

https://dspace.org/

Strategies for designing useful interfaces for software in high voltage applications and a case study of these principles applied to the alternative transients program (ATP).

Orbach, Tamir

1995
https://hdl.handle.net/10539/27908

Downloaded from a DSpace repository. https://dspace.org/

STRATEGIES FOR DESIGNING USEFUL INTERFACES FOR SOFTWARE IN HIGH VOLTAGE APPLICATIONS AND A CASE STUDY OF THESE PRINCIPLES APPLIED TO THE ALTERNATIVE TRANSIENTS PROGRAM (ATP)

Tamir Orbach

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, 1995

DECLARATION

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

Tamir Orbach

15 day of June 1995

ABSTRACT

ATP (Alternative Transients Program) has been used as a case study to show the development of strategies for writing usable and maintainable software in the High Voltage field. This was done by developing a graphical user interface (GUI) for ATP as a pre-processor to the program. The GUI was developed using modern software quality principles, as well as modern usability and maintainability trends including the use of object-oriented programming and design. The implementation of the GUI was performed by first writing the documentation for the program including the User Requirements Specifications, the User Reference Manual, and the Technical Reference Manual, and then coding the actual GUI. The implementation is shown to be successful in that the program is very usable, easily maintainable, and adheres to internationally accepted standards for software. It is shown that the techniques used for this design can easily be used for other High Voltage software which suffers from setbacks similar to those of ATP.

DEDICATION

To Shana, Ima and Aba

ACKNOWLEDGMENTS

The supervision, guidance and encouragement given by Doctor Ian Jandrell has been a tremendous help, and is gratefully acknowledged.

Thanks are also due to Doctor Barry Dwolatzky for his great help on the software aspects of the project as well as the papers published.

I would also like to thank Doctor John Van Coller whose help with ATP was invaluable.

Finally, I would like to thank my fiance, Shana, and my parents, Yael and Elie, whose motivation and encouragement were a great source of help for me.

ACKNOWLEDGMENTS

The supervision, guidance and encouragement given by Doctor Ian Jandrell has been a tremendous help, and is gratefully acknowledged.

Thanks are also due to Doctor Barry Dwolatzky for his great help on the software aspects of the project as well as the papers published.

I would also like to thank Doctor John Van Coller whose help with ATP was invaluable.

Finally, I would like to thank my fiance, Shana, and my parents, Yael and Elie, whose motivation and encouragement were a great source of help for me.

TABLE OF CONTENTS

DECLARATION ii

ABSTRACT iii

DEDICATION iv

ACKNOWLEDGMENTS v

TABLE OF CONTENTS vi

TABLE OF FIGURES xii

1 INTRODUCTION 1

1.1 Problem Statement 1

1.2 Aim 2

1.3 Overview of the Thesis 2

2 THE NEED FOR USABILITY AND MAIN-TAINABILITY IN HIGH VOLTAGE SOFTWARE 4

2.1 Introduction 4

2.2 The Importance of Software in the High Voltage and Power Systems Fields . 5

2.3 Modern GUIs 6

2.4 Usability 7

2.5 Maintainability 7

2.6 Some Examples of High Voltage Software 8

2.6.1 ATP 8
2.6.2 ERACS 9
2.7 Chapter Summary 11

3 DESIGNING SOFTWARE FOR USABILITY AND MAINTAINABILITY 13
3.1 Introduction 13
3.2 Implementing Usability According to the IBM CUA and Microsoft Application Design Guide 13
3.3 Designing for Maintainability 15
3.4 Using Software Quality Principles 15
3.4.1 The User Requirements Specification 16
3.4.2 The User Reference Manual 16
3.4.3 The Technical Reference Manual 16
3.4.4 The Product Test Plan and Specification 17
3.5 Chapter Summary 17

4 ATP 18
4.1 Introduction 18
4.2 What ATP can simulate 18
4.3 How ATP works 19
4.4 The ATP data card interface 24
4.5 Chapter Summary 26

5 THE PRESENT GUI FOR ATP (ATPDRAW) 27
5.1 Introduction 27
5.2 The Design of ATPDRAW 27
5.3 The ATPDRAW Interface 28
5.4 The Advantages of ATPDRAW over ATP 30
5.5 The Disadvantages of ATPDRAW 30

5.6 Chapter Summary 31

6 OBJECT ORIENTED PROGRAMMING AND DESIGN 33
6.1 Introduction 33
6.2 Object Oriented Programming Concepts 33
6.2.1 Abstraction 33
6.2.2 Classes 34
6.2.3 Inheritance 34
6.2.4 Polymorphism 35
6.2.5 Message Passing 36
6.2.6 Late Binding 37
6.3 Booch Diagrams and Class Structure Design 37
6.3.1 Classes 37
6.3.2 Class Relationships 38
6.4 The Advantages of Using OOD/OOP 40
6.5 Object-oriented Tools 41
6.6 Chapter Summary 42

7 USING VISUAL BASIC TO WRITE OBJECT-BASED AND EVENT DRIVEN SOFTWARE 43
7.1 Introduction 43
7.2 Visual Basic 43
7.3 Visual Basic Controls 44
7.4 Object Based and Event Driven Programming with Visual Basic 45
7.5 Comparing Visual Basic Maintainability to C++ Maintainability 47
7.5.1 The FormSimpleRLCInitialize Code 49
7.5.2 The FormDistributedRLCInitialize Code 51
7.5.3 Initializing the Two Forms Using C++ 52
7.6 Comparing Visual Basic Usability to C++ Usability 53

7.7 Chapter Summary 53

8 ATP FOR WINDOWS 54
8.1 Introduction 54
8.2 Requirements for ATP for Windows 54
8.2.1 Ensuring Usability 55
8.2.2 Ensuring Maintainability 56
8.3 Implications of the Requirements 56
8.4 Chapter Summary 57

9 THE OBJECT-ORIENTED DESIGN OF THE DATA CARD UNIT 58
9.1 Introduction 58
9.2 Using The Structure of ATP Data Cards to Select Objects 59
9.3 Object-oriented Design in ATP for Windows 61
9.3.1 The Power_System_Data_Card class 62
9.3.2 The Branch_Card Class 64
9.4 Chapter Summary 65

10 THE OBJECT-BASED DESIGN OF ATP FOR WINDOWS 66
10.1 Introduction 66
10.2 The Structure of ATP for Windows Forms 66
10.3 The Design of ATP for Windows Modules 67
10.3.1 The Transmission_Line_Module (TRNSLINE.BAS) 69
10.4 Using MS Access Databases to Store Circuit Information 81
10.5 Chapter Summary 82

11 THE IMPLEMENTATION OF THE INTERFACE 83
11.1 Introduction 83
11.2 Using Software Quality Principles 83

11.3 The ATP for Windows User Requirements Specifications 84
11.4 The ATP for Windows User Reference Manual 84
11.5 The ATP for Windows Technical Reference Manual 85
11.6 The ATP for Windows Product Test Specifications and Plan 85
11.7 Using Visual Basic as Opposed to C++ to Implement ATP for Windows . . 86
11.8 ATP for Windows: The Final Product 86
11.9 Chapter Summary 88

12 ASSESSMENT OF THE IMPLEMENTATION 89
12.1 Introduction 89
12.2 Assessing the Usability Achieved 89
12.3 Assessing the Maintainability Achieved 91
12.4 Comparing ATP for Windows to ATPDRAW 92
12.5 Chapter Summary 94

13 POTENTIAL IMPROVEMENTS TO ATP FOR WINDOWS AND ATP 95
13.1 Introduction 95
13.2 Making the Interface More Maintainable 95
13.3 Using Delphi to implement the GUI 95
13.4 Other Potential Improvements to ATP 96
13.5 Chapter Summary 97

14 IMPLICATIONS FOR FUTURE WORK 98
14.1 Introduction 98
14.2 General Strategies Used 98
14.2.1 Ensuring Usability 99
14.2.2 Ensuring Maintainability 99
14.3 Strategies Which Can be Used in Other High Voltage Applications 99
14.3.1 Improving ERACS 99

14.4 Chapter Summary 100

15 CONCLUSION 101

16 REFERENCES 104

APPENDIX A - THE ATP FOR WINDOWS USER REQUIREMENTS SPECIFICATIONS 108

APPENDIX B - THE ATP FOR WINDOWS USER REFERENCE MANUAL 125

APPENDIX C - THE ATP FOR WINDOWS TECHNICAL REFERENCE MANUAL . 126

APPENDIX D - THE OBJECT-ORIENTED DATA CARD UNIT 127

APPENDIX E - ATP CASE STUDIES 128

APPENDIX F - A COMPARISON OF ATPDRAW AND ATP FOR WINDOWS 135

APPENDIX G - ADDING AN ELEMENT TO ATP FOR WINDOWS TO ASSESS THE MAINTAINABILITY OF THE PROGRAM 139

TABLE OF FIGURES

Figure 1 - Simple ATP Circuit Simulation 8
Figure 2 - The ERACS Main Screen 11
Figure 3 - Simple Network Around Node 1 20
Figure 4 - Inheritance 35
Figure 5 - Class Icon Representation 37
Figure 6 - Class Relationships in Booch Diagrams 38
Figure 7 - Internet News Reader Booch Class Diagram 39
Figure 8 - Visual Basic Controls 44
Figure 9 - Booch Diagram of a Visual Basic Form 45
Figure 10 - The Properties Inherited by the cmd3dCircuitPanel Control 46
Figure 11 - The Simple RLC Branch Element Details Screen 48
Figure 12 - The Single Transmission Line Conductor Element Details Screen 49
Figure 13 - The Booch Class Diagram for the Data_Case Class 59
Figure 14 - The Power_System_Data_Card Booch Class Diagram 60
Figure 15 - The Transmission Line Element Details Screen 67
Figure 16 - The Booch Diagram Showing Relationship Between Element Details Forms and Corresponding Modules and Forms 68
Figure 17 - The Transmission Conductor Element Details Screen 70
Figure 18 - The ATP For Windows Main Screen 87
Figure 19 - The Main Screen of ATPDRAW 94

1 INTRODUCTION

1.1 Problem Statement

The High Voltage field has become largely dependant on software applications as many simulations and calculations have become too tedious to do by hand. As a result of this, many new High Voltage and Power Systems software applications are now in use on a daily basis. Many High Voltage software applications suffer from a number of setbacks, that often cost users time and money, which can be eliminated or improved using modern software principles. One such application is the Alternative Transients Program (ATP). ATP is a very widely used computer program written in Fortran, which is capable of simulating transient conditions and phenomena in power machinery and distribution networks. ATP can solve any network consisting of interconnections of resistances, inductances, capacitances, single and multiphase π circuits, distributed (or lumped) parameter lines, and certain other elements (Dommel 1986).

ATP was developed in the 1960's, and at the time made use of data cards to input information. The present version of ATP makes use of very strict format data files to input data (see Appendix D - ATP Case Studies). This type of user interface is extremely time consuming to use and to learn, as well as being very user-unfriendly (Orbach 1995b). Although the Bonneville Power Administration (distributors of ATP) are at present supplying a graphical user interface for ATP, this interface suffers from numerous setbacks, and has not been widely accepted by ATP users worldwide (Orbach 1995). Many other High Voltage software applications such as PSSE and ERACS also have poor interfaces.

1.2 Aim

A graphical user interface (GUI) for ATP has been developed as part of the author's MSc (Eng) Elec degree. The GUI was developed with an emphasis on maintainability and usability whilst maintaining ISO 9001 Software Quality Principles. The strategies used for the development of the GUI were noted, and the implementation was evaluated. The aim of this thesis is to show the importance of software in the High Voltage and Power systems fields, and the need for the use of software engineering principles in the development of this software. The thesis also presents the background to the development of the GUI as well as the various stages involved in the design of the GUI including the literature survey, the factors influencing design decisions, and the implementation of the design. These are then analyzed with respect to applying similar principles to make other High Voltage software applications more usable and maintainable.

1.3 Overview of the Thesis

Chapters 2 to 7 cover the background information to the final development process and implementation.

Chapter 2 discusses the need for maintainability and usability in High Voltage software with particular reference to ATP and ERACS.

Chapter 3 covers the methods used and the standards available for modern software to ensure that software is usable and maintainable.

Chapter 4 describes ATP, what it is and how it works at present, as well as the setbacks it suffers at present.

Chapter 5 analyses the present GUI for ATP describing its advantages and disadvantages, and justifying the need for a new GUI for ATP.

Chapter 6 introduces the concepts of object-oriented programming and design (OOP and OOD).

Chapter 7 describes the event driven and object based structure of Visual Basic, and justifies the use of this rapid application development tool in this project.

Chapter 8 introduces the reader to the interface being designed, and the requirements for ATP for Windows at the outset of the project.

Chapter 9 describes the object-oriented design of the Data Card module of the interface including a detailed analysis of this design.

Chapter 10 details the design of the interface module of the program.

Chapter 11 describes the actual implementation of the GUI, while chapter 12 involves an assessment of the implementation of the interface.

Chapter 13 describes improvements which can be made to the GUI for ATP, and chapter 14 details the general strategies which were used to develop the GUI for ATP, that can be used to improve other High Voltage software applications such as ERACS.

Chapter 15 concludes the thesis.

2 THE NEED FOR USABILITY AND MAINTAINABILITY IN HIGH VOLTAGE SOFTWARE

2.1 Introduction

Denny stated that Electric power systems are growing in size and becoming increasingly complex in function. Because of these advances, more advanced computer applications will be required to monitor, control, and analyze the vast electrical power systems which future supplies will demand. Operators will have to contend with new uncertainties, and decision making will increasingly have to depend on computer analysis and simulation (Denny 1988). This has been proved to be true. However, much of the software which is written in the High Voltage and Power Systems fields is never used, or used at a high monetary cost and time penalty to the user. This is due to the lack of usability and maintainability of this software (Orbach 1995b).

It is now widely recognized in the computer industry that usability of programs is one of the most important issues of software in the 90's (Hicks, 1991). Maintainability of software is also seen as a very important aspect of modern software. It is therefore these two factors which should be considered when designing modern software, and particularly when designing graphical user interfaces, since it is the user interface of the program which determines its usability (Orbach 1995). The following chapter includes the following:

- A description of the importance of software in the High Voltage and Power Systems fields.
- The definition of the concepts of usability and maintainability, and
- a description of modern graphical user interfaces.

It is then shown that some High Voltage applications which are still very widely used do not

adhere to either modern usability or modern maintainability standards.

2.2 The Importance of Software in the High Voltage and Power Systems Fields

Software is one of the most important aspects of modern power engineering design. A testimony to this fact is that the IEEE (Institute of Electrical and Electronic Engineers) has devoted a monthly publication entirely to the issues surrounding software in the Power Engineering field (IEEE Computer Applications in Power). A good example of the importance of software in the Power Engineering field today is that to keep up with the more stringent design requirements that the industry demands, designers of electrical machinery are turning to computer aided design (Salon 1990). Distribution Systems also make extensive use of software: For example, automatic circuit reclosers were developed to reduce circuit outages and increase the reliability of distribution systems. The settings of these circuit reclosers must be coordinated to provide ideal behaviour. The variability in settings, combined with the multiple reclosing events, must be considered when coordinating the recloser with source-side and/or load-side fuses and reclosers. An expert system built into a coordination program can help distribution engineers establish a functional protection system, and numerous such software systems are used today (Witte et al 1992).

Software is used for simulating effects in power systems (de Mello et al 1992), as well as for load management (Scott Davis 1990), Finite Element Analysis of electrical machinery (Salon 1990), assessing transient stability of power systems (Rahimi et al 1991), optimizing Energy Management Systems (Lee Smith 1989), and many other applications.

Although this is the case, many Power Engineering software applications still suffer from numerous setbacks, of which the most important is the lack of a graphical user interface, or the existence of a very poor interface. This makes much of this software time consuming to use and

to learn. After a graphical user interface was developed for the Generator Artificial Intelligence Diagnostics System (GenAID) in 1990, Gray wrote that the graphical user interface represents a significant improvement over existing technology (Gray and King 1991). This has been the case in almost all attempts to develop user interfaces for Power Engineering software.

2.3 Modern GUIs

Modern Graphical User Interfaces (GUIs) adhere to the IBM CUA (Common User Access) guideline (which defines user interface components that should be consistent within and across applications (Lacob 1993)), while Windows applications generally adhere to the Microsoft Application Design Guide. The IBM Common User Access is a set of selected software interfaces, conventions, and protocols that serves as a common framework for application development (IBM 1987). It focuses on two fundamental aspects of usability (IBM 1987):

- Users can develop a conceptual model of the interface - In order to reinforce the users' conceptual model of the user interface a number of techniques including using metaphors, or analogies, to connect otherwise unrelated concepts, and having consistency amongst different areas of the interface, are used.
- Users can be in control of the dialog, and should be whenever possible - Users should be allowed to switch from one activity to another, change their minds, and stop activities they no longer want to continue.

The main purpose of the Microsoft Application Design Guide is to provide visual and functional consistency within and across Windows based applications (Lacob 1993). The Guide replaces IBM's CUA Guidebook that until 1992 served as Microsoft's design standard (Sherer 1992).

2.4 Usability

Usability is the extent to which an end user is able to carry out required tasks using the system successfully and without difficulty (Ravden 1989). Eason defines usability as 'an attempt to study the quality of human-computer interaction' (Eason 1991). A product which is not usable costs companies time and money in extra training costs, using costs and maintenance costs.

2.5 Maintainability

Another very important aspect of modern software including GUIs, is the maintainability of the software. Any successful software will eventually enter a prolonged and costly maintenance phase, which will involve the GUI in almost every case. Any modification to a software product after its first installation to correct faults, to improve performance or other attributes, or to adapt the product to a changing environment can be considered as software maintenance (Lacob 1993).

Lehman states that a program that is used in a real-world environment necessarily must change or become less and less useful in that environment (Lehman 1980). As an evolving program changes, its structure becomes more complex unless active efforts are made to avoid this phenomenon (Boocn 1994). During maintenance of software, developers will be asked to make continual improvements to an existing system; these maintainers are often a different group of people than the original developers.

2.6 Some Examples of High Voltage Software

2.6.1 ATP

The Alternative Transients Program (ATP - also known as EMTP) is a power systems transient analysis program, which can predict variables of interest within electric power networks as functions of time, typically following some disturbance such as the switching of a circuit breaker, or a fault (Leuven EMTP Center, 1992).

ATP is very widely used by power distributors, universities and research institutions, as well as industry. However, ATP suffers from a number of setbacks which could potentially be improved. The most important problem with ATP, is its user interface. This interface is based on inputting data cards / data files in DOS text form. It is very tedious to use, and is extremely outdated. A typical EMTP/ATP data file can be seen in the following example:

In order to simulate the circuit seen in *figure 1* (lightning striking a resistor):



Figure 1 - Simple ATP Circuit Simulation

The following data case is required:

```
BEGIN NEW DATA CASE
C Hello
$OPEN, UNIT=4 FILE=c:\tut1.pl4 FORM=FORMATTED
  2.E-7  1.E-4
    1    1          3                    1
00A                10.0                          3
BLANK card ending branch cards
BLANK card ending switch cards
15A   -1   3.2E4   -2.E6   -1.E7
BLANK card ending source cards
 1
BLANK card ending plot cards
BEGIN NEW DATA CASE
BLANK card ending session
```

As can be seen from this interface example, ATP has a user interface which is not very usable, and is time consuming to use and to learn. ATP does not include any of the properties described in the IBM CUA guideline as essential for a usable program (see section 3.2). It is extremely difficult to use, as well as being very time consuming to learn. ATP was written in FORTRAN, and is hence very difficult to maintain. ATP is described in greater detail in chapter 4 - ATP.

2.6.2 ERACS

ERA technology, a UK based company, has developed a suite of software programs, collectively known as ERACS, which allows the user to design and analyze electrical power system networks under steady state and transient operating conditions (ERA Technologies 1994). ERACS includes three full program features:

- Loadflow analysis
- Short Circuit analysis, and
- Transient Stability analysis

ERACS is a menu driven program which runs in a mouse driven environment. It allows the user to place elements on the screen by selecting the element from the add menu, and then using the mouse to position it (see *figure 2* - The ERACS main screen). Although ERACS is more user-friendly than many other such applications on the market, there are still a number of flaws which can be attributed to it:

- ERACS is a DOS based program, which does not have an interface that conforms to IBM CUA standards in all respects.
- This means that ERACS has an interface which is not consistent with modern graphical user interfaces, which are usually based on a graphical platform such as MS Windows, Apple MAC, or SUNOS - Open Windows..

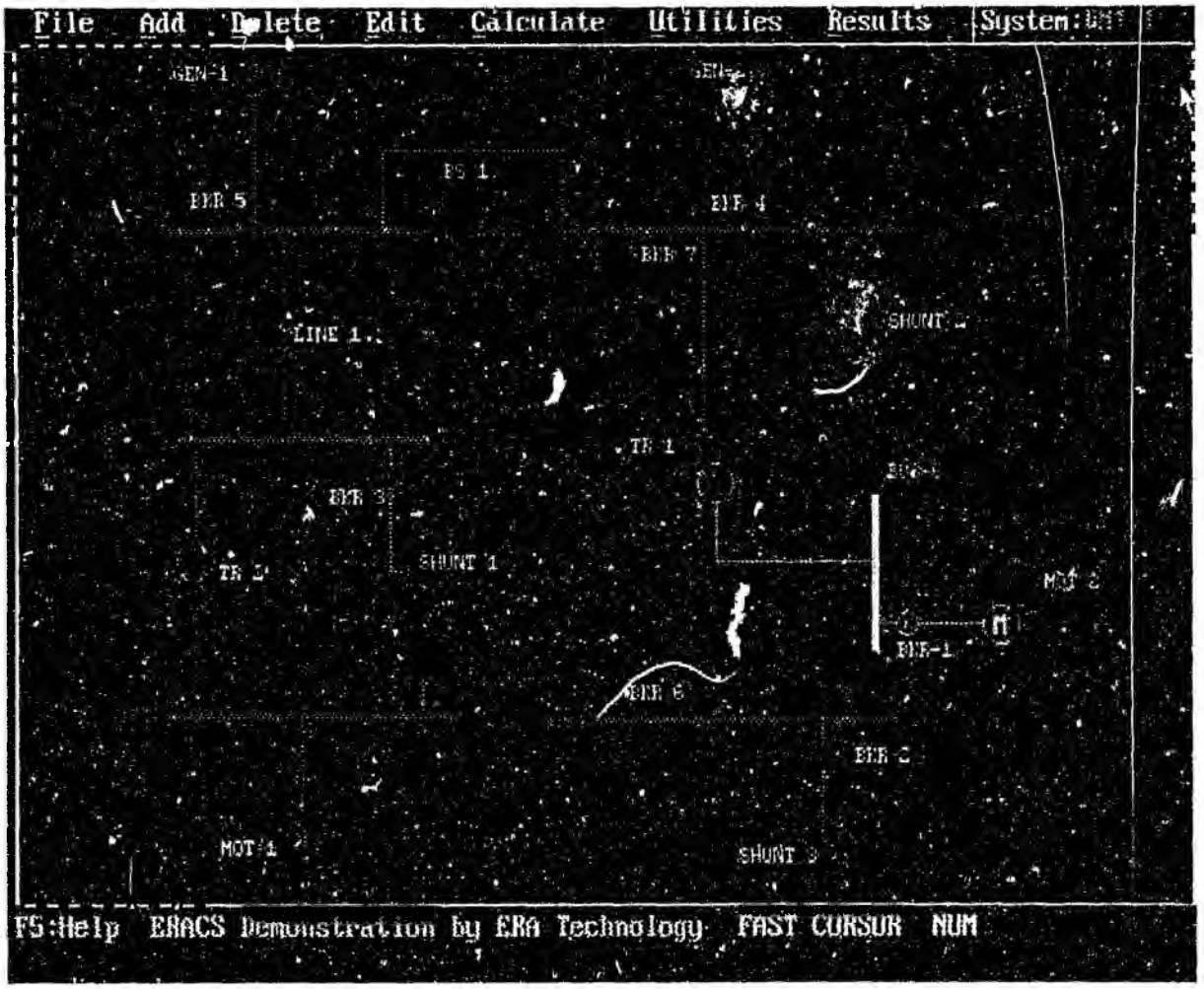


Figure 2 - The ERACS Main Screen

2.7 Chapter Summary

Software is becoming an increasingly important aspect of the High Voltage and Power Systems fields. However, many software applications in these fields suffer from numerous setbacks including the lack of usability and maintainability of these applications. Usability and maintainability are two of the most important aspects of modern software design. Usability determines the ease of use of a product and can often determine the success or failure of a

software product which is commercially sold. Maintainability is a measure of the ability to change a software product after it has been completed by adding to it, or simply changing existing programming functions. Maintainability is directly proportional to the cost of developing and maintaining a product since a product which is not easily maintainable costs more to maintain and to use in the long run than one which is easily maintainable. A number of High Voltage and Power Systems software applications were presented, and it was shown that these products are neither usable nor maintainable. This has probably resulted in a great waste of both time and money. In the light of the discussion contained in this chapter, it is clear that it is important to analyze High Voltage software more closely, to determine methods for solving the problems associated with this software. To aid in the development of strategies for making High Voltage software more usable and maintainable, it was decided to write a graphical user interface for ATP as a pre-processor to ATP. In order to aid the development of this GUI, it was necessary to analyze ATP in detail, as well as analyzing the factors which make software usable and maintainable.

3 DESIGNING SOFTWARE FOR USABILITY AND MAINTAINABILITY

3.1 Introduction

The significant use of software in the High Voltage field has made it crucial that this software is developed to be as usable and maintainable as possible. Because of this, it is essential to analyze the processes involved in designing software for usability and maintainability. The following chapter describes the different aspects of designing software for usability and maintainability including:

- implementing usability according to existing guidelines,
- designing software for maintainability, and
- using software quality principles to ensure that the software is maintainable.

3.2 Implementing Usability According to the IBM CUA and Microsoft Application Design Guide

There is no doubt that graphical user interfaces are becoming increasingly popular with users and a number of surveys have pointed to the superiority of such interfaces over command based interfaces with regard to such qualities as consistency, ease of use, and reduced learning curve (Urlocker 1990). This indicates that the first step to making software usable, is to design a graphical user interface for the software. As mentioned previously, the standard guideline for interface design is the IBM CUA.

Lacob identifies a number of criteria as being important for adhering to the IBM CUA usability guideline (Lacob 1993), including:

- Visual clarity - This refers to the clear displaying of information on the screen, making the screen appear uncluttered, and enabling the users to find required information quickly and easily.
- Consistency - This refers to keeping consistency amongst different screens in the application, thus allowing the user to feel comfortable with any screen once the user has used one or two of the available screens. Consistency can greatly increase speed of learning, and significantly reduce the likelihood of errors (it is the key to the effectiveness of the Microsoft Windows platform).
- Informative feedback - This indicates that the users should be given clear and informative feedback on where they are in the system, what actions they have taken, whether these actions have been successful, and what actions they should take next.
- Explicitness - The way in which the system works should be very clear to the user. This makes the interface 'transparent' to the user, and allows the user to develop a clear and accurate mental representation of the interface.
- Appropriate functionality - The interface must include functionality which the users feel they require in order to carry out the tasks at hand.
- Flexibility and control - The system should be as flexible as possible to ensure that any user feels comfortable using the system, and feels in control of the system.
- Error prevention and correction - The system should be designed to minimize the possibility of user error, with built-in error handling for those errors which do occur. Users should be able to check their inputs and correct them if necessary before the inputs are processed.
- User guidance and support - Easily accessible, and informative help should be available for the user at any point or situation in the system (e.g. Windows On-line Help).

These criteria can be used both as a guideline for developing usable software, and as a method of testing whether or not software is indeed usable.

- Visual clarity - This refers to the clear displaying of information on the screen, making the screen appear uncluttered, and enabling the users to find required information quickly and easily.
- Consistency - This refers to keeping consistency amongst different screens in the application, thus allowing the user to feel comfortable with any screen once the user has used one or two of the available screens. Consistency can greatly increase speed of learning, and significantly reduce the likelihood of errors (it is the key to the effectiveness of the Microsoft Windows platform).
- Informative feedback - This indicates that the users should be given clear and informative feedback on where they are in the system, what actions they have taken, whether these actions have been successful, and what actions they should take next.
- Explicitness - The way in which the system works should be very clear to the user. This makes the interface 'transparent' to the user, and allows the user to develop a clear and accurate mental representation of the interface.
- Appropriate functionality - The interface must include functionality which the users feel they require in order to carry out the tasks at hand.
- Flexibility and control - The system should be as flexible as possible to ensure that any user feels comfortable using the system, and feels in control of the system.
- Error prevention and correction - The system should be designed to minimize the possibility of user error, with built-in error handling for those errors which do occur. Users should be able to check their inputs and correct them if necessary before the inputs are processed.
- User guidance and support - Easily accessible, and informative help should be available for the user at any point or situation in the system (e.g. Windows On-line Help).

These criteria can be used both as a guideline for developing usable software, and as a method of testing whether or not software is indeed usable.

3.3 Designing for Maintainability

Any changes to the final product constitute maintenance. Maintenance is an integral part of the life cycle of software, and thus it is imperative that the design should as far as possible take further enhancement into account. It has been claimed that software in general, developed using Object-oriented Design is easier to change and hence more easily maintained than structured code (Lacob 1993). The most important factor to be taken into account, from the design perspective, when designing software for maintainability, is to attempt where possible to use Object-oriented Design and programming. However, an object-oriented program would still be difficult to maintain if the design of the program is not properly documented.

3.4 Using Software Quality Principles

Walker defines software quality assurance as a planned and systematic pattern of all actions necessary to provide adequate confidence that the software conforms to established technical requirements (Walker 1993). One of the most important of these technical requirements is maintainability. To ensure completion of a software project, and to ensure maintainability of the product, a software product should always start with a set of formal requirement and specification documents including the following essential documents (Walker 1993):

- The User Requirements Specification
- The User Reference Manual
- The Technical Reference Manual
- The Product Test Plan and Specification

3.4.1 The User Requirements Specification

The User Requirements Specification is used to clearly and unambiguously specify the requirements of the project from the customer's standpoint, and will normally involve repeated discussion and clarification of details with the customer. If a clear and systematic technique is not used, there is a danger of a great deal of time being wasted and little real progress being made (Walker 1993).

3.4.2 The User Reference Manual

The User Reference Manual is the instruction guide for using a program. The advantage of producing a User Manual at this early stage in the life cycle of the software is that it provides a means of double checking the system requirements (Walker 1993). Also, any subsequent changes to the design can be made to the User Manual as opposed to the actual code, saving the designer time and money.

3.4.3 The Technical Reference Manual

The principle use of the Technical Reference Manual is to successively refine the design down to its lowest logical level whilst ensuring that it remains wholly consistent with the system specifications (see User Requirements Specification and User Reference Manual). The overall system design will need to be reviewed in terms of completeness, consistency, necessity feasibility, and correctness in much the same way as the system requirements. It is also important that the system is designed to consist of well defined logical and physical modules to maximize testability during this and subsequent phases. Each module interface must be well defined at all levels (Walker 1993).

3.4.4 The Product Test Plan and Specification

The Product Test Specification must contain a basic set of test cases to clarify and determine the measurability of each requirement. Acceptance criteria need to be decided upon and accompanied by details of input data and expected results. The Product Test Plan must include details of what is to be tested, and expected results and acceptance criteria (Walker 1993).

3.5 Chapter Summary

The increasing trend towards solving complex High Voltage and Power Systems problems using software makes the software in this area a very important aspect of the High Voltage field in general. Because of this, it is essential that software in this field is as usable and maintainable as possible. To design software for usability, it is essential to adhere to design strategies which ensure that the software has visual clarity, consistency, informative feedback, explicitness, appropriate functionality, flexibility and control, error prevention and correction, and user guidance and support. To design software for maintainability, the design should be kept as close as possible to Object-oriented design and the product should be well designed including documenting the User Requirements Specification, the User Reference Manual, the Technical Reference Manual, and the Product Test Specification and Plan prior to beginning the coding. Since the required task is to write a graphical user interface for ATP, it is essential to analyze ATP before the design of the GUI begins.

4 ATP

4.1 Introduction

ATP is a computer program written in Fortran in the 1960s, which is capable of simulating transient conditions and phenomena in power machinery and distribution networks (Dommel 1986). The following chapter includes descriptions of:

- what ATP can simulate and solve,
- the algorithms used in ATP and how ATP works, and
- the data card interface of ATP.

4.2 What ATP can simulate

ATP can simulate the following machines and elements:

linear, uncoupled lumped elements such as

- resistances,
- capacitances and inductances,

linear, coupled lumped elements including

- multi-phase nominal π circuits,
- overhead transmission lines and underground cables,
- transformers,
- simple voltage and current sources,
- dynamic rotating machinery including the three-phase synchronous machine,
- the universal machine,
- switches and diodes, and
- surge arrestors and protective gaps.

as well as:

- Resistances with user-defined V-I characteristics,
- time-dependant resistances,
- saturating inductances, and
- hysteresis and residual flux in inductances

(Dommel 1986).

Another option which is offered is Transient analysis of control systems (TACS). TACS offers the ability to simulate Fortran relationship elements, controller topology elements, and a number of special control devices such as the relay controlled switch, the level-triggered switch, the simple differentiator, and the controlled integrator.

4.3 How ATP works

ATP simulations work on the basis of node voltages. To keep the explanations in this section brief, only single-phase network elements will be considered. In a simple circuit such as the circuit in *figure 3* below, if the voltages and currents have already been computed at time instances 0, δt, $2\delta t$, etc., up to $t-\delta t$, and then the solution must be found at instant t, a set of simple equations can be used to solve the nodal voltages and currents as follows:

At any time t, using KCL (Kirchoffs Current Law),

$$i_{12}(t)+i_{13}(t)+i_{14}(t)+i_{15}(t)=i_{1}(t) \quad \ldots\ldots(1)$$

Where $i_{12}(t)$ is the branch current between nodes 1 and 2 etc.

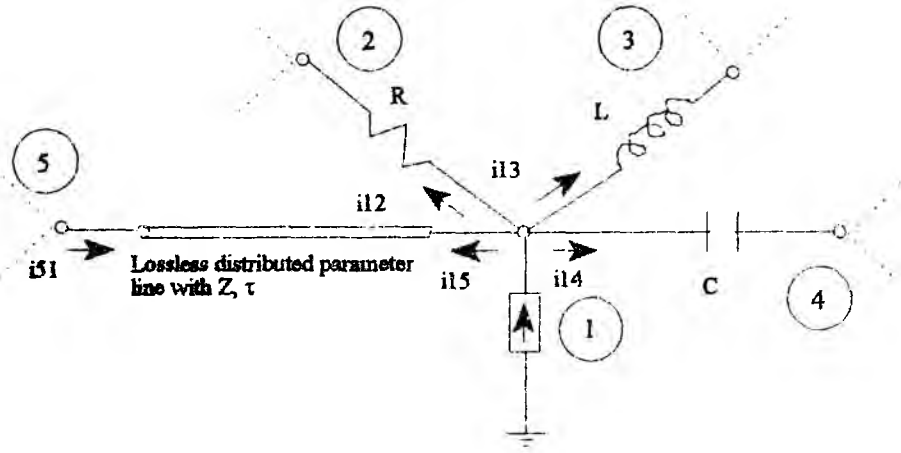


Figure 3 - Simple Network Around Node 1

Node voltages are used as state variables in ATP. It is therefore necessary to express the branch currents as functions of the node voltages. For resistances, this is done using the equation:

$$i_{12}(t) = \frac{1}{R}[v_1(t) - v_2(t)]$$

For the branch with the inductive component, the equation solved is:

$$v = L\frac{di}{dt}$$

with a central difference equation (or Trapezoidal Rule equivalent):

$$\frac{1}{2}[v(t) + v(t-\delta t)] = \frac{L}{\delta t}[i(t) - i(t-\delta t)] \quad \ldots\ldots(2)$$

In the case of the circuit in *figure 3*, this equation can be written as:

$$i_{13}(t) = \left(\frac{\delta t}{2L}\right)[v_1(t) - v_3(t)] + hist_{13}(t-\delta t) \qquad \ldots\ldots (3a)$$

with

$$hist_{13}(t-\delta t) = i_{13}(t-\delta t) + \left(\frac{\delta t}{2L}[v_1(t-\delta t) - v_3(t-\delta t)\right] \qquad \ldots\ldots (3b)$$

Similarly for the branch 1-4, the equation for the capacitive component can be derived as:

$$i_{14}(t) = \left(2\frac{C}{\delta t}\right)[v_1(1) - v_4(t)] + hist_{14}(t-\delta t) \qquad \ldots\ldots (4a)$$

with $hist_{14}(t - \delta t)$ again known from the preceding time steps:

$$hist_{14}(t-\delta t) = -i_{14}(t-\delta t) - \left(2\frac{C}{\delta t}\right)[v_1(t-\delta t) - v_4(t-\delta t)] \qquad \ldots\ldots (4b)$$

For the transmission line between nodes 1 and 5, it can be shown that:

$$i_{15}(t) = \left(\frac{1}{Z}\right)v_1(t) + hist_{15}(t-\tau) \qquad \ldots\ldots (5a)$$

where

$$hist_{15}(t-\tau) = -\left(\frac{1}{Z}\right)[v_5(t-\tau)] - i_{15}(t-\tau) \qquad \ldots\ldots (5b)$$

where:

τ = (line length/wave velocity)

As can be seen from the above equations, using values of i and v from previous time steps, and using equation (1), the network can be solved.

For any type of network with n nodes, a system of n such equations can be formed,

$$[G]\,[v(t)] = [i(t)] - [hist], \qquad \text{.....(6a)}$$

where

[G]= n*n symmetric nodal conductance matrix,
[v(t)]= vector of n node voltages,
[i(t)]= vector of n current sources, and
[hist]= vector of n known "history" terms.

Usually, some nodes have known voltages either because they have voltage sources connected to them, or because the nodes are connected to ground. If this is the case, equation (6) is partitioned into a set A of nodes with unknown voltages, and a set B, of nodes with known voltages. The unknown voltages are then found by solving the equation:

$$[G_{AA}][v_A(t)] = [i_A(t)] - [hist_A] - [G_{AB}][v_B(t)] \quad \text{.....(6b)}$$

for $[v_A(t)]$.

Originally, the ATP was written for cases starting from zero initial conditions, making terms like $hist_{13}$, $hist_{14}$ and $hist_{15}$ in the previous equations equal to zero. This is not the case however, in ac steady state initial conditions, where simulations are started from power frequency. Although it is possible to read in such ac steady state initial conditions, this is very troublesome, and a routine which works out these conditions was incorporated into ATP. Using node equations again, where in the equation:

$$I_{12} + I_{13} + I_{14} + I_{15} = I_1, \quad \text{.....(7)}$$

the currents I are complex phasor quantities $|I|.e^{j\alpha}$, it can be shown that again, for any type of network with n nodes, a system of n equations can be formed with:

$$[Y][V] = [I]. \quad \text{.....(8)}$$

where

[Y] = symmetric nodal admittance matrix, with complex elements,
[V] = vector of n node voltages (complex phasor values), and
[I] = vector of n current sources (complex phasor values).

Again, equation 8 is separated into a set A of nodes with unknown voltages, and a set B of nodes with known voltages. The unknown voltages are then found by solving the equation:

$$[Y_{AA}][V_A] = [I_A] - [Y_{AB}][V_B].$$

(Dommel 1986)

4.4 The ATP data card interface

ATP works on the basis of ASCII data files. Each 80 column line in the user's data file is known as a data card. The totality of data cards in the user data file defining the simulation is known as a data case. The data cards fall into four 'classes':

- Power system data cards,
- ATP control cards ($-cards),
- ATP setup cards (request cards), and
- Comment cards

The data cards have to be placed in a definite sequential order according to class:

First in the sequence are the initializing cards,

second the Power system branch cards. These declare

- linear and non-linear power system branches
- transmission lines and cables
- transformers

Third are the Power system switch cards. These declare:

- circuit breakers (switches)
- diodes
- SCR's

Fourth in the sequence are Power system source cards. These declare:

- voltage sources
- current sources
- rotating machinery

Fifth in the sequence are Initial condition cards (optional). These declare the initial conditions, overriding the default initial conditions (zero).

Sixth are the Output variable request cards. These are used to either specify the variables individually, or to request the voltages of all the nodes.

Seventh in the sequence are the User-defined source cards (optional). These are used to define a source point by point, one data card per time-step with multiple user-defined sources accommodated on the same data card (a termination card is used to indicate that this part of the sequence is over).

