Solver	48	388	11	189	20	30
Drawer	39	2959	10	865	16	176
Graphs	149	479	54	169	26	49
Help	38	217	11	86	17	25
Inputter	54	69	19	22	15	19
Menus	88	1153	58	317	15	68
Monte	58	1837	18	779	16	56
Netlist_Supporter	46	682	12	213	16	27
Netlists	319	2645	117	1105	70	299
Optimiser	54	1866	16	752	14	74
Scheduler	34	488	9	163	15	31
Screen_Objects	365	1436	148	567	20	181
Screens	144	1304	46	474	19	139
Tutorials	37	265	9	103	19	24

Table 4-2 Line, Statement and Comment count for Elector Packages


Scheduler
Drawer
Analysers
Plotter
Monte
Optimiser
Central_Solver
Netlist_Supporter
Netlists
Screen_Supporter
Inner
Outer
Screen_Objects
Buttons
Boolean_Buttons
Input_Lines
Pick_Lists
Forms
Windows
Tutorial
Help
Inputter
Menu
Stacked
Spread
Graph
Screens
Errors
Prompts
Status
Main
Ada_Extensions
Dos_Environment

4.3 Ada_Extensions

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
	Math_Lib Text_IO	Analysers Central_Solver Dos_Environment Drawer Graphs Help Inputter Menus Monte Netlist_Supporter Netlists Optimiser Screen_Objects Screens Tutorials

4.3.1 Purpose

The base or foundation object on which all the other objects are build. It contains several data types, as well as the dynamic string and maths objects. (See the section on dynamic strings and maths in Chapter 2)

4.3.2 Construction

In the Texts package, two generic packages are instantiated for the conversion of integer and real values to strings.

```
package Real_In_Out    is new Text_IO.Float_IO (REAL_TYPE);
package Integer_In_Out is new Text_IO.Integer_IO (INTEGER_TYPE);
```

4.3.3 Data Types and Structures

The FOREGROUND_TYPE and BACKGROUND_TYPE define the colours for the screen. The LIST_LINE_TYPE is used for lists which are passed to the PROMPTS subscreen which gets choices from the user. The FIELD type is used for forms which are passed to the PROMPTS subscreen.

```
subtype INTEGER_TYPE     is INTEGER;
subtype REAL_TYPE        is FLOAT;
subtype COLUMN_TYPE      is INTEGER      range 1..80;
subtype NODE_TYPE        is INTEGER      range 0..250;
subtype OPERATION_TYPE   is INTEGER;
subtype ROW_TYPE         is INTEGER      range 1..25;

type FOREGROUND_TYPE is
   (BLACK,          BLUE,              GREEN,
   CYAN,            RED,               PURPLE,
   BROWN,           LIGHT_GRAY,        DARK_GRAY,
   LIGHT_BLUE,      LIGHT_GREEN,       LIGHT_CYAN,
   LIGHT_RED,       LIGHT_PURPLE,      YELLOW,
   WHITE);

subtype BACKGROUND_TYPE is FOREGROUND_TYPE range BLACK..LIGHT_GRAY;


type LIST_LINE_TYPE (TEXT_LENGTH : POSITIVE := 30) is
   record
      TEXT     : STRING (1..TEXT_LENGTH);
      STATE    : BOOLEAN := FALSE;
   end record;

type LIST_ARRAY_TYPE is array (POSITIVE range <>) of LIST_LINE_TYPE;


type FIELD (VALUE_LENGTH     : POSITIVE := 10;
      TEXT_LENGTH            : POSITIVE := 10;
      IS_INTEGER_TYPE        : BOOLEAN := TRUE) is
   record
      ACTUAL_LENGTH          : NATURAL;
      ACCEPTING              : BOOLEAN;
      ROW                    : ROW_TYPE;
      TEST_AGAINST           : INTEGER_TYPE;
      VALUE                  : STRING (1..VALUE_LENGTH);
      TEXT                   : STRING (1..TEXT_LENGTH);
      case IS_INTEGER_TYPE is
         when TRUE =>
            MAX_INTEGER,
            MIN_INTEGER,
            INTEGER_VALUE    : INTEGER_TYPE;
         when others =>
            MAX_REAL,
            MIN_REAL,
            REAL_VALUE       : REAL_TYPE;
      end case;
   end record;

type FIELDS_TYPE is array (POSITIVE range <>) of FIELD;
```