Eighth and last in the sequence are the Plotter directive cards. These simply specify whether a plot is required, and the output to which this plot should be written. (Van Coller 1993)

The output file from ATP is in a form known as .4, which is a form of data file. Clearly this form of user interface is very time consuming to learn as well as to use. For more detail on ATP data cards, see Appendix E - ATP Case Studies.

2.5 Chapter Summary

ATP performs simulations by solving node currents and voltages using the Trapezoidal rule. ATP has a data card interface which involves writing data cards or lines, and placing them in a strict sequential order. The output file from ATP is in a format known as pl4, which is a DOS text format. Since ATP was written in the late 1960s, a number of attempts have been made to write interfaces for ATP. The latest interface for ATP is called ATPDRAW, and is at present shipped with ATP.

5 THE PRESENT GUI FOR ATP (ATPDRAW)

5.1 Introduction

The user interface currently supplied by the Bonneville Power Administration or BPA (distributors of ATP) is a GUI (called ATPDRAW), which runs on the MS DOS platform. The interface is a pre-processor to ATP, and as such runs totally separately from ATP. ATPDRAW works on the General Interface Generating System or GIGS (Hoidalen 1992). GIGS is developed by the DSL (Center for computer aided learning) at the university of Trondheim in Norway, where ATPDRAW was developed. GIGS consists of a resident graphical driver and a library of routines to be used for this platform. A user-created application can be run on the GIGS platform. GIGS handles the mouse cursor, performs pop-up menus and supports a large run-time library of graphical routines which can be used by the application. Although this system is far better than the standard version of ATP, it is lacking in many respects which has led to this interface not being widely accepted as a replacement for the present ATP interface. The following chapter details:

- the design of ATPDRAW,
- the usability of ATPDRAW,
- the advantages of ATPDRAW over ATP, and
- the disadvantages of ATPDRAW.

5.2 The Design of ATPDRAW

The design and implementation of ATPDRAW were based entirely on the so-called 'structured approach' (Hoidalen 1992). It was initially written in Turbo Pascal 5.5 and later converted to Turbo Pascal 7.0, both versions being written in the procedural paradigm, making it less maintainable than an equivalent GUI written in the object oriented paradigm (Orbach 1993).

Structured programming and design involves breaking a program up into functions, which are separate from the application's data. This means that if the data is changed, the function code must also be changed. Every instance of the data being passed to a function within the program requires changing of the function code every time the data is changed. Adding functionality to a structured program would involve understanding the entire program architecture (and details of the entire program), to ensure that all variables are taken into account. In structured programming, a common occurrence is that pieces of data flow unchanged or unwanted through the majority of the modules they pass through (Jamsa 1984). Object-oriented programming (OOP) and design (OOD), on the other hand, involves the encapsulation of both data and functions into entities called objects (Booch 1994). This means that if a variable name is changed, only the object in which this variable is defined must be changed. Adding functions does not require an understanding of the entire program, only of a single object. Clearly the fact that ATPDRAW is based entirely on a procedural paradigm design is a disadvantage in terms of its maintainability.

5.3 The ATPDRAW Interface

ATPDRAW is a menu driven, DOS based interface which allows the user to select an element from a list of different element types, and then to place it on the circuit at any position. ATPDRAW conforms to some of the IBM CUA standards while it does not conform to others (see Chapter 2.3):

- Visual clarity - Although most of the information available to the user is uncluttered, there are numerous situations in which circuit elements appear extremely cluttered on the screen, without the option of changing this (a good example of this is when a three phase transmission line is connected to three separate simple RLC branches). Another important area where this field is lacking is that the entries required to describe each element are written in abbreviated format, which makes ATPDRAW very difficult to use by anyone who is unaccustomed to ATP itself. Clearly this is a very serious flaw in ATPDRAW.

- Consistency - ATPDRAW is consistent in it's screen layout within itself.
- Informative feedback - In the area of informative feedback, ATPDRAW is lacking in many respects. No information is given to the user at any time. For users who are unaccustomed to ATPDRAW, this lack of feedback could be very time consuming to overcome.
- Explicitness - ATPDRAW does not have a very clear and explicit methodology. At first glance it is not clear how to use ATPDRAW, and the user would be required to study the user manual in detail. Much of this lack of explicitness is due to the use of the GIGS platform, which is not a standard platform known by many people.
- Appropriate functionality - There is much functionality which is missing from ATPDRAW, and it's general functional flexibility is very low.
- Flexibility and control - ATPDRAW is extremely inflexible, and it is not possible to change any aspect of the interface to make it more comfortable for specific people to use.
- Error prevention and correction - Error trapping in ATPDRAW is not sufficient, with many serious errors such as crashes occurring during the running of ATPDRAW. However, users are able to check their inputs and correct them if necessary before the inputs are processed.
- User guidance and support - ATPDRAW has absolutely no On-line help. This is a critical part of modern interfaces, which is an extremely important aspect of making a program usable.

If the same interface was written for the Windows platform, many of these problems would automatically be solved, however it would still be necessary to add features such as on-line help.

5.4 The Advantages of ATPDRAW over ATP

Since ATPDRAW is a graphical user interface, it has many advantages over the standard data card / data file ATP interface:

- In ATPDRAW, to describe an element, it is required to click on that element. This produces a screen relating to that specific element which allows the user to enter the relevant details in any order, and in any form. This is a very significant advantage over the element details entry in ATP itself since in that case the same details would be in a specific point of a specific line in the data file. However, the use of abbreviations to label the entries in these screens means that a user must be accustomed to using ATP itself in order to know what each entry means, or to have the user reference manual of ATPDRAW available at every step of the simulation setup.
- Other advantages of ATPDRAW over ATP include the fact that ATPDRAW is graphical, and because of this, a circuit need not be drawn on paper separately to the simulation itself.
- Also, ATPDRAW allows for automatic node naming, which is a very useful feature.
- Other features of ATPDRAW are that it is interactive, menu driven, and works in a mouse driven environment (Hoidalen 1992).

5.5 The Disadvantages of ATPDRAW

As has been shown, ATPDRAW has a significant number of disadvantages which make it less appealing to the average ATP user. The disadvantages of ATPDRAW include the following:

- The fact that the ATPDRAW software was designed in a structured way makes it less maintainable than an equivalent GUI written in the object-oriented paradigm.
- In an important High Voltage software application such as ATP, new functions are often added to the application as research progresses. The poor maintainability of an interface

means that it will be difficult and very time consuming to add the operation of these functions to the GUI (unless the original coder of the program is available to do this). Also if any problems are encountered with the GUI, it would be difficult to correct these problems.

- Although ATPDRAW is a great improvement on ATP itself, it is in many cases not very flexible in functionality. A good example of this is that it comes with only 12 printer drivers, and if the user wishes to use any printer other than t[illegible] specified, he/she has to write their own printer driver. Another example of this is that it does not allow for interfacing with a CAD package or database type files. This means that if a circuit was initially drawn in a CAD package and now requires an ATP simulation, it would have to be redrawn in the GUI.
- Also, It does not have any on-line Help, and it does not have any feedback messages to help the user during the running of ATPDRAW.

The user's ability to learn an interface is crucial to its acceptance (Ege et al 1992). Since ATPDRAW is graphical, it is relatively easy to use. It is however, still time consuming to learn how to use such a GUI, since it does not operate in a very common standardized manner (such as programs which run on the Microsoft Windows platform) Regular users of ATP who are already accustomed to using the data file interface would therefore have to learn how to use this GUI, and this could be time consuming. Since most software users are accustomed to the MS Windows platform, an equivalent Windows GUI would not suffer from the same setback.

5.6 Chapter Summary

The interface which is presently supplied with ATP is called ATPDRAW. ATPDRAW has several advantages over the standard ATP interface, including that it is menu driven and works in a mouse driven environment, and most importantly that it is graphical. However, ATPDRAW suffers from several setbacks with respect to its usability, many of which are very serious. These include the lack of proper feedback to the user, the use of abbreviations for labels describing element detail entries, the lack of on-line Help, and the inflexibility of ATPDRAW. Also, ATPDRAW was

designed entirely using the procedural or 'structured' paradigm, making it less maintainable than the equivalent object-oriented interface. To ensure that the proposed GUI for ATP is more maintainable than ATPDRAW, it was found necessary to analyze the concepts of object-oriented programming and object-oriented design in more detail.

6 OBJECT ORIENTED PROGRAMMING AND DESIGN

6.1 Introduction

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships (Booch 1994 and Orbach 1995).

Object-oriented programming uses objects, not algorithms, as its fundamental logical building blocks.

Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design (Booch 1994). The following chapter introduces the basic concepts of object-oriented programming and design, as well as Booch class structure diagrams.

6.2 Object Oriented Programming Concepts

6.2.1 Abstraction

The essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply-defined conceptual boundaries relative to the perspective of the viewer; the process of focusing upon the essential characteristics of an object. Abstraction is one of the fundamental elements of the object model (Booch 1994).

6.2.2 Classes

A class is a set of objects that share a common structure and a common behaviour. The terms *class* and *type* are usually (but not always) interchangeable; a class is a slightly different concept than type, i.e. that it emphasizes the classification of structure and behaviour (Booch 1994).

E.g.. Pascal

```
VAR  A, B, C : REAL
```

In this example, REAL can be said to be a class, and A, B, and C objects of type REAL. The object oriented approach extends this concept so that the user can define classes. For e.g..

```
triangle1, triangle2, triangle3 : TRIANGLES
```

A class is a black box which can give information about itself. The procedures and techniques which the class uses to get this info are totally hidden from other classes (or users).

6.2.3 Inheritance

This is a relationship among classes, wherein one class shares the structure or behaviour defined in one (single inheritance) or more (multiple inheritance) other classes. Inheritance defines an "is-a" or "type of" hierarchy among classes in which a subclass inherits from one or more generalized superclasses; a subclass typically specializes its superclasses by augmenting or redefining existing structure and behaviour (Booch 1994).

e.g.

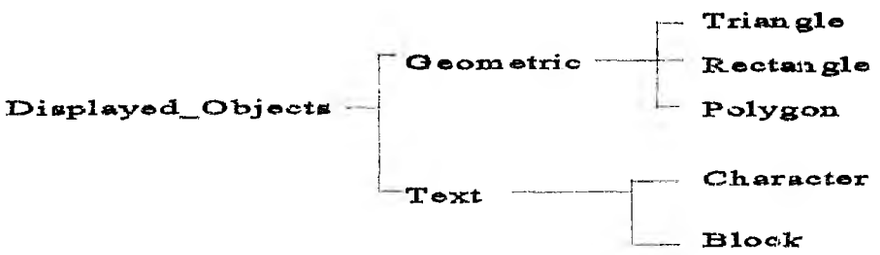


Figure 4 - Inheritance

In *Figure4* above, Triangle, Rectangle and Polygon are all of type Geometric, Geometric is a type of Displayed_Objects. Similarly Character and Block are both of type Text, and Text again is of type Displayed_Objects. This means that the Triangle, Rectangle and Polygon classes all inherit the Geometric class, which along with the Text class inherits the Displayed_Objects class.

Although it is possible for a class to have more than one 'parent' (only inherits properties of one class), this will usually create problems. Because of this, a class usually only has one parent.

6.2.4 Polymorphism

This is a concept in type theory, according to which a name (such as a variable declaration) may denote objects of many different classes that are related by some common superclass; thus, any object denoted by this name is able to respond to some common set of operations in different ways (Booch 1994).

E.g..

A:=B+C

- If A, B and C are REAL, then '+' implies Floating Point addition.
- If A, B and C are INTEGER, then '+' implies integer addition.
- If A, B and C are STRING, then '+' implies Concatenation.

This means that the '+' symbol is polymorphic, ie. it can mean different things in different cases.

In Object-oriented programming a method can be defined such that it is polymorphic.

6.2.5 Message Passing

All approaches to software design are based on some 'computational metaphor'. Most programmers will be familiar with the 'Procedure Calling' metaphor. In the Object-oriented approach, we usually (but not always) use the 'Message Passing' metaphor. Instead of calling procedures, we send messages to objects and ask the objects to give the required output. (Dwolatzky 1994)

E.g.. Menu_Show_yourself (blue, white) could be asking a menu to show itself with a blue foreground and a white background, whereas in procedural programming this would be done with a procedure: Show_Menu.

6.2.6 Late Binding

Late or dynamic binding, denotes the association of a name (such as a variable declaration) with a class; dynamic/late binding is a binding in which the name/class association is not made until the object designated by the name is created at execution time (Booch 1994).

6.3 Booch Diagrams and Class Structure Design

Grady Booch developed a graphical system whereby one can draw the class structure of a project. A *class diagram* is used to show the existence of classes and their relationships in the logical view of a system. A single class diagram represents a view of the class structure of a system. The two essential elements of a class diagram are classes and their basic relationships. (Booch 1994)

6.3.1 Classes

Figure 5 below shows the icon which is used to represent a class in a class diagram. Its shape is that of a cloud. Each class is required to have a unique name (especially when using programming languages such as C++ or Smalltalk).

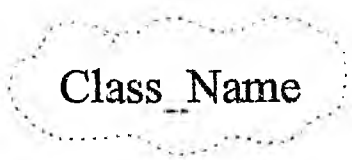


Figure 5 - Class Icon Representation

6.3.2 Class Relationships

Classes rarely stand alone; instead they collaborate with other classes in a variety of ways. For the purpose of this thesis, it is only essential to cover three of these collaborations. These are 'inheritance', 'has', and 'uses'. The symbols used for these three collaborations can be seen in *figure 6* below. In the figure, Class A 'Has' or contains Class D, Class A 'Uses' Class C, and Class B inherits the methods and properties of class A.

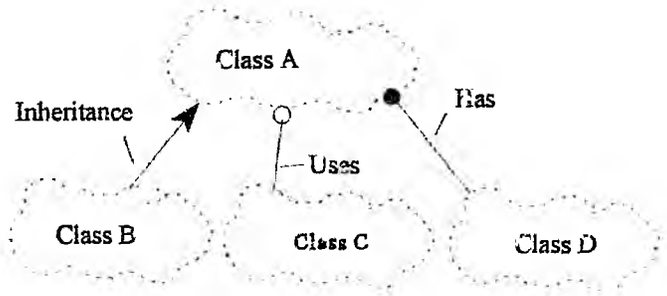


Figure 6 - Class Relationships in Booch Diagrams

To explain how these relationships work, it is best to use an example. The example discussed can be seen in *figure 7* on the following page. It involves a very simple interface for the Internet News, and its basic class structure.

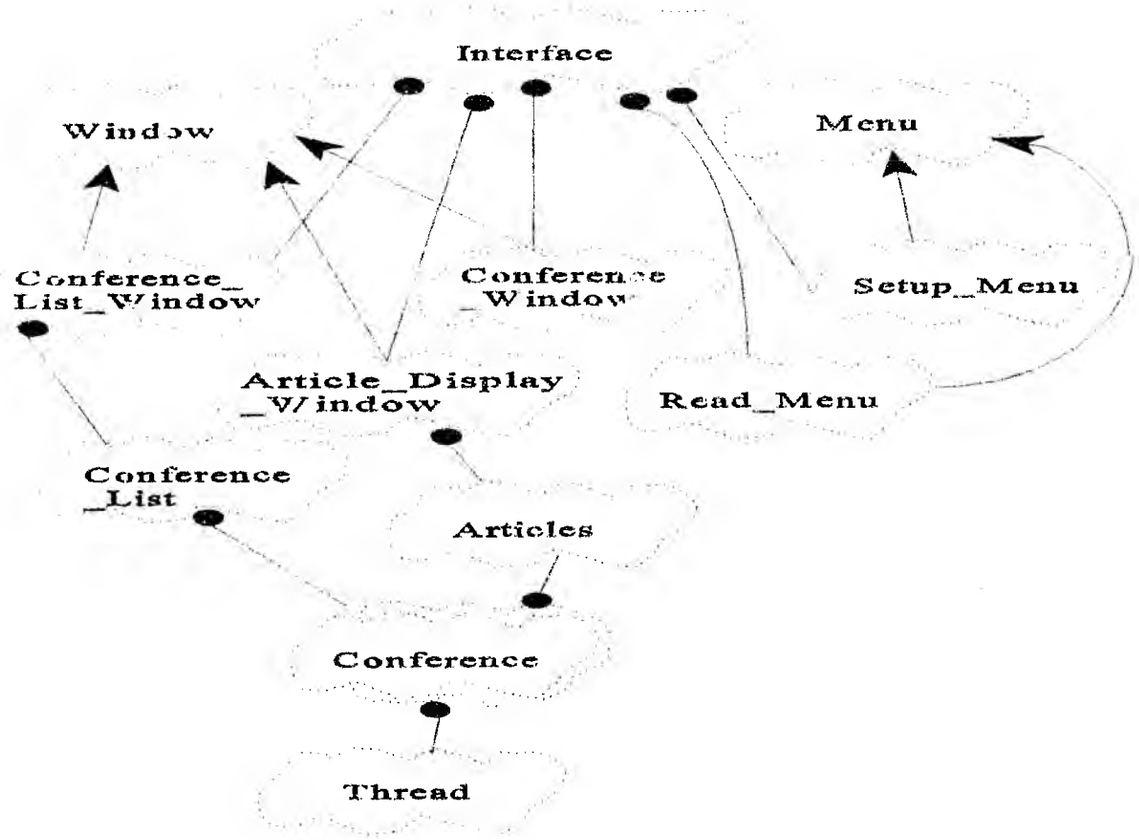


Figure 7 - Internet News Reader Booch Class Diagram

From *figure 7*, it can be seen that the class hierarchy is as follows:

The main class is the *Interface* class. This class contains or 'has' the *Conference List Window* class, the *Conference Window* class, the *Article Display Window* class, the *Setup Menu* class, and the *Read Menu* class.

Since the classes *Conference List Window*, *Conference Window*, and *Article Display Window* are all types of Windows, they all inherit the class *Window*. Similarly, the classes *Setup Menu*, and *Read Menu* are both Menus, and both inherit the *Menu* class.

The class *Conference List Window*, contains the class *Conference List*. The class *Conference Window*, contains all the *Conference* classes, which are also contained by the *Conference List* class. The class *Article Display Window* contains the class *Articles*, and the class *Conference* contains the class *Thread*, which contains the class *Articles*.

This hierarchy can be explained as follows:

The user of the program deals with the interface only. This means that only the class Interface really deals with the outside world.

If a conference (or group) list is needed, the interface will call the class *Conference List Window* to show itself. The *Conference List Window* will in turn call on the *Conference List* class to show itself within the Conference List Window class. The *Conference List* then calls all of the *Conference* classes within the *Conference List* class to announce themselves. This will give the user a list of all the conferences or groups available in the News package. If the user then pages through the list, each time there is a change, the Interface class calls the *Conference List Window* class again, and this in turn gets the *Conference List* to remove whatever conferences are no longer meant to be on the list, and to add the relevant conferences on to the list. The rest of the hierarchy of the system works in a similar fashion.

6.4 The Advantages of Using OOD/OOP

In comparing the structured (or 'procedural') paradigm to the object-oriented paradigm, (Wilde et al 1993) state that object-oriented methods provide better data abstraction and better information hiding, allowing for concurrency more readily, and respond better to changes in the real world. Wilde et al also describe the goal of object-orientation as being to make change easier, stating that objects in the program match objects in the real world more closely, so real world changes should

be easier to map to program modifications. This is clearly the most important aspect of the object-oriented paradigm since one of the largest problems with modern software is the lack of maintainability. However, (Booch 1994) states that both the object-oriented, and the structured paradigm are important, each having its own particular advantages over the other.

Another advantage of using OOP and OOD is that graphical environments have an inherent complexity that makes them good candidates for the benefits of OOP (Urlocker 1990). This complexity also translates directly to the need for high maintainability of GUIs, which is provided by the object-oriented paradigm. As a result, OOP has caught on widely on platforms such as the Mac, Microsoft Windows, and the NeXT machine.

6.5 Object-oriented Tools

The range of object-oriented tools on the market is very large. There are object-oriented environments, object-oriented applications, object-oriented databases, architectures and user-interfaces. Then of course there are object-oriented programming languages, from the conservative Ada, to the radical Smalltalk, with C++, Objective C, and Object Pascal somewhere in between (Cox 1990). Another recently released product which could soon become very popular is Borland's Delphi programming environment. Delphi combines the best features of the object-oriented Turbo Pascal for Windows language with a top-notch graphical development environment that resembles - then goes beyond - Visual Basic (see chapter 7) (Powell 1995).

C++ is by far the most popular object-oriented programming language available, with the number of users doubling every 7.5 to 9 months (Cline 1995).

6.6 Chapter Summary

Object oriented programming makes use of a number of new concepts to change the perspective of programming from the procedural paradigm to the object-oriented paradigm. The concepts used are *abstraction*, *classes*, *inheritance*, *polymorphism*, *message passing* and *late binding*. Booch class structure diagrams are used to represent class structures using inheritance and contains lines, and clouds or blocks to represent classes. Object-oriented programming has several advantages over 'structured' or procedural programming, including the better maintainability of large object-oriented programs than the equivalent 'structured' programs. The most popular object-oriented programming tool is C++, and had the program been written in an entirely object-oriented language, it woud have been in C++ due to the platform porting capabilities of this language.

7 USING VISUAL BASIC TO WRITE OBJECT-BASED AND EVENT DRIVEN SOFTWARE

7.1 Introduction

A fairly recent tool to emerge as a popular and extremely powerful product is Visual Basic 3.0. Visual Basic is an interface design tool which is primarily used to write front ends for accessing Unix and DOS based databases in a user-friendly environment. One other use of Visual Basic however is to create applications that fully exploit the graphical user interface (Microsoft 1993). The following chapter includes descriptions of:

- The Visual Basic programming system, Visual Basic controls, and forms,
- the object-based nature of Visual Basic,
- a comparison of the maintainability of Visual Basic software and C++ software, and
- a comparison of the usability of Visual Basic and C++.

7.2 Visual Basic

Visual Basic is a programming system which can be used to build complete applications for the Windows environment. In Visual Basic, the layout of an application's user interface is designed with a set of interactive graphical tools, such as dialog editors. The interactive tools used in Visual Basic are known as Controls. By selecting the controls such as text boxes, check boxes, and command buttons, the user can build a complete user interface interactively (Microsoft 1993). These controls will be described briefly in the following section. The first step to creating a Visual Basic application is to create the interface: the forms, the controls, and other objects the user will see and use. This is followed by setting the properties and writing Visual Basic code based on events associated with each control to make the interface active. A Visual Basic form is any screen

which is designed using Visual Basic.

7.3 Visual Basic Controls

The Visual Basic Toolbox contains the tools which are used to draw controls on the application's forms. Some of the controls available with the Professional Edition of Visual Basic 3.0 are as seen in *figure 8* below:

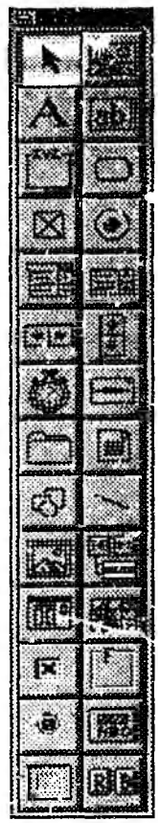


Figure 8 - Visual Basic Controls

Forms and Controls make up the entire visual construction of Visual Basic programs.

7.4 Object Based and Event Driven Programming with Visual Basic

The general structure of a Visual Basic form can be seen in the following Booch diagram which represents the frmATP form (the main screen of ATP For Windows - see Chapter 10).

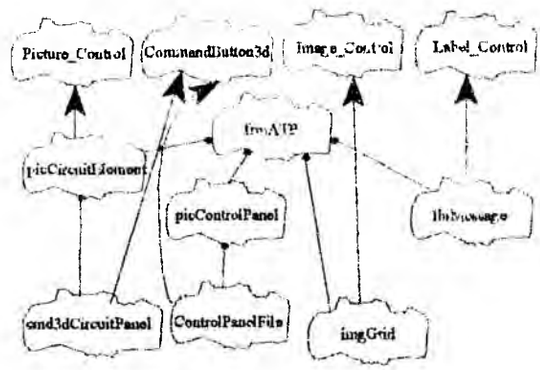


Figure 9 - Booch Diagram of a Visual Basic Form

As can be seen in the above Booch diagram (*figure 9*), The frmATP form contains the controls required for the screen (see section 10.2). Each control including the frmATP control, inherits the events and properties of the general control of its type. An example of this is that the cmd3dCircuitPanel control, which is an array of 3d command buttons, inherits the properties of the general control called CommandButton3d. This inheritance includes inheriting events such as the Click event which occurs every time a user clicks on this command button, and the DragDrop event which occurs every time something which is being dragged, is dropped onto the command button. This means that if the programmer would like something to occur every time the user of the program clicks on the command button with the mouse, this event is already automatically available to the programmer, and he/she simply has to fill in what should happen at every instance of this event. This functionality is equivalent to overriding an existing inherited method in C++.

Similarly, the properties of the command button such as the background colour, height, top, etc.,

are automatically assigned to this control because of the inheritance of the general CommandButton3d control, and these can be specified for each 3d command button placed in the form. The list of these properties can be seen in *figure 10* below.

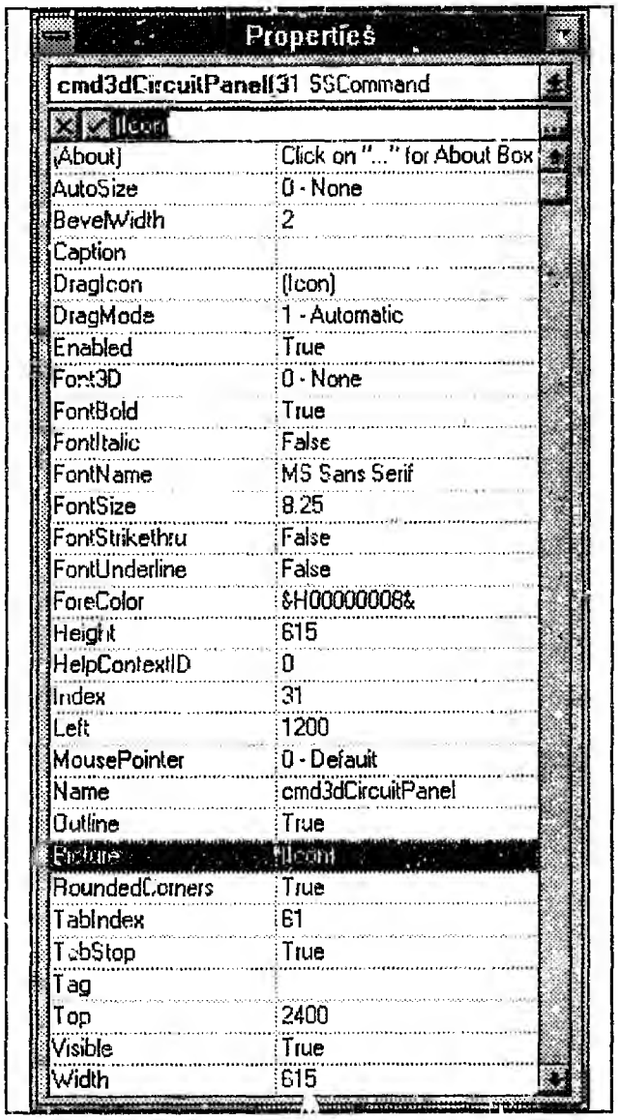


Figure 10 - The Properties Inherited by the cmd3dCircuitPanel Control

The inheritance shown in *figure 10* occurs in every screen or form throughout the Visual Basic module of ATP for Windows, and is clearly a very useful feature of Visual Basic which makes it extremely easy to maintain.

7.5 Comparing Visual Basic Maintainability to C++ Maintainability

Although Visual Basic is object based, and is easy to maintain, it has several disadvantages in the area of maintainability when compared to object-oriented languages such as C++. The most important disadvantage of Visual Basic is that although each 'visual' control in Visual Basic inherits properties, events and methods from the general control of the same type, once code is written to a specific control, this code can no longer be inherited by other controls which are similar in behaviour, and the code for the new controls has to be rewritten. Also, if code is written in 'modules' (which are used for code not directly related to one specific control), this code is by definition structured. (although it is still possible to make it easily maintainable). An example of this can be seen when comparing the FormSimpleRLCInitialize method in the Simple RLC Branch form of ATP for Windows, to the FormDistributedRLCInitialize method of the Distributed RLC Branch / Single Transmission Line Conductor form of ATP for Windows. The Simple RLC Branch form or screen is as follows:

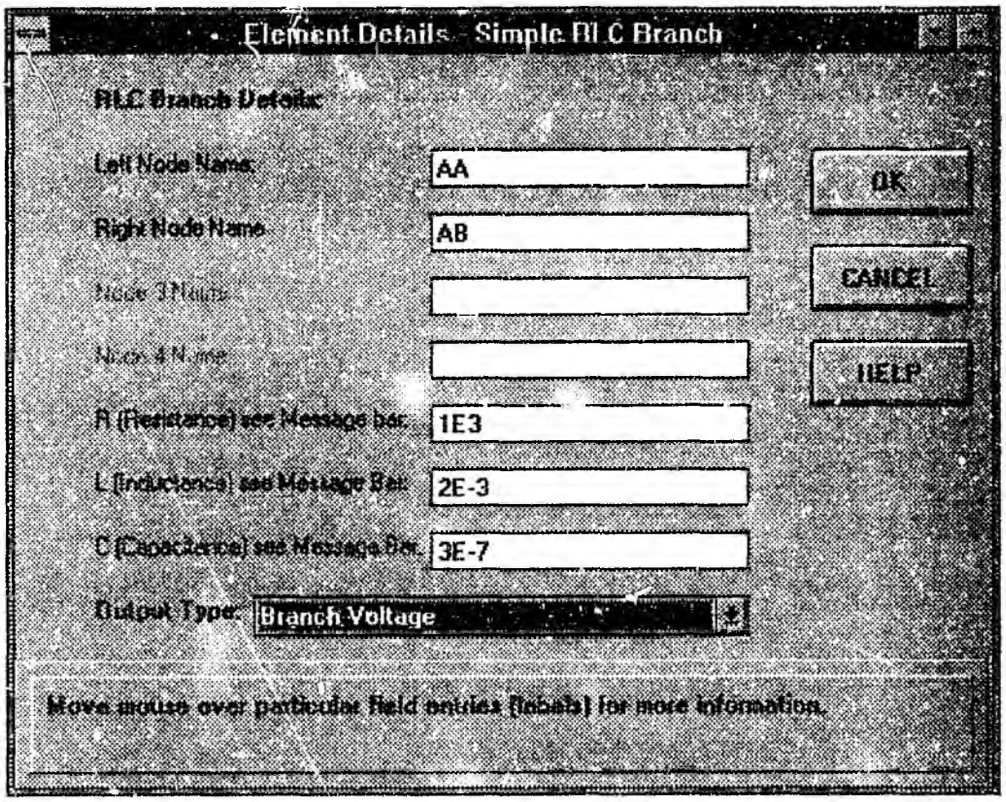


Figure 11 - The Simple RLC Branch Element Details Screen

The single transmission line conductor form is as follows:

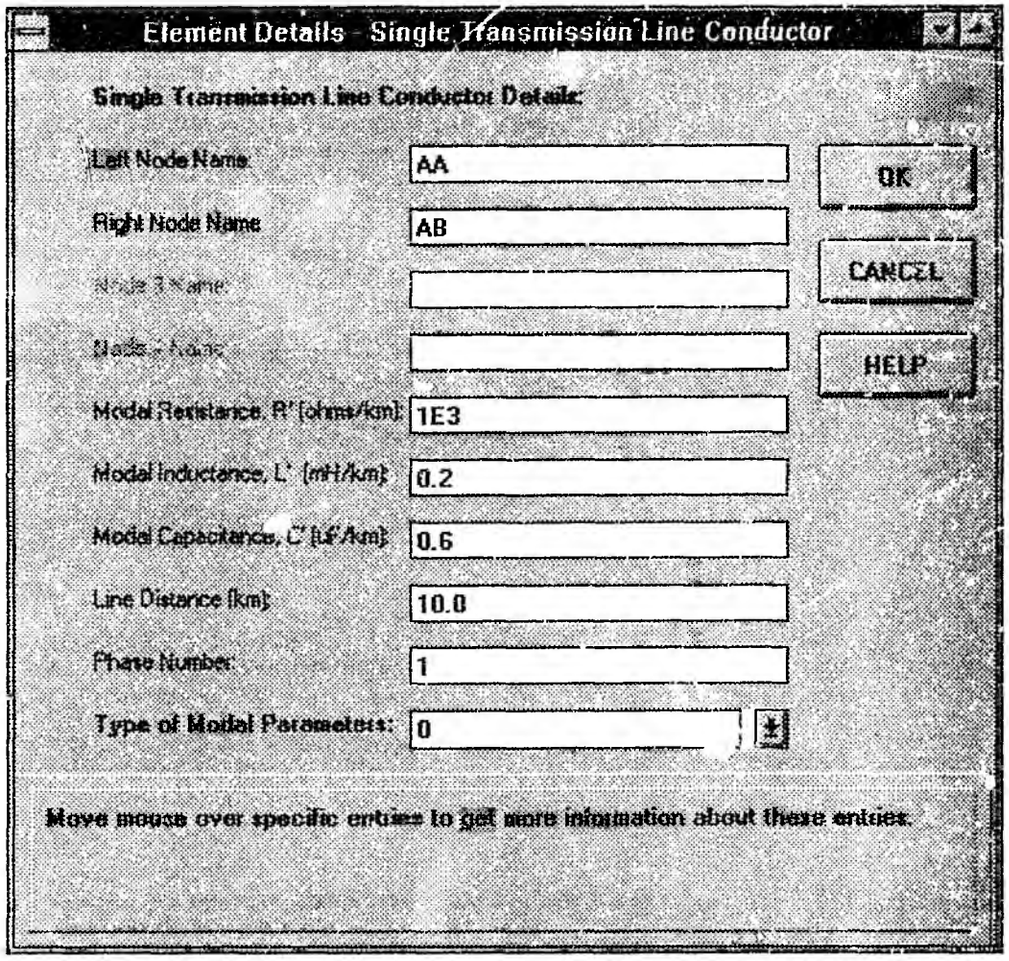


Figure 12 - The Single Transmission Line Conductor Element Details Screen

This form is clearly very similar to the form for the Simple RLC branch, and comparing the code to initialize the two forms, it is clear that inheritance would be advantageous in this case.

7.5.1 The FormSimpleRLCInitialize Code

The code used to initialize this form places the required options in the combo boxes on the Element Details screen, and checks whether or not the element being pointed to is a new element

or an existing element. If the element is an element which has previously been defined, this method places the existing information into the relevant text boxes in the Element Details screen.:

```
'Place items in combo boxes
frmSimpleRLC.cmbOutputType.AddItem ("None")
frmSimpleRLC.cmbOutputType.AddItem ("Branch Current")
frmSimpleRLC.cmbOutputType.AddItem ("Branch Voltage")
frmSimpleRLC.cmbOutputType.AddItem ("Branch Voltage & Current")
frmSimpleRLC.cmbOutputType.AddItem ("Inst net power flow into branch")

'Check if element is a new element
Dim NewElement As Integer
SimpleRLCNumber = -1
NewElement = True
SimpleRLCCount = SimpleRLCCount + 1

If SimpleRLCCount = 0 Then
   If frmATP.imgGrid(picGridIndex).Tag = -1 Then
      NewElement = True
   Else
      SimpleRLCNumber = 0
      NewElement = False
   End If
End If

If SimpleRLCCount > 0 Then
   If frmATP.imgGrid(picGridIndex).Tag = -1 Then
      NewElement = True
   Else
      SimpleRLCNumber = frmATP.imgGrid(picGridIndex).Tag
      NewElement = False
      SimpleRLCCount = SimpleRLCCount - 1
   End If
End If

'If element is not new, then place existing values into form/screen
If NewElement = False Then
   frmSimpleRLC.txtLeftNodeName.Text = RLCs(SimpleRLCNumber).LeftNodeName
   frmSimpleRLC.txtRightNodeName.Text = RLCs(SimpleRLCNumber).RightNodeName
   frmSimpleRLC.txtNode3Name.Text = RLCs(SimpleRLCNumber).Node3Name
   frmSimpleRLC.txtNode4Name.Text = RLCs(SimpleRLCNumber).Node4Name
   frmSimpleRLC.txtResistance.Text = RLCs(SimpleRLCNumber).Resistance
   frmSimpleRLC.txtCapacitance.Text = RLCs(SimpleRLCNumber).Capacitance
   frmSimpleRLC.txtInductance.Text = RLCs(SimpleRLCNumber).Inductance
   frmSimpleRLC.cmbOutputType.ListIndex = RLCs(SimpleRLCNumber).OutputType
   frmSimpleRLC.cmdOK.Enabled = True
Else

   frmSimpleRLC.cmbOutputType.ListIndex = 0
   frmSimpleRLC.cmdOK.Enabled = False
   SimpleRLCNumber = SimpleRLCCount
   frmATP.imgGrid(picGridIndex).Tag = SimpleRLCNumber
End If
```

7.5.2 The FormDistributedRLCInitialize Code

The code used to initialize this form places the required options in the combo boxes on the Element Details screen, and checks whether or not the element being pointed to is a new element or an existing element. If the element is an element which has previously been defined, this method places the existing information into the relevant text boxes in the Element Details screen:

```
'Add items to combo boxes
frmDistributedRLC.cmbModalParametersType.AddItem ("0")
frmDistributedRLC.cmbModalParametersType.AddItem ("1")
frmDistributedRLC.cmbModalParametersType.AddItem ("2")
frmDistributedRLC.cmbModalParametersType.ListIndex = 0

'Check if element is new
Dim NewElement As Integer
DistRLCNumber = -1
NewElement = True
DistRLCCount = DistRLCCount + 1

If DistRLCCount = 0 Then
   If frmATP.imgGrid(picGridIndex).Tag = -1 Then
      NewElement = True
   Else
      DistRLCNumber = 0
      NewElement = False
   End If
End If

If DistRLCCount > 0 Then
   If frmATP.imgGrid(picGridIndex).Tag = -1 Then
      NewElement = True
   Else
      DistRLCNumber = frmATP.imgGrid(picGridIndex).Tag
      NewElement = False
      DistRLCCount = DistRLCCount - 1
   End If
End If

'If element is not new then place existing values into form / screen
If NewElement = False Then
   Dim EmptyRest As Integer
   frmDistributedRLC.txtLeftNodeName.Text = DRLCs(DistRLCNumber).LeftNodeName
   frmDistributedRLC.txtRightNodeName.Text = DRLCs(DistRLCNumber).RightNodeName
   frmDistributedRLC.txtPhaseNumber.Text = DRLCs(DistRLCNumber).PhaseNumber

   If Not DRLCs(DistRLCNumber).Node3Name = "0" Then
      EmptyRest = True
   Else
      EmptyRest = False
   End If
   If Not DRLCs(DistRLCNumber).Node4Name = "0" Then
      EmptyRest = True
   Else
```

```
        EmptyRest = False
    End If

    If EmptyRest = False Then
        frmDistributedRLC.txtModalResistance.Text = DRLCs(DistRLCNumber).ModalResistance
        frmDistributedRLC.txtModalCapacitance.Text = DRLCs(DistRLCNumber).ModalCapacitance
        frmDistributedRLC.txtModalInductance.Text = DRLCs(DistRLCNumber).ModalInductance
        frmDistributedRLC.txtLineDistance.Text = DRLCs(DistRLCNumber).LineDistance

        frmDistributedRLC.cmbModalParametersType.ListIndex = DRLCs(DistRLCNumber).ModalParametersType
        frmDistributedRLC.cmdOK.Enabled = True
    Else
        frmDistributedRLC.txtNode3Name.Text = DRLCs(DistRLCNumber).Node3Name
        frmDistributedRLC.txtNode4Name.Text = DRLCs(DistRLCNumber).Node4Name
        frmDistributedRLC.lblModalResistance.Enabled = False
        frmDistributedRLC.txtModalResistance.Enabled = False
        frmDistributedRLC.lblModalCapacitance.Enabled = False
        frmDistributedRLC.txtModalCapacitance.Enabled = False
        frmDistributedRLC.lblModalInductance.Enabled = False
        frmDistributedRLC.txtModalInductance.Enabled = False
        frmDistributedRLC.lblLineDistance.Enabled = False
        frmDistributedRLC.txtLineDistance.Enabled = False
    End If

Else
    frmDistributedRLC.cmdOK.Enabled = False
    DistRLCNumber = DistRLCCount
    frmATPDrawingGrid(picGridIndex).Tag = DistRLCNumber
End If
```

This is clearly very similar to the code used for the simple RLC form.