4.3.4 Methods

- *Angle* : Calculates the angle in degrees from a X and Y coordinate.
- *Find_Magnitude* : Calculates the magnitude from an X and Y component.
- *Log* : Calculates the logarithm to base 10 of a number by finding the natural logarithm and then converting it to the new base.
- *Check_Value* : The passed value is checked against a minimum which is assigned to it if it is smaller. Otherwise, it is checked against the maximum which is assigned to it if it is larger.

This function deals only with integer type values.

- *Check_Value* : The passed value is checked against a minimum which is assigned to it if it is smaller. Otherwise, it is checked against the maximum which is assigned to it if it is larger. This function deals only with real type values.
- *Swop_Node* : The value of NODE1 is swopped with the value of NODE2.

4.4 Analysers

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Central_Solver Math_Lib Netlist_Supporter Screens	Scheduler

4.4.1 Purpose

A collection of analysers which calculates the current, power and sensitivity of a circuit. The nodes to be used in a calculation can also be set.

4.4.2 Methods

- *All_Calculation_Nodes* : Sets all the possible nodes in a circuit to be analyzed.
- *Calculate_Current* : Calculates the current in the branches and displays the results on the main screen.
- *Calculate_Power* : Calculates the power used in the components and displays the results on the main screen.
- *Calculate_Voltage* : Calculates the voltage of the nodes and displays the results on the main screen.
- *Sensitivity* : Calculates the sensitivity of the node voltages of defined node pairs to perturbations in component values. The results are displayed on the main screen.
- *Set_Calculation_Nodes* : Sets the circuit nodes which will be analyzed.
- *Show_Nodes* : Displays the circuit nodes which will be analyzed on the main screen.

4.5 Central_Solver

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Math_Lib Netlist_Supporter Screens Unchecked_Deallocation	Analysers Monte Optimiser Scheduler

4.5.1 Purpose

It does the *solving* of the circuit and is used as the core process in any analysis. It solves the circuit by determining the voltages for the nodes of the circuit.

4.5.2 Data Types and Structures

The VOLTAGE_ACCESS_TYPE is a dynamic access type which points to a record VOLTAGE_TYPE. This record contains two arrays - one for the real voltage component values and one for the imaginary voltage component values. These arrays store the voltage values for all the nodes in the circuit. The MATRIX_GB_ACCESS type is used in the *Solve_AC_Network* method. It is a dynamic access array type which holds the conductance matrix (G) and the susceptance matrix (B).

```
MAX_NODES_1 : constant := MAX_NODES + 1;

type VOLTAGE_ARRAY_TYPE is array (1..MAX_NODES_1) of REAL_TYPE;

type VOLTAGE_TYPE is
   record
      VOLTAGE_REAL,
      VOLTAGE_IMAG : VOLTAGE_ARRAY_TYPE;
   end record;

type VOLTAGE_ACCESS_TYPE is access VOLTAGE_TYPE;

NODE : VOLTAGE_ACCESS_TYPE;


type MATRIX_GB_TYPE is array (0..MAX_NODES_1, 0..MAX_NODES) of
   REAL_TYPE;

type MATRIX_GB_ACCESS is access MATRIX_GB_TYPE;

B_MATRIX, G_MATRIX   : MATRIX_ACCESS;
```