7.5.3 Initializing the Two Forms Using C++

Given a situation in which the same two forms were designed and written in C++, the code for initializing the two forms would only be written once, and then inherited in the other case. This would save time and programming lines, and make it easier to change anything related to both of the initialize methods. With both the SimpleRLC class and the DistributedRLC class capable of calling these methods. This would be the ideal way of performing this task if C++ were as easy to use as Visual Basic.

7.6 Comparing Visual Basic Usability to C++ Usability

Writing applications for Windows using Borland C++ 4.5, is a lengthy and difficult task. It is done using a class library called the Object Windows Library or OWL, which is distributed with Borland C++ 4.5. Where in Visual Basic, a control is placed for every element in the screen, the equivalent process in C++ is far more lengthy, and requires more code manipulation. Duntemann claims that C++ projects often take 10 to 30 times longer to complete than Visual Basic projects (Duntemann 1995). Our own experiences have also shown that using Visual Basic for front ends of Windows applications (its main purpose), is far easier than using C++ for the same task (Orbach 1995b). For this reason, although C++ applications are more easily maintained than Visual Basic applications when it comes to the code, it is still less time consuming to code and to maintain Visual Basic front ends than the equivalent C++ front ends. Any code which is not directly linked to the interface, such as code which solves numerically intensive algorithms, will be more maintainable if written in C++ or another object-oriented language, than if it were written in Visual Basic.

7.7 Chapter Summary

Visual Basic is a visually oriented programming system which can be used to build front ends very quickly and efficiently. Although Visual Basic applications are easy to maintain, this maintainability is due to its ease of use, and object based nature for programming and designing front ends, and this maintainability does not stretch over to code which is not directly related to the front end of an application.

8 ATP FOR WINDOWS

8.1 Introduction

As has been shown, software is a very important aspect of the High Voltage and Power Systems fields, and the two most important aspects of modern software are maintainability and usability. To satisfy the usability requirements, it was decided to write the proposed graphical user interface for ATP in the Windows domain. Due to this platform selection, this GUI has been named ATP for Windows. The following chapter describes:

- the requirements for ATP for Windows including
- ensuring usability in ATP for Windows,
- ensuring maintainability in ATP for Windows, and
- the implications of the requirements on the final design and implementation.

8.2 Requirements for ATP for Windows

A graphical user interface is required for the Fortran based ATP. The interface must be capable of converting standard circuit diagrams along with the corresponding circuit information into the required input data files, and must also be capable of reading the output data files, and converting these files to graphical and numerical data within the interface. The most important general requirements for ATP for Windows are as follows:

- It must perform on the Microsoft Windows (3.1 or later) platform.
- It must be as usable and flexible as possible.
- It must be as maintainable as possible.
- It must be capable of interfacing with as many external CAD packages (such as Intergraph Microstation) as possible.

To ensure that the final product meets all the requirements, the formal requirements for ATP for Windows were written during the first stage of the project. These requirements were formalised in a document titled the User Requirements Specification (see Appendix A - The User Requirements Specification). The User Requirements Specification is a list of the formal requirements, and does not include such items as usability and maintainability.

8.2.1 Ensuring Usability

To ensure that ATP for Windows is as usable as possible, it is necessary to satisfy all of the usability standards described in chapter 2 (see section 2.3):

- Visual clarity - ATP for Windows must have the standard CUA layout which is used in Windows applications.
- Consistency - Consistency is the key to the effectiveness of the Microsoft Windows platform. One of the reasons for the choice of the Windows platforms for the GUI is to ensure consistency of ATP for Windows with other applications as well as within itself. The consistency of each screen in ATP for Windows with other screens in ATP for Windows should be ensured by designing the interface carefully, and completing the design on paper before any of the coding is performed, as well as by conforming to the IBM CUA and the Microsoft Application Design Guide for each and every screen.
- Informative feedback - Informative feedback must be ensured by adding a message bar to the interface. The message bar should guide the user by giving the user the next step possible as well as by describing the different options as the user traverses over these toolbar options with the mouse. Screen specific On-line Help should be available for the user at any stage of the program to aid in the required feedback.
- Explicitness - For ATP for Windows, it is necessary to ensure that the system is similar in many respects to other Windows applications.
- Appropriate functionality - To ensure appropriate functionality, the User Reference Manual

must be thoroughly reviewed by a number of potential users before the actual coding begins.

- Flexibility and control - To ensure flexibility of the system, the user must be given the option to change the layout of the system settings as far as possible.
- Error prevention and correction - The system must be designed to minimize the possibility of user error, with built-in error handling for those errors which do occur. Users should be able to check their inputs and correct them if necessary before the inputs are processed.
- User guidance and support - To ensure proper user guidance and support, easily accessible, and informative on-line help should be available for the user at any point or situation in the system.

8.2.2 Ensuring Maintainability

To ensure maintainability of the program, a number of steps are required:

- The GUI should be written in an object-oriented or otherwise maintainable language where possible.
- The GUI should be designed carefully before the actual coding begins and the design process should ensure that maintainability is emphasized.

8.3 Implications of the Requirements

To satisfy the requirements mentioned above, the GUI was implemented using Software Engineering Quality Principles (see Chapter 11) on the Windows platform. The GUI was written in Visual Basic due to the ease of use of Visual Basic, and the subsequent ease of maintainability of the program. A separate entirely object oriented unit which writes the data files required by ATP, was designed for comparison purposes.

8.4 Chapter Summary

To ensure the usability and maintainability of ATP for Windows, ATP for Windows must be written for the Windows platform using Visual Basic. The requirements for the interface are detailed in a formalised User Requirements Specifications document (see Appendix A - the User Requirements Specification). The actual designs achieved using these requirements as a basis can be seen in the chapters following this one.

9 THE OBJECT-ORIENTED DESIGN OF THE DATA CARD UNIT

9.1 Introduction

One of the designs considered for ATP for Windows is an entirely object-oriented design (for a more detailed description, see Appendix D - The Object-Oriented Design of the Data Card Unit). The reasons for this include the ease of maintaining object-oriented software. Although the final design of ATP for Windows is object-based as opposed to object-oriented, if C++ had been used to write ATP for Windows, it is very likely that the following design would have been used. The design was completed as a comparison to the Visual Basic design used, as well as a prospective design for future versions of ATP for Windows if these versions are developed in an entirely object-oriented language such as C++ (or Delphi).

ATP for Windows is a pre-processor to ATP, and must thus convert graphical information received from the user, to the data cards / data files needed to run ATP simulations. The unit which was designed using object oriented design would be the back end of the interface since it reads the circuit information stored in the MS Access tables, and converts this information to the required ATP data files. This section describes this unit including:

- An indication of the general structure used for the object-oriented design of the Data Card unit, as well as the structure that would be used for a complete object-oriented design of the entire program
- An explanation of the use of inheritance in this unit, and the advantages of using inheritance.
- A detailed description of two classes showing how the use of object-oriented programming and inheritance saves lines of code as well as programming time.

9.2 Using The Structure of ATP Data Cards to Select Objects

ATP data cards all have a fairly similar structure to each other, but different types of cards such as Branch cards or Source cards, have a fixed structure within their own card type (See Appendix E - ATP Case Studies). This means that the data cards can be split up into definite types, which can then be used as classes in the object-oriented approach to the design of the interface.

This approach has led to the high level design seen in *figure 13 and figure 14.*

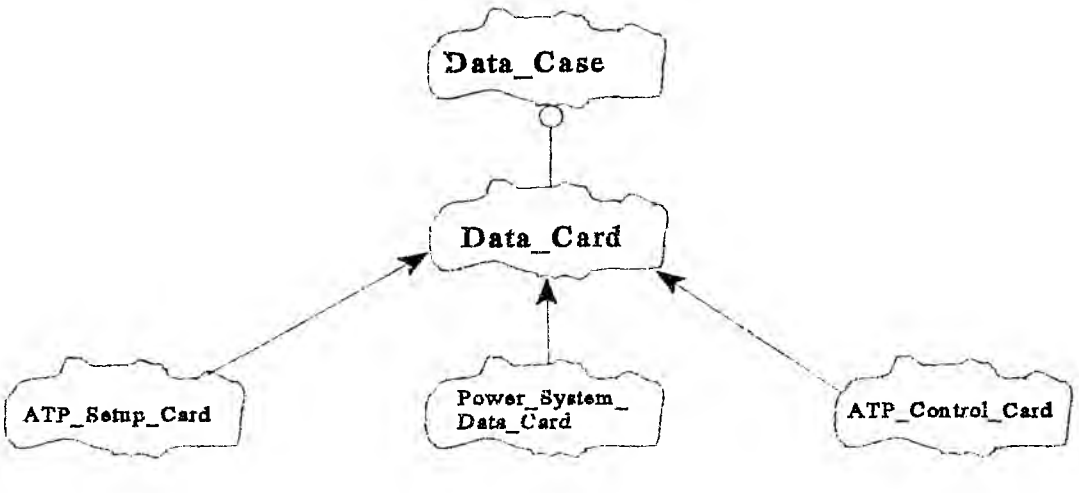


Figure 13 - The Booch Class Diagram for the Data_Case Class

In *figure 13* and *figure 14*, it can be seen that the *Data_Case* class contains the Data Card classes (*Power_System_Data_Card, ATP_Setup_Card,* and *ATP_Control_Card*). Each type of card inherits the general properties of the *Data_Card* class, the *Branch_card* class inherits the *Power_System_Data_Card* class (and hence automatically also inherits the *Data_Card* class properties and methods), and each type of Branch card inherits the properties of the *Branch_Card* class. This means that using inheritance, one can write the *RLC_Branch* class by using the general existing properties and methods of the *Branch_Card* class, and adding whatever member functions / methods are necessary.

Similarly, if at a later stage anyone wishes to add an element which is a branch card, he/she can do this, without knowing the internal workings of the rest of the program, by simply adding whatever

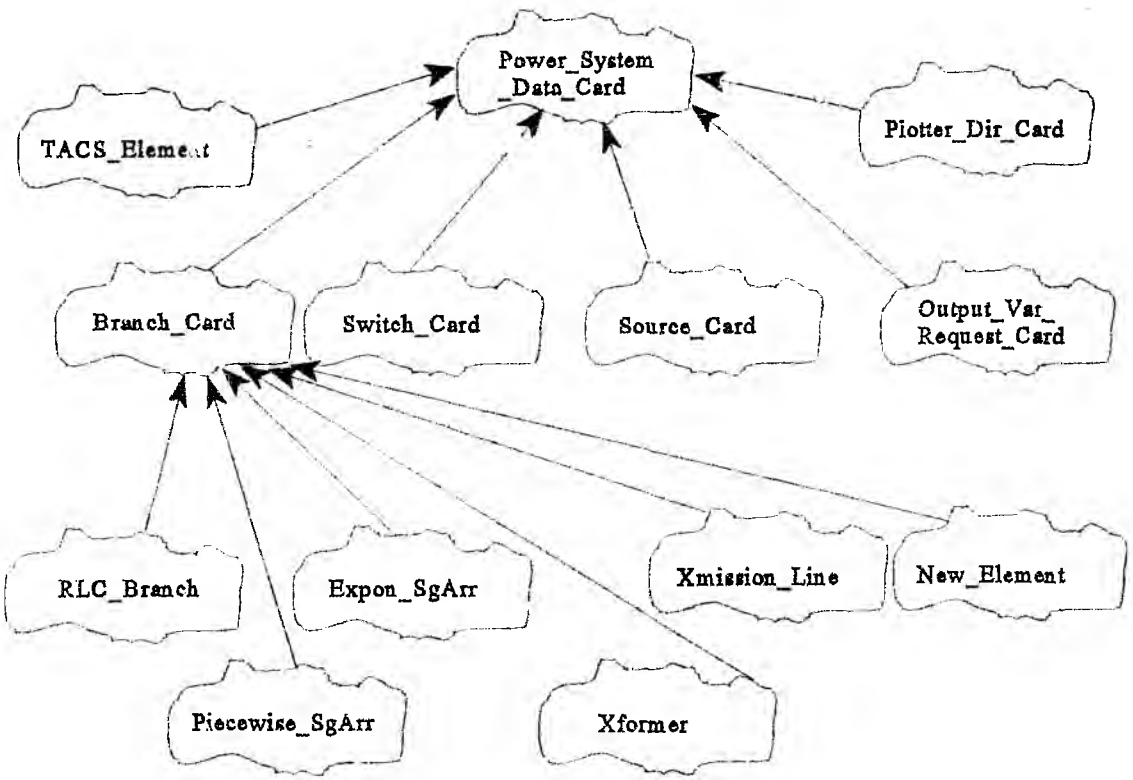


Figure 14 - The Power_System_Data_Card Booch Class Diagram

member functions are required. In the same way, anyone can add any type of ATP element, such as source and switch, to the interface without understanding the rest of the program details, using inheritance. This is shown in more detail in the next section.

This use of inheritance is one of the properties of object-oriented programming and design which makes OOP programs extremely maintainable. (For a more detailed description of this unit, see Appendix D - The Object-Oriented Design of the Data Card Unit).

9.3 Object-oriented Design in ATP for Windows

Using the structure described above, a good example of the advantages of this design format can be seen in the *Power_System_Data_Card* class, and its inherited classes (see *figure 14*).

As can be seen in *figure 14*, the classes which inherit the properties and methods of the *Power_System_Data_Card* class include the *Branch_Card* class, the *TACS_Element* class, the *Switch_Card* class, the *Source_Card* class, the *Output_Variable_Request_Card*, and the *Plotter_Directive_Card* class. The descriptions of these classes are as follows (see Appendix D - The Object-Oriented Data Card Unit):

Class Name	Description
Branch_Card	This class inherits the methods and data members of the Power_System_Data_Card class. It contains functionality to write all details which are specific to ATP Branch cards and are not available in the Power_System_Data_Card class (see List of Methods).

Class Name	Description
Switch_Card	This class contains functionality to write all details which are specific to ATP Switch cards and are not available in the Power_System_Data_Card class (see List of Methods).
Source_Card	This class contains functionality to write all details which are specific to ATP Source cards and are not available in the Power_System_Data_Card class (see List of Methods).
Output_Var_Request_Card	This class contains functionality to write all details which are specific to ATP Output Variable Request cards and are not available in the Power_System_Data_Card class (see List of Methods).

For the purpose of this example, it is only necessary to describe the Power_System_Data_Card class, and the Branch_Card class in detail (for more details see Appendix D - The Object-Oriented Data Card Unit).

9.3.1 The Power_System_Data_Card class

The *Power_System_Data_Card* class contains functionality which must be general to all the different classes which inherit this class. This means that the functionality included in this class is required by all of the inherited classes, but it need not be re-written in these classes. The methods used in the *Power_System_Data_Card* class are as follows:

Method Name	Description
void Power_System_Data_Card()	Constructor
~Power_System_Data_Card()	Destructor
void WriteLeftNodeName(char *Node1)	This method is used to write the left / top node names of all circuit elements to the correct position in the data card.
void WriteRightNodeName(char *Node2)	This method is used to write the right / bottom node names of all circuit elements to the correct position in the data card
void WriteNode3Name(char *Node3)	This method is used to write the left / top node names of an element which shares all of the properties of the present element, if such an element exists, to the correct position in the data card.
void WriteNode4Name(char *Node4)	This method is used to write the right / bottom node names of an element which shares all of the properties of the present element, if such an element exists, to the correct position in the data card.
int IsSimilarElementUsed()	This method can be called from any [illegible] class to check whether or not the present [illegible]ing written uses the properties of a similar element.

All of the above methods are also required by the *Branch_Card*, but this class has access to these via the inheritance relationship. This means that although every branch card has a field for the left node name, a field for the right node name, and a field for Node 3, and Node 4, these fields can be entered by calling the existing *Power_System_Data_Card* functionality although it is not actually present in the *Branch_Card* class.

9.3.2 The Branch_Card Class

The *Branch_Card* class involves functionality to write two of the fields available in ATP. These are the *ITYPE* field, which is used by ATP to describe each element using a two letter code, and the *P* variable, which determines what outputs are required for this particular branch. Since these two fields are not relevant to the other classes which inherit the *Power_System_Data_Card* class, it is necessary to write these methods in the *Branch_Card* class only (although all of the classes which inherit the *Branch_Card* class again have access to these methods - see Appendix D - The Object-Oriented Data Card Unit).

Method Name	Description
void Branch_Card()	Constructor
~Branch_Card()	Destructor
void DeleteElement()	This method is called to delete a branch card and all of it's properties.
void WriteItype(int Itype)	This method is used to write the Itype variable which every branch card needs to identify it.
void WriteP(int P)	This method is used to write the variable P (Column 80) to the data card. The variable P is a flag for specifying what outputs are required from this branch. See User Manual for what each number stands for.

From the above descriptions, it is clear that the decision to use the actual card types as objects in the object-oriented design of this unit was a logical decision since card types are hierarchically broken up into different types. The comparison clearly shows the advantages of using object-oriented programming for this unit.

9.4 Chapter Summary

In order to analyze the use of object-oriented design in the implementation of ATP for Windows, an object-oriented unit which converts circuit information from MS Access databases to the required ATP format was designed. ATP data cards have a similar structure to each other, whilst different card groups have similarities within their group. This structure allows for the definition of data cards within ATP as classes or objects in ATP for Windows. This has simplified the design of the Data Card unit somewhat, and allowed for a logical selection of classes in the object-oriented design of this unit. This unit was not used due to the relative ease of use of Visual Basic for implementing graphical user interfaces when compared to C++ and other object-oriented languages. The module which was used for the implementation of ATP for Windows was designed using object-oriented principles, but was implemented in Visual Basic which is an object-based programming system.

10 THE OBJECT-BASED DESIGN OF ATP FOR WINDOWS

10.1 Introduction

This section describes the design of the ATP for Windows program as it was implemented. Since Visual Basic is a 'visual' language, each screen in the interface is implemented using the form control in Visual Basic. Different controls such as label controls, text box controls, combo box controls, and command buttons are used in each of the screens as required. The chapter includes descriptions of:

- the structure of the ATP for Windows forms,
- the design of ATP for Windows modules, and
- saving the circuit information in MS Access databases.

10.2 The Structure of ATP for Windows Forms

The structure of ATP for Windows is identical to the structure of standard Visual Basic forms (see section 7.4). The design of ATP for Windows is such that the typical circuit element in ATP for windows is represented by an icon on the screen, and has a corresponding Element Details screen which is shown when the user clicks on the element using the right mouse button. The Element Details Screen for the Transmission Line element can be seen in *figure 15*.

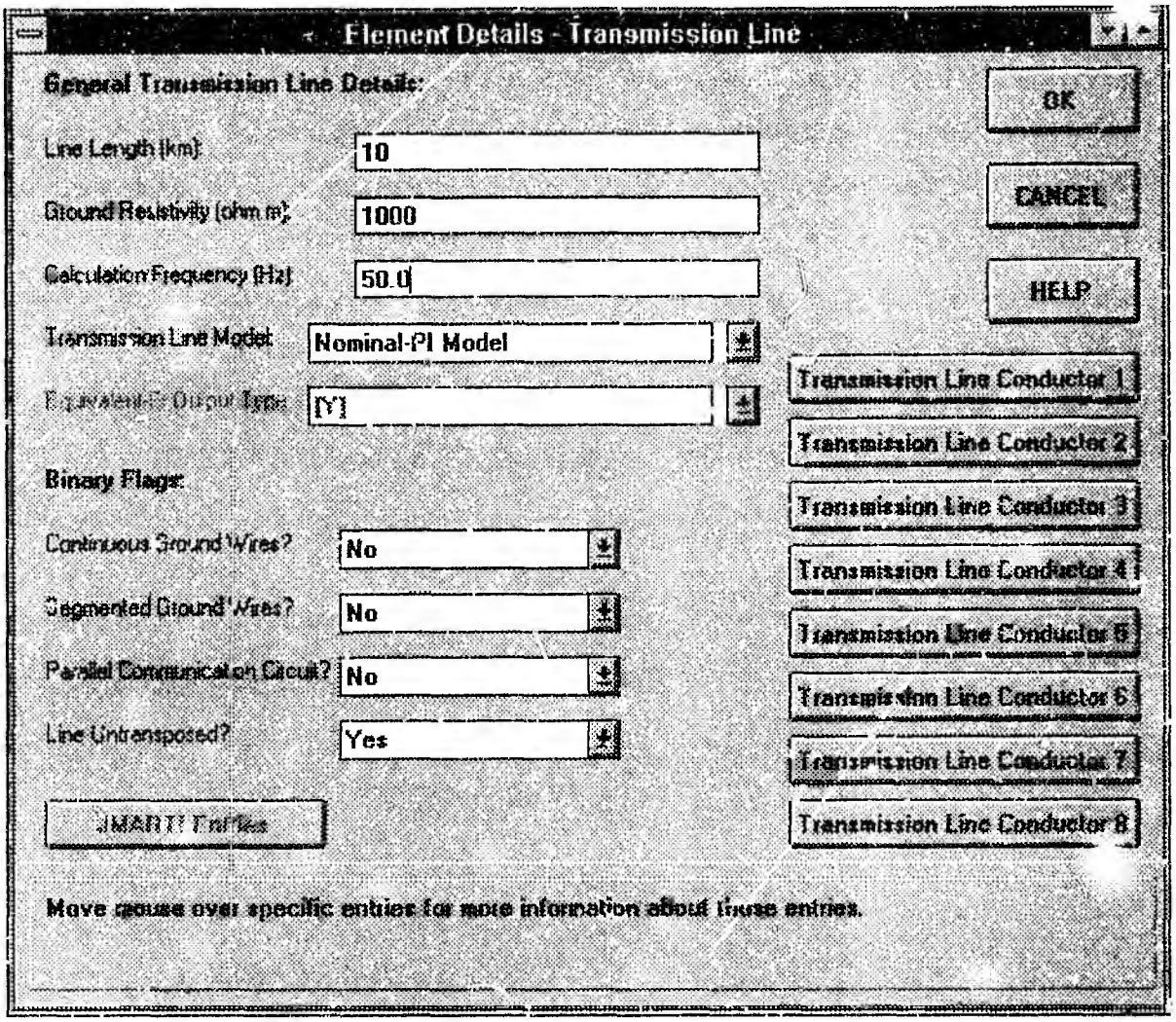


Figure 15 - The Transmission Line Element Details Screen

Each element also has a module (also known as a BAS file) associated with it.

10.3 The Design of ATP for Windows Modules

Visual Basic modules or BAS files are sections of code which are not directly tied to forms or other Visual Basic controls. ATP for Windows modules are generally associated with the Element

Details screen of the corresponding element as mentioned above. The main module in ATP for Windows is the ATP_Module, or ATP.BAS file. This module is used to perform such general tasks as scrolling, redrawing elements on the screen when an existing simulation is opened etc. The relationship between the general Element Details form (or screen) and it's corresponding module file can be seen in the following Booch diagram (*figure 16*), which depicts the relationship between the Transmission Line element and it's corresponding module. Note also that every circuit element is connected directly to the main screen (frmATP), and the main ATP for Windows module (ATP.BAS).

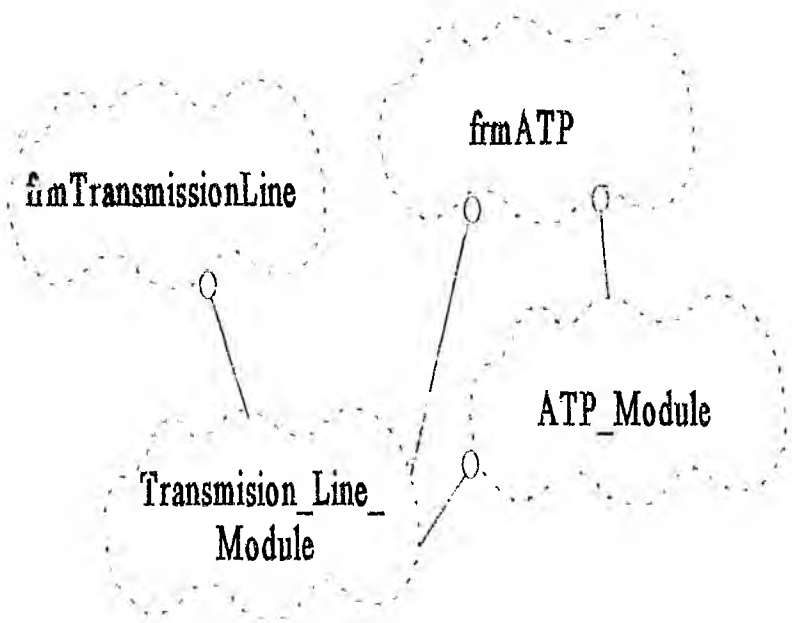


Figure 16 - Booch Diagram Showing Relationship Between Element Details Forms and Corresponding Modules and Forms

As can be seen in the above figure, the Transmission Line's form, frmTransmissionLine, 'uses' the Transmission_Line_Module, and both the frmATP form and the ATP_Module also use this class or module. Each module contains functions (known as Subs or Functions) which are called

from the corresponding Element Details screens or from methods / functions in the ATP_Module to perform certain tasks. Since the functionality for most elements is very similar, looking at just one element will show the design principles which were used for this unit. All circuit information is saved in MS Access type database files.

10.3.1 The Transmission_Line_Module (TRNSLINE.BAS)

The Transmission Line element is a standard transmission line which can be modelled in three ways including the Nominal-Pi model, the Equivalent-Pi model, and the JMARTI modal parameters model. The screen for the Transmission Line element can be seen in *figure 15*. The functionality is such that the user must enter all of the details describing the element in the relevant text entry boxes. To enter the details of the transmission line conductors, the user must click on the command buttons corresponding to the conductor required. This produces the Transmission Conductor Element Details screen, figure 17 below.

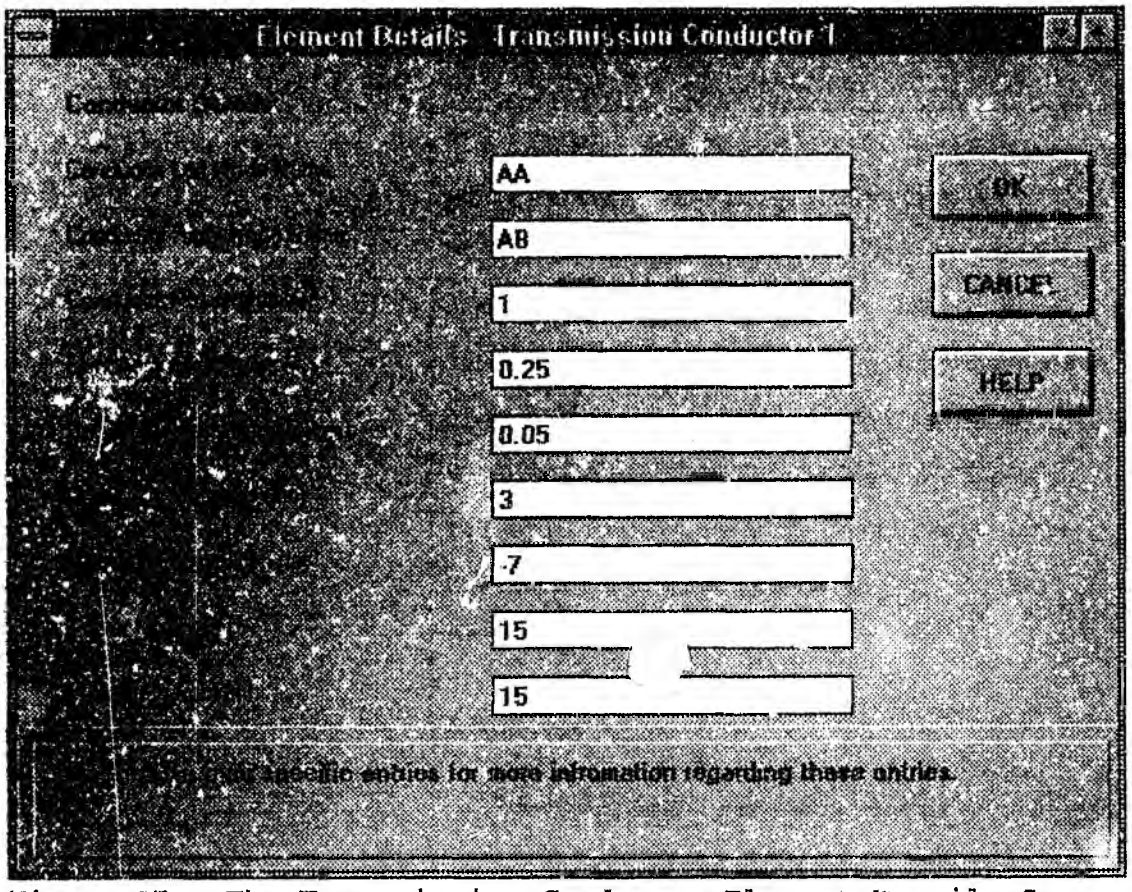


Figure 17 - The Transmission Conductor Element Details Screen

The module which contains the code used for this element is similar in almost all respects to other circuit elements available in ATP for Windows. The methods or functions used in the Transmission_Line_Module are as follows:

Method Name	Description

FormJMartiInitialize( )	This method is used to initialize the Jmarti form whenever this form is called from withm the frmTransmissionLine form (ie. When the modal parameter model of a transmission line is selected, and then the user clicks on the JMARTI command button). If the Jmarti details for this particular transmission line element have been previously entered by the user, these values are placed in the relevant text box or combo box.
	Eg.
	.....
	If NewElement = False
	frmJMarti.txtDiagonalizationFrequency.Text =
	Transmiss onLines(i).DiagonalizationFrequency
	.....
	Else
	.....
	End If

Method Name	Description

FormTransmissionConductorInitialize ( )	This method is used to initialize the frmTransmissionConductor form whenver it is loaded from the frmTransmissionLine form. The method checks what number of conductor is being loaded, changes the caption of the form accordingly, and places the values for the particular conductor in the relevant text boxes and combo boxes if the conductor has been previously defined.
	Eg.
	.....
	If NewElement = False
	frmTransmissionLine.txtLineLength.Text = TransmissionLines(i).LineLength
	.....
	Else
	.....
	End If

Method Name	Description

FormTransmissionLineInitialize ( )	This method is used to initialize the frmTransmissionLine form whenever it is loaded. The method checks whether the element is a new element. If it is a previously defined element, the stored values for this element are placed in the relevant text boxes and combo boxes.
	Eg.
	.....
	If NewElement = False
	frmTransmissionConductor.txtLeftNodeName.Text = TransmissionLines(i).LeftNodeName
	.....
	Else
	.....
	End If

Method Name	Description
JmartiOK ( )	This method is called when the user has clicked on the OK button in the frmJmarti form to indicate that he/she is finished entering the Jmarti details. The method then stores the information in memory by allocating the text in each of the screen text boxes and the screen combo boxes to the corresponding variable in the TransmissionLine(100) array (see following section describing the Transmission Line data members). Eg. TransmissionLine(i).DiagonalizationFrequency = frmJMarti.txtDiagonalizationFrequency.Text .....
TransmissionConductorOK ( )	This method is called when the user has clicked on the OK button in the frmTransmissionConductor form to indicate that he/she is finished entering the conductor details for that particular conductor. The method then stores this information in memory by allocating the text in each of the screen text boxes and combo boxes to the corresponding variable as described above.
TransmissionLineOK ( )	This method is called when the user has clicked on the OK button in the frmTransmissionLine form to indicate that he/she is finished entering all of the transmission line details for that particular transmission line element. The method then stores this information in memory as described above.

Method Name	Description
TransmissionLineOpenCircuit ( )	This method is used to retrieve information of all the transmission line elements in a particular simulation, from disk (if this information has been previously saved to disk). The method opens the table corresponding to the transmission line element in the Microsoft Access database in which the circuit has been stored using the TransmissionLineSaveCircuit method (see next method). Eg. Table = Circuit.OpenTable ("Transmission Line") TransmissionLines(i).DiagonalizationFrequency = CctTable![DiagonalizationFrequency] ..... Table.MoveNext
TransmissionLineSaveCircuit (Position As String, i As Integer)	This method is used to save the transmission line details to disk. This is done by opening the table corresponding to the transmission line elements in the Microsoft Access database which is currently open (opened in the File Save command), and then saving the details of each transmission line element to this table (including position in the circuit). Eg. Table = Circuit.OpenTable("Transmission Line") CctTable![Diagonalization Frequency] = TransmissionLines(i).DiagonalizationFrequency ..... Table.MoveNext

Method Name	Description
TransmissionLineWriteCircuit (Position As String, i As Integer)	This method is used to write the transmission line details to the main ATP data case (ie. The actual data file). This is done by opening the data file in append mode (ie. A mode which allows you to add details to the file), and then writing the stored details of each transmission line to this data file. Eg. Open FileName For Append As fhandle Print #fhandle, "00"; Spc(6 - LeftNodeNameLength); TransmissionLines(i).LeftNodeName(j); Spc(6)..... .....
TransmissionLineWriteLineConstants (Position As String, i As Integer)	This method is used to write the Line Constants support program details to the Line Constants data file. This is done by opening a new Line Constants data file, and then writing the required details to this file. (LINE CONSTANTS is an add on / support program which uses the ATP data file format, and is used to calculate the current transformation matrices required by the main ATP data file). Eg. Open FileName For Output As fhandle Print #fhandle, "00"; Spc(6 - LeftNodeNameLength); TransmissionLines(i) LeftNodeName(j); Spc(6)..... .....

Method Name	Description
TransmissionLineWriteNodeNames (Number As Integer)	This method is used by the ATP.BAS (main module) to indicate in the Setup Simulation screen, those (transmission line) nodes for which the user has requested some form of output such as branch voltage or branch current. Eg. frmSetupSimulation.lstAvailableNodes.AddItem (TransmissionLines(Number).ConductorPightNodeName(0))

Many of the methods which are used in this module are very similar to the equivalent methods in other elements. Although not every element has a support program associated with it, the TransmissionLineWriteLineConstants ( ) method is similar to the methods used in elements which do have support programs associated with them such as the surge arrestor (ZnOFitter support program).

The Transmission_Line_Module data members or variables are as follows:

Member / Const Name	Description
Type TransmissionLineType ElementName As String LineLength As String GroundResistivity As String CalculationFrequency As String TransmissionLineModel As Integer ContinuousGroundWires As Integer SegmentedGroundWires As Integer ParalleiCommsCct As Integer LineUntransposed As Integer DiagonalizationFrequency As String SteadyStateFrequency As String LowestFrequency As String NumberOfDecades As String PointsPerDecade As String ConductorLeftNodeName(7) As String ConductorRightNodeName(7) As String ConductorPhaseNumber(7) As String ConductorSkinRatio(7) As String ConductorResistance(7) As String ConductorDiameter(7) As String HorizontalSeparation(7) As String TowerHeight(7) As String MidspanHeight(7) As String EquivalentPiType As Integer LineConstantsName As String IncludeFileName As String End Type	This type defines the stored variables of the Transmission Line Element (see TransmissionLines(100)). The LeftNodeName and RightNodeName variables are similar to those availabled in each element. The rest of the entries are specific to the transmission line element. These entries are placed in the TransmissionLineType because each transmission line element has these characteristics, and each TransmissionLine(100) element is then capable of storing these characteristics. Eg. TransmissionLines(i).LeftNodeName = frmTransmissionConductor.txtLeftNodeName.Text
TransmissionLineNumber As Integer	This variable is used to store the number of transmission line elements which have been created in the present simulation. It is then used in the procedure which determines whether or not an element is a new element.

Member Name	Description
TransmissionLines(100) As TransmissionLineType	This array is the defining array of the transmission line type (see above). It is an array of transmission line elements. Each element of this array includes values for many or all of the variables listed in the Type definition for the TransmissionLineType shown above.
NewElement As Integer	This variable is an integer variable used by the FormTransmissionLineInitialize method to distinguish between an existing element, and one which has previously been created. It is declared (locally) in every ATP element, and used by that element's initialize method.
NewJMarti As Integer	This variable is used by the FormJMartiInitialize method to determine whether the Jmarti details have been previously defined or not.
NewConductor(7) As Integer	This variable array is used by the FormTransmissionConductorInitialize method to determine whether each of the 8 conductors' conductor details have been previously defined or not.

Other modules in ATP for Windows have similar methods to perform the required tasks (for a more details description of the entire design, see Appendix C - The Technical Reference Manual).

As can be seen in the above description, although Visual Basic is not an object-oriented language, the design should still be easily maintainable. This is true because of the following:

- Structured programming and design involves breaking a program up into functions, which are

separate from the application's data (Orbach 1995). In the case of ATP for Windows, the design of the available modules is object-based, and the module functions are only called once the object or circuit element corresponding to the relevant module, has been created. which is also true for object-oriented programming.

- The variables or data members used for a module are all kept defined within the same particular module, and no two circuit element modules will pass each other information (global variables are kept to a minimum although some are necessary, but only get used in the ATP.BAS module, and a maximum of one circuit element module each). This is an advantage because it overcomes the main problem with structured programming, which is that in many cases, pieces of data flow unchanged or unwanted through the majority of the modules they pass through (Jamsa 1984).
- The only passing of information occurs between the main ATP.BAS module, and each circuit element module.
- Most of the circuit elements have extremely similar modules and the methods within these modules are similar in each element to other element's methods.

This becomes clearer when looking at the steps required to add an element to ATP for Windows. Since different elements within ATP for Windows code never call functions other than those within their own modules, if a new element is added, the programmer will have to do the following:

- Create a new module for the new element.
- Create a new form for the element.
- Write the relevant code in the new element's module, and
- write the calls for the new functionality in the ATP.BAS module.

No other coding will be necessary. (For a more detailed description of adding an element to ATP for Windows, see Appendix G - Adding an Element to ATP For Windows to Assess the Maintainability of the Program). This means that the programmer will only have to understand the functionality involved in creating the new element, and not the rest of the program. The above

mentioned property, is one of the reasons that object-oriented programs are easily maintainable, and using this aspect of object-oriented programming in the Visual Basic design of ATP for Windows, ensures that the program is almost as maintainable as the equivalent, entirely object-oriented program would have been.

The entire design of ATP for Windows can be seen in the Technical Reference Manual for ATP for Windows (See Appendix C - The ATP for Windows Technical Reference Manual).

10.4 Using MS Access Databases to Store Circuit Information

Saving the circuit information to disk from the Visual Basic front end to ATP for Windows, is performed by writing the information to an MS Access database which contains one table for each type of ATP for Windows element available. Each table has the fields relevant to its corresponding circuit element details. The use of a database to store the circuit information has many advantages including the following:

- The circuit information can easily be accessed by the Data Card unit (see chapter 10 - the object-oriented design of the Data Card unit)
- only one function in the Data Card unit must be called from the Interface unit. This function simply passes the Data Card unit the file name of the database in which the circuit information is stored (whereas using records would have meant that one function is required per variable which needs to be written to the ATP data file).
- Importing files into ATP for Windows from other CAD packages involved writing code in the actual CAD packages which stores circuit information in MS Access tables (which is a very common format).
- The circuit information can be browsed in a convenient database format.
- Small changes to the circuit can be made by simply opening the database or by running ATP for Windows.
- Existing ATP files can easily be converted into the MS Access database format.

- Circuits can be created entirely in MS Access, and then opened in ATP for Windows, or simply passed to the Data Card unit which would then run the simulation normally.

10.5 Chapter Summary

The design of ATP for Windows is based around each circuit element as a separate object having a corresponding Element Details screen, as well as a corresponding module or BAS file. The functionality of the Element Details forms is similar to that of the standard Visual Basic form, while the functionality of the modules involves functions to initialize the forms involved, to save the details of the element to memory, and to open the details of the element from the MS Access database file involved. The functions are written so that all function calls to circuit element modules are made either from within the specific modules themselves, or from the ATP.BAS module. This ensures that the modules are easy to maintain (although an entirely object-oriented unit would still be easier to maintain than one which is not entirely object-oriented such as this unit). The circuit information is stored in MS Access databases, which is a convenient format that allows for easy manipulation of the information.

11 THE IMPLEMENTATION OF THE INTERFACE

11.1 Introduction

As was previously mentioned, ATP for Windows was written using Microsoft Visual Basic 3.0. In order to develop a successful GUI, it was decided that recent Software Engineering quality principles be used for the project. The following chapter describes the implementation of ATP for Windows including:

- the use of software quality principles for the design of ATP for Windows
- the User Requirements Specification,
- the User Reference Manual,
- the Technical Reference Manual,
- the Product Test Specifications and Plan,
- the use of Visual Basic as opposed to C++ for the GUI, and
- the final product acheived.

11 2 Using Software Quality Principles

Recent developments in the Software Engineering field have shown that it is very important to design a large software program very thoroughly before the actual coding or implementation of the program begins (Orbach 1995). Most importantly, it is crucial to complete the documentation which was traditionally written only after the coding, before the coding stage begins. Using the above mentioned methodology, the implementation of a software product should involve the following stages:

- Documenting the User Requirements Specification, the User Reference Manual, the Technical Reference Manual, and the Product Test Specification and Plan.

- Coding of the GUI.
- Testing of the GUI.
- Final revision of documentation.

To ensure that ATP for Windows is as maintainable as possible, and that the implementation process is as quick as possible, the above mentioned stages were completed in that order.

11.3 The ATP for Windows User Requirements Specifications

The User Requirements Specification for ATP for Windows was completed in June 1994 (See Appendix A - The ATP for Windows User Requirements Specification). While the system specifications were being established, careful attention was given to what was required from the system, and a very basic design as well as a selection of techniques and tools to be considered were documented. It was found that it was difficult to define the specifications clearly before the actual program was designed for the User Reference Manual. The first detailed design of ATP for Windows was described in the User Reference Manual, and this played a large role in the definition of the final product. This product definition is usually associated primarily with the User Requirements Specification.

11.4 The ATP for Windows User Reference Manual

The User Reference Manual for ATP for Windows was completed in November 1994 (see Appendix B - The ATP for Windows User Reference Manual). It is a highly detailed document which involved detailed design of every aspect of the interface and included research into modern user interface trends, and human-computer interaction. The User Reference Manual was updated twice before the actual coding began, proving that there was indeed a time saving in writing the

User Manual before beginning the coding (since updating the code twice would have been far more time consuming).

11.5 The ATP for Windows Technical Reference Manual

The ATP for Windows Technical Reference Manual was completed in January 1995 (see Appendix C - The ATP for Windows Technical Reference Manual). It involves a detailed description of the well defined logical and physical design of ATP for Windows, as required. The design is shown down to as low a level as required. The design involved was described in chapter 10 of this thesis.

11.6 The ATP for Windows Product Test Specifications and Plan

Although it is customary in many cases to produce a Product Test Specification and Plan, this is only usually done if the product is being developed by a team of software developers. Since in this case ATP for Windows was developed only by the author of this thesis, it was decided in the case of ATP for Windows, to release the first version as a 'beta' version instead of writing the Product Test Specifications and Plan. This means that the users of the first version are aware that extensive testing has not been performed, and are willing to report faults that they find in exchange for getting the program free of charge. The reason for this choice of testing procedure is that the size of ATP for Windows, implies that extensive testing, and the documentation of the required tests would take up more time than the value of such testing. Since testing of some form is essential, the option of 'beta' testing was selected. Rudimentary testing will however be performed before the product is shipped to ensure that there are not an unacceptable number of errors present in the initial release of ATP for Windows.

11.7 Using Visual Basic as Opposed to C++ to Implement ATP for Windows

Both Visual Basic and C++ were considered as potential tools for the implementation of ATP for Windows. It was shown that the object-oriented design for the Data Card unit of ATP for Windows (see chapter 9) would be easily maintainable. However, as was explained in section 7.6, projects developed using Visual Basic have been found to take up to thirty times less than the equivalent C++ programs due to the ease of use of Visual Basic (Duntemann 1995). Because of this, it was decided that the GUI be developed using Visual Basic instead of C++. The definition of maintainability of a program must include the ease of use of the tool with which the program was implemented since this also has a large influence on the time taken to maintain a program. Because of this, it is felt that the final program is more maintainable than the equivalent GUI written in C++.

11.8 ATP for Windows: The Final Product

The final product is a graphical user interface on the MS Windows platform, which is as flexible as possible, easily maintainable, and adheres to the principles of the IBM CUA, and Microsoft Application Design Guide, making it very usable. Although not every element in ATP is available for simulation in ATP for Windows at this stage, due to the nature of the design, further elements can easily be added at a later stage, and this is in the pipeline for future work. The general format of ATP for Windows can be seen in *figure 18* (whereas the general format of ATPDRAW, the previous GUI for ATP can be seen in *figure 19* ahead).

As can be seen in the *figure 18*, ATP for Windows is mouse driven. To select an element, the user must click on the element from the Circuit Element Panel on the left hand side of the screen, and drag the element onto the circuit region. To describe the element, the user must click on the right

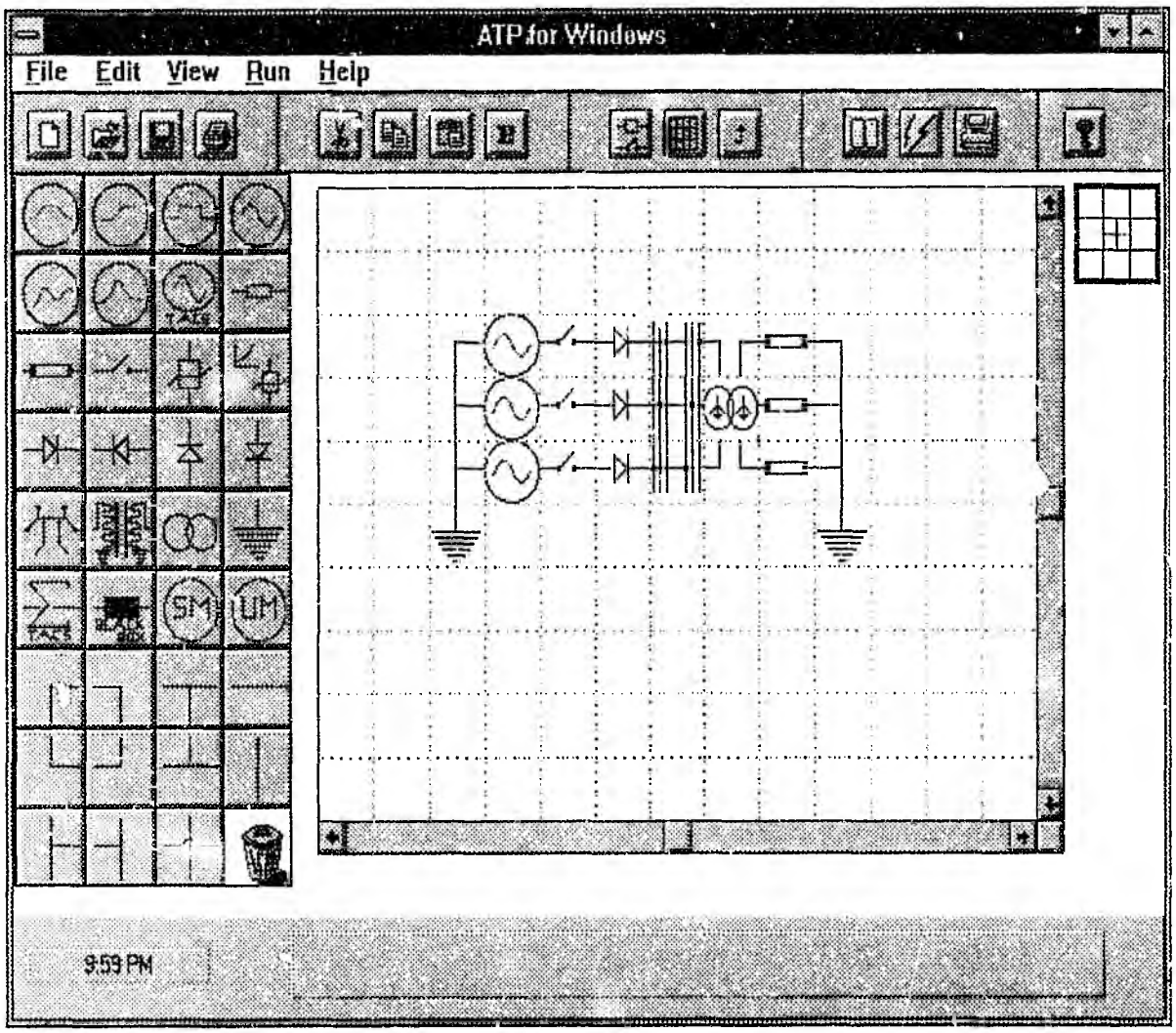


Figure 18 - The ATP For Windows Main Screen

mouse button while the mouse pointer is pointing to the required element. This will produce an Element Details screen which is specific to that particular element. ATP for Windows is as flexible as possible, with the user being capable of selecting an element either using the Circuit

Element Panel, or the menu, as well as being able to change the circuit icons appearing on the Circuit Element Panel and having other such options which allow for flexibility (see Appendix B - the User Reference Manual). One feature which also makes ATP for Windows easy to use is the extensive use of informative feedback. Each ATP for Windows screen has a message bar which gives the user information regarding the particular situation involved.

11.9 Chapter Summary

The implementation of ATP for Windows involved writing the User Requirements Specification, the User Reference Manual, and the Technical Reference Manual, and then coding the program. These stages were followed to ensure compliance with recent Software Engineering Quality principles. The implementation was successful, and the final product was assessed using the criteria of usability and maintainability which were described in previous chapters in this thesis.

12 ASSESSMENT OF THE IMPLEMENTATION

12.1 Introduction

ATP for Windows is a user-friendly GUI for ATP, which is a pre-processor to ATP, and converts graphical and numerical information received from the user, to the data cards / data files required to run ATP. As was described in previous chapters, ATP for Windows is a menu driven GUI which runs in a mouse driven environment with the goal of the design being to make ATP for Windows as usable, flexible, and maintainable as possible. The following chapter describes the steps taken to assess the success of the implementation of ATP for Windows including:

- assessing the usability achieved using the IBM CUA guideline,
- assessing the maintainability achieved, and
- comparing ATP for Windows to ATPDRAW.

12.2 Assessing the Usability Achieved

As can be seen from the implementation (see Appendix B - The User Reference Manual), the goal of usability for the interface has been achieved to a very large degree. In order to properly asses the usability of ATP for Windows, it is necessary to analyse the usability standards defined in section 2.3:

- Visual *clarity* - ATP for Windows has the standard CUA layout which is used in Windows applications.
- Consistency - The consistency of each screen in ATP for Windows with other screens in ATP for Windows was ensured by designing the interface carefully, and completing the design on paper before any of the coding was performed, as well as by conforming to the IBM CUA and the Microsoft Application Design Guide for each and every screen.

Analyzing the different screens in ATP for Windows (such as the different Element Details screens), it is clear that there is consistency amongst these screens (see Appendix B - The ATP for Windows User Reference Manual).

- Informative feedback - Informative feedback is ensured by having a message bar which informs the user what element is presently being pointed to, and often informs the user of the steps the user is required to take. Each Element Details screen has its own message bar which gives the user information regarding the entries in that screen. Also, the user is often asked for confirmation before making any changes which might result in loss of information. Context sensitive on-line help is available to the user at any stage of running the program, while general ATP for Windows help is also available at any time.
- Explicitness - ATP for Windows has many similarities in structure to Windows CAD packages, and is extremely similar to other Windows packages in menu layout and structure, as well as general operation.
- Appropriate functionality - To ensure appropriate functionality, the User Reference Manual was thoroughly reviewed by a number of potential users before the actual coding began, and several changes were made to the system due to this review.
- Flexibility and control - To ensure flexibility of the system, the user is given the option to change the layout of the system settings including changing the elements which appear in the Circuit Element Panel, removing or adding the grid, selecting an element using the menus etc (see Appendix B - The ATP for Windows User Reference Manual).
- Error prevention and correction - ATP for Windows includes error checking and trapping in every screen and has error handling for every error which is not related to ATP itself (as opposed to ATP for Windows).
- User guidance and support - To ensure proper user guidance and support, easily accessible, and informative on-line help is available for the user at any point or situation in the system. The help available is context sensitive if the user presses the 'F1' key at the point where help is required, and general help if the user selects the Help menu or the Help toolbar button.

As can be seen in the above analysis, ATP for Windows satisfies all of the defined criteria for usability. It can therefore be concluded that the implementation of ATP for Windows has been successful with respect to usability and flexibility.

12.3 Assessing the Maintainability Achieved

When considering the maintainability of ATP for Windows, it is important to note that the immediate disadvantage of ATP for Windows is the fact that Visual Basic is an object-based language, and not entirely object-oriented. However, Visual Basic is a very easy language to learn and to use, and maintaining the program has been found to be very easy

Adding functionality to the interface is a simple task which involves the following step:

- overriding existing events and properties in the Visual Basic module.

Adding actual elements to the interface involves the following steps:

- Designing new forms in the Visual Basic module. These forms are not connected in any way to the rest of the program so that the maintainer need not understand the rest of the program.
- Creating a new module for the element.
- Writing a module for this element which includes the appropriate functionality.
- Adding calls for the element to the ATP.BAS module.

An example of adding an element to ATP for Windows is shown in Appendix G - Adding An Element to ATP for Windows. This is a step by step description of what was required to add the double exponential function (which is now available in ATP for Windows) to the program.

12.4 Comparing ATP for Windows to ATPDRAW

In comparing ATP for Windows to ATPDRAW, it is essential to compare the factors in software which are considered crucial to the success of the software, ie. usability and maintainability, since the success of ATP for Windows in the actual 'market place' will not be ascertained until it is distributed extensively. It has been shown in section 4.3 of chapter 4, that ATPDRAW is not very usable (according to the definitions of usability), and also not easily maintainable due to the use of structured Pascal 5.0 for the programming of ATPDRAW (see section 4.2). It is easy to see that ATP for Windows is a more user-friendly program than ATPDRAW when looking at the main screens of the two programs: The main screen of ATP for Windows can be seen in *figure 18* in section 11.8. The main screen for ATPDRAW can be seen in *figure 19* below.

As can be seen from the comparison of the two screens, ATF for Windows makes more use of graphics as opposed to text, has a clearer screen, and offers on-line Help among other features which are not available in ATPDRAW. For a further comparison, see Appendix F - A Case Study using ATP, ATPDRAW and ATP for Windows.

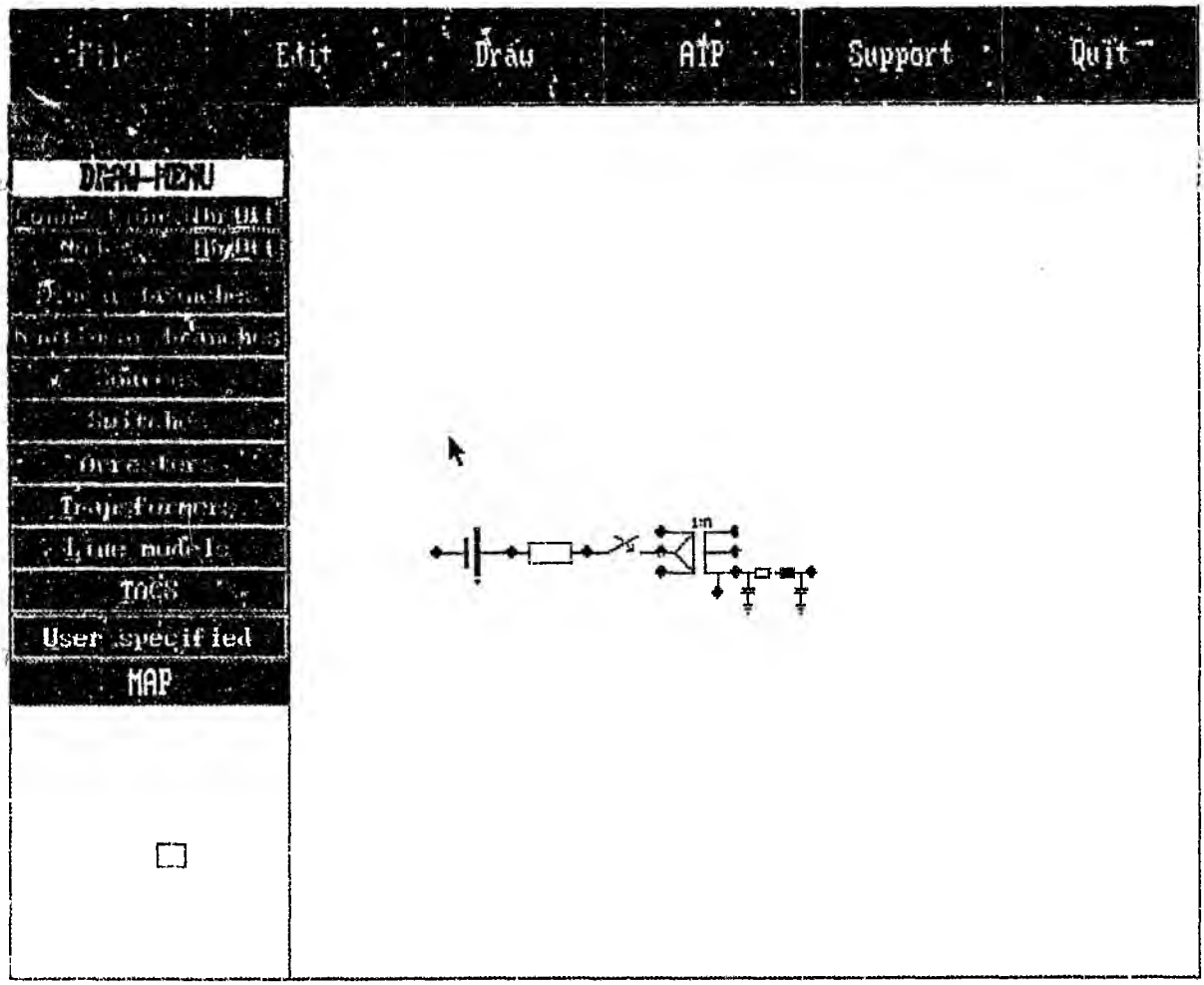


Figure 19 - The Main Screen of ATPDRAW

12.5 Chapter Summary

ATP for Windows is as required, a usable and maintainable product which adheres to the IBM CUA, and maintainability standards. Comparing ATP for Windows to ATPDRAW, it is clear that ATP for Windows is a huge improvement over ATPDRAW and ATP itself in the fields of maintainability and usability. Although ATP for Windows could potentially be improved in the maintainability field if there was a tool which could combine the object-oriented nature of C++, and the ease of use of Visual Basic, ATP for Windows is considered to be a successful product which presents strategies that can be used on many other High Voltage and Power Systems software applications to improve their interfaces as well as to improve the programs in general

13 POTENTIAL IMPROVEMENTS TO ATP FOR WINDOWS AND ATP

13.1 Introduction

Although ATP for Windows is very usable, and easy to maintain, since the development of ATP for Windows began, a new product has been launched on the market, which could markedly improve the maintainability of ATP for Windows. This product is known as the Delphi programming system. The following chapter details the improvements which could potentially be made to ATP for Windows if Delphi was used to code the program as well as other potential improvements which could be made in future work on ATP. The chapter focuses on:

- making ATP for Windows more maintainable,
- using Delphi to implement ATP for Windows, and
- other potential improvements which could be made to ATP.

13.2 Making the Interface More Maintainable

As was previously mentioned, the only problem with the maintainability of ATP for Windows is that ATP for Windows was written using Visual Basic, which is not an entirely object-oriented language. Although Visual Basic is easy to learn, this is still a disadvantage which could be overcome given the right tool.

13.3 Using Delphi to implement the GUI

In March 1995, Borland launched a new Pascal based compiler and programming system called

Delphi. This system combines the object-oriented nature of Object Pascal (and C++), and the 'visual' programming, event driven nature of Visual Basic. Since Borland Delphi combines the useful features of both C++ and Visual Basic, it would have been advantageous to implement ATP for Windows using only Delphi, and the existing object oriented design of the Data Card unit could be used for that part of the interface. The only disadvantage of Delphi is that it is entirely platform sensitive (it can only run on the MS Windows platform). This means that if ATP for Windows was written for a Unix machine or the Apple Macintosh at a later stage, the code would have to be rewritten from scratch. This would not be the case if the object oriented Data Card unit designed for the purpose of this project were implemented as it stands, using C++ code, but it would be the case for ATP for Windows as it stands, since Visual Basic is also entirely platform sensitive (to the MS Windows platform, although it can run on Windows NT on many machines including the DEC Alpha). Since the modern versions of ATP only work on the DOS platform, the possibility of rewriting ATP for Windows for a Unix platform is low unless ATP is rewritten for the Unix platform, or ATP for Windows is run on a Unix platform while the file which ATP for Windows produces is transferred to the DOS platform, and run separately.

13.4 Other Potential Improvements to ATP

One of the most useful potential improvements to ATP itself, is to convert the code of ATP to C++ code. Since C++ is not platform sensitive, this would mean that ATP could be run from a DOS based machine, a Unix based machine, or an Apple Macintosh based machine without requiring much of the code to be changed in each transfer. Although this is true for ATP itself, this is not true for ATP for Windows since most Unix machines have their own graphical operating system similar to MS Windows, which is not compatible with the code written for the MS Windows platform. In the case of ATP for Windows, it can be said that one advantage of having designed the Data Card module using object-oriented design, is that if this module was implemented, it could be ported across platforms if necessary, and only the actual Interface

module would have to be rewritten on the new platform. Since most visual or graphical platforms are very similar in structure, the specifications used to write ATP for Windows could be used to write the new interface unit, with the User Reference Manual as the guideline to this interface.

13.5 Chapter Summary

One disadvantage of the implementation of ATP for Windows is that the program was written using Visual Basic, which is not entirely object-oriented. A potential solution for this problem has recently been introduced with the launch of Borland's Delphi programming system. Delphi has the object-oriented advantages of C++ as well as the ease of use and event driven nature of Visual Basic. This means that ATP for Windows could potentially be implemented using Delphi alone, and utilizing the object-oriented design of the Data Card unit. A potential improvement of ATP itself would be to rewrite ATP using C++ This would allow for the use of ATP on a number of platforms including DOS, Unix, and the Apple Macintosh. Since Unix machines have faster processors than their DOS counterparts, this could speed up the running time of ATP simulations significantly. ATP for Windows would have to be rewritten for the new platform if ATP was ported to a platform other than DOS, but it could be rewritten using the same specifications as set out in the User Requirements Specification and the User Reference Manual of ATP for Windows, and the object-oriented Data Card unit which was designed could remain almost unchanged.

14 IMPLICATIONS FOR FUTURE WORK

14.1 Introduction

The main goal of writing a user interface for ATP, was to define strategies for designing useful interfaces for software in high voltage applications. This includes High Voltage applications other than ATP, such as ERACS. The following chapter describes the strategies which were successfully used to develop ATP for Windows, and can also be used to improve other High Voltage and Power Systems applications. The chapter focuses on:

- the general strategies used for ATP for Windows, and
- strategies which could be used on other High Voltage and Power Systems applications.

14.2 General Strategies Used

It is now an accepted fact that software has led to profound and dramatic changes in the power system operations and the high voltage field (Denny 1988). Since most High Voltage software is written by High Voltage engineers as opposed to software engineers, High Voltage software is often neither usable, nor maintainable, and in many cases is hacked together in the minimum amount of time possible. This trend can waste an enormous amount of time. The fact that many High Voltage applications perform the same functions as each other is a sign in itself that users felt a need to rewrite software which already existed because that software was either difficult to use, or difficult to maintain. It is essential therefore that High Voltage engineers study some aspects of software engineering, to avoid these situations.

14.2.1 Ensuring Usability

As discussed in previous chapters, one of the most important factors of modern software, is usability. Graphical user interfaces have been used successfully to ensure usability in modern software, although an interface is not necessarily usable just because it is graphical. Many other usability criteria must be considered when designing modern software, and it is essential that high voltage software follow the usability trend to save users time, money, and frustration. The most important factor when considering usability is to use an interface which is commonly used for other applications such as Microsoft Windows, and adheres to guidelines such as the IBM CUA and the Microsoft Application Design Guide.

14.2.2 Ensuring Maintainability

In the High Voltage field, new developments can often lead to the need for large changes in simulation software such as ATP. This indicates that it is more critical in the High Voltage field than in many others, that software be as maintainable as possible. Since most changes to software also involve changes to the interface, it is crucial that interfaces to High Voltage software are very easily maintainable. The use of object-oriented or object-based design, and the use of software engineering quality principles in high voltage software is essential.

14.3 Strategies Which Can be Used in Other High Voltage Applications

14.3.1 Improving ERACS

As was mentioned in section 2.5.2, although ERACS is a program which is relatively easy to use, users would still require many hours to learn how to use this program. ERACS could be improved

if it was written for a graphical platform such as MS Windows or IBM OS/2. The disadvantage of these operating systems is that they decrease the speed of the actual running of programs due to the massive memory requirements of such graphical systems. However, with powerful computers becoming more and more accessible to the general public, the benefits of a consistent graphical platform, as well as the almost automatic adherence of such software to usability guidelines, the advantages of using such a platform for ERACS and other Power Systems or High Voltage software would out-way the time losses involved.

14.4 Chapter Summary

Since High Voltage software is more often prone to the need for changes than most other software, it is critical that High Voltage engineers adhere to several software engineering principles when designing High Voltage software. These principles include ensuring that the software is usable and adheres to accepted software standards such as the IBM CUA and the Microsoft Application Design Guide, and ensuring that the software is as maintainable as possible to avoid the unnecessary rewriting of software which often occurs because of difficulty in maintaining a software product.

15 CONCLUSION

Software is one of the most important aspects of the High Voltage and Power Systems fields today. However, it has been shown that many High Voltage and Power Systems software applications are designed without regard for modern software strategies. The most important software strategies which are not taken into account are ensuring usability and maintainability of the software.

To determine strategies for developing usable and maintainable interfaces in High Voltage software, ATP (the Alternative Transients Program) was used as an example of High Voltage software which has a poor interface, and a graphical user interface (GUI) for ATP as a pre-processor to the program, was proposed. ATP is a power systems transient analysis package which is used to simulate transient conditions such as lightning and switching conditions in distribution networks and machines. ATP was written in Fortran (in the late '60s), and makes use of a data file interface. This makes it very difficult to use the program, and very time consuming to learn how to use it. Although a user interface has been developed for ATP before (called ATPDRAW), this interface suffers from a number of setbacks including that it does not conform to software usability standards, and it is difficult to maintain.

The factors which make an interface usable and maintainable were studied, and it was found that it is important to adhere to software usability guidelines such as the IBM CUA, and the Microsoft Application Design Guide to ensure usability. It was also found that it is necessary to use object-oriented or object-based software design as well as thorough design and documentation of the software before the coding is begun, to ensure that the software is maintainable. The analysis of modern usability standards led to the proposition that the interface be developed for the Windows platform, which in itself ensures that a number of the factors that make a program usable are taken into account.

Also, software tools were analyzed to determine the best tool to use for the implementation of the GUI for ATP, and it was determined that Visual Basic should be used for the implementation of this GUI, while the MS Access database (within the Visual Basic code) should be used to store circuit information. Visual Basic is an object-based and event driven programming system which is extremely easy to use for interface programming. Although it was found that programs developed using object-oriented design and programming are usually more easily maintainable than structured programs, programs written in Visual Basic were found to be easy to maintain if they involve the visual part of an interface. For comparison purposes, as well as for potential future use, a unit which converts circuit information from the MS Access format to the ATP data file format was designed for an entirely object-oriented system such as C++. This design was found to be easily maintainable in theory.

The GUI for ATP, which is called ATP for Windows, was designed using the techniques which ensure that it is both very usable, and easily maintainable. To satisfy modern software quality trends, ATP for Windows was designed thoroughly before the coding began. This involved documentation including the User Requirements Specification, the User Reference Manual, and the Technical Reference Manual.

ATP for Windows was assessed, and was found to be very easy to use, and easily maintainable. In comparing ATP for Windows to ATPDRAW, it was found that ATP for Windows is much easier to use, and more easily maintainable. The limitations of ATP for Windows, are that it was written using a software tool which is not entirely object-oriented. Also, ATP for Windows is slow in operation unless it is run on a computer with a large bank of RAM (which is a setback that all MS Windows programs suffer from).

The strategies used to design ATP for Windows were proposed as general strategies for improving interfaces of other High Voltage applications. It was recognized that software is one of the most important aspects of High Voltage and power systems calculations. Due to the progress of research in the High Voltage field, changes are often required in High Voltage software applications, making the need for the ease of maintainability in these applications very high. Also, it was noted that making High Voltage applications more usable, will save High Voltage engineers time and money.

16 REFERENCES

1. Booch Grady(1994). "Object-Oriented Analysis and Design with Applications" Second Edition, The Benjamin/Cummings Publishing Company, Redwood City, California, 1994.

2. Cline MP (1995). "Frequently Asked Questions (FAQs) for comp.lang.c++ on the USENET" Paradigm Shift Inc., Norwood New York, March 1995.

3. Cox BJ (1990). "There is a Silver Bullet" *Byte* Magazine, October 1990.

4. de Mello FP, Feltes JW, Laskowski TF and Oppel LJ (1992). "Simulating Fast and Slow Dynamic Effects in Power Systems", IEEE Computer Applications in Power, Volume 5, Number 3, July 1992.

5. Denny FI (1988). "Future Applications in Power" IEEE Computer Applications in Power. Volume 1, No. 1, IEEE, January 1988.

6. Dommel HW (1986). "Reference Manual for EMTP" Vancouver Canada, 1986.

7. Duntemman J (1995). "Visual Development Tools", PC Techniques Magazine, Vol.5, No.6, Feb/March 1995.

8. Dwolatzky B (1994). "Software Engineering Concepts" Course Notes, CEE, University of the Witwatersrand, JHB, 1994.

9. Eason KD (1991). "Ergonomic perspectives on advances in human computer interaction", *Ergonomics*, vol. 34, no. 6, pp. 721-741, 1991.

10. Ege R. and Stary C (1992). "Designing maintainable, reusable interfaces" IEEE Software, pp24-32, Nov 1992.

11. Gerald CF(1973). "Applied Numerical Analysis" California State Polytechnic, Addison Wesley publishing company, Reading Massachusetts, 1973.

12. Gray RF and King IJ (1991). "User Interface Graphically Improves Generator AI Diagnostics" IEEE Computer Applications in Power, Volume 4, Number 3, July 1991.

13. Hoidalen HK (1992). "ATP-Draw - Graphical Preprocessor to ATP" Preliminary Release, University of Trondheim, Division of Electrical Power Engineering, Norway, 1992.

14. International Business Machines (IBM) (1987). "Common User Access Panel Design and Interaction" First Edition, IBM, Florida, 1987.

15. Jamsa K A (1984). "Object-oriented Design vs. Structured Design - A Student's Perspective" ACM Sigsoft Engineering Notes, Vol9, No 1, pp43, Jan 1984.

16. Lacob RI (1993) "Designing the User Interface of an Automated System for Non-destructive Testing (NDT) to Take User Limitations Into Account" University of the Witwatersrand, Johannesburg, 1993.

17. Lee Smith H and Modzelewski T (1989). "Enhancing Energy Management Systems with Advanced RTU Capabilities", IEEE Computer Applications in Power, Volume 2, Number 4, October 1989

18. Leuven EMTP Center(1992). "Alternative Transients Program Rule Book" Leuven, Belgium, June 1992.

19. Lehman (1980). "Programs, lifecycles and laws of software evolution", Proc. of the IEEE, vol. 68, no. 9, , pp1060-1076, September 1980.

20. Microsoft (1993). "Microsoft Visual Basic Programmer's Guide" Microsoft Corporation 1993.

21. Orbach T (1993). "Using Interpolation to Speed up NEC2 simulations" University of the Witwatersrand, Johannesburg, South Africa, December 1993.

22. Orbach T (1995). "Writing an Object-oriented Graphical User Interface for ATP as a Pre-processor to the Program" SAUPEC, January 1995.

23. Orbach T (1995b). "Strategies for Designing Useful Interfaces for Software in High Voltage Applications", International Symposium on High Voltage (ISH) '95, Graz, Austria, August 1995.

24. Powell JE (1995). "Promising Windows Development" Windows Magazine, pp 130, February 1995.

25. Rahimi FA, Balu NJ and Lauby MG (1991). "Assessing Online Transient Stability in Energy Management Systems" IEEE Computer Applications in Power, Volume 4, Number 3, July 1991.

26. Ravden S (1989). "Evaluating Usability of the Human Computer Interface", Ellis Horwood Limited, Chichester, 1989.

27. Salon SJ (1990). "Finite Element Analysis of Electric Machinery", IEEE Computer Applications in Power, Volume 3, No. 2, IEEE, April 1990.

28. Scott Davis A (1990). "Florida Power Corporation's Load Management System" IEEE Computer Applications in Power, Volume 3, Number 2, April 1990.

29. Sherer P (1992). "Microsoft seeks GUI uniformity" PC Week, pp57-59, July 13, 1992.

30. Urlocker Z (1990). "Object-oriented Programming for Windows", Byte, pp287-294, May 1990.

31. Van Coller JM (1993). "An Introduction To EMTP (Electromagnetic Transients Program)" Revision 3.0, Department of Electrical Engineering, University of the Witwatersrand, Johannesburg, South Africa, 1993.

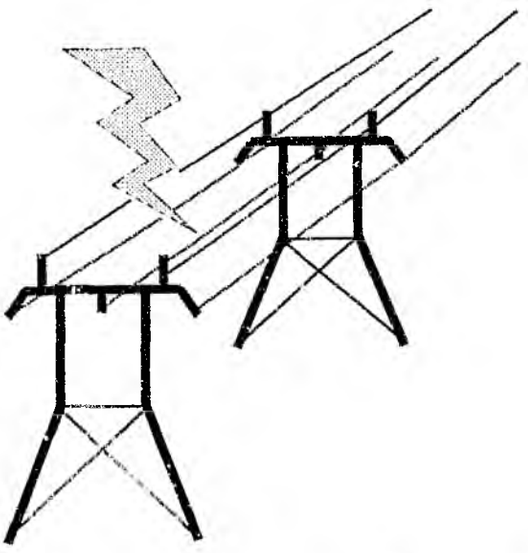
32. Walker AJ (1993). "Computer Engineering Course Notes" Department of Electrical Engineering, University of the Witwatersrand, Johannesburg, South Africa, 1993.

33. Wilde N, Matthews P, and Huitt R (1993). "Maintaining Object-Oriented Software", IEEE Software, Jan 1993.

34. Witte JF, Mendis SR, Bishop MT and Kischefsky JA (1992). "Computer-Aided Recloser Applications for Distribution Systems" IEEE Computer Applications in Power, Volume 5, Number 3, July 1992.

APPENDIX A - THE ATP FOR WINDOWS USER REQUIREMENTS SPECIFICATIONS





User Requirements Specifications

1 Problem statement

1.1 What problem is the product meant to solve?

The problem is a graphical user interface for the High Voltage Analysis package - Alternative Transient Analysis Package. The interface must have the following properties:

i) It must perform on the MS Windows platform.

ii) It must be simple and it must conform to the IBM Common User Access standards.

2 Environment

2.1 Cost

2.1.1 What is the limit of expenditures to be used to create the product? (i.e. Rands and/or engineering hours).

Approximately two man-years has been set aside for the completion of this project.

2.1.2 What is the limit of expenditures to be used to maintain the product? (i.e. Rands and/or engineering hours per unit time of use).

R 0-00 per year for modifications/upgrades.

2.1.3 What other resources are available for the product development? (i.e. facilities/personnel and quantitative or qualitative limits).

2.1.3.1 Project supervisors: Dr. I. Jandrell and Dr. B. Dwolatzky

2.1.3.2 The University of the Witwatersrand Library.

2.1.3.3 Borland C++ version 4.0.

2.1.4 What other resources are available for maintenance use during the product service life? (i.e. facilities/personnel and quantitative or qualitative limits).

None.

2.1.5 What methods of progress measurement does the customer require to track the development task? (e.g. status reports, progress reports).

2.1.5.1 MSc. project proposal.

2.1.5.2 Interim reports handed in periodically.

2.1.6 What are the milestones that shall be met? (i.e. schedules, deliveries).

2.1.6.1 User Reference Manual - 01/12/94

2.1.6.2 Technical Reference Manual - 01/02/95

2.1.6.2 Final Product - 01/07/95

2.2 Hardware/Software

2.2.1 What computers shall the product run on? (e.g. makes, models, particular installations).

80486 PC and a minimum of 8Mb RAM.

2.2.2 What peripherals are required to interface with the product? (e.g. terminals, hardcopy devices, tape drives, disk drives, displays, etc. Specify makes and models).

i) A mouse or other pointing device.

ii) Super VGA display.

iii) Printers and plotters with drivers for use with Windows 3.1 may optionally be used.

2.2.3 What unique interfaces does the product need to access? (e.g. cockpit hardware, microcomputer, test benches etc. Specify makes and models).

i) The EMTP/ATP Fortran Data Card Interface.

ii) Microsoft Access 2.0 Database.

2.2.4 On what operating system shall the product run? (i.e operating system ID, revision level).

MS Windows 31. (or better).

2.2.5 With what other external (foreign to this product) software products shall this product interface? (e.g. routines, libraries, product names, manufacturers, suppliers, revision levels, etc.)

Microsoft Windows 3.1., Microsoft Access 2.0

2.2.6 What are the size restrictions for the finished product? (e.g. bytes of code, text, embedded data etc.)

There is no size restriction at this point.

2.2.7 How transportable shall the product be? (i.e. range of hardware, software environments under which the product will run).

The product will be able to run under any operating system that supports Windows 3.1. This includes OS/2 version 2.1 and Windows NT. The product will not run under Windows 3.0.

2.3 Required Design Restrictions

2.3.1 What are the language restrictions for the product? (e.g. language names, versions; or maximum number of languages, etc.)

None at this stage.

2.3.2 To what standards shall the product conform? (e.g., document titles, sources, revision levels, etc.)

ISO9001 software quality standards.

2.4 Customers

2.4.1 Who is commissioning the product? (e.g. identify signatures of authority etc.)

Dr. I. Jandrell, University of the Witwatersrand, Johannesburg.

2.4.2 Who is to accept delivery? (e.g. identify signatures of authority etc.)

As for 2.4.1.

2.4.3 What is the profile of the typical end user? (e.g. years of experience in some specified job, years of schooling etc.)

Typically the product will be used by an electrical engineer who is simulating transmission networks at a power utility.

2.4.4 What are the human factors criteria and how shall the success of the product in meeting the criteria be determined? (e.g. non-direct paths, non-standard user-interface, etc.)

The user interface is to conform to the IBM Common User Access and the Microsoft Applications Design Guide specifications.

2.5 Performance factors

2.5.1 What human support is required to operate the product? (e.g. Identify any runtime operations which the user or support personnel shall physically carry out in order to achieve any specified results etc.)

None.

2.5.2 What periods of time shall the product be available for operation? (e.g. hours per day, weekly schedule, etc.)

Continuous.

2.5.3 What criteria must the product meet regarding reliability of operation?

The product should be able to perform a small simulation from within the Windows platform, and larger simulations from the DOS platform.

2.6 Operational considerations

2.6.1 What methods shall be used to deliver the product? (e.g. data storage or transfer medium)

High density stiffy disks. (3.5 inch).

2.6.2 What are the restrictions on the installation of the product? How easy shall the product be to install? (e.g. classes of users, numbers of users, access limitations etc.)

There are no access limitations.

2.6.3 To what degree shall the operation of the product be successfully restricted to authorized users only? (e.g. classes of users, numbers of users, access limitations etc.)

There are no access limitations.

2.6.4 How shall the product integrity be ensured? (e.g. data recovery, source code recovery, etc.)

A backup floppy/stiffy containing source code and executable files will be supplied with the product. Another floppy containing the source for all the documentation supplied with the program will also be supplied.

2.7 Maintainability

2.7.1 What support of the product after turnover to the customer is required? (e.g. training support, computer personnel, etc.)

User Reference Manual, Technical Reference Manual, Product Test Specification.

2.7.2 Where might the requirements for the product expand in the future? (i.e. percentage chance of change).

2.7.2.1 New modules may be added to the ATP package, which would require the updating of the interface to allow for the use of these modules within the interface.

2.7.2.2 Improved interfacing capabilities may be introduced; this implies the incorporation of new import and export data formats.

2.7.2.3 The interface may be change to make it more compatible to MS Windows 4.0 (Chicago) if this version of Windows is vastly different from version 3.1.

2.8 Product Acceptance

2.8.1 Who is responsible for accepting the product?

Dr. I. Jandrell.

2.8.2 What test, or tests, must be applied to establish acceptability?

Beta Testing

2.8.3 What is the sample data to be supplied for acceptance testing?

Beta Testing data (ie. Random)

3 Inputs

3.1 What are the input media? (e.g. touch screen, keyboard, tape etc.)

Normal QWERTY keyboard, mouse, floppy disk drive, stiffy disk drive or hard drive.