4.5.3 Methods

- *Get_Voltages* : Gets the node voltage and adjusts it if the LOW_NODE is not the reference node.
- *Solve_AC_Network* : Calculates the AC solution of the network by using a conductance matrix (G) and a susceptance matrix (B). The results are displayed on the main screen. There are three internal procedures :
  - *Store_GB_Matrices*
  - *Solve_Linear_Equations*
  - *Find_GB_Matrices*

4.6 Dos_Environment

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions		Drawer Help Inputter Menus Screen_Objects Screens Tutorials

4.6.1 Purpose

Provides high-level access to low-level services provided by the underlying computer and operating system. These services are divided into two groups :

- BIOS services which are concerned with handling a function call.
- Buffer commands which are used for the user interface.

4.6.2 Construction

This module consists of subprogram declarations and interface pragmas. Interface pragmas connect the Ada subprograms to the corresponding assembler code subroutines. The assembler subroutines have comments for each line of assembly code. There is a prologue comment for each subroutine as well. (See Chapter 7)

4.6.3 Data Types and Structures

The ASCII_CHARACTERS type defines the ASCII character range. The FRAME_TYPE contains ASCII characters used in drawing the frame for subscreens and menus. The SCREEN_COLOUR_TYPE defines the foreground, background and blink attributes for screen characters.

```
subtype ASCII_CHARACTERS is INTEGER range 0..255;
type FRAME_TYPE is array (1..10) of ASCII_CHARACTERS;

type SCREEN_COLOUR_TYPE is
   record
      FOREGROUND  : FOREGROUND_TYPE     := LIGHT_GRAY;
      BACKGROUND  : BACKGROUND_TYPE     := BLACK;
      BLINK       : BOOLEAN             := FALSE;
   end record;
```

4.7 Drawer

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Dos_Environment Graphs Inputter Screens Unchecked_Deallocation	Scheduler

4.7.1 Purpose

It draws a schematic of the circuit on the main screen. The circuit is drawn on a graphic screen instead of the text screen used by the other processes. The circuit can be scrolled and zoomed.

4.7.2 Data Types and Structures

The DRAW_ACCESS is a dynamic access type which points to a record DRAW_CIRCUIT_TYPE. This record consists of three array types which hold the locations of the components in the circuit.

4.7.3 Methods

- *Ellipse* : Draws an ellipse where XC and YC are the centre coordinates, and AR and BR are the radii of the X axis and Y axis respectively. (Adapted from Meridian Graphics Library)

```
MAXIMUM_COUNT : constant := 3000;

type DRAWING_COMPONENT is
   record
      NUMBER      : INTEGER_TYPE := 0;
      HIGH_NODE,
      LOW_NODE    : NODE_TYPE := 0;
   end record;

type DRAWING_NODE_TYPE is array (1..MAX_NODES) of INTEGER_TYPE;
type GROUP_TYPE is array (1..MAX_LOW, 1..MAX_HIGH) of INTEGER_TYPE;
type NUMBER_NODES_TYPE is array (1..MAXIMUM_COUNT) of
     DRAWING_COMPONENT;

type DRAW_CIRCUIT_TYPE is
   record
      DRAWING_NODES  : DRAWING_NODE_TYPE;
      NUMBER_NODES   : NUMBER_NODES_TYPE;
      GROUP          : GROUP_TYPE;
   end record;

type DRAW_ACCESS is access DRAW_CIRCUIT_TYPE;

DRAW_CIR : DRAW_ACCESS;
```

- *Circle* : Draws a circle where XC and YC are the centre coordinates, and R is the radius.
- *Draw_Circuit* : Draws all the components of the circuit. The next 36 subprograms are declared local within it.
- *Draw_Voltage* : Draws a voltage source given certain parameters.
- *Draw_Dependent* : Draws a voltage controlled current source given certain parameters.
- *Draw_Current* : Draws a current source given certain parameters.
- *Draw_Passive* : Draws a passive component given certain parameters. Passive components are capacitors, inductors and resistors.
- *Draw_1_RLC* : Draws a passive component for a branch in the circuit given certain parameters.
- *Draw_2_RLC* : Draws two different passive components for a branch in the circuit given certain parameters.
- *Draw_3_RLC* : Draws three different passive components for a branch in the circuit given certain parameters.
- *Draw_Dependent_1_RLC* : Draws a voltage controlled current source and a passive component given certain parameters.
- *Draw_Dependent_2_RLC* : Draws a voltage controlled current source and two different passive components given certain parameters.
- *Draw_Dependent_3_RLC* : Draws a voltage controlled current source and three different passive components given certain parameters.
- *Draw_Current_1_RLC* : Draws a current source and a passive component given certain parameters.
- *Draw_Current_2_RLC* : Draws a current source and two different passive components given certain parameters.
- *Draw_Current_3_RLC* : Draws a current source and three different passive components given

certain parameters.

- *Draw_Voltage_1_RLC* : Draws a voltage source and a passive component given certain parameters.
- *Draw_Voltage_2_RLC* : Draws a voltage source and two different passive components given certain parameters.
- *Draw_Voltage_3_RLC* : Draws a voltage source and three different passive components given certain parameters.
- *Draw_Dependent_Voltage_1_RLC* : Draws a voltage controlled current source, a voltage source, and a passive component given certain parameters.
- *Draw_Dependent_Voltage_2_RLC* : Draws a voltage controlled current source, a voltage source, and two different passive components given certain parameters.
- *Draw_Dependent_Voltage_3_RLC* : Draws a voltage controlled current source, a voltage source, and three different passive components given certain parameters.
- *Draw_Current_Voltage_1_RLC* : Draws a current source, a voltage source, and a passive component given certain parameters.
- *Draw_Current_Voltage_2_RLC* : Draws a current source, a voltage source, and two different passive components given certain parameters.
- *Draw_Current_Voltage_3_RLC* : Draws a current source, a voltage source, and three different passive components given certain parameters.
- *Draw_Current_Dependent* : Draws a voltage controlled current source and a current source given certain parameters.
- *Draw_Current_Dependent_1_RLC* : Draws a voltage controlled current source, a current source, and a passive component given certain parameters.
- *Draw_Current_Dependent_2_RLC* : Draws a voltage controlled current source, a current source, and two different passive components given certain parameters.
- *Draw_Current_Dependent_3_RLC* : Draws a voltage controlled current source, a current source, and three different passive components given certain parameters.
- *Draw_Current_Dependent_Voltage_1_RLC* : Draws a voltage controlled current source, a current source, a voltage source, and a passive component given certain parameters.
- *Draw_Current_Dependent_Voltage_2_RLC* : Draws a voltage controlled current source, a current source, a voltage source, and two different passive components given certain parameters.
- *Draw_Current_Dependent_Voltage_3_RLC* : Draws a voltage controlled current source, a current source, a voltage source, and three different passive components given certain parameters.
- *Col* : Finds if the specified node is in the list of drawing nodes.
- *Draw_Branch* : Works out if the branch is vertical or horizontal, and then determines which components are in the branch.
- *Find_Number_Nodes* : Finds the branches and the components in those branches that are to be drawn.
- *Draw_Netlist* : Draws the individual branches of the circuit.
- *Label_Nodes* : Labels the nodes of the circuit schematic drawn on the main screen.

- *Zoom* : Sets and changes the values of several variables used in drawing the circuit. This allows the drawing to be expanded or contracted.
- *Draw_The_Netlist* : Draws the circuit and gives an error message if the drawing node ordering is not acceptable.

4.8 Graphs

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions	Dos_Environment Math_Lib Screens	Drawer Monte Plotter Scheduler

4.8.1 Purpose

It is the foundation object for displaying plots and circuits on the graphic screen. It contains subprograms to change the screen to graphics mode and back to text mode, as well as to plot points, draw lines etc. Some of these subprograms call Assembly Language procedures which do the actual work.