3.2 What is the format of the input?

See User Reference Manual.

3.3 What are the input items?

See User Reference Manual.

3.4 What are the characteristics of each input item? (e.g. units, data type, sampling rate etc.)

See User Reference Manual.

3.5 For each item what is the acceptable level of accuracy (tolerance)?

See User Reference Manual.

3.6 For each numeric item, how many significant digits are required?

See User Reference Manual.

3.7 What are the descriptions and range for each item? (e.g. month of the year, file spec. etc.)

See User Reference Manual.

4 Process

4.1 Function

4.1.1 What processes is the product to perform?

4.1.1.1 The conversion of Windows graphics input to ATP data files.

4.1.1.2 The conversion of Intergraph Microstation graphics files to DBase III type files.

4.1.1.3 The conversion of Dbase III type files to ATP data files.

4.1.1.4 The conversion of ATP output files to written and graphical data on the Windows platform.

4.1.2 What are the algorithms?

None.

4.1.3 What are the associations between functions and processes?

See User Reference Manual, Technical Reference Manual.

4.1.4 What are the modes of operation? (e.g. interactive, batch etc.)

4.1.4.1 For the entry of a problem geometry by the user the application will function interactively.

4.1.4.2 During the solution of a problem, the application will follow this sequence:

4.1.4.2.1 The writing of the ATP data files

4.1.4.2.2 The running of the ATP simulation using the data files in 4.1.4.2.1

4.1.4.2.3 The reading of the output data files from text files written by ATP

4.1.4.2.4 The writing of the output graphically on the Windows platform within the interface

4.1.5 What options are available prior to startup? (e.g. switches, job parameters, etc.)

See User Reference Manual.

4.1.6 What options are available during runtime? (e.g. selection of I/O, job

parameters, etc.)

See User Reference Manual.

4.1.7 How does the product use all the input data?

See User Reference Manual.

4.1.8 How does the product define all output data?

See User Reference Manual.

4.2 Timing

4.2.1 What are the constraints on response time? (e.g. acknowledge time, results time, etc.)

In all situations where the response time exceeds 0.5s, the cursor turns into an hourglass icon indicating that the program is busy.

4.2.2 What are the constraints on iteration rate? (e.g. frames per second)

None.

4.2.3 What is the latency required? (e.g. data transmission in multi-processing)

Not applicable.

4.2.4 What are the CPU usage constraints for each process?

4.2.4.1 During interactive processing (i.e., entry of problem geometry) non-preemptive multitasking will take place therefore CPU usage will be partial.

4.2.4.2 During running of ATP, non-preemptive multitasking will take place therefore CPU usage will be partial.

4.2.5 What are the CPU usage constraints for the product?

There are no CPU usage constraints for the product as a whole.

4.3 Error Handling

4.3.1 What defines an error? (internal, user induced, system induced, disaster recovery, etc.)

See User Reference Manual.

4.3.2 How is each error communicated? (e.g. setting error flags, sending message to user, etc.)

See User Reference Manual.

4.3.3 How does the product handle error condition? (e.g. program abort, require new input, restore original environment, etc.)

See User Reference Manual.

5 Output

5.1 What are the output media? (e.g. touch screen, keyboard, tape, etc.)

Screen, printer, plotter, disk.

5.2 What is the format of the output? (e.g. record layout, etc.)

See User Reference Manual, Technical Reference Manual.

5.3 What are the output items?

See User Reference Manual.

5.4 What are the characteristics of each output item? (e.g. units, data type, sampling rate, etc.)

See User Reference Manual, Technical Reference Manual.

5.5 What is the acceptable level accuracy? (e.g. tolerance)

Not Applicable.

5.6 For each numeric item, how many significant digits are required?

See User Reference Manual, Technical Reference Manual.

5.7 What are the descriptions and range for each data item? (e.g. month of the year, file spec. etc.)

See User Reference Manual, Technical Reference Manual.

5.8 What are the expected values of output for a given set of inputs?

See User Reference Manual, Technical Reference Manual, Product Test Specification.

6 Documentation

6.1 Requirements

6.1.1 What documents have been referenced in the requirements document?

6.1.1.1 MSc Preliminary Report.

6.1.1.2 User Reference Manual.

6.1.1.3 Technical Reference Manual.

6.1.2 What additional documents are recommended for the development of the product?

6.1.2.1 Literature survey on how ATP works

6.1.2.2 Manuals associated with the relevant programming tools.

6.1.3 What are the definitions of all the acronyms and technical terms?

Refer to the User Reference Manual, Technical Reference Manual.

6.1.4 What technical or legal authorizations are required to develop the product?

None.

6.1.5 Who has created and revised the requirements document and what are the dates of creation and revision?

Tamir Orbach

Date of creation: 6 June 1993.

Date of revision: N/A.

6.2 Support

6.2.1 What type of support documentation shall the developer provide with the product?

6.2.1.1 User Reference Manual.

6.2.1.2 Technical Reference Manual.

6.2.1.3 Product Test Specification.

6.2.2 To what standards shall the support documentation for the product conform?

ISO9001

APPENDIX B - THE ATP FOR WINDOWS USER REFERENCE MANUAL

SEE VOLUME 1 OF SEPARATE VOLUMES

APPENDIX C - THE ATP FOR WINDOWS TECHNICAL REFERENCE MANUAL

SEE VOLUME 2 OF SEPARATE VOLUMES

APPENDIX D - THE OBJECT-ORIENTED DATA CARD UNIT

SEE VOLUME 3 OF SEPARATE VOLUMES

APPENDIX E - ATP CASE STUDIES

1. Problem: Lightning strikes a resistor as shown below. Determine the waveforms of the voltage at the top of the resistor and of the current through the resistor. Also determine the total energy dissipated in the resistor.



Assume that the lightning current waveform can be approximated as

$$i(t) = 3.2E4[e^{(-2\ E6)t} - e^{(-1.E7)t}]$$

Solution:

Every ATP data case begins with the following lines

BEGIN NEW DATA CASE

$OPEN , UNIT=4 FILE=a:tut1.pl4 FORM=UNFORMATTED

(where a:tut1.pl4 is the filename selected by the user to which the pl4 (ATP output data file) is written).

This is followed by the floating-point miscellaneous data card

This card comprises six variables which must be placed between specific columns in the line, and must be right justified in the line. These are:

- DELTAT (Columns 1-8) - The simulation time step in floating point format, in seconds.
- TMAX (9-16) - Required duration of simulation in floating point format, in seconds.
- XOPT (17-24) - This is a flag for indicating the form of inductance input - whether inductance in linear branches is to be specified in mH (ix XOPT is set to 0.0) or in Ohms - inductive reactance (if XOPT is set to 50.0 then the inductive reactance is for a frequency of 50Hz).
- COPT (25-32) - same as above but indicates the form of capacitance input - whether capacitance in linear branches is to be specified in μF (if COPT is set to 0.0) or in μmhos - capacitive susceptance (if COPT is set to 50.0 then the capacitive susceptance is for a frequency of 50Hz)
- EPSILN (33-40) - near zero tolerance that is used to test for singularity of the real coefficient matrix within the time step loop. A blank field or zero value will result in the default value being used (1.E-8).
- TOLMAT (41-48) - near zero tolerance that is used to test singularity of the complex admittance matrix [Y] of the steady state, phasor solution. A blank field or zero value will result in the value of EPSILN being used.

Next comes the integer miscellaneous data card. This card comprises ten entries placed in eight column fields:

- IOUT (1-8) - Defines the frequency of output relative to the simulation time step.
- IPLOT (9-16) - Defines the frequency of saving simulation values for later plotting.
- IDOUBL (24) - This is a flag for controlling the output of a table showing network connectivity.
- KSSOUT (32) - This is a flag for controlling the printout of the steady-state phasor solution. There are three basic types of outputs: branch flows, switch flows, and nodal injections. These can be controlled by the KSSOUT value as follows:

0 - no steady state solution printout

1 - print the complete steady state solution: branch flows, switch flows and source injections

2 - print switch flows and source injections, but not branch flows.

3 - print branch flows requested by column 80 flags on the branch cards, switch flows, and source injections.

- MAXOUT(40) - This is a flag for controlling the printout of extrema at the completion of the simulation. A 0 will suppress such output whereas a 1 will request it.
- IPUN (47-48) - This is a flag for controlling the presence of an extra following card to manipulate the printout frequency. A -1 indicates an extra card while a blank or 0 indicates no card.
- MEMSAV (56) - This is a flag for controlling the dumping of ATP memory onto disk at the end of te simulation for subsequent use with the START AGAIN request. Enter a 1 if such a memory is required, or a 0 if it is not.
- ICAT (64) - This is a flag for the generation of a plot file. A 0 will supress the generation of a plot ile, while a 1 will generate te plot file but ignore any batch mode plot cards that may be present, and a 2 will generate a plot file and honour and batch mode plot cards that may be present.
- NENERG (65-72) - A flag which is used to indicate the number of iterations required when performing Monte Carlo analysis using a switch with a systemattically increasing closing time. Usually left blank.

These are then followed by the branch cards. The branch card for a resistor is the Simple RLC branch. This card includes the following entries:

- ITYPE (1-2) - This is a code which is specific to each type of element. In the case of a simple RLC branch, the code is "00".
- NODE1(3-8) and NODE2(9-14) - These are the node names of the two terminating nodes of the RLC branch. If the node2 name is left blank it is assumed to be connected to earth.
- NODE3(15-20) and NODE4(21-26) - These are the node names of any reference RLC

branch whose properties this branch will have.

- R (27-32) - The resistance specified in floating point format in ohms.
- L (33-38) - The inductance specified in floating point format in mH or in Ohms (depending on the specification of XOPT in the floating point miscellaneous card).
- C (39-44) - The capacitance specified in floating point format in μF or μmhos (depending on the specification of COPT in the floating poinr miscellaneous card).
- P (80) - This is a flag for specifying what outputs are required from this branch. Ths options are

  0 - none

  1 - branch current

  2 - branch voltage

  3 - branch current and voltage

  4 - instantaneous power into branch and net energy flow into branch

Next a blank card ending the branch cards:

BLANK card ending branch cards

then a blank card ending the switch cards (in this case there are no switch cards, the but the blank card is still needed.

BLANK card ending switch cards

This is followed by the source cards. The source card for a double exponential function includes the following entries:

- ITYPE (1-2) - Same as previous ITYPE, but "15" in this case
- NODE (3-8) - name of the node to which the source is connected (the source is placed between this node and local ground).
- IV (9-10) - This is a flag to specify whether the source is a current source or a voltage source. A 0 indicates a voltage source, while a -1 indicated a current source.]
- A (11-20), B(21-30), C (31-40) - These are the defining variables of the equation, ie.

  $$f(t) = A[e^{Bt} - e^{Ct}]$$

- TSTART - The delay before the source is activated, in floating point format, in seconds.
- TSTOP - The time at which the source is terminated - source value is pulled to zero, in floating point format, in seconds.

The source cards are then followed by a Blank card ending the source cards:

BLANK card ending source cards

which is then followed by the following:

BLANK card ending output request cards

BEGIN BEW DATA CASE

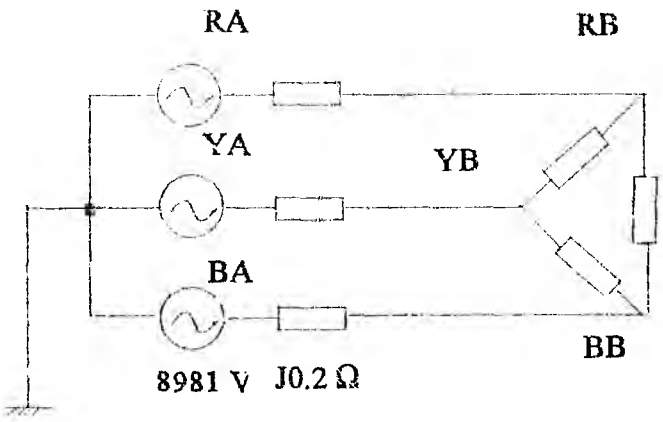
BLANK card ending session

All of the above entries will cause an error if they are entered in an incorrect format.

Using these entries, the required data file for this example is as follows:

```
C C:\VB\ATPWIN\TUT1.DAT
BEGIN NEW DATA CASE
$OPEN, UNIT=4 FILE=c:TUT1.pl4 FORM=UNFORMATTED
  2.E-7  1.E-4     0     0
    1     1     0     3     0              1
00A                 10.                          3
BLANK Card ending branch cards
BLANK Card ending switch cards
15A    -1    3.2E4    -2.E6    -1.E7
BLANK Card ending source cards
 1
BLANK Card ending plot cards
BEGIN NEW DATA CASE
BLANK Card Ending Session
```

2. **Problem:** A 50kW delta-connected load with a power factor of 0.9 lagging is connected to the 11kV generator below. Determine the waveforms of the line and phase currents.



Solution:

$$50000 = 3V_{phase}I_{phase}\cos\theta$$
$$= 3\,(11000)I_{phase}0.9$$

therefore $I_{phase} = 1.68A$

and $Z = (11000/1.68) = 6548\Omega$

and $R = 6548 * 0.9 = 5893\Omega$

and $X = 6548 * \sin(\arccos(0.9)) = 2863\Omega$

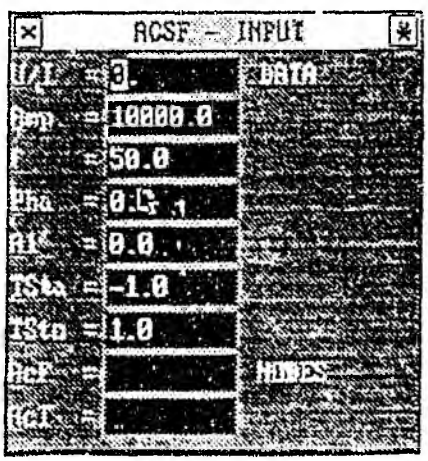
This requires the following data case:

```
C C:\VB\ATPWIN\TUT2.DAT
BEGIN NEW DATA CASE
$OPEN, UNIT=4 FILE=c:TUT2.pl4 FORM=UNFORMATTED
  .001    .2   50.0     0
    1     1     0     3     0              1     0
00RA   RB                 0.2                          1
00RB   YB           5893. 2863.                          1
00YA   YB   RA   RB                                     1
00RB   BB   RB   YB                                     1
00BB   YB   RB   YB                                     1
00BA   BB   RA   RB                                     1
BLANK Card ending branch
BLANK Card ending switch cards
14RA   0   8981.    50.0     0.0      0          -1.      0.
14YA   0   8981.    50.0   120.0      0          -1.      0.
14BA   0   8981.    50.0   240.       0          -1.      0.
BLANK Card ending source cards
BLANK Card ending plot cards
BEGIN NEW DATA CASE
BLANK Card Ending Session
```

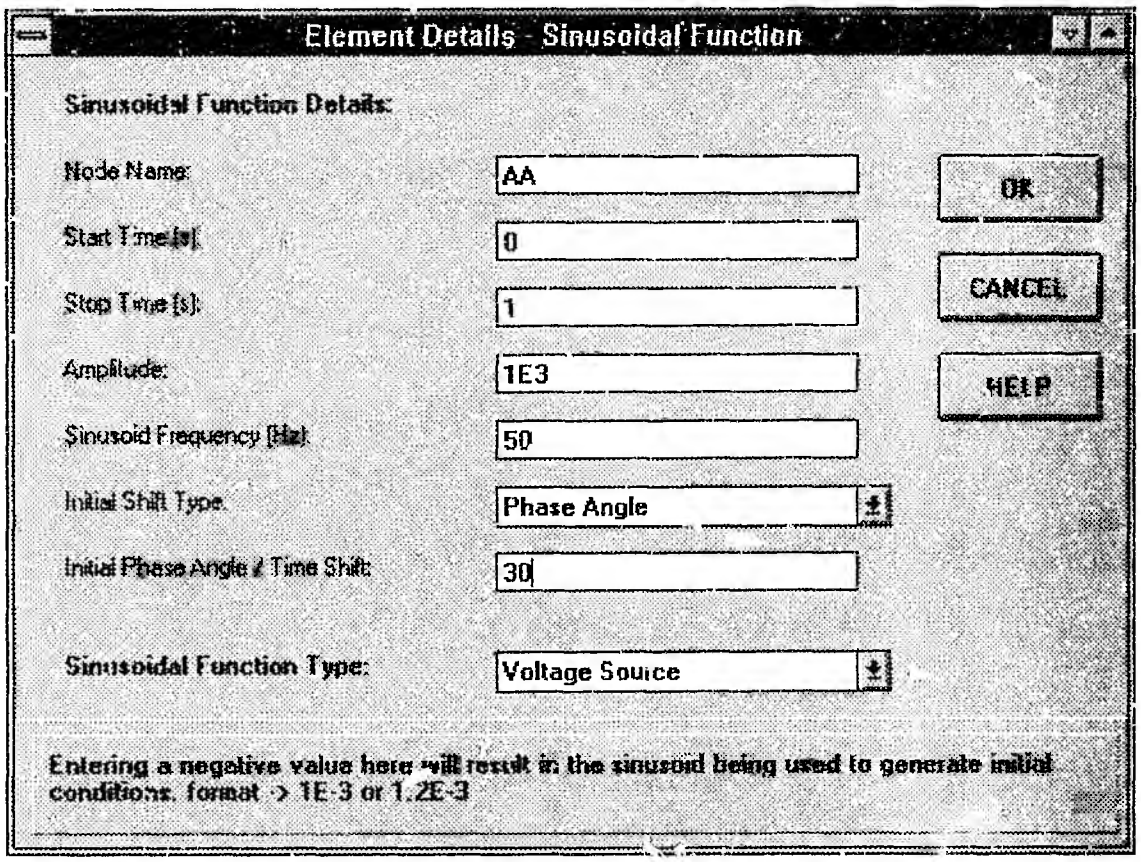
All examples seen above taken from (Van Coller 1993).

APPENDIX F - A COMPARISON OF ATPDRAW AND ATP FOR WINDOWS

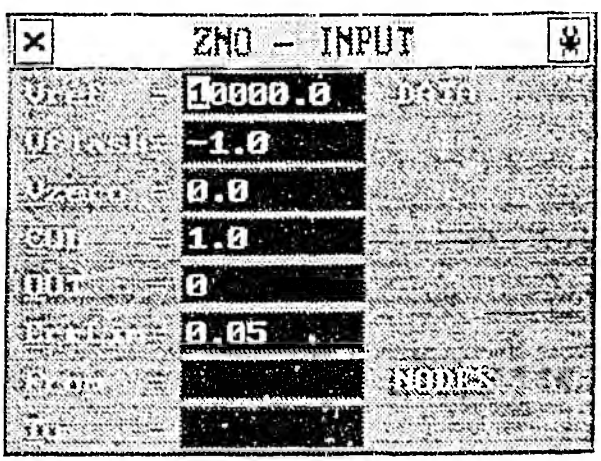
In order to compare ATPDRAW and ATP for Windows, it is necessary to look at some of the typical input screens which these two programs make use of. Since both of these interfaces are graphical, they make use of the principle of dragging and dropping circuit elements to draw the circuit, and then clicking on the right mouse button to obtain an input screen for the data related to the particular element which the user is pointing to. A good example of an input screen is the Element Details screen of the Sinusoidal Source / Sin Function. The screen for this entry in ATPDRAW is as follows:



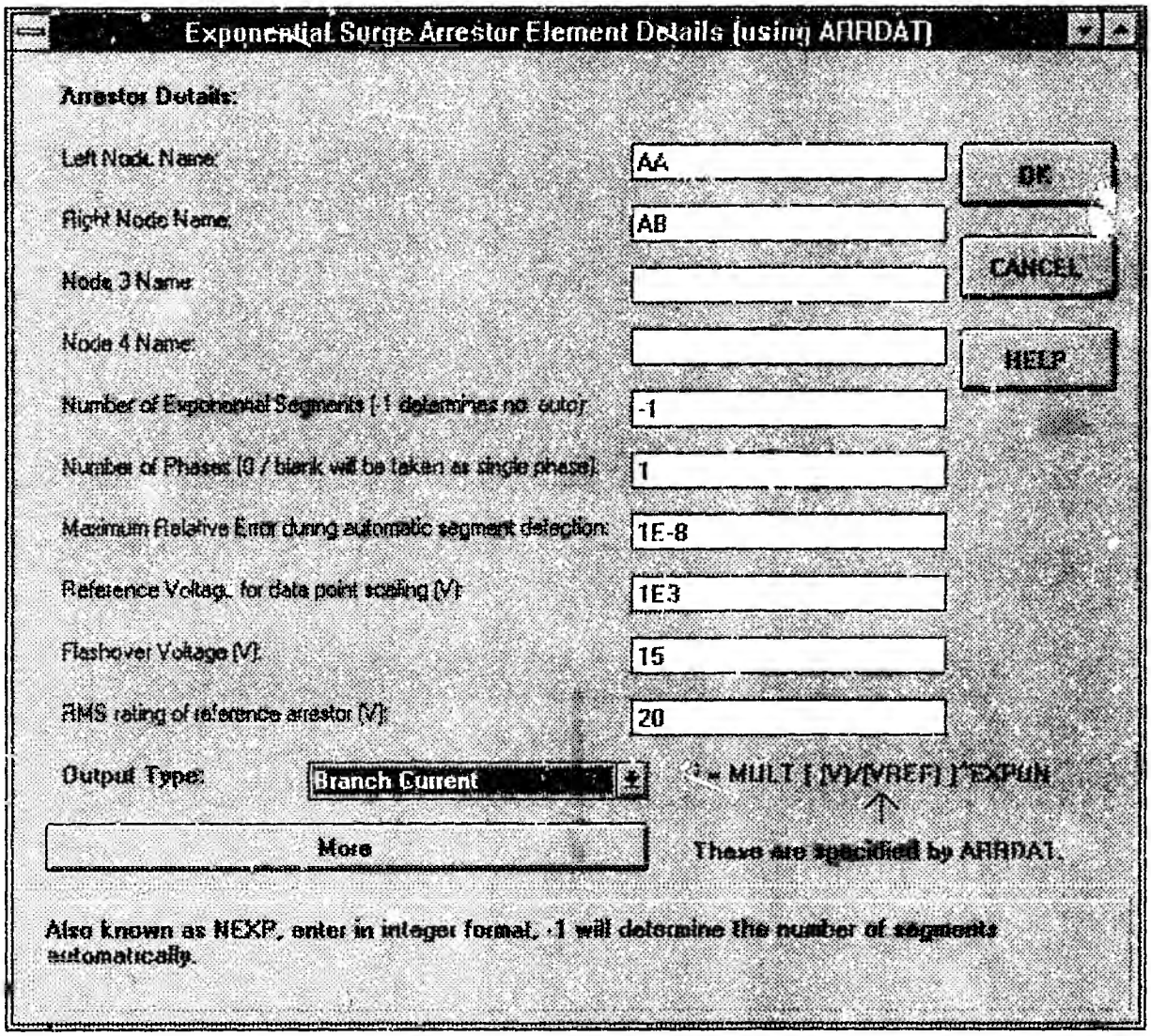
As can be seen in this screen, the input labels are all abbreviated. The use of abbreviations for all the entries is one of the factors which make ATP itself difficult to use, and most of the abbreviations used in ATPDRAW correspond to those used in ATP itself. The equivalent screen in ATP for Windows is as follows:



The entries in ATP for Windows are clear, and the message bar at the bottom of the screen gives the user any other necessary information. Another example which show the difference between these two interfaces is the ZnO exponentially modelled surge arrestor input screen. The screen for this element in ATPDRAW is as follows:



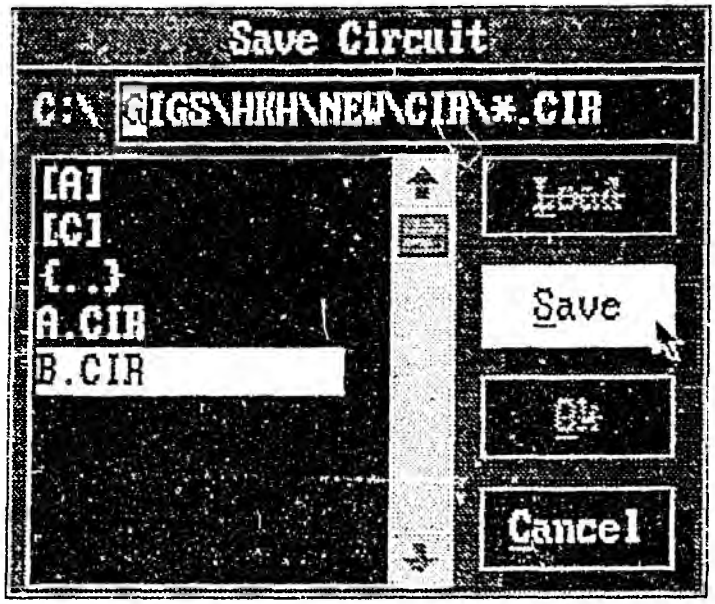
The corresponding screen in ATP for Windows can be seen in the figure below:



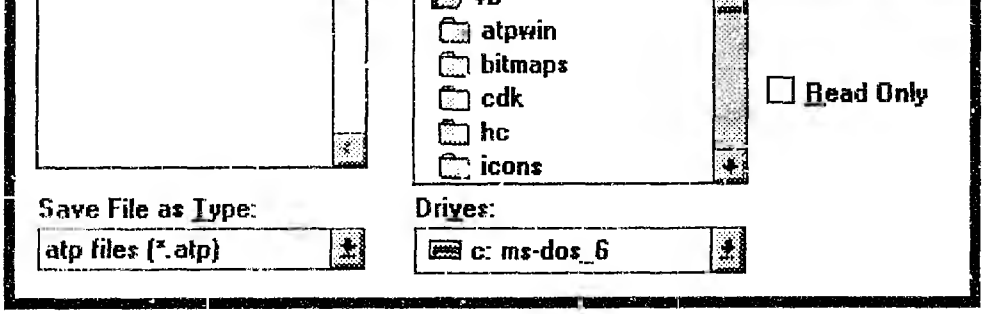
Again, ATP for Windows has clear labels, and a message bar at the bottom of the screen which informs the user information specific to each entry as the user traverses over these entries with the mouse.

Other typical screens in ATPDRAW include the Save screen, which can be seen in the figure

below:



The corresponding screen in ATP for Windows can be seen in the figure below:



It is clear from the above comparison that ATP for Windows is more user-friendly than ATPDRAW.

APPENDIX G - ADDING AN ELEMENT TO ATP FOR WINDOWS TO ASSESS THE MAINTAINABILITY OF THE PROGRAM

To show the steps which are required when adding a new element to ATP for Windows, the Double Exponential Function will be used.

Adding an element to ATP for Windows involves the following steps:

- Drawing the icon on an icon drawing program such as Iconworks (which is a sample program shipped with Visual Basic 3.0). The icons used in ATP for Window are *.ico files. The icon for the Double Exponential Function is can be seen in figure G-1 below.
- Placing a new 3d command button in the circuit panel (using the existing array of 3d command buttons - ie. copy and existing button, and paste it into the circuit panel).
- Setting the 'Picture' property and the 'Dragicon' property of the new 3d command button to the file name of the icon drawn for the new element.
- Setting the 3d command button's 'Dragmode' property to *Automatic*. This allows the icon to automatically drag when the user clicks on the 3d command button associated with this element.

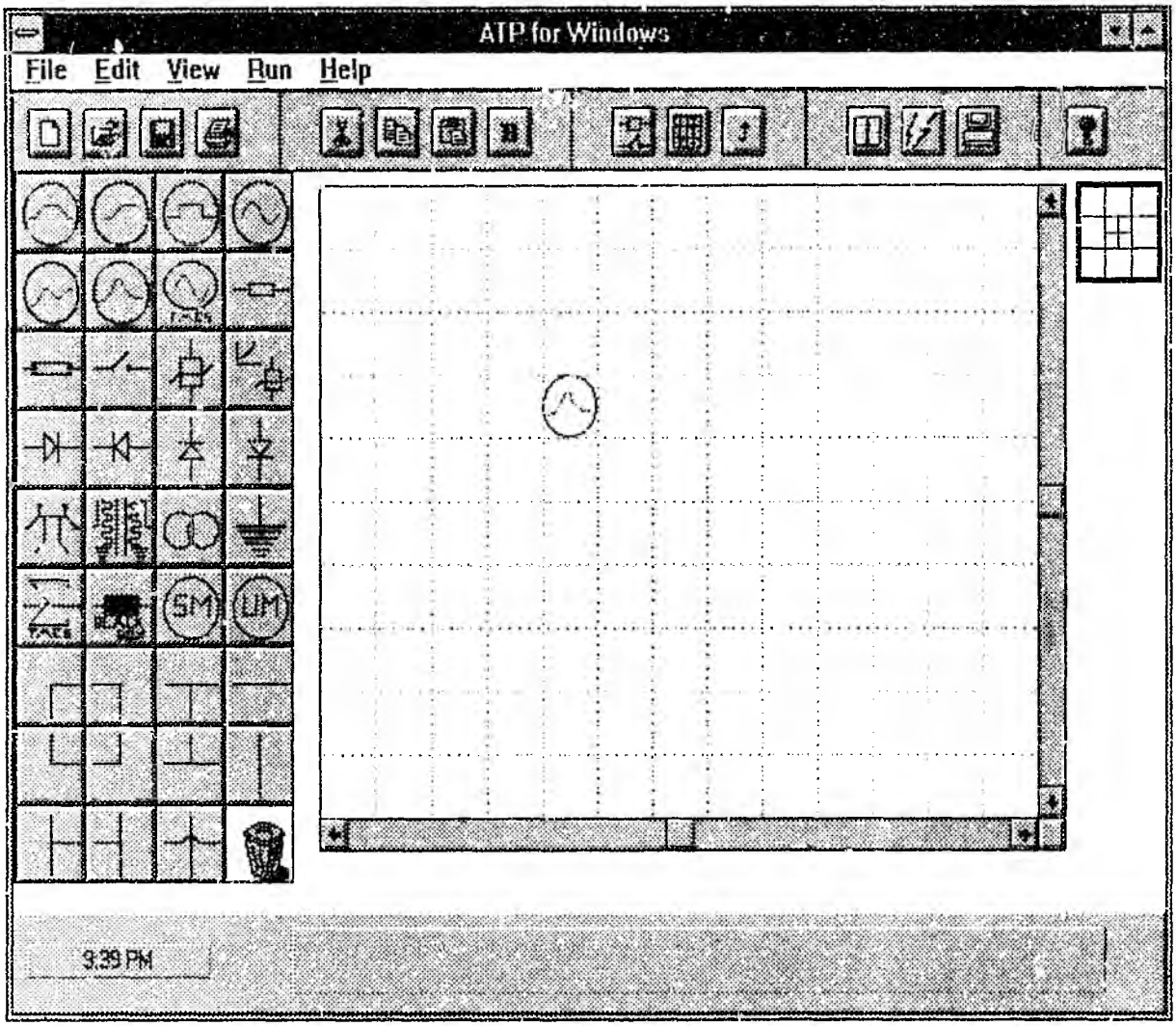


Figure G-1 - The Main Screen of ATP for Windows with the Double Exponential Function Icon

- Design a new form (select New Form from the toolbox), placing the necessary command buttons including OK, CANCEL, and HELP on the screen. Place labels for each entry relating to a double exponential function in ATP on the form. Then place a text box or a combo box next to each label (depending on whether an entry is required, or the variable is a flag, and name the text box using the prefix txt (the combo boxes with the prefix cmb), and then the name of the text box's corresponding variable. Create a message bar by placing two 3d panels on top of each other, and then a label control on top of the 2nd panel. Name the label lblMessage. Change the 'mouseover' events of each label and text

box / combo box on the screen so that the caption of the lblMessage label changes to whatever is relevant to that entry when the user traverses the mouse over the entry. The final screen is as follows:

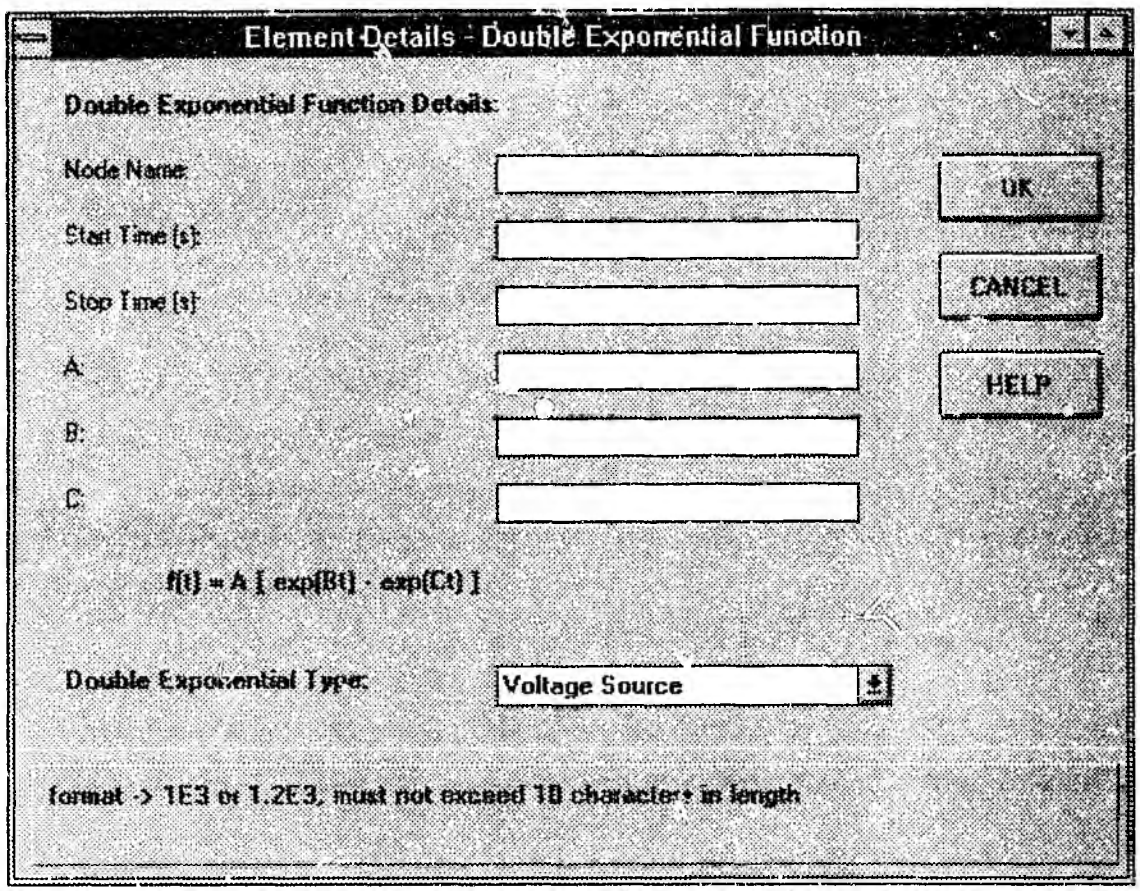


Figure G-2 - The Double Exponential Function Element Details Screen

- Using the Data Manager of Visual Basic, create a new table in the circuit.mdb database included with ATP for Windows. Name the table 'Double Exponential Function', and add fields to the table for every entry required by ATP for the double exponential function.
- In the ATP.BAS module (the main module), create a new Global Const called DOUBLEEXPONENTIALFUNCTION, and set it's value to the 'Index' property of the corresponding 3d command button. ie.

```
Global Const DOUBLEEXPONFUNCTION = 12
```

- Create a global integer variable in the ATP.BAS module called *ElementName*Count

```
Global DoubleExponentialFunctionCount As Integer
```

- In the ATP.BAS module, go to the *InitializeAllVariables* method, and put in a line which initializes the DoubleExponentialFunctionCount variable to -1 (As soon as a Double Exponential Function element is created, this variable is incremented to 0 etc.)
- In the ATP.BAS module, go to the 9 methods called DrawCircuit, DrawTopCircuit, DrawBottomCircuit, etc. and place a Case statement to draw the element in the particular screen required:

```
Case DOUBLEEXPONFUNCTION
    frmBottomImages.imgGridTemp(j).Picture =
frmATP.cmd3dCircuitPanel(DOUBLEEXPONFUNCTION).DragIcon
```

- In the ATP.BAS module, go to the OpenCircuit method. Add a call for the DoubleExponentialFunction. ie.

```
DoubleExponentialFunctionOpenCircuit
```

- In the ATP.BAS module, go to the SaveCircuit method. Place a CASE statement in each For loop as follows

```
Case frmATP.cmd3dCircuitPanel(DOUBLEEXPONENTIALFUNCTION).DragIcon
    DoubleExponFunctionSaveCircuit "Main", i
```

The position you place in the "Main" section depends on which For loop this statement is in. For eg., the LeftTop part of the circuit area has its own For loop, which will have the statement as follows:

```
Case frmATP.cmd3dCircuitPanel(DOUBLEEXPONENTIALFUNCTION).DragIcon
    DoubleExponFunctionSaveCircuit "LeftTop", i
```

- In the ATP.BAS module, go to the WriteNodeNames method, and add a Case statement to call the double exponential function's WriteNodeNames method (see later):

```
Case frmATP.cmd3dCircuitPanel(DOUBLEEXPONENTIALFUNCTION).DragIcon
    DoubleExpFuncWriteNodeNames i
```

- Create a new module. Define an *ElementName*Number integer variable, and define a type called the DoubleExponentialFunctionType and include all the variables required for a double exponential by ATP, and then define an array of approximately 100 elements of this type:

```
Dim DblExpFncNumber As Integer
```

```
Type DoubleExpFnc
        NodeName As String
        StartTime As String
        StopTime As String
        A As String
        B As String
        C As String
        FunctionType As Integer
End Type
Dim DblExpFncs(100) As DoubleExpFnc
```

This type is used to save all the elements details in the array of double exponential functions. The DblExpFuncNumber variable is used to keep track of how many double exponential functions are available in the circuit, as well as to access the information of the correct element at different points of the simulation.

- Write a method which initializes the form you have created. This method must check whether or not the element is a new element. This is done as follows:

```
Sub FormDblExpFuncInitialize
        'This method checks if the double exponential function is a
        'new element. If it is not a new element, the text box
        'for this element is filled in with whatever was in the
        'element before. Also, if it is a new element, the
        'black box count element is incremented.

        frmDoubleExponentialFunction.cmbDoubleExponentialType.AddItem ("Voltage Source")
        frmDoubleExponentialFunction.cmbDoubleExponentialType.AddItem ("Current Source")
        frmDoubleExponentialFunction.cmbDoubleExponentialType.ListIndex = 0

        Dim NewElement As Integer
        DblExpFncNumber = -1
        NewElement = True
        DblExpFncCount = DblExpFncCount + 1

        If DblExpFncCount = 0 Then
           If frmATP.imgGrid(picGridIndex).Tag = -1 Then
              NewElement = True
           Else
              DblExpFncNumber = 0
              NewElement = False
           End If
        End If

        If DblExpFncCount > 0 Then
           If frmATP.imgGrid(picGridIndex).Tag = -1 Then
                 NewElement = True
          Else
                 DblExpFncNumber = frmATP.imgGrid(picGridIndex).Tag
                 NewElement = False
                DblExpFncCount = DblExpFncCount - 1
          End If
        End If

        If NewElement = False Then
```

```
            frmDoubleExponentialFunction.cmdOK.Enabled = True
            frmDoubleExponentialFunction.txtNodeName.Text = DblExpFncs(DblExpFncNumber).NodeName
            frmDoubleExponentialFunction.txtStartTime.Text = DblExpFncs(DblExpFncNumber).StartTime
            frmDoubleExponentialFunction.txtStopTime.Text = DblExpFncs(DblExpFncNumber).StopTime
            frmDoubleExponentialFunction.txtA.Text = DblExpFncs(DblExpFncNumber).A
            frmDoubleExponentialFunction.txtB.Text = DblExpFncs(DblExpFncNumber).B
            frmDoubleExponentialFunction.txtC.Text = DblExpFncs(DblExpFncNumber).C
          frmDoubleExponentialFunction.cmbDoubleExponentialType.ListIndex =
        DblExpFncs(DblExpFncNumber).FunctionT
          Else
            frmDoubleExponentialFunction.cmdOK.Enabled = False
            DblExpFncNumber = DblExpFncCount
            frmATP.imgGrid(picGridIndex).Tag = DblExpFncNumber
          End If
End Sub
```

- Write a method which saves the values on the Double Exponential Function screen when the user clicks on OK:

```
Sub DblExpFuncOK
        DblExpFncs(DblExpFncNumber).NodeName = frmDoubleExponentialFunction.txtNodeName.Text
        DblExpFncs(DblExpFncNumber).StartTime = frmDoubleExponentialFunction.txtStartTime.Text
        DblExpFncs(DblExpFncNumber).StopTime = frmDoubleExponentialFunction.txtStopTime.Text
        DblExpFncs(DblExpFncNumber).A = frmDoubleExponentialFunction.txtA.Text
        DblExpFncs(DblExpFncNumber).B = frmDoubleExponentialFunction.txtB.Text
        DblExpFncs(DblExpFncNumber).C = frmDoubleExponentialFunction.txtC.Text
        DblExpFncs(DblExpFncNumber).FunctionType =
         frmDoubleExponentialFunction.cmbDoubleExponentialType.ListIndex
        Unload frmDoubleExponentialFunction
End Sub
```

- Write a method which saves the values of an element to disk (to the MS Access table):

```
Sub DoubleExpFuncSaveCircuit (Position As String, i As Integer)
        Dim Number As Integer
        Dim BBTable As Table
        Select Case Position
        Case "Main"
                Number = frmATP.imgGrid(i).Tag
        Case "Top"
                Number = frmTopImages.imgGridTemp(i).Tag
        Case "Bottom"
                Number = frmBottomImages.imgGridTemp(i).Tag
        Case "Left"
                Number = frmLeftImages.imgGridTemp(i).Tag
        Case "Right"
                Number = frmRightImages.imgGridTemp(i).Tag
        Case "LeftTop"
                Number = frmLeftTopImages.imgGridTemp(i).Tag
        Case "LeftBottom"
                Number = frmLeftBottomImages.imgGridTemp(i).Tag
        Case "RightTop"
                Number = frmRightTopImages.imgGridTemp(i).Tag
        Case "RightBottom"
                Number = frmRightBottomImages.imgGridTemp(i).Tag
        End Select

        Set BBTable = Circuit.OpenTable("Double Exponential Function")
        BBTable.AddNew
```

```
        BBTable!Index = i
        Select Case Position
        Case "Main"
                BBTable!Position = "Main"
        Case "Top"
                BBTable!Position = "Top"
        Case "Bottom"
                BBTable!Position = "Bottom"
        Case "Left"
                BBTable!Position = "Left"
        Case "Right"
                BBTable!Position = "Right"
        Case "LeftTop"
                BBTable!Position = "LeftTop"
        Case "LeftBottom"
                BBTable!Position = "LeftBottom"
        Case "RightTop"
                BBTable!Position = "RightTop"
        Case "RightBottom"
                BBTable!Position = "RightBottom"
        End Select
        If Not Number = -1 Then
                BBTable![Node Name] = DblExpFncs(Number).NodeName
                BBTable![Start Time] = DblExpFncs(Number).StartTime
                BBTable![Stop Time] = DblExpFncs(Number).StopTime
                BBTable!A = DblExpFncs(Number).A
                BBTable!B = DblExpFncs(Number).B
                BBTable!C = DblExpFncs(Number).C
                BBTable![Function Type] = DblExpFncs(Number).FunctionType
                BBTable!Exists = 1
        Else
                BBTable!Exists = 0
        End If


        BBTable.Update
End Sub
```

- Write a method which retrieves this information from the database:

```
Sub DblExpFncOpenCircuit ()
        Dim CctTable As Table
        Dim LoopEnd As Integer
        Dim Exists As Integer
        Dim Index As Integer
        Dim Number As Integer

        Set CctTable = Circuit.OpenTable("Double Exponential Function")
        If Not CctTable.EOF Then
        CctTable.MoveFirst
        Do
         If CctTable!Exists = 0 Then
                Exists = False
         ElseIf CctTable!Exists = 1 Then
                Exists = True
         End If
         LoopEnd = False
         If Exists = True Then
                DblExpFncCount = DblExpFncCount + 1
```

```
            Number = DblExpFncCount
        Else
            Number = -1
        End If
        Index = CctTable!Index
        Select Case CctTable!Position
            Case "Main"
                DrawSavedPicture Index, DOUBLEEXPONFUNCTION
                frmATP.imgGrid(Index).Tag = Number
            Case "Left"
                DrawSavedLeftPicture Index, DOUBLEEXPONFUNCTION
                frmLeftImages.imgGridTemp(Index).Tag = Number
            Case "Right"
                DrawSavedRightPicture Index, DOUBLEEXPONFUNCTION
                frmRightImages.imgGridTemp(Index).Tag = Number
            Case "Top"
                DrawSavedTopPicture Index, DOUBLEEXPONFUNCTION
                frmTopImages.imgGridTemp(Index).Tag = Number
            Case "Bottom"
                DrawSavedBottomPicture Index, DOUBLEEXPONFUNCTION
                frmBottomImages.imgGridTemp(Index).Tag = Number
            Case "LeftBottom"
                DrawSavedLeftBottomPicture Index, DOUBLEEXPONFUNCTION
                frmLeftBottomImages.imgGridTemp(Index).Tag = Number
            Case "RightBottom"
                DrawSavedRightBottomPicture Index, DOUBLEEXPONFUNCTION
                frmRightBottomImages.imgGridTemp(Index).Tag = Number
            Case "LeftTop"
                DrawSavedLeftTopPicture Index, DOUBLEEXPONFUNCTION
                frmLeftTopImages.imgGridTemp(Index).Tag = Number
            Case "RightTop"
                DrawSavedRightTopPicture Index, DOUBLEEXPONFUNCTION
                frmRightTopImages.imgGridTemp(Index).Tag = Number
            Case Else
                MsgBox "Error, the position has not been saved"
                Exit Sub
        End Select

        If Exists = True Then
            DblExpFncs(DblExpFncCount).NodeName = CctTable![Node Name]
            DblExpFncs(DblExpFncCount).StartTime = CctTable![Start Time]
            DblExpFncs(DblExpFncCount).StopTime = CctTable![Stop Time]
            DblExpFncs(DblExpFncCount).A = CctTable!A
            DblExpFncs(DblExpFncCount).B = CctTable!B
            DblExpFncs(DblExpFncCount).C = CctTable!C
            DblExpFncs(DblExpFncCount).FunctionType = CctTable![Function Type]
        End If

        CctTable.MoveNext
        If CctTable.EOF Then
            LoopEnd = True
        End If

        Loop Until LoopEnd = True
    End If

End Sub
```

- Write a method which writes the circuit to the data card:

```
Sub DblExpFncWriteCircuit (Position As String, i As Integer)

    Dim FileName As String
    Dim Number As Integer
    Dim fhandle As Integer

    Select Case Position
       Case "Main"
          Number = frmATP.imgGrid(i).Tag
       Case "Top"
          Number = frmTopImages.imgGridTemp(i).Tag
       Case "Bottom"
          Number = frmBottomImages.imgGridTemp(i).Tag
       Case "Left"
          Number = frmLeftImages.imgGridTemp(i).Tag
       Case "Right"
          Number = frmRightImages.imgGridTemp(i).Tag
       Case "LeftTop"
          Number = frmLeftTopImages.imgGridTemp(i).Tag
       Case "LeftBottom"
          Number = frmLeftBottomImages.imgGridTemp(i).Tag
       Case "RightTop"
          Number = frmRightTopImages.imgGridTemp(i).Tag
       Case "RightBottom"
          Number = frmRightBottomImages.imgGridTemp(i).Tag
    End Select

    Dim NodeNameLength As Integer
    Dim StartTimeLength As Integer
    Dim StopTimeLength As Integer
    Dim ALength As Integer
    Dim BLength As Integer
    Dim CLength As Integer
    Dim FunctionTypeLength As Integer

    If Not Number = -1 Then
       If DblExpFncs(Number).FunctionType = 1 Then
          DblExpFncs(Number).FunctionType = -1
       End If
       DblExpFncs(Number).StartTime = ConvertString(DblExpFncs(Number).StartTime)
       DblExpFncs(Number).StopTime = ConvertString(DblExpFncs(Number).StopTime)
       DblExpFncs(Number).A = ConvertString(DblExpFncs(Number).A)
       DblExpFncs(Number).B = ConvertString(DblExpFncs(Number).B)
       DblExpFncs(Number).C = ConvertString(DblExpFncs(Number).C)

       NodeNameLength = Len(DblExpFncs(Number).NodeName)
       StartTimeLength = Len(DblExpFncs(Number).StartTime)
       StopTimeLength = Len(DblExpFncs(Number).StopTime)
       ALength = Len(DblExpFncs(Number).A)
       BLength = Len(DblExpFncs(Number).B)
       CLength = Len(DblExpFncs(Number).C)
       FunctionTypeLength = Len(DblExpFncs(Number).FunctionType)

       fhandle = FreeFile
       Open ActiveDataFileName For Append As fhandle
       'Print #fhandle,
       Print #fhandle, "15" + DblExpFncs(Number).NodeName; Spc(6 - NodeNameLength + 2 - FunctionTypeLength);
DblExpFncs(Number).FunctionType; Spc(9 - ALength); DblExpFncs(Number).A; Spc(10 - BLength);
```

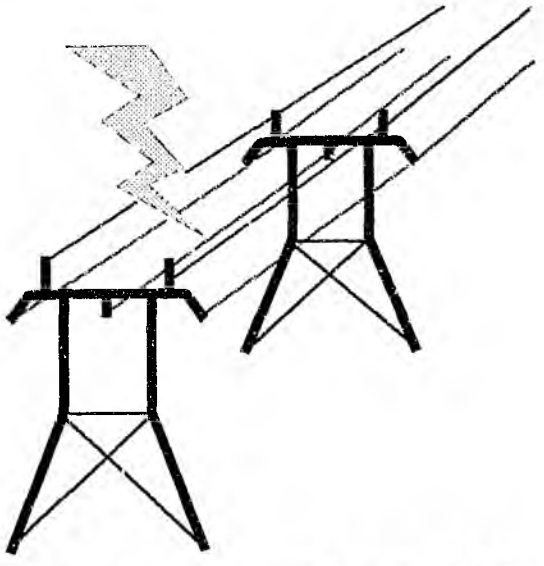
```
DblExpFncs(Number).B; Spc(10 - CLength); DblExpFncs(Number).C; Spc(39 - StartTimeLength);
DblExpFncs(Number).StartTime; Spc(10 - StopTimeLength); DblExpFncs(Number).StopTime
Close #fhandle    ' Close file.


  End If
  If DblExpFncs(Number).FunctionType = -1 Then
     DblExpFncs(Number).FunctionType = 1
  End If


End Sub
```

- Write a method which writes the node names used by this element to the Setup Simulation screen:

```
Sub DblExpFncWriteNodeNames (Number As Integer)
   frmSetupSimulation.lstAvailableNodes.AddItem (DblExpFncs(Number).NodeName)
End Sub
```

Volume 1 - Appendix B

ATP FOR WINDOWS



USER REFERENCE MANUAL

ACKNOWLEDGEMENTS

I would like to acknowledge that extensive use was made of 'AN INTRODUCTION TO EMTP (ELECTROMAGNETIC TRANSIENTS PROGRAM)' by J.M. VAN COLLER (1993) for many of the descriptions of the elements used in ATP, as well as the entries required for these elements.Much help on ATP was received from John Van Coller, and for this I am also very grateful.

DOCUMENT STATUS CONTROL

Configuration Control

Project / Owner:	Tamir Orbach
Title:	*ATP for Windows*
Doc. Ref. Number:	ATP001
Version Number:	2.0
Version Release Date:	16 June 1995
Author:	Tamir Orbach
Location and Filename:	C:\OFFICE\WPWIN\WPDOCS\ URM.001

Chapter Status Control

Chapter	Revision	Date
1	2.0	16/06/95
2	2.0	16/06/95
3	2.0	16/06/95
4	2.0	16/06/95
5	2.0	16/06/95
6	2.0	16/06/95
7	2.0	16/06/95
8	2.0	16/06/95
9	2.0	16/06/95
10	2.0	16/06/95

Document Change History

Document Change History

DCR	Date	Chapter	Page	Change Description / Reason	Old Rev	New Rev
ATP for Windows	18/10/94	All	All	More Detail Needed	0.1	0.2
ATP for Windows	18/01/95	All	All	Screen Dumps from program	0.2	1.0
ATP for Windows	16/06/95	All	All	Program Finished, many changes	1.0	2.0

TABLE OF CONTENTS

1 INTRODUCTION 1-1
1.1 Background 1-1
1.2 Features 1-1
1.2.1 CAD Input Interface 1-1
1.2.2 Compatibility with other CAD packages 1-2
1.2.3 Graphical Solution Visualisation 1-2
1.2.4 Limitations 1-2
1.3 Hardware Requirements 1-3
1.3.1 Processor Type 1-3
1.3.2 Hard Disk Space 1-3
1.3.3 Display 1-3
1.3.4 Mouse 1-3
1.3.5 Printers 1-3
1.3.6 Operating System 1-3
1.4 Gettmg Started 1-4
1.4.1 Installation 1-4
1.5 User Interface Terminology 1-4
1.6 Application Window 1-4
1.6.1 Main Menu 1-4
1.6.2 The Control Panel / Toolbar 1-4
1.6.3 The Circuit Region 1-5
1.6.4 The Circuit Element Panel 1-5
1.6.5 The Mini-Circuit Map 1-6

2 FILE 2-1
2.1 Introduction 2-1
2.2 How To Use It 2-1
2.3 Remarks 2-1
2.4 New 2-2
2.5 Open 2-3
2.6 Save 2-4
2.7 Save As 2-6
2.8 Import 2-7
2.9 Print 2-8
2.10 Exit 2-9

3 EDIT 3-1
3.1 Introduction 3-1
3.2 How To Use It 3-1

3.3 Remarks 3-1
3.4 Cut 3-1
3.5 Copy 3-3
3.6 Paste 3-3
3.7 Block 3-4
3.8 Select All 3-5
3.9 Unselect All 3-5
3.10 BlackBox Circuit 3-6
3.11 Unblackbox Circuit 3-6

4 VIEW 4-1
4.1 Introduction 4-1
4.2 How To Use It 4-2
4.3 Show/Hide Control Panel 4-2
4.4 Edit Circuit Panel 4-2
4.5 Show / Hide Grid 4-4
4.6 Rotate Element 4-4

5 RUN 5-1
5.1 Introduction 5-1
5.2 How To Use It 5-1
5.3 Remarks 5-1
5.4 Select Element 5-1
5.4.1 Function 5-1
5.4.2 How To Use It 5-2
5.4.2.1 The Simple Sources and Functions Library 5-3
5.4.2.2 The Impedance Elements & Surge Arrestors Library 5-4
5.4.2.3 The Switches and Diodes & SCRs Library 5-5
5.4.2.4 The Transformer Library 5-7
5.4.2.5 The TACS Element Library 5-7
5 4.2.6 The TACS Source Library 5-8
5.4.2.7 The Transmission Line Library 5-9
5.5 Setup Simulation 5-10
5.6 Run Simulation 5-13
5.7 Read Output 5-14

6 HELP 6-1
6.2 How To Use It 6-1
6.3 Remarks 6-1
6.4 Help Contents 6-1
6.5 Name Search 6-3

6.6 Tutorial 6-4

7 ELEMENT DETAILS 7-1
7.1 How To Use It 7-1
7.2 Sources 7-1
7.2.1 The Step Function 7-1
7.2.2 The Ramp Function 7-3
7.2.3 The Dual-slope Ramp Function 7-4
7.2.4 The Double Exponential Function 7-5
7.2.5 The Sinusoidal Source 7-7
7.2.6 The User Defined Voltage or Current Source 7-8
7.3 Branch Card Type Elements 7-10
7.3.1 The Simple RLC Branch 7-10
7.3.2 Distributed Parameter RLC Branch / Single Transmission Line Conductor 7-13
7.3.3 The Piecewise-linear model Surge Arrestor 7-15
7.3.4 The Exponential Model Surge Arrestor 7-18
7.3.5 Transformers 7-22
7.3.6 The Three Phase Transformer 7-26
7.3.7 Black Box Element 7-29
7.4 Switch Card Elements 7-30
7.4.1 Switches 7-30
7.4.1.1 The Standard Switch (Definite instant - switch operation) 7-33
7.4.1.2 Closing Switch (Random instant - switch operation -> closing only 7-35
7.4.1.3 Opening Switch (Random instant - switch operation -> opening only) 7-36
7.4.1.4 The Slave Closing Switch 7-37
7.4.1.5 The Slave Opening Switch 7-40
7.4.1.6 Systematically increasing instant of operation -> closing switch 7-41
7.4.1.7 The Constant Delay Slave Switch 7-42
7.4.1.8 The Diode/SCR Element 7-44

8 USING TACS (TRANSIENT ANALYSIS OF CONTROL SYSTEMS) ELEMENTS
8.1 How To Use It 8-1
8.2 Creating a FORTRAN relationship TACS element 8-3
8.3 Creating a TACS element which uses control topologies 8-6
8.4 Creating a TACS element which is a special device 8-10
8.4.3 The Relay-operated Switch 8-11
8.4.4 The Level-triggered Switch 8-14

8.4.5 The Controlled Integrator 8-16
8.4.6 The Simple Differentiator 8-19
8.5 Creating a TACS controlled component 8-21
8.5.3 The TACS controlled resistor 8-22
8.5.4 The TACS controlled switch 8-22

9 USING TACS (TRANSIENT ANALYSIS OF CONTROL SYSTEMS) SOURCES AND WAVEFORMS 9-1
9.1 How To Use It 9-1
9.2 Creating a DC level or single pulse 9-2
9.3 Creating a Sinusoidal Wave Generator 9-4
9.4 Creating a Repetitive Pulse 9-6
9.5 Creating a Repetitive Ramp 9-8
9.6 Creating a TACS controlled voltage or current source 9-10

10 MODELLING TRANSMISSION LINES ON ATP FOR WINDOWS 10-1
0.1 How To Use It 10-1

1 INTRODUCTION

1.1 Background

ATP or the Alternative Transients Program is a computer program written in Fortran, which is capable of simulating transient conditions and phenomena in power machinery and distribution networks. **ATP** can solve any network consisting of interconnections of resistances, inductances, capacitances, single and multiphase π circuits, distributed (or lumped) parameter lines, and certain other elements (Dommel 1986:1-1).

ATP was developed in the 1960's, and at the time made use of data cards to input information. The present version of **ATP** makes use of data files to input data. **ATP for Windows** is a pre-processor to **ATP** which allows the user to enter data graphically and numerically on the Windows platform, using a menu driven, and object-oriented graphical interface.

1.2 Features

1.2.1 CAD Input Interface

The primary feature of **ATP for Windows** is that it has a **CAD** (Computer Aided Drawing) style, user-friendly graphical interface which allows the user to define transient analysis problems quickly and easily. The interface is menu driven and adheres to the **IBM CUA** (Common User Access) the Microsoft Application Design Guide and the general Windows environment.

1.2.2 Compatibility with other CAD packages

ATP for Windows has the potential to interface with any CAD package which is capable of converting graphical input and numerical information to **Microsoft Access** files. This means that any circuit geometry which has been drawn in Intergraph can be used directly in **ATP for Windows** without the user having to redraw the circuit if the mdl code is written in Intergraph to enable the storage of circuit information in MS Access tables.

1.2.3 Graphical Solution Visualisation

ATP for Windows is capable of displaying the **ATP** output graphically and/or numerically.

1.2.4 Limitations

ATP for Windows requires Microsoft DOS 5.0 (or better) and **Microsoft Windows 3.1** (or better) in order to run. **ATP for Windows** can also be run on operating systems which are compatible with MS Windows 3. such as OS/2 2.1, Windows NT, Windows 4.0 (Chicago) etc.

ATP for Windows will not have all of the ATP elements available for use initially. These are planned for future work in **ATP for Windows.** Some functions listed in this User Reference Manual may not be available at the time of first issue of ATP for Windows.

1.3 Hardware Requirements

1.3.1 Processor Type

ATP for Windows requires the use of an 80486 processor with a minimum of 4Mb of RAM (although it is recommended that you use 8Mb of RAM). For large problems, 4Mb of RAM on a 486 DX may not be sufficient, and it is recommended that if large simulations are anticipated at the time of purchasing of the hardware, a Pentium processor with a minimum of 8Mb RAM is purchased.

1.3.2 Hard Disk Space

At least 2 Megabytes of hard disk space is required for the executable code of the **ATP for Windows** pre-processor, although this does not include the space required for the **ATP** program itself, or the space required for MS Windows.

1.3.3 Display

ATP for Windows requires at least a Super VGA display card and corresponding monitor since colours are used extensively in the interface.

1.3.4 Mouse

A mouse or trackball is required to use **ATP for Windows.** The mouse used must be compatible with Windows 3.1, and it is recommended that a high resolution mouse (800 dpi or greater) be used.

1.3.5 Printers

All printers which are supported by Windows 3.1 are supported by **ATP for Windows.**

1.3.6 Operating System

ATP for Windows runs under the Windows 3.1 (or better) platform as well as under the MS DOS 5.0 (or better) platform. Earlier versions of DOS should work if they support MS Windows 3.1, and operating systems which support Windows 3.1, such as OS/2 2.1 and Windows NT can also be used.

1.4 Getting Started

1.4.1 Installation

- Load MS Windows.
- From Windows select the File- Run option, and run a:\setup.exe
- This will automatically create a sub-directory called ATPWIN, and install the pre-processor / interface

1.5 User Interface Terminology

ATP for Windows makes use of the standard Windows 3.1 interface, and it is assumed that the user is familiar with Windows and typical Graphical User Interface functions. The meaning of terms specific to Windows such as icon or dialogue box can be found in the Windows 3.1 User Manual.

1.6 Application Window

The **ATP for Windows** main window, which shows the menu layout, can be seen in Figure 1.1. This window is divided into the following sections:

1.6.1 Main Menu

The main menu provides access to all the functions offered by **ATP for Windows** Functions that are used often can also be accessed by the **Control Panel / Toolbar** (see Section 1.6.2). The entire menu layout showing all sub-menus can be seen in Appendix A - The ATP Menu Layout.

1.6.2 The Control Panel / Toolbar

The Control Panel / Toolbar is a panel of speed buttons which allow the user to perform any function without accessing that function on the menu. The user has the option of not seeing the control panel at all (*View - Show / Hide Control Panel*). Each button is a graphical representation of the function which it performs.

1.6.3 The Circuit Region

This region contains the main problem definition and solution display area. All circuits are drawn in this area, and numerical as well as graphical solutions are displayed in this area. A drawing grid is displayed during the input of circuits for simulation.

1.6.4 The Circuit Element Panel

The Circuit Element Panel is a panel of speed buttons (or a tool bar) which allow the user to

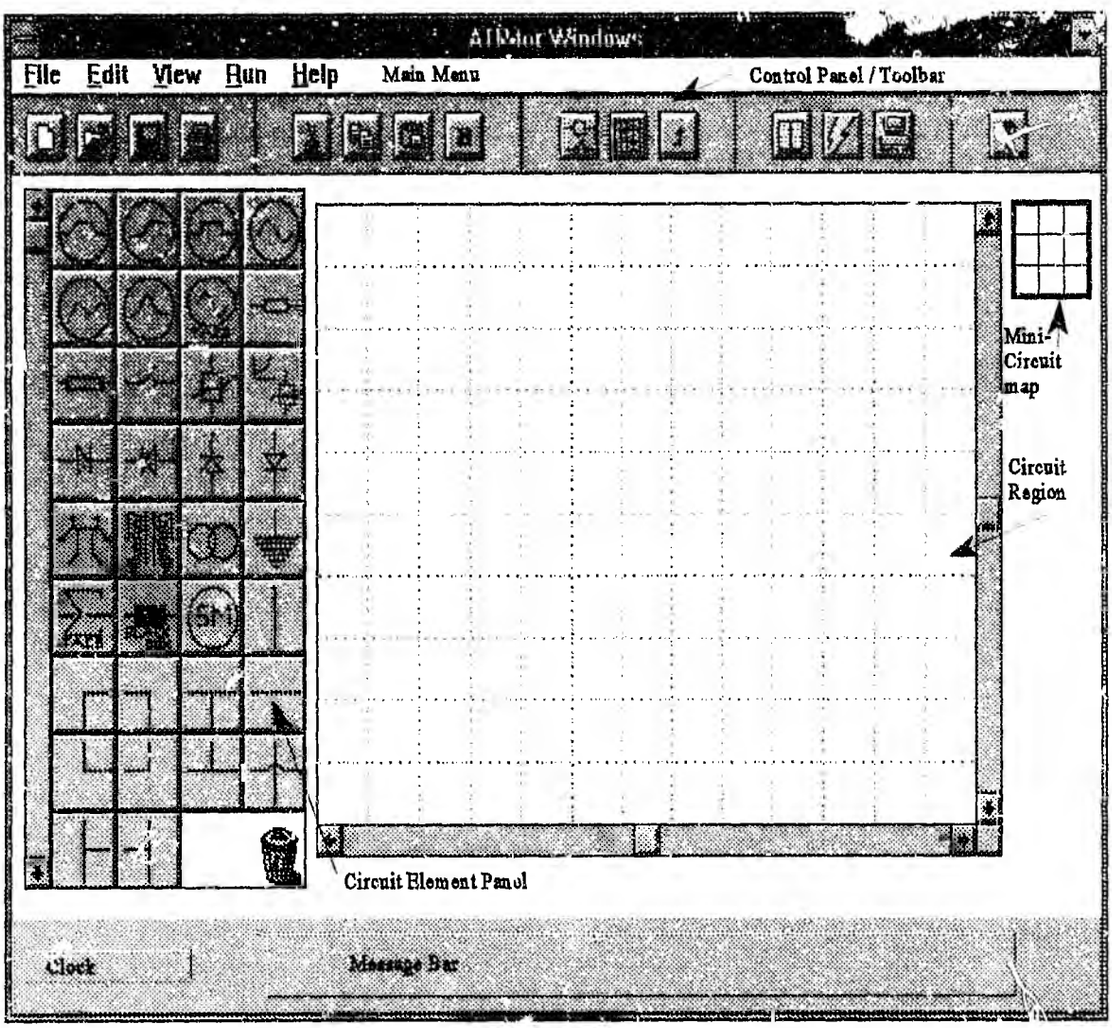


Figure 1.1 - The Application Window

select a circuit element which the user would like to place in the circuit. Each button is a graphical representation of the circuit element it represents, and a description of each element is given on the Message Bar as the user traverses over the element with the mouse. To draw a circuit, click on the element you require using the left mouse button. Then drag the element icon to the required grid position on the visible circuit region, and release the mouse button. This will drop the element in the visible circuit region. To see a description of the element details corresponding to this element, click on the element with the right mouse button once the element is in the actual visible circuit region (see Chapter 7).

1.6.5 The Mini-Circuit Map

The Mini-Circuit Map is used to indicate where in the entire circuit region, circuit elements exist in the present simulation. Since only a small part of the screen is actually visible at one time, the mini circuit map indicates where any circuit elements exist with respect to the current position. The current position is always the Center of the Mini-Circuit Map. To show that circuit elements exist in different sections of the circuit region, a cross appears in the mini-map area corresponding to all the areas in which circuit elements exist.

2 FILE

2.1 Introduction

The File option provides a pull down menu of commands related to file operations. These include:

- Commands to create a new simulation (by opening a *problem definition* file), open an existing simulation. and save an existing simulation (*problem definition file*).
- A command to print the current output graph or problem circuit.
- A command to exit **ATP for Windows.**

The *File* sub-menu is shown in Figure 2.1.

2.2 How To Use It

To bring down the file menu options, select *File* from the main menu with the mouse, or type <ALT>-F.

2.3 Remarks

Files containing the particulars of a problem are termed *ATP* files, and have an extension **ATP.** The geometry of a problem as well as the properties of every element in the circuit are stored in an ATP file, which is an MS Access compatible database file.

The solution of a problem is stored in a file of the same name as the problem definition file, but with the extension **OUT.** This file is the output file from **ATP,** and it is not changed for the purpose of the **ATP for Windows** pre-processor. The user does not have access to the **OUT** files from within **ATP for Windows,** but the information in these files can be displayed numerically or graphically within **ATP for Windows.** The information in the **.ATP** files is converted to **ATP** data files with the extension **DAT** when the command is given to start the simulation, and the user, again, does not have access to the input data file from within the **ATP for Windows** pre-processor.

2.4 New

2.4.1 Function

The *New* command creates a new .ATP file and replaces the current problem.

2.4.2 How To Use It

Select the *New* command from the *File* sub-menu.

2.4.3 Remarks

If any changes to the current problem have not been saved, the user will be queried as to whether or not the problem should be saved before a new problem is created. If the answer is **YES** and the current problem has not been named yet, a *Save As* dialogue box will appear, prompting the user for a name (see section 2.7 - Save As).

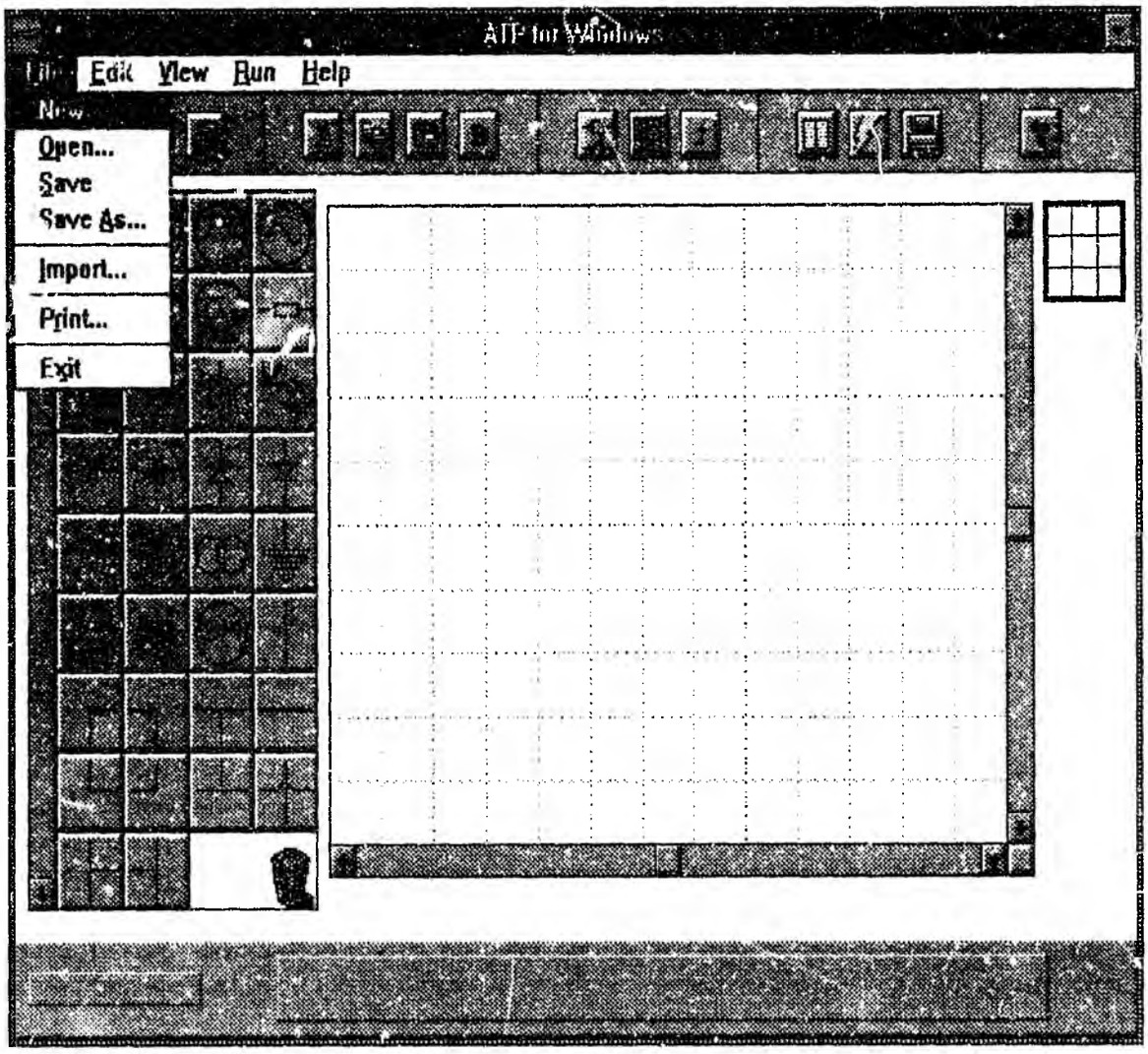


Figure 2.1 - The File Sub-menu

2.5 Open

2.5.1 Function

The *Open* command loads an existing **.ATP** (ATP for Windows) file from disk.

2.5.2 How To Use It

Select the *Open* command from the *File* sub-menu. An *Open File* dialogue box will appear (see

Figure 2.2) prompting the user for the name of the file to open. All the files in the current directory with the extension **ATP** will be shown. Typing a directory followed by the *.atp extension will list all the atp files in that directory. The default directory is C:\ATPWIN\FILES*.atp.

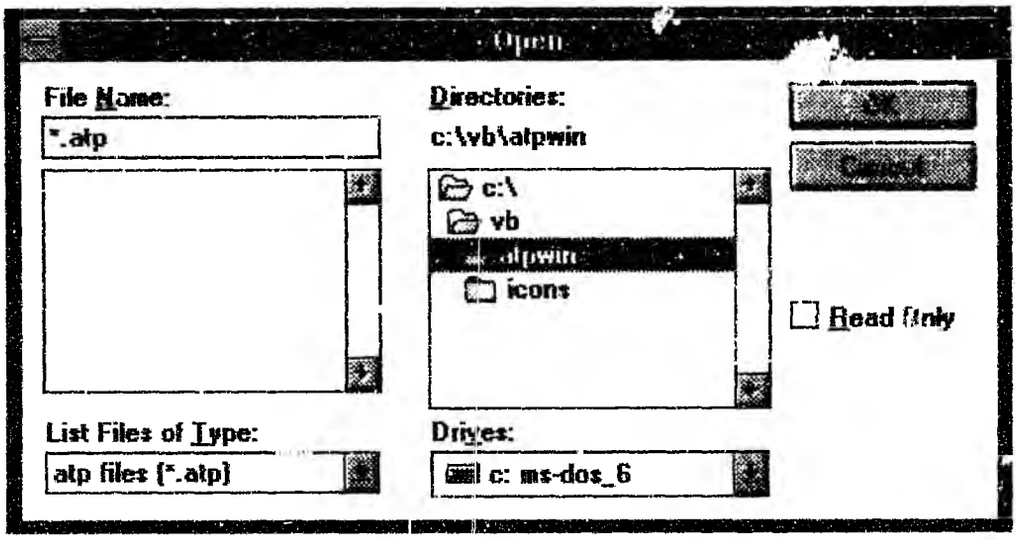


Figure 2.2 - The Open File Dialogue Box

2.5.3 Remarks

If any changes to the current problem have not been saved, the user will be queried as to whether or not the problem should be saved before a new problem is created. If the answer is **YES** and the current problem has not been named yet, a *Save As* dialogue box will appear, prompting the user for a name. Once the problem has been loaded, the corresponding circuit will be shown in the *Circuit Region* of the main window.

2.6 Save

2.6.1 Function

The *Save* function allows the user to save to disk the problem which is currently open.

2.5.2 How To Use It

Select the *Save* menu option from the *File* sub-menu. This will produce the Save screen (see figure 2.3)

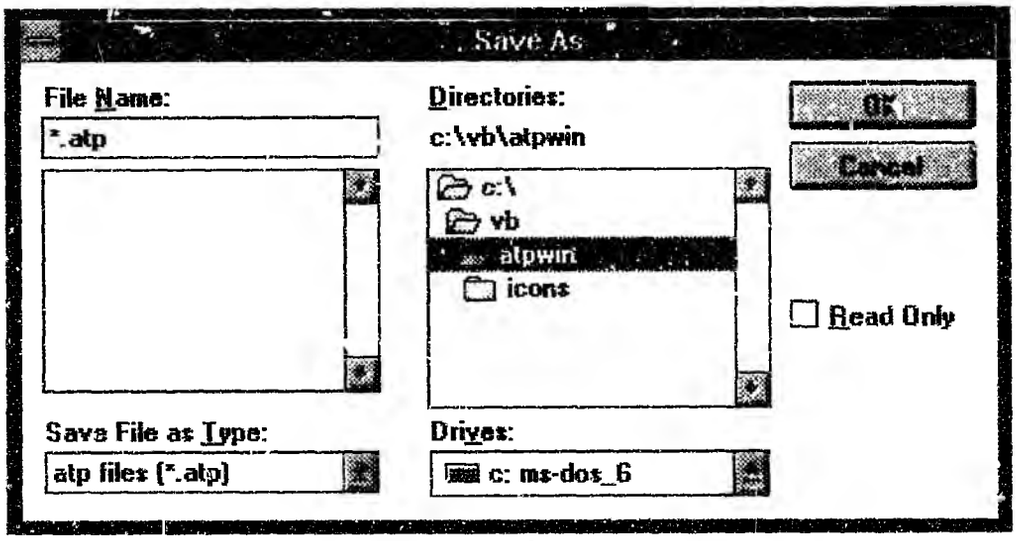


Figure 2.3 - The Save File Dialogue Box

2.6.3 Remarks

If the current problem is a new problem which has not been previously saved, a Save As dialogue box will be opened, allowing the user to name the problem before saving it. Note that the chosen file name does not have to have the extension **ATP** (See section 2.7- Save As).

2.6.4 Error Messages

There are three possible errors which can occur when saving a problem definition:

- **Invalid filename**: If the filename entered in the filename edit control of the *Save* dialogue box is invalid, a beep is sounded , and the *Save* dialogue box is returned to. The invalid filename is highlighted.
- **I/O Error**: This error is caused by a general write error which is often due to the disk drive not being closed. When this error occurs, the user is given the choice of cancelling or retrying the operation.
- **Insufficient Disk Space**: If there is insufficient disk space to save the file, a message is displayed, and the operation is aborted.

2.7 Save As

2 7.1 Function

The *Save As* command allows the user to save the problem currently in the work area to be saved under a name other than that already allocated to this problem.

2.7.2 How To Use It

Select the *Save As* option from the *File* sub-menu. The *Save As* dialogue box will be displayed (see figure 2.3 - The Save File Dialogue Box) allowing the entering of a new file name.

2.7.3 Remarks

Once the user has selected a new name for the problem file, the old name is replaced, and the problem can then only be accessed by it's new name. If the name selected corresponds to the name of an existing file, the user is queried as to whether or not this file should be overwritten. If the **NO** option is selected, the operation is aborted.

2.7.4 Error Messages

There are three possible errors which can occur when saving a problem definition:

- **Invalid filename:** If the filename entered in the filename edit control of the *Save As* dialogue box is invalid, a beep is sounded , and the Save As dialogue box is returned to. The invalid filename is highlighted.
- **I/O Error:** This error is caused by a general write error which is often due to the disk drive not being closed. When this error occurs, the user is given the choice of cancelling or retrying the operation.
- **Insufficient Disk Space:** If there is insufficient disk space to save the file, a message is displayed, and the operation is aborted.

2.8 Import

2.8.1 Function

The *Import* command allows files in the Drawing Interchange format (**DXF**) to be read, and converted to the internal data file format used by **ATP**. This command is not available at this stage.

2.8.2 How To Use It

Select the *Import* option from the *File* sub-menu. The *Import* dialogue box will be displayed (see figure 2.5 - The Import Dialogue Box). If an invalid file is selected, an error will occur informing the user that this has happened.

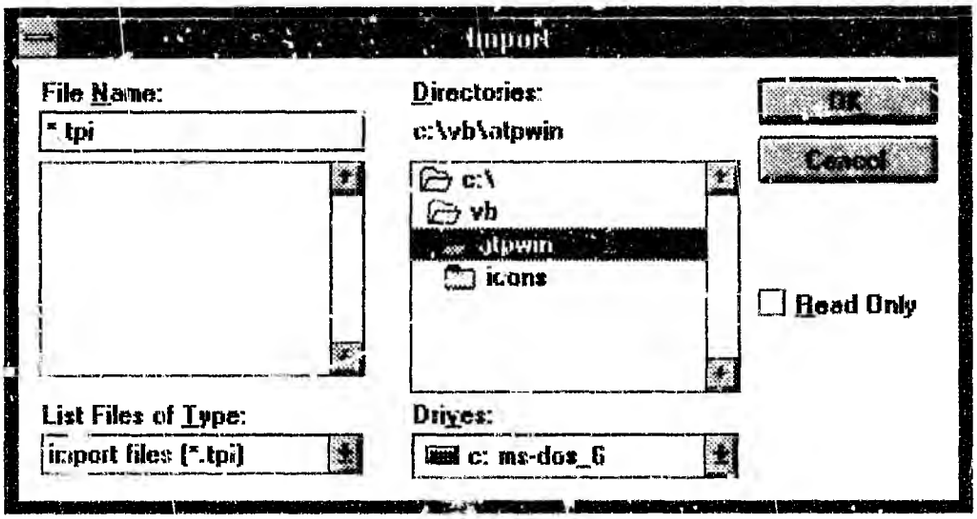


Figure 2.4 - The Import Dialogue Box

To import a file, select the file name and file type, and then select the *OK* button.