4.8.2 Methods

- *Change_Screen* : Changes the screen text to graphic mode or graphic to text mode.
- *Character_To_Graphic* : Writes a text character to the graphic screen.
- *Dot_Horizontal_Line* : Draws a dotted horizontal line with adjustable spacing between the dots on the graphic screen.
- *Dot_Vertical_Line* : Draws a dotted vertical line with adjustable spacing between the dots on the graphic screen.
- *Line* : Draws a solid line on the graphic screen.
- *Plot* : Plots a point on the graphic screen in the present plot colour.
- *Plot_Four_Pixels* : Plots 4 points symmetrically around a central dot on the graphic screen in the present plot colour.
- *Set_Plot_Colour* : Sets the present plot colour.
- *Set_Plot_Window* : Sets a window in which plotting takes place. Any point outside this window is not plotted.
- *Setup* : Sets up the user interface configuration as well as determining the values of certain variables used for drawing graphs.
- *String_To_Graphic* : Writes a text string to the graphic screen at the position and colour

specified.

- *Draw_X_Axis* : Draws the X axis on the graphic screen in the present plot colour.
- *Draw_Y_Axis* : Draws the Y axis on the graphic screen in the present plot colour.
- *Fix_Scale* : Determines the scale of the graph.
- *Get_Vertical_Inc* : Gets the maximum and minimum values for the horizontal axis and the x increment.
- *Get_X_Frequency* : Gets the starting and end frequency of the graph.
- *Label_Linear_X* : Labels a linear X axis.
- *Label_Log_X* : Labels a log X axis.
- *Label_Y* : Labels the Y axis.
- *Zero_Line* : Draws the zero line axis for Y = 0.

4.9 Help

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions	Dos_Environment Inputter Screen_Objects Sequential_IO	Menus Screens

4.9.1 Purpose

It displays help text relating to a subscreen or a highlighted option on a menu. (See Chapter 9)

4.9.2 Construction

This package instantiates the SEQUENTIAL_IO generic package. It uses the disk device to read the ELECTOR.HLP file which contains the help text.

4.10 Inputter

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Dos_Environment		Drawer Menus Screen_Objects Screens

4.10.1 Purpose

It is the object that collects input from the keyboard and the mouse. (See Chapter 8)

4.11 Menus

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions	Dos_Environment Help Inputter	Scheduler

4.11.1 Purpose

It displays menus and handles input directed to them. These menus display which options can be selected, and which cannot. (See Chapter 11)

4.12 Monte

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Bit_Ops Central_Solver Graphs Math_Lib Netlist_Supporter Screens Unchecked_Deallocation	Scheduler

4.12.1 Purpose

It performs statistical analysis using the Monte-Carlo simulation method for specified deviations in component values. The graphs are drawn on the graphics screen while text is sent to a text screen.

4.12.2 Data Types and Structures

The MONTE_ACCESS_TYPE is a dynamic access type which points to the record MONTE_DATA_STRUCTURE_TYPE. This record consists of three array types and two netlist types. The REAL_VOLTS and IMAG_VOLTS arrays hold voltage values. The Y_VALUES array holds the graph values.

```
MAXIMUM_SIMULATIONS : constant := 500;

type VOLT_ARRAY_TYPE is array (1..MAXIMUM_SIMULATIONS) of REAL_TYPE;

type Y_VALUES_ARRAY_TYPE is array (0..NX_PIXELS, 1..2) of REAL_TYPE;

type MONTE_DATA_STRUCTURE_TYPE is
   record
      Y_VALUES        : Y_VALUES_ARRAY_TYPE;
      REAL_VOLTS,
      IMAG_VOLTS      : VOLT_ARRAY_TYPE;
      NETLIST2,
      NETLIST3        : NETLIST_TYPE;
   end record;

type MONTE_ACCESS_TYPE is access MONTE_DATA_STRUCTURE_TYPE;
```

4.12.3 Methods

- *Bar_3D* : Draws a 3D bar with a certain depth.
- *Clear_Variables* : Clears all the components in the branches of NETLIST2 and NETLIST3.
- *Destroy_Data_Structure* : Destroys the dynamic data structure used for the Monte-Carlo calculations. This frees memory for other memory intensive operations. A flag is cleared showing that the data structure no longer exists.
- *Random* : Produces a *random* value using an integer seed.
- *Gauss* : Calculates a Gauss distribution.
- *Setup_Monte* : Creates NETLIST2 and NETLIST3 so that they have the same number of branches as NETLIST2.
- *Insert* : Inserts data for a component for the Monte simulation.
- *Load* : Loads a monte netlist from a disk file for monte simulations.
- *Perform* : Performs the Monte - Carlo simulations.
- *Save* : Saves the monte netlist used in monte simulations to a disk file.
- *Show* : Displays the components, their tolerances and types of distributions that are selected for the Monte - Carlo simulations.
- *Test_Random* : Tests the random number generator used in the monte analysis.
- *Worst* : Determines the *Worst* and *Absolute Worst* case monte simulation.

4.13 Netlist_Supporter

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Math_Lib Screens	Analysers Central_Solver Monte Optimiser Scheduler

4.13.1 Purpose

It provides methods for use with the NETLIST_TYPE abstract data type. The methods are separated from those in the Netlists package because they do not need to know the implementation details of the NETLIST_TYPE.

4.13.2 Methods

- *Change_Magnitude_Phase* : Changes the magnitude or phase of a component in the Netlists.
- *Insert_Component* : Inserts a component into the netlist. This may involve many calculations to determine mutual inductances and other factors in the circuit.
- *List_Netlist* : Lists all the components of a certain type on the main screen. The listing can also be in the form of equivalent inductors or admittances.

4.14 Netlists

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions	Math_Lib Unchecked_Deallocation Screens (Screen_Supporter) Screens (Outer) Sequential_IO (Outer)	Analysers Central_Solver Drawer Monte Netlist_Supporter Optimiser Scheduler

4.14.1 Purpose

It contains the data type definition and methods for instantiations of the NETLIST_TYPE abstract

data type. (See Chapter 6)

4.14.2 Construction

The NETLISTS object is constructed from three objects : a SCREEN_SUPPORTER, an INNER and an OUTER object. The SCREEN_SUPPORTER and OUTER objects can use the low-level services of the user interface, while the INNER object cannot.

4.15 Optimiser

Needed Units (Interface)	Needed Units (Body)	Units Using This Unit
Ada_Extensions Netlists	Central_Solver Math_Lib Netlist_Supporter Screens	Scheduler

4.15.1 Purpose

It optimises a circuit's performance by tuning component values and/or frequency.

4.15.2 Data Types and Structures

```
type OPTIMISE1_TYPE is (FIND,  MAXI,  MINI,  SETO);
type OPTIMISE2_TYPE is (MAXIM, MINIM, SETT);
type RESULTS_TYPE   is (DECIB, IMAGN, MAGNI, PHASES, REALS);

type INTEGER_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE) of
     INTEGER_TYPE;
type REAL_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE) of REAL_TYPE;
type MX_ARRAY_TYPE is array (1..MAX_OPTIMUM_VARIABLE) of ELEMENTS;

ANGLE_E,   ANGLE_I,   ANGLE_Y,
MAGN_E,    MAGN_I,    MAGN_Y,
PRTURB,    SC,        XX          : REAL_ARRAY_TYPE;
IDEG,      LPAR                   : INTEGER_ARRAY_TYPE;
MX                                : MX_ARRAY_TYPE;
```

4.15.3 Methods

- *Adjust_Magnitude_Phase* : Adjusts the magnitude and phase of a component.