2.8.3 Remarks

Since not all of the information required can be described in the presently available **CAD** packages, it may be necessary for the user to add some information to the circuit parameters once the circuit has been converted from **DXF** format to the **ATP** data file format. The extent

of the information required depends on the number of attributes of the problem described in the CAD package used.

2.8.4 Limitations

The **DXF** import facility is not currently available

2.9 Print

2.9.1 Function

The *Print* command allows the user to select the current printer required, as well as to set up parameters for the current printer or plotter, and then sends the contents of the Circuit Region to the currently selected hardcopy device. It is also possible to print output from a simulation.

2.9.2 How To Use It

Select the *Print* option from the *File* sub-menu. The *Print* dialogue box, will be displayed (see figure 2.6 - The Print Screen). To Abort the printing process, select the *Abort* button witnin the Print dialogue box.

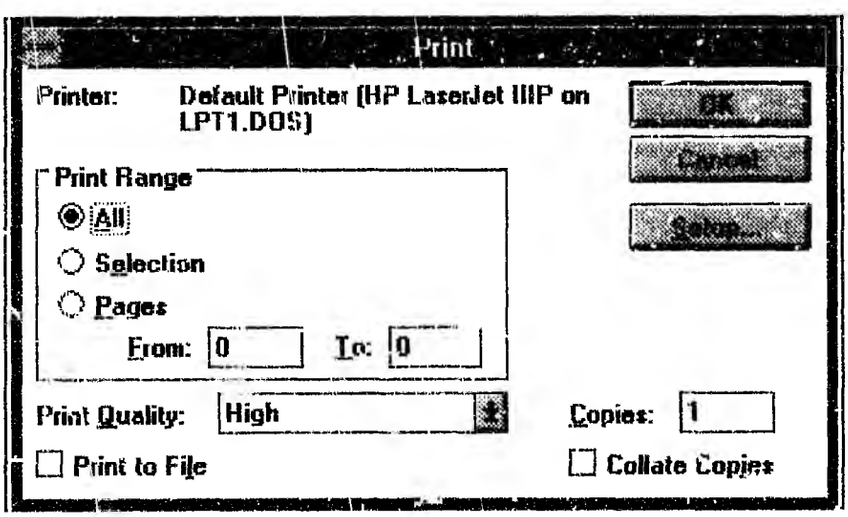


Figure 2.6 - The Print Screen

To print a circuit, select the name of the circuit and the number of copies, and then select the

OK button.

2.9.3 Remarks

The changes made to the printer setup are not permanent, and only apply to the instance of **ATP for Windows**. The printer setup does not get saved with the current problem definition.

2.9.4 Limitations

Only printers which support graphics can be used. Line printers or daisy-wheel printers are not supported.

2.9.5 Error Messages

The following errors can occur when the Print command is invoked:

- **No Primary Hardcopy Device**. This error occurs if a hardcopy device has not been registered with Windows. Install the necessary device driver using the Windows Control Panel.
- **Graphics Not Supported**. If the printer being used does not support graphics, an error message is displayed and the operation is aborted.

2.10 Exit

2.10.1 Function

The *Exit* command exits **ATP for Windows.**

2.10.2 How To Use It

Select the *Exit* option from the *File* sub-menu. If the most recent changes to the problem have not been saved, the user is queried as to whether or not this should be done.

2.10.3 Remarks

Windows Profile Files (files with the INI extension) are used to store configuration information for Windows applications.

3 EDIT

3.1 Introduction

The *Edit* menu option pulls down the commands related to object editing:

- Commands to cut, copy and paste objects in the current problem definition.
- Commands to select and unselect all objects in the current problem definition.

The *Edit* sub-menu can be seen in **figure 3.1**

3.2 How To Use It

To bring down the edit menu options, select *Edit* from the main menu with the mouse, or type <ALT>-E.

3.3 Remarks

In order to edit the characteristics of objects (circuit elements or others), the objects need to be selected. This is done using the by holding the left mouse button down and at the same time, pressing the <SHIFT> key while moving across the required element(s).

3.4 Cut

3.4.1 Function

The Cut function is used to copy selected objects to the clipboard and delete them from the circuit while keeping them in the clipboard for future use.

3.4.2 How To Use It

Ensure that all the objects you would like to cut have been selected using the *Block* command or using the mouse and the shift button (see section 3.7). Select the *Cut* option from the *Edit* sub menu by using the mouse or pressing <ALT>-C, or select the Cut toolbar option from the Control Panel / Toolbar (scissors).

3.4.3 Remarks

When an object is selected from the Circuit Region, the object as well as it's characteristics (which have been entered using the *Element Details Menu* - See section 7) are stored in the clipboard (*this is not the case with the Copy command*).

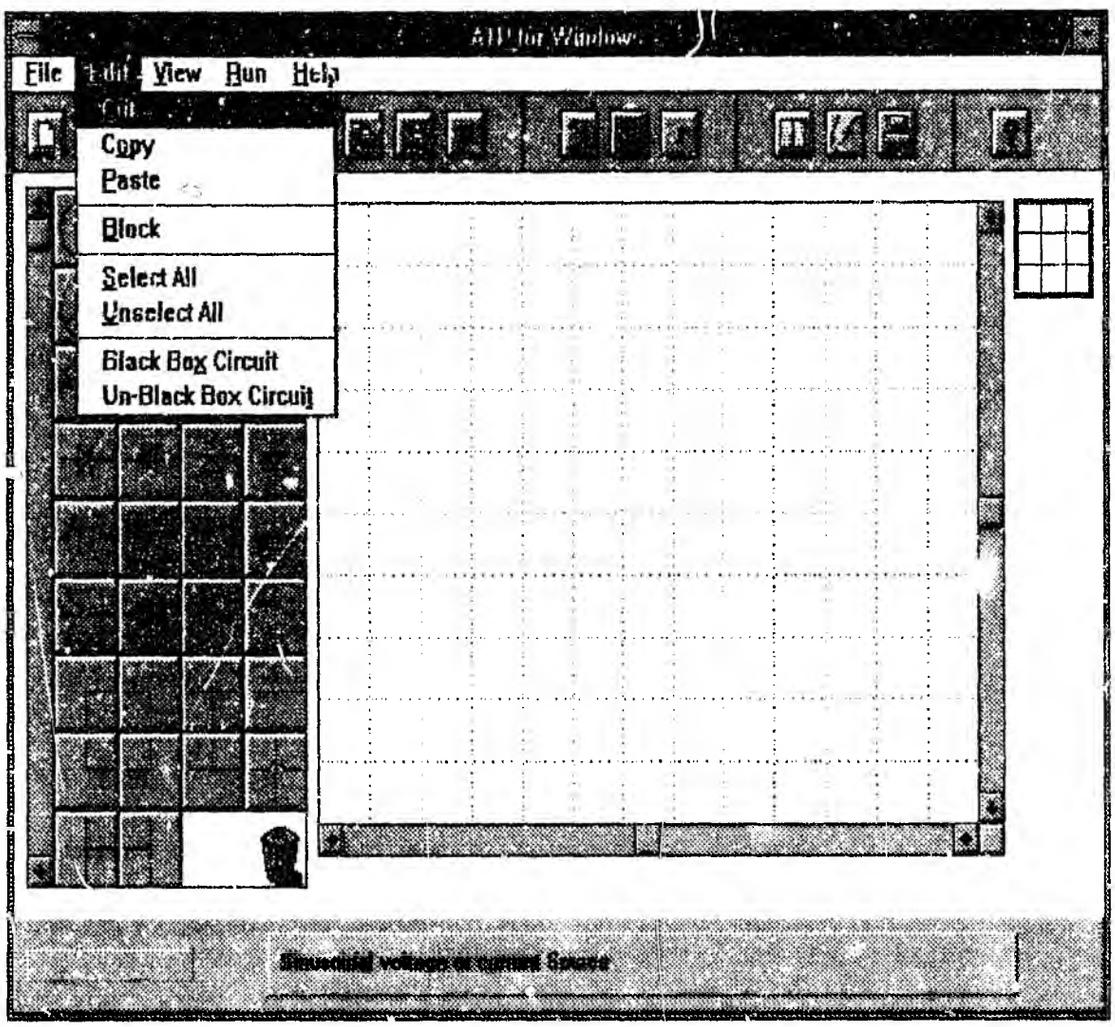


Figure 3.1 - The Edit Sub-menu

3.4.4 Error Messages

The following errors may occur when attempting to Cut objects:

- **Insufficient Memory.** If Windows is low on memory, there may not be enough memory to carry out the operation. In this case an appropriate message is displayed.

3.5 Copy

3.5.1 Function

The Copy command allows selected objects to be copied to the clipboard while leaving the selected objects completely intact in the Circuit Region.

3.5.2 How To Use It

Ensure that all the objects you would like to copy have been selected using the *Block* command or using the mouse and the shift button (see section 3.7). Select the *Copy* option from the *Edit* sub-menu by using the mouse or pressing <ALT>-O, or select the Copy toolbar option from the Control Panel / Toolbar.

3.5.3 Remarks

When an object is selected from the Circuit Region, the object only is stored in the clipboard (and not the element details of this object - although this is planned for future versions of ATP for Windows).

3.5.4 Error Messages

The following errors may occur when attempting to copy objects:

- **Insufficient Memory.** If Windows is low on memory, there may not be enough memory to carry out the operation. In this case an appropriate message is displayed.

3.6 Paste

3.6.1 Function

The Paste command allows objects which have previously been stored on the clipboard to

be pasted anywhere in the Circuit Region and into the current problem.

3.6.2 How To Use It

Select the point in the current problem Circuit Region where you would like to paste the object. This can be done by double clicking the mouse when the mouse pointer is in the required place. The particular position which you have selected will be highlighted. To unselect this, click once anywhere in the circuit region, or double click on the same spot again.

Now select the *Paste* option from the *Edit* sub-menu. If no valid **ATP for Windows** objects are present in the clipboard, the *Paste* menu option will be unavailable.

3.6.3 Remarks

Objects that were cut or copied to the clipboard from other applications can not be pasted in the ATP Circuit Region.

3.7 Block

3.7.1 Function

The Block function is used to select one or more objects from the Circuit Region.

3.7.2 How To Use It

Keep the <SHIFT> button selected, click on the mouse at the left top corner of the section you would like to block, and drag the mouse down and to the right while keeping the mouse button as well as the <SHIFT> key pressed down. To stop blocking, release the <SHIFT> key and the mouse button The selection will be highlighted. Once all the objects you would like to select have been selected, choose the *Edit* option you require. To unselect an object, select the *Unselect All* option from the *Edit* menu, or click on a blank area of the screen with the left mouse button.

3.7.3 Remarks

The *Block* command is not available in menu form and works only using the mouse.

3.8 Select All

3.8.1 Function

This can be used to select all the objects in the (visible) Circuit Region at once. This may be useful if the complete visible circuit is to be stored on the clipboard or moved.

3.8.2 How To Use It

Once the *Select All* function has been selected, the complete circuit in the Circuit Region will be highlighted by a black dotted frame. Using the mouse, move accross the circuit region by scrolling, and paste the circuit wherever you would like to paste it by moving the mouse pointer to the position on which you would like to have the upper left corner of the circuit as it stands, double clicking on the left button, and selecting the *Edit - Paste* option. A circuit can not be pasted over an existing element, or if any part of the circuit will not fall into the visible circuit region.

3.8.3 Remarks

Once the Select All function has been chosen, it can be cancelled by either using the Unselect All command or by clicking on a blank area of the screen.

3.9 Unselect All

3.9.1 Function

This is used to unselect all of the objects which have been selected by either the *Select All* command or the Block (using the mouse) commands.

3.9.2 How To Use It

Choose the *Unselect All* option from the *Edit* sub-menu.

3.10 BlackBox Circuit

3.10.1 Function

The BlackBox Circuit option is used to modularise a section of the circuit into one black box element. This function will be available in ATP for Windows ver 1.5 (should be available by December 1995). This function will also allow the user to place the new Black Box into the Circuit Panel, and draw an icon for it.

3.11 Unblackbox Circuit

3.11.1 Function

The Unblackbox Circuit option is used to expand a circuit which has been modularised using the BlackBox Circuit function.

4 VIEW

4.1 Introduction

The *View* menu option pulls down the commands related to the viewing of the screen.

- Commands to show and hide the speed button Control Panel / Toolbar.
- Commands to edit the Circuit Panel, and
- commands to rotate a circuit element.

The *View* sub-menu can be seen in **figure 4.1**.

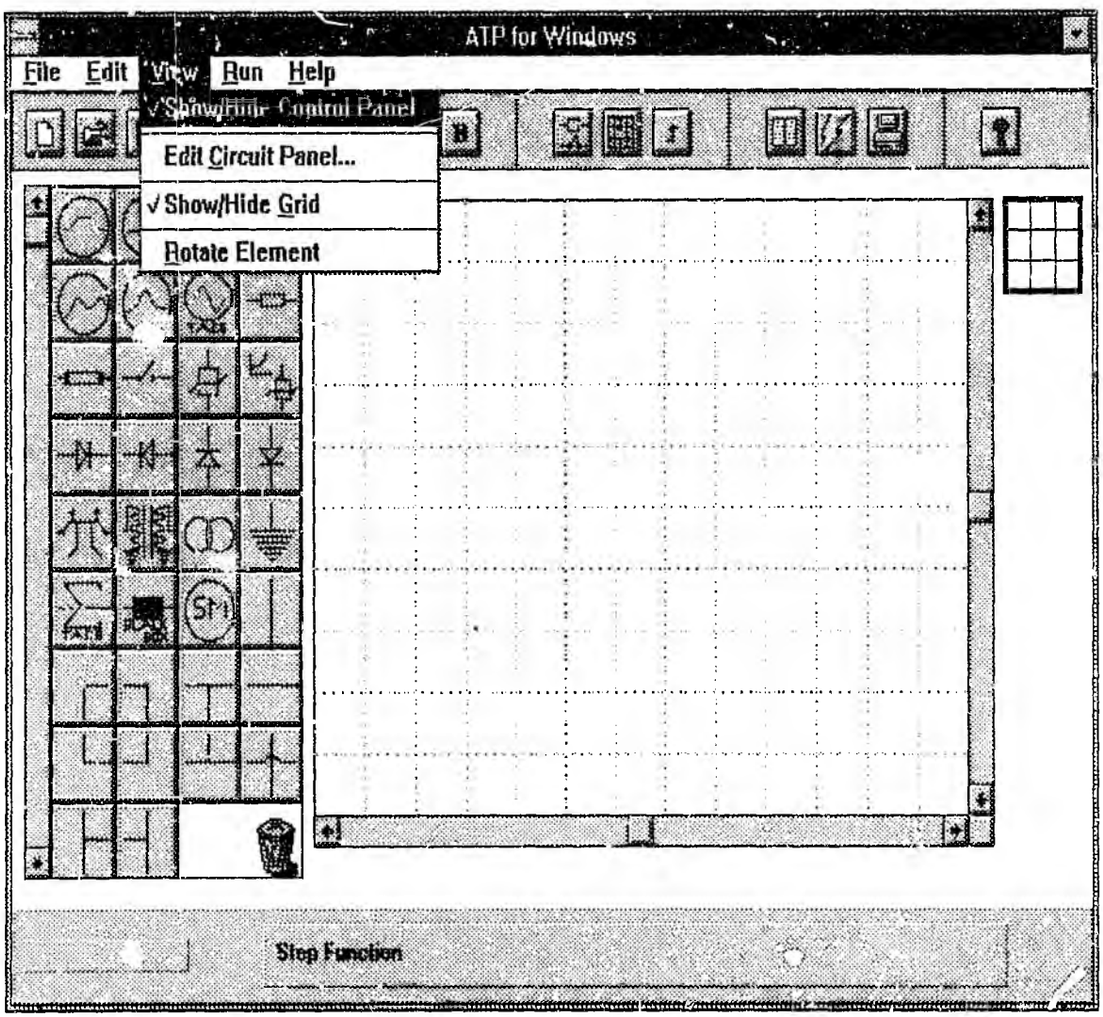


Figure 4.1 - The View Sub-menu

4 VIEW

4.1 Introduction

The *View* menu option pulls down the commands related to the viewing of the screen:

- Commands to show and hide the speed button Control Panel / Toolbar.
- Commands to edit the Circuit Panel, and
- commands to rotate a circuit element.

The *View* sub-menu can be seen in **figure 4.1**.

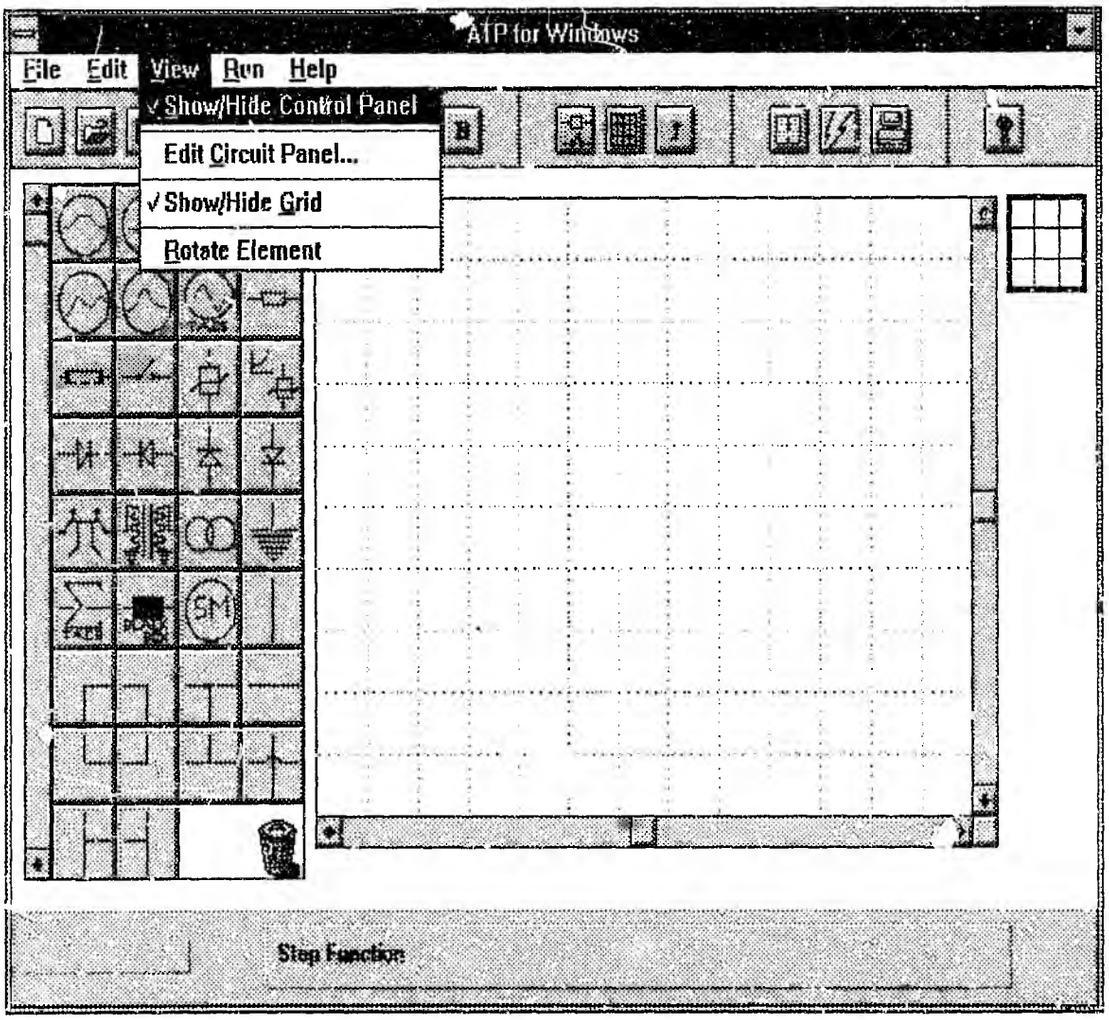


Figure 4.1 - The View Sub-menu

4.2 How To Use It

To bring down the *View* menu options, select *View* from the main menu with the mouse, or type <ALT>-V.

4.3 Show/Hide Control Panel

4.3.1 Function

The Show/Hide Control Panel option is used to show or take off the control panel on top of the screen in **ATP for Windows**. The Control Panel contains speed buttons which perform any function available in the pull down menu.

4.3.2 How To Use It

Select the *Show Control Panel* option from the *View* sub-menu using the mouse or the arrows and the <ENTER> key or type <ALT>-S.

4.3.3 Remarks

The *Show Control Panel* function is used to show the Control Panel as well as to remove it from the screen.

4.4 Edit Circuit Panel

4.4.1 Function

The *Edit Circuit Panel* option is used to edit the Circuit Panel speed buttons by adding or removing circuit elements which are not needed by the user for most simulations.

4.4.2 How To Use It

Once the *Edit Circuit Panel* option is selected from the *View* sub-menu, the *Edit Circuit Panel* screen will appear (see figure 4.3). This screen contains a menu of the Circuit Panel settings already selected. It has a default setting which is automatically used if this choice is not changed.

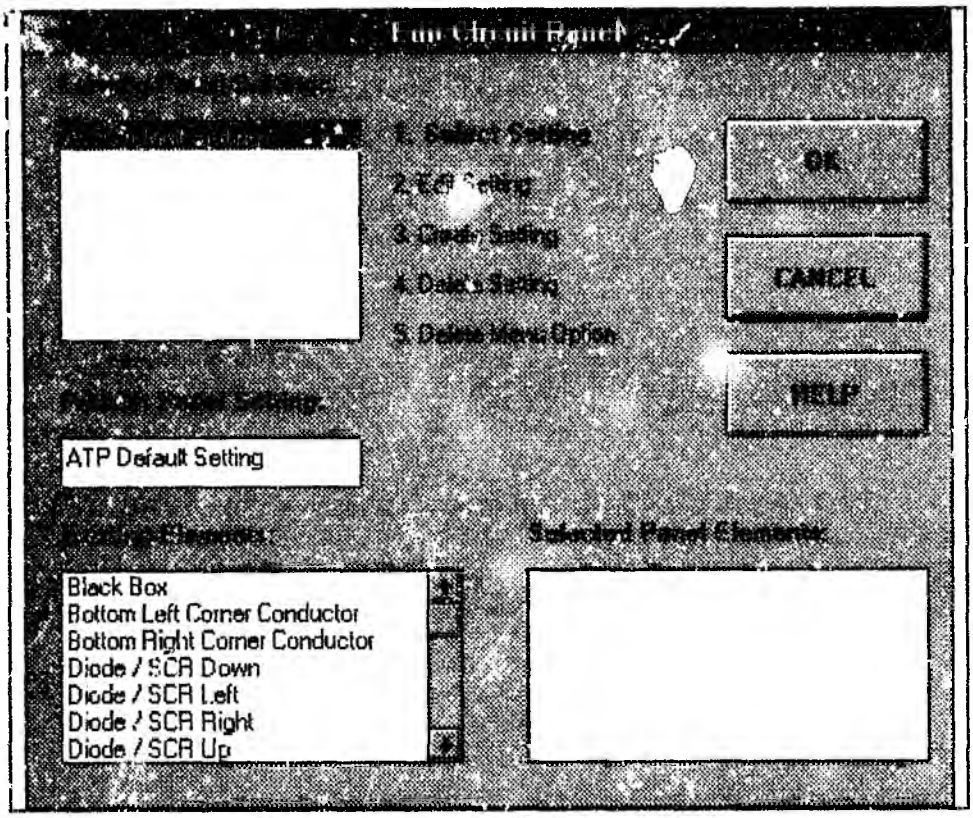


Figure 4.2 - The Edit Circuit Panel Screen

- To create a new setting, select the *Create Setting* option from the *Edit Circuit Panel* screen. This will ask the user to name the new setting, and then automatically pull down a list on the left hand side of the screen which contains all the elements that have been entered into the pre-processor as well as all the standard elements in ATP.
  - To edit an existing option other than the *ATP default* setting, first select the setting from the Existing Panel Settings list, and then select the *Edit Setting* option from the *Edit Circuit Panel* screen. To select an element which you would like to put in the *Circuit Element Panel*, click on the element from the Existing Elements List, and drag it to the *Selected Panel Elements* list . The elements selected for this setting will appear on the *Selected Panel Elements* list as well as the corresponding icon appearing in the actual Circuit Element Panel. To delete an element from a setting, click on the element in the *Select Panel Elements* list, and drag it to the *Garbage Bin* on the left of the screen, or click on the element and select the *Delete Element* option.
  - Once a *Circuit Panel* setting has been edited or created, it can be selected by

moving the mouse pointer to this setting in the *Panel Settings* list, and then clicking on it and choosing the *Select* option from the menu on the right of the screen.

- To delete an existing setting, follow the instructions for selecting a setting but select the *Delete* option from the menu on the right of the screen.

To exit from the *Edit Circuit Panel* screen click on the *OK* button and to cancel the operation at any point, click on the *Cancel* button.

4.5 Show / Hide Grid

4.5.1 Function

The *Show / Hide Grid* option is used to show or hide the Circuit Region Grid.

4.5.2 How To Use It

Select the Show / Hide Grid option from the View menu by clicking on it, or type <ALT> - G, or select the grid button from the Control Panel / Toolbar.

4.5.3 Remarks

None.

4.6 Rotate Element

4.6.1 Function

The Rotate Element function is used to rotate a Simple RLC branch or a Distributed RLC branch 90 degrees. This allows for the drawing of these elements vertically as well as horizontally. This function does not operate on other elements since it is not necessary for other elements.

4.6.2 How To Use It

Once an element has been dropped in the circuit, it can be rotated immediately by selecting the *Rotate Element* option from the *View* menu or by clicking on the *Rotate* option in the

Toolbar (see Message Bar when traversing over toolbar options).

4.6.3 Remarks

None

5 RUN

5.1 Introduction

The *Run* menu option pulls down the commands related to the running of the simulation through the pre-processor:

- Commands to view any elements from the Element Libraries of **ATP for Windows** and commands to select any general element from these libraries.
- Commands to set up and run the simulation.
- Commands to read the results graphically (not available yet).

The *Run* sub-menu can be seen in **figure 5.1.**

5.2 How To Use It

To bring down the *Run* menu options, select *Run* from the main menu with the mouse, or type <ALT>-R.

5.3 Remarks

Due to the numerous possibilities available in a simulation in ATP, the setup simulation function must be completed before a simulation can be performed. Many of the entries in the setup simulation screen however will have default settings if they are not numerical entries.

5.4 Select Element

5.4.1 Function

The *Select Element* option can be used to view or select any element available in **ATP for Windows**, and to place this element in the *Circuit Region*. This function is an alternative to using the *Circuit Element Panel* since it's functionality is the same.

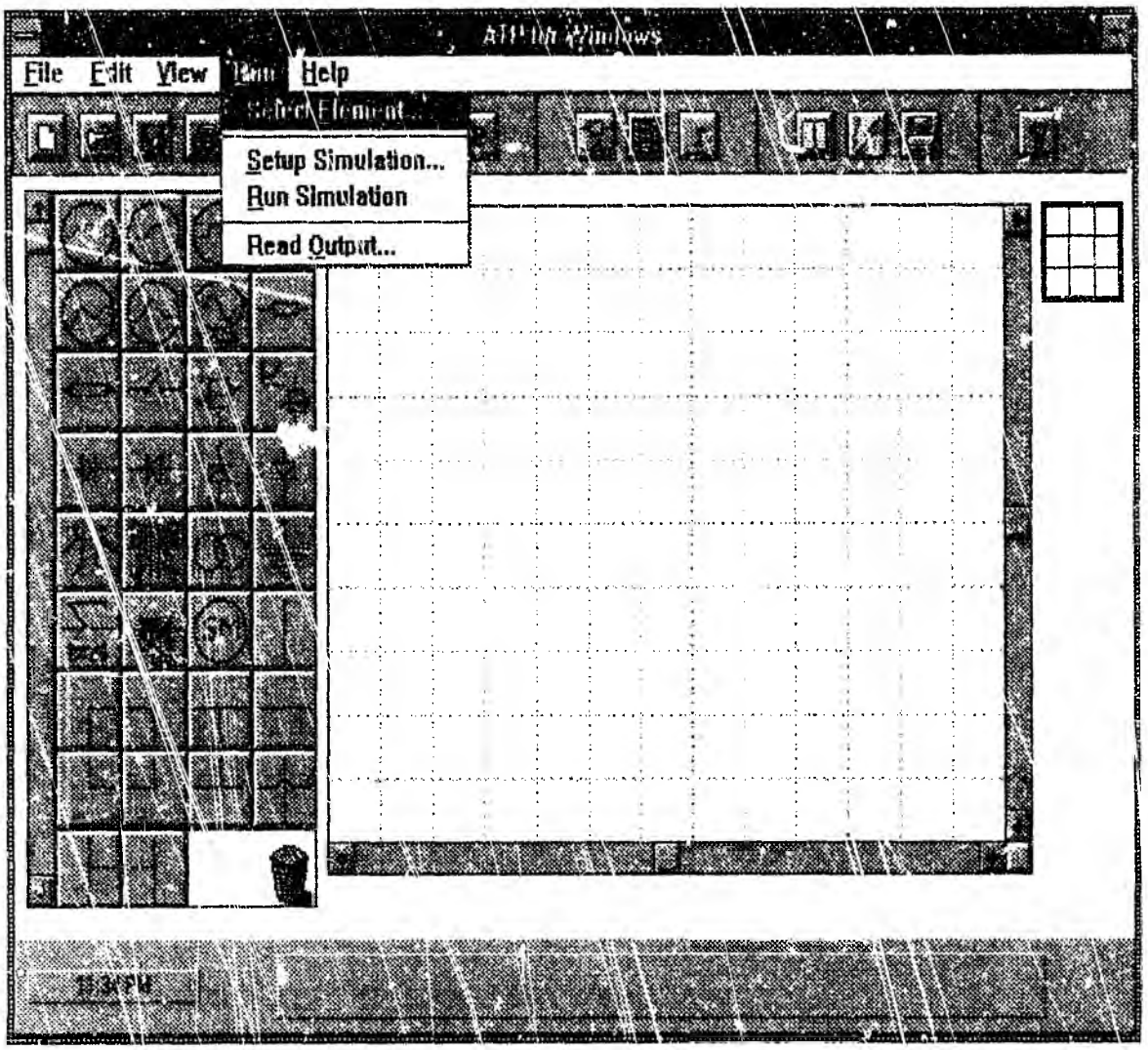


Figure 5.1 - The Run Sub-menu

5.4.2 How To Use It

Select the *Select Element* option from the Run menu by clicking on it with the left mouse button or using the arrows, or by typing <ALT>-E. This will produce the Element Library List Screen (see figure 5.2).

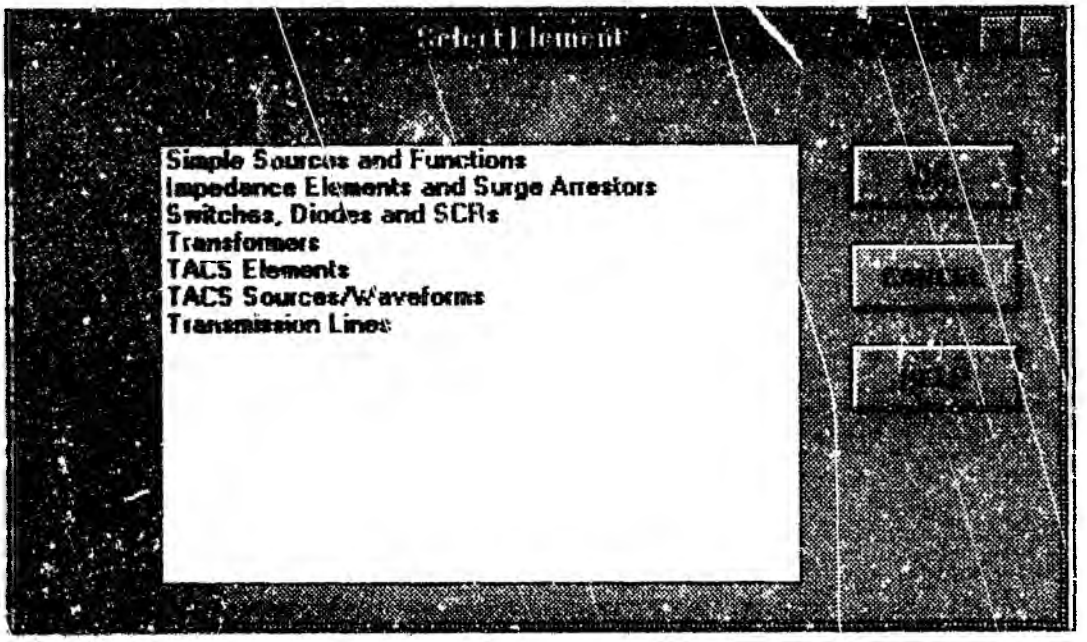


Figure 5.2 - The Select Element Screen

To select a particular library, click on library in the Element Libraries list, and select the *OK* button on the right hand side of the screen. Note: Although the transformer, switch, TACS elements, TACS sources and transmission lines have libraries, these are only for viewing purposes, and the particular type of element in these particular libraries can only be selected in the *Element Details* screen of this element.

5.4.2.1 The Simple Sources and Functions Library

Selecting the Simple Sources and Functions Library will produce the Simple Sources and Functions Library Screen (see figure 5.3).

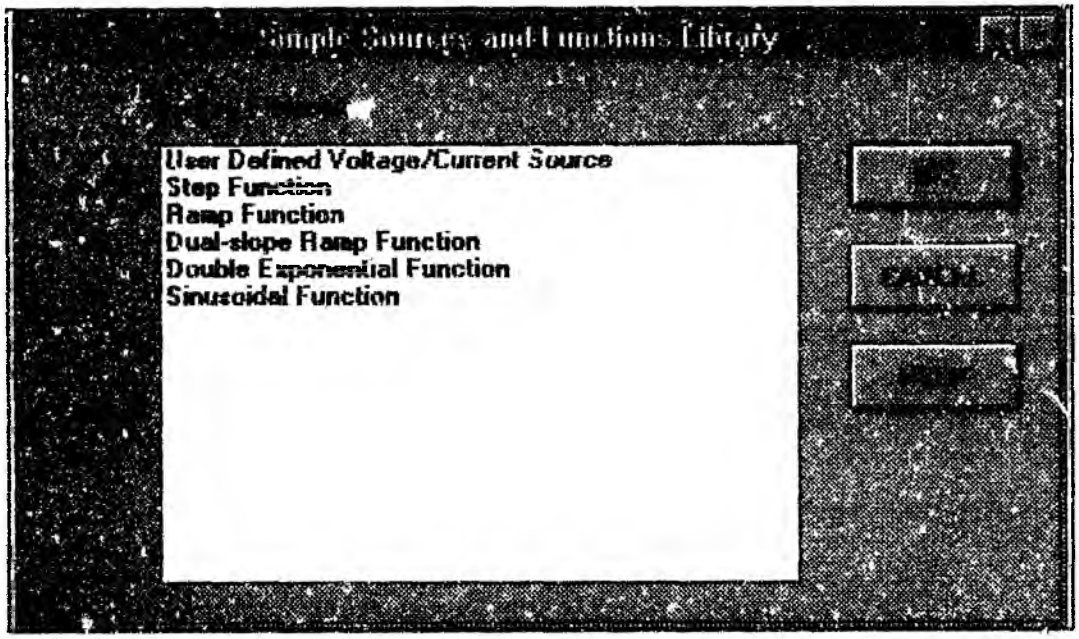


Figure 5.3 - The Simple Sources & Functions Library List

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.4.2.2 The Impedance Elements & Surge Arrestors Library

Selecting the Impedance Elements and Surge Arrestors Library will produce the Impedance Elements and Surge Arrestors Library Screen (see figure 5.4).

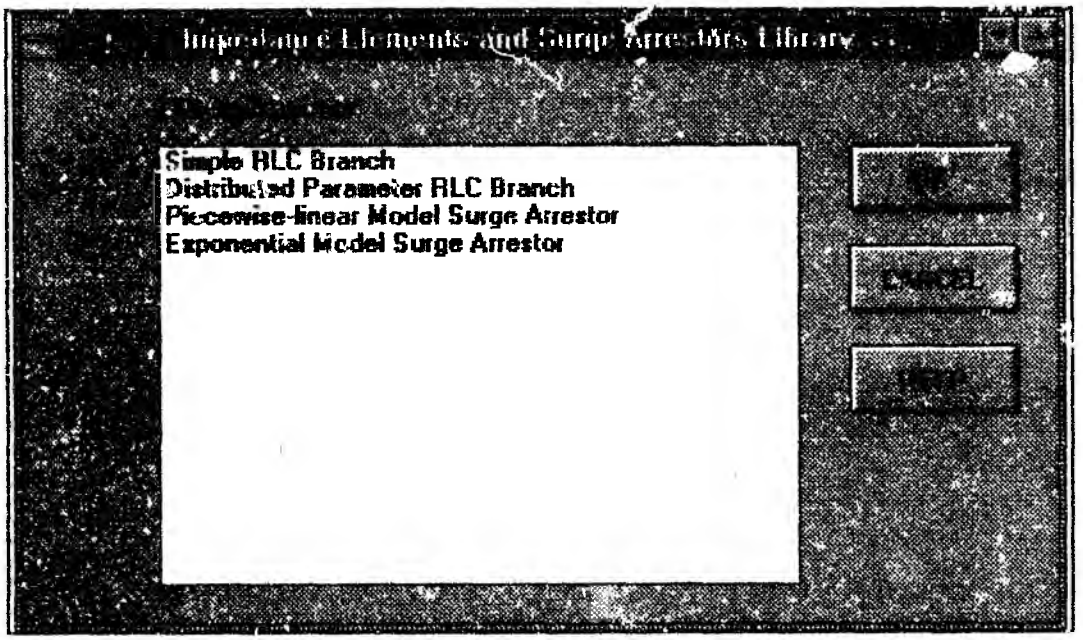


Figure 5.4 - The Impedance Elements & Surge Arrestors Library

To select an element click on the radio button corresponding to the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.4.2.3 The Switches and Diodes & SCRs Library

Selecting the Switches and Diodes & SCRs Library will produce the Switches and Diodes & SCRs Library Screen (see figure 5.5).

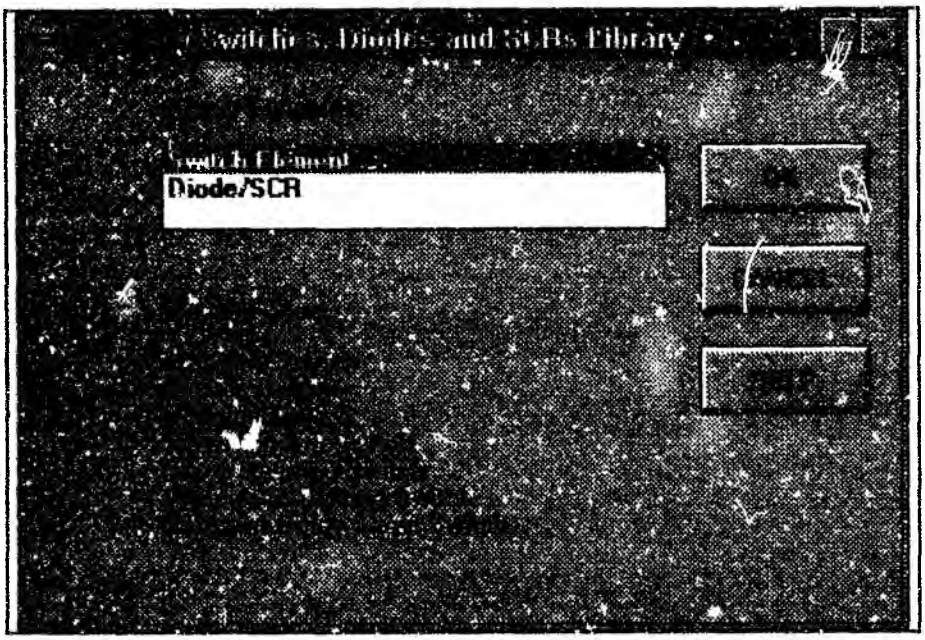


Figure 5.5 - The Switches and Diodes & SCRs Library

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Move this icon using the mouse to the required position in the *Circuit Region*. Note that selecting a switch requires the selection of the general *Switch Element* option, and the particular type of switch will be chosen in the *Element Details* screen corresponding to this element. To obtain on-line Help at any time select the *HELP* button.

5.4.2.4 The Transformer Library

Selecting the Transformer Library will produce the Transformer Library Screen (see figure 5.6).

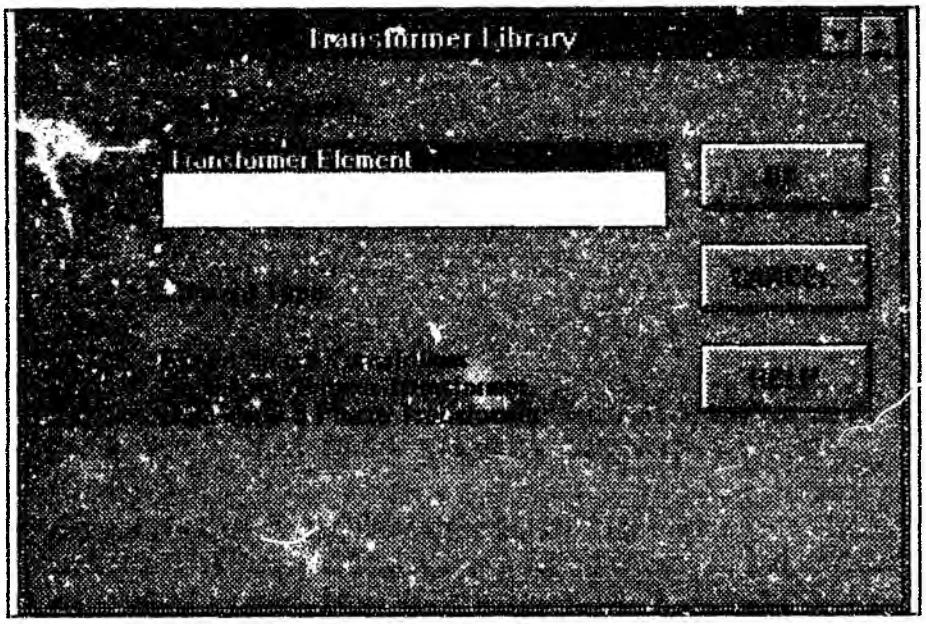


Figure 5.6 - The Transformer Library Screen

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Note that selecting a transformer requires the selection of the general *Transformer Element* option, and the particular type of transformer will be chosen in the *Element Details* screen corresponding to this element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.4.2.5 The TACS Element Library

Selecting the TACS Element Library will produce the TACS Element Library Screen (see figure 5.7).

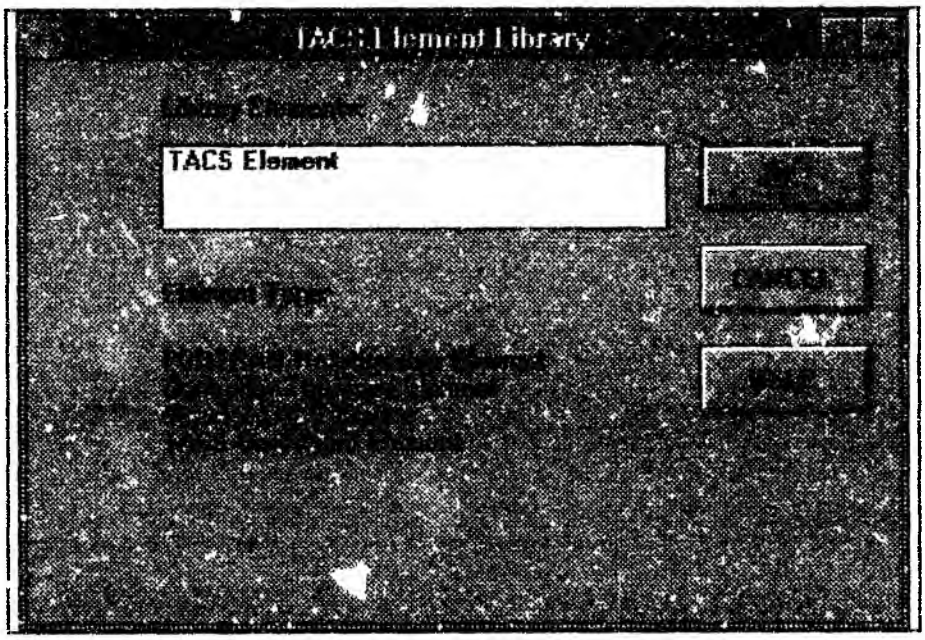


Figure 5.7 - The TACS Element Library Screen

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Note that selecting a TACS Element required the selection of the general *TACS Element* option, and the particular type of TACS Element will be chosen in the *Element Details* screen corresponding to this element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.4.2.6 The TACS Source Library

Selecting the TACS Source Library will produce the TACS Source Library Screen (see figure 5.8).

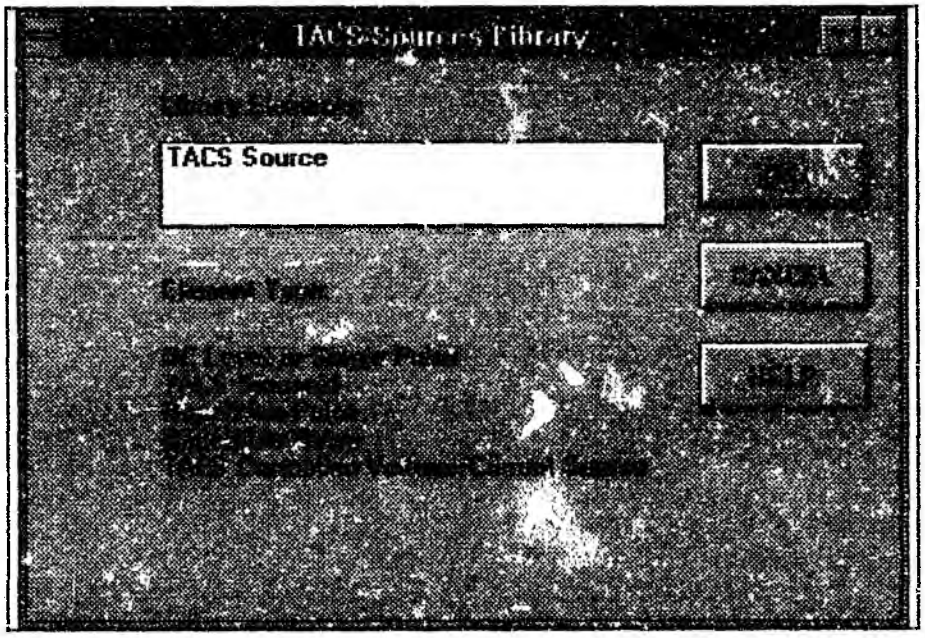


Figure 5.8 - The TACS Sources Library Screen

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Note that selecting a TACS Source requires the selection of the general *TACS Source* option, and the particular type of TACS Source will be chosen in the *Element Details* screen corresponding to this element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.4.2.7 The Transmission Line Library

Selecting the Transmission Line Library will produce the Transmission Line Library Screen (see figure 5.9).

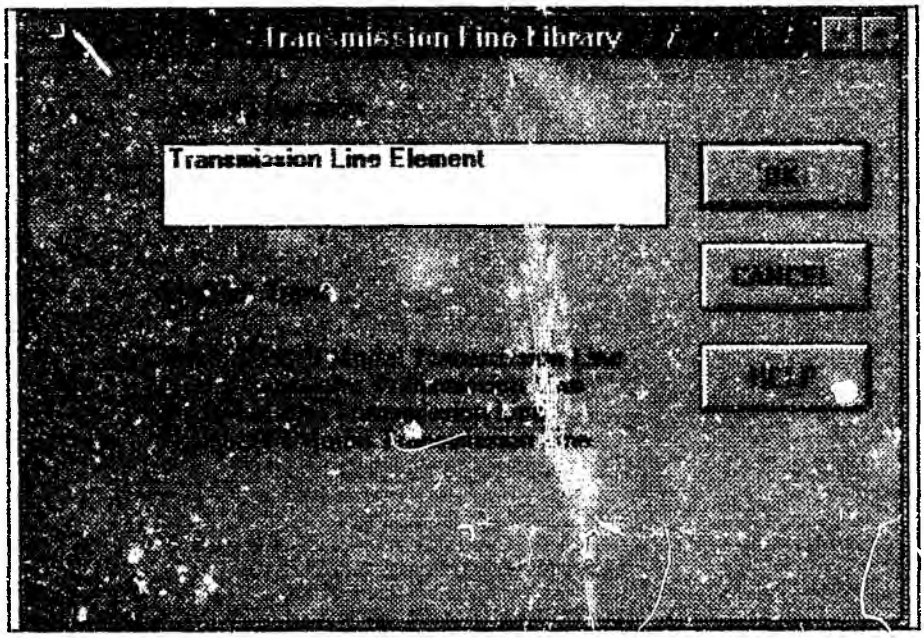


Figure 5.9 - The Transmission Line Library Screen

To select an element click on the required element, and then click on the *CREATE* button on the right hand side of the screen. This will exit from this screen, and produce an icon representing the required element. Note that selecting a transmission line requires the selection of the general *Transmission Line* option, and the particular type of transmission line will be chosen in the *Element Details* screen corresponding to this element. Move this icon using the mouse to the required position in the *Circuit Region*. To obtain on-line Help at any time select the *HELP* button.

5.5 Setup Simulation

5.5.1 Function

The *Setup Simulation* option is used to enter the general information required for every simulation performed on ATP such as frequency of simulation and time steps required. The entries required correspond to the entries of the cards in ATP which are not specifically associated with a branch card, switch card, source card etc. The information which is usually placed in the *Variable Output request card* is entered here for example. Before the *Run Simulation* function can be used, the *Setup Simulation* option has to be completed.

5.5.2 How To Use It

Select the *Setup Simulation* option from the *Run* sub-menu. Once this has been selected, the *Setup Simulation* screen will appear (see figure 5.10).

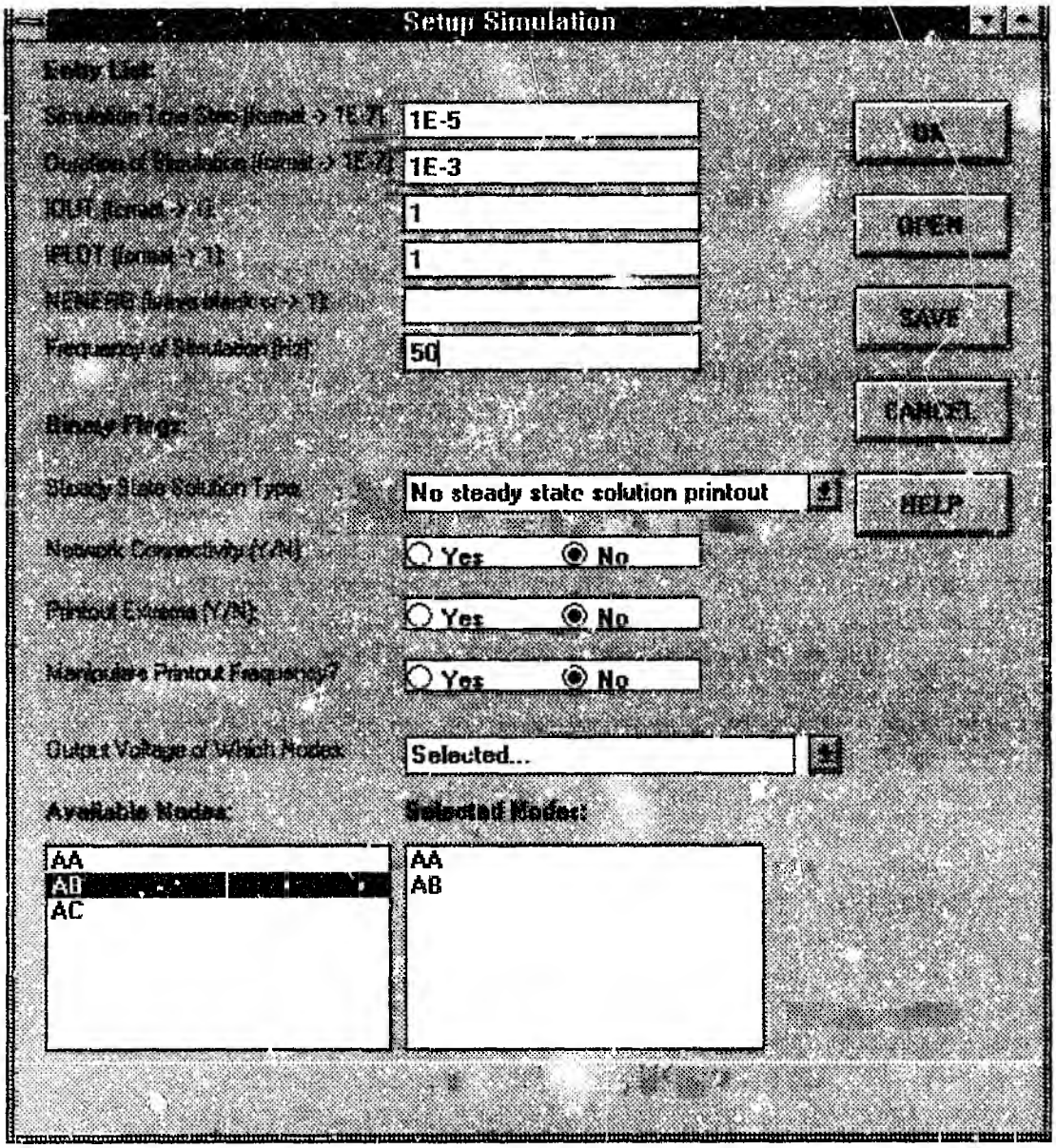


Figure 5.10 - The Setup Simulation Screen

This screen allows the user to enter or modify the following fields:

- **Simulation time step** - This is the time step required between each simulation of the circuit/network.
- **Duration of Simulation** - This is the total duration required for the simulation.
- **IOUT** - This defines the frequency of output, relative to the simulation time step (a 3 in this field will request that the values for every third simulation time step be printed). A 0 will be treated as a 1.
- **IPLOT** - This defines the frequency of saving simulation values for later plotting (a 3 will in this field will request that the value of every third simulation time step be saved). Again, a 0 is treated as a 1.
- **NENET** - Leave blank unless required by element (see section 7.3.13.5)
- **Frequency of Simulation** - This defines the frequency of the simulation.
- **Inductance input (Ω/mH)** radio button- This is a flag to indicate whether the inductance in linear branches is to be specified in mH or in Ω. If the inductance is to be specified in Ω, the user will be asked to enter the frequency of the simulation.
- **Capacitance input (Ω/μF)** radio button - This is a flag to indicate whether the capacitance in linear branches is to be specified in μF or in μΩhos (capacitive susceptance). If the capacitance is to be specified in μΩhos, the user will be asked to enter the frequency of the simulation.
- **Network Connectivity(Y/N)** radio button - This is a flag for controlling the output of a table showing network connectivity. The options available in this field are to suppress the output or to produce the output.
- **Steady State Solution Type** radio button - This is a flag for controlling the printout of the steady state phasor solution. There are three basic types of outputs: branch flows, switch flows, and nodal injections. Theses can be controlled by the choice of this field. The default choice is *no steady state solution.*
- **Printout Extrema (Y/N)?** radio button - This is a flag for controlling the printout of extrema in the simulation. The default state is to suppress this output.
- **Manipulate printout freq?** radio button - If the printout frequency is non-standard (ie. the time increment is not constant), select yes in this option. This will open a dialogue box which requires the entry of each time step at which a printout is required. **NOT AVAILABLE YET.**
- **Output voltage of all nodes** radio button - This is used to specify that the voltage output required includes every node in the circuit/network.
- **Output Voltage of Specified Nodes** - This input is used to specify which of the node voltages are required. Selecting this option will open the

Available Nodes, and the *Selected Nodes* lists. The available nodes will list all of the nodes presently being used in the circuit. To move an available node to the *Selected Nodes* list, drag the node from the *Available Nodes* list, and drop it into the *Selected Nodes* list.

During running of the simulation, move the mouse over specific entries in this screen for more information about the entries. To modify any of these settings/values, select the setting/value you wish to modify, and then modify it using the keyboard. To select an existing setup, select the *Open* button on the right of the screen. This will produce a screen similar to the *Open* option in the *File* sub-menu. To save a setup, select the *Save* button. This will produce a *Save* screen similar to the *Save* screen in the *File* sub-menu. The Setup simulation setting files have the extension ssf (setup simulation files). To obtain on-line Help, select the *HELP* button at any time.

5.5.3 Remarks

It is important to note that the *Run Simulation* option will remain grayed out (unusable) until the *Setup Simulation* option has been completed.

5.6 Run Simulation

5.6.1 Function

The *Run Simulation* function is used to call up **ATP** from within Windows, and run the simulation presently in the *Circuit Region*.

5.6.2 How To Use It

Select the *Run Simulation* option from the *Run* sub-menu. This will begin the simulation process. Before the simulation is run, a screen similar to the Save As screen will appear asking the user to select a file name for the data file which ATP for Windows will create. Once the simulation is complete, clicking on the Read Output option will run PCPLOT, which allows the user to read the output. ATP for Windows will have its own output reader from version 1.5. A recommended Windows output reader until then is Electrotek's TOP (available free to universities through EPRI and the EMTP Users Group).

5.6.3 Errors

If the fields in the Simulation Setup option have not been completed, the Run Simulation option will not be available to the user. If there is not enough virtual memory to run the simulation, exit **ATP for windows**, and run ATP using the data file created by **ATP for Windows**. Once this has been done. Errors that occur after running ATP are most likely to be ATP errors, and consulting the EMTP Users Group should clear up the problem.

5.7 Read Output

5.7.1 Function

The Read Output function is used to portray the results of the simulation either numerically or graphically. This function is not available yet.

5.7.2 How To Use It

Once a simulation has been performed, select the Read Output option from the Run sub-menu. This will automatically call the Read Output screen (see figure 5.3). To view the voltages or currents of a simulation select the variables from the *Select Variables* list and then select to view these results either graphically or numerically. Once the type of viewing has been selected, select the OK option. You will then be asked to name the graph/list before viewing it. This graph/list will automatically be saved for later printing use. To print a graph, select the print option and select the relevant graph/list from the list of files given. To compare to variables, select each variable, and then select the *Compare Graphs* option. This will plot each variable vs time on the same screen.

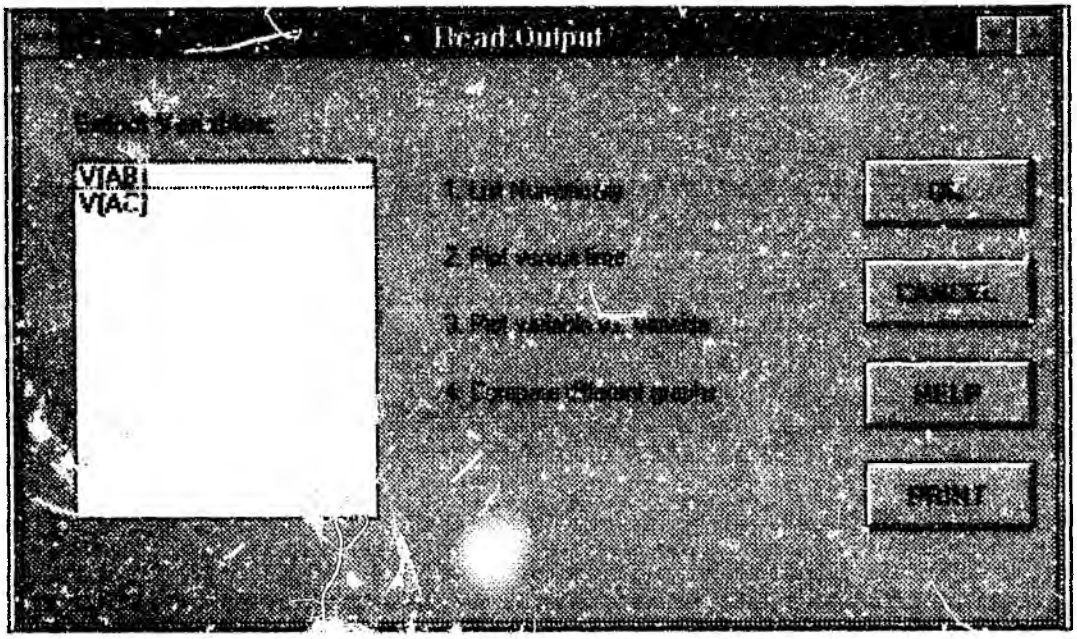


Figure 5.3 - The Read Output Screen

5.7.3 Remarks

The *Read Output* option will c[illegible] the user if the circuit/network which is currently in the *Circuit Region* [illegible] simulated using **ATP for Windows.**

6 HELP

6.1 Introduction

The Help menu option pulls down the commands related to the on-line help available to the user of **ATP for Windows**:

- Commands to set up the Help Index and Name Search functions.
- Commands to setup the on-line Tutorial.

The *Help* sub-menu can be seen in **figure 6.1**.

6.2 How To Use It

To bring down the *Help* menu options, select *Help* from the main menu with the mouse, or type <ALT>-H.

6.3 Remarks

Those functions that are not as yet in the on-line Help should be available in this User Reference Manual.

6.4 Help Contents

6.4.1 Function

The *Help Contents* is a list of all the functions available in **ATP for Windows**. It allows the user to move through the list and select an entry for which the user needs help.

6.4.2 How To Use It

Select the *Help Contents* from the *Help* sub-menu. Once this has been selected, a list of all the functions available in **ATP for Windows** will appear (see figure 6.2). To move through the list, use the vertical scroll bar on the right hand side of the screen or use the arrows on the keyboard. To select an entry, click on that entry with the mouse, or press <ENTER> when the cursor is on that entry, and then select the *GOTO* button. To move to a previous screen select the *PREVIOUS* button.

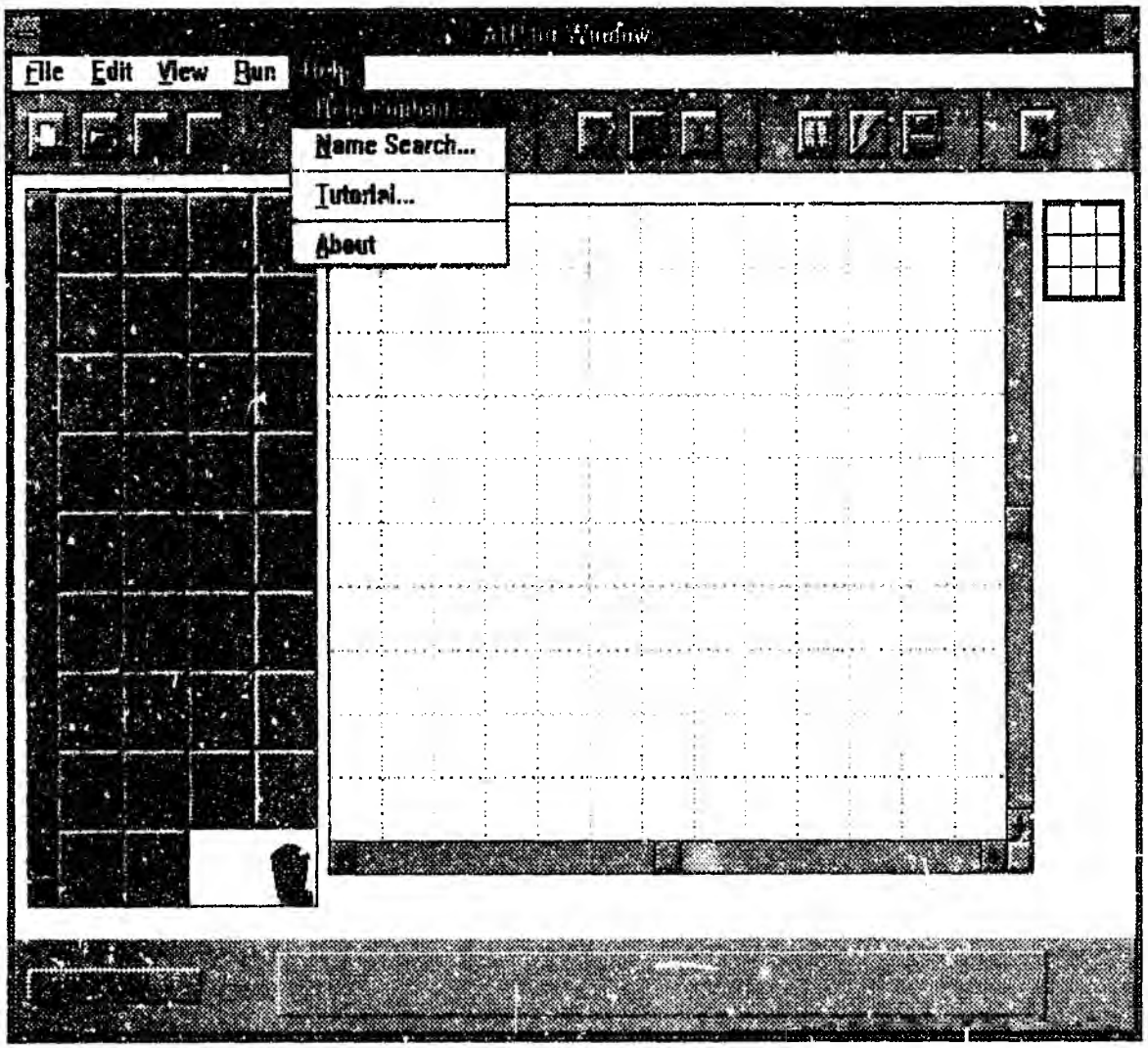


Figure 6.1 - The Help Sub-menu

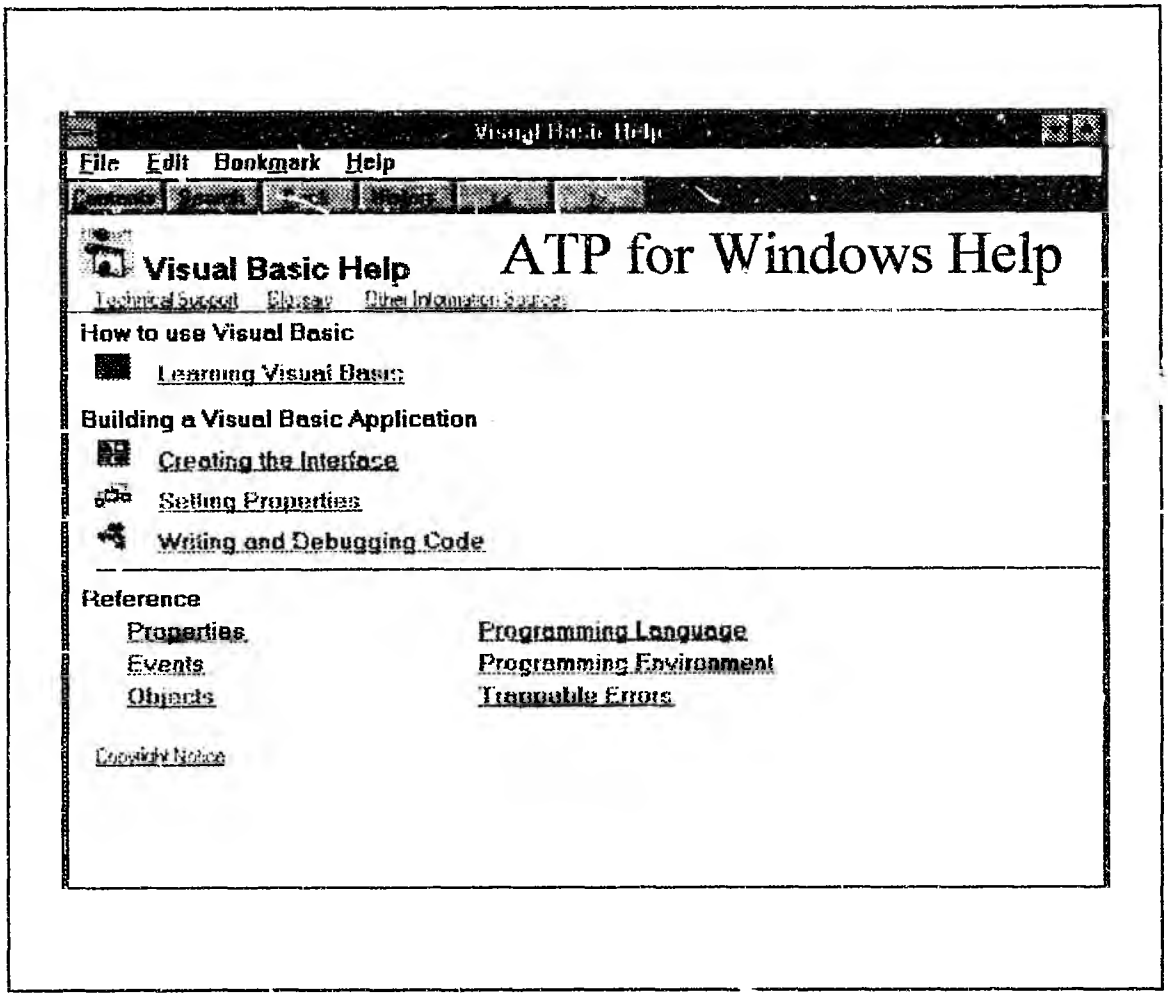


Figure 6.2 - The Help Contents Screen

6.5 Name Search

6.5.1 Function

Name Search is used to obtain help on a function or topic by naming this topic.

6.5.2 How To Use It

Select the *Name Search* function from the *Help* sub-menu. In the *Name Search* screen, type in the name of the function/topic you would like to get help on. Once the first letters of any function or topic available are typed, that topic will appear in the function list (see Figure

7.3 below). As soon as the topic or function which you would like help on appears in the list, click on the *GOTO* button.

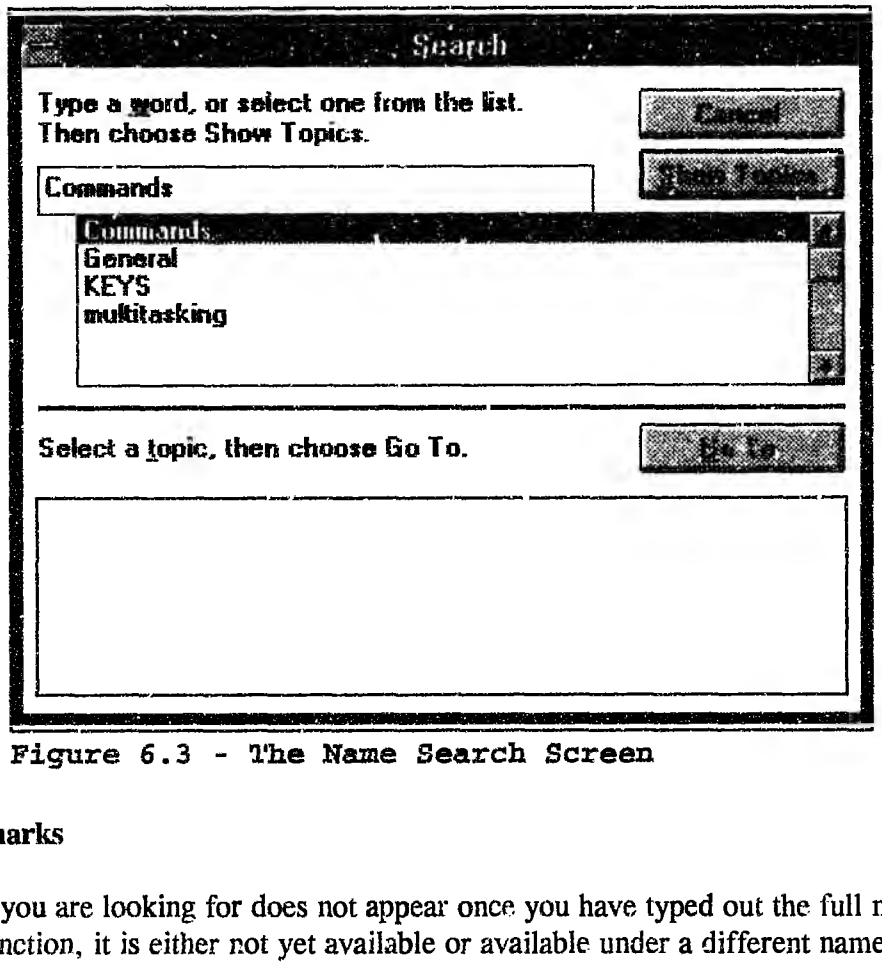


Figure 6.3 - The Name Search Screen

6.5.3 Remarks

If the topic you are looking for does not appear once you have typed out the full name of the topic/function, it is either not yet available or available under a different name.

6.6 Tutorial

6.6.1 Function

The Tutorial is available to help the user get well acquainted with **ATP for Windows**. It is a step by step instruction manual involving problem definition as well as complete simulation of a number of problems.

6.6.2 How To Use It

Select the Tutorial option from the *Help* sub-menu. Once this has been selected, select either the tutorial help option, or the required tutorial question from the Tutorial list. The help option will explain the function of the on-line Tutorial, while selecting a tutorial question will provide a step by step tutorial through the selected problem. **Note: this function is not available yet.**

6.6.3 Remarks

Once all the problems in the tutorial have been completed, the user should be fairly proficient in **ATP for Windows.**

7 ELEMENT DETAILS

The Element Details option pulls down the commands related to the circuit element description:

- Commands to enter the numerical details of the circuit element as well as the names of the nodes touching the element.

The *Element Details* function is used to enter the numerical details of the relevant circuit element. For this reason, the content of the *Element Details* screen differs for each and every type of element. The structure of the screen however is similar in most cases. If the relevant circuit element is a resistor, the resistance of the resistor as well as the names of the two nodes touching this resistor will be entered in this screen. If the relevant element is a voltage or current source, the type of source, as well as the equation governing this source must be entered in the *Element Details* screen. Every screen has standard user-defined entries as well as default value entries. Default value entries are entries for which **ATP for Windows** has allocated an automatic default choice. These entries are generally combo box entries.

7.1 How To Use It

To get to the Element Details screen of an element, click on that element with the right mouse button.

7.2 Sources

7.2.1 The Step Function

The step function starts at zero and reaches its final amplitude in one simulation time step, and maintains this value. To select the step function, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the Step Function Details Screen will appear (see figure 7.1).

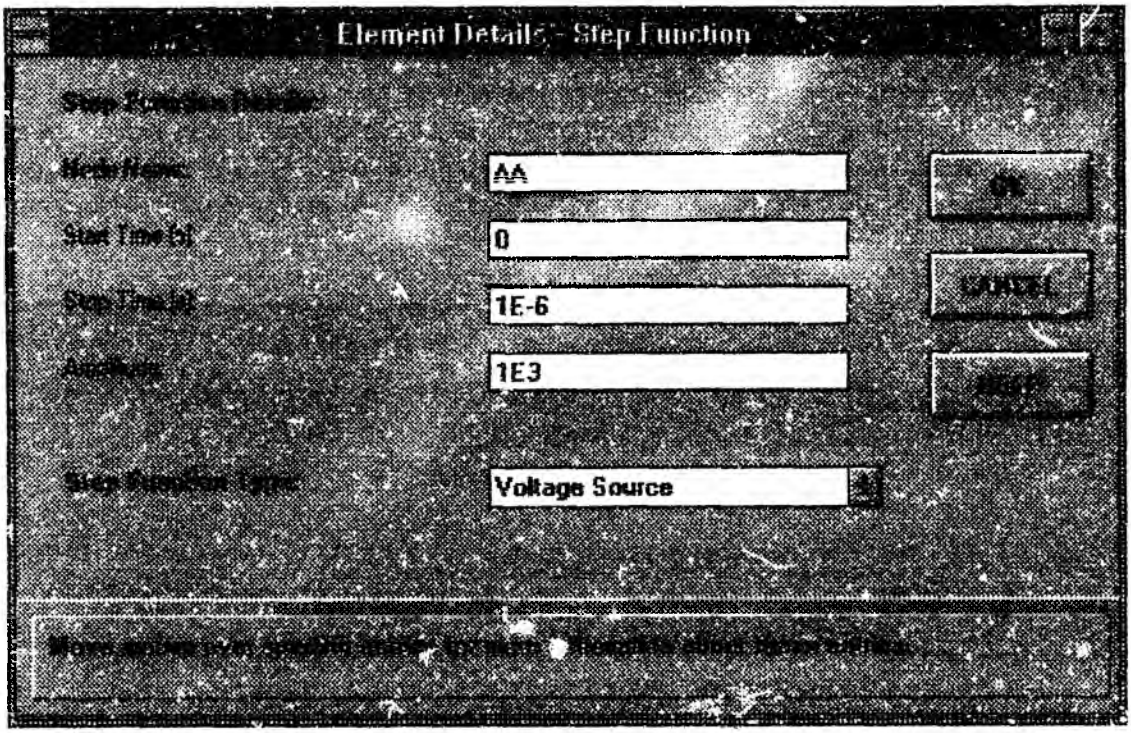


Figure 7.1 - The Step Function Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.
- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. If left blank the source will continue indefinitely.
- **Amplitude** - The final amplitude in volts or amps.
- **Step Function Type** Combo Box - Select one of these entries to choose between a voltage or current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.2.2 The Ramp Function

The ramp function starts at zero and reaches its final amplitude in t seconds (specified by the user), and then maintains this value. To select the ramp function, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the Ramp Function Details Screen will appear (see *figure 7.2*).

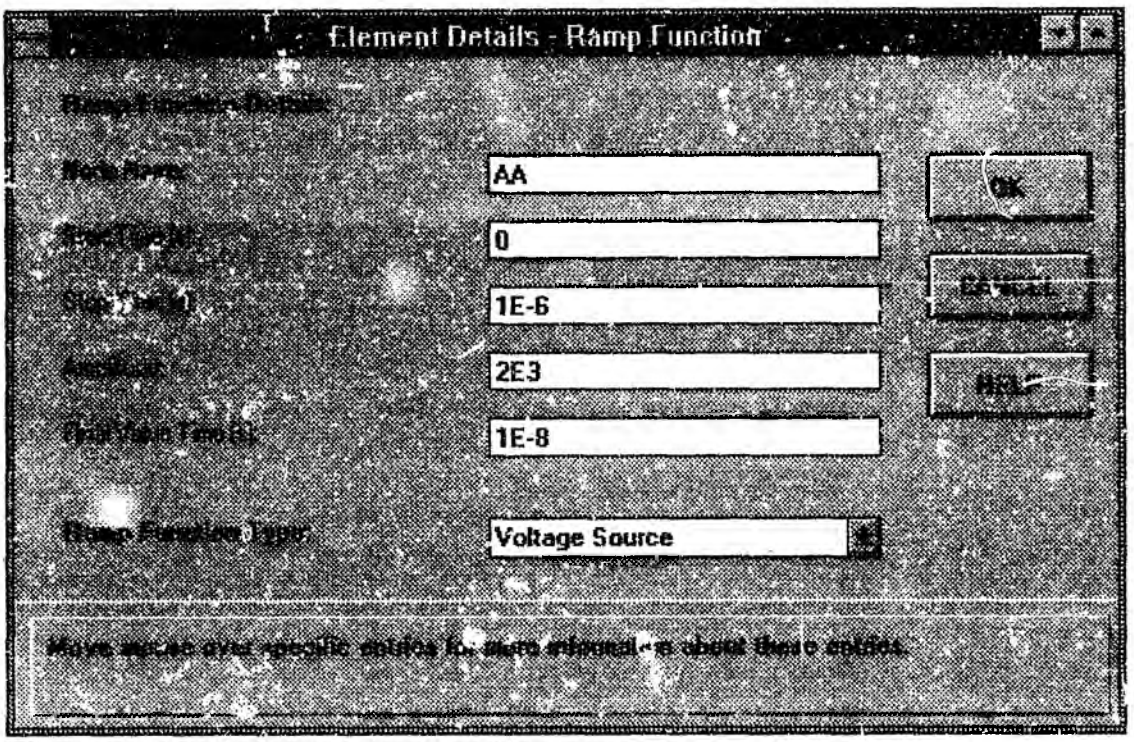


Figure 7.2 - The Ramp Function Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.
- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. If left blank the source will continue indefinitely.
- **Amplitude** - The final amplitude in volts or amps.
- **Final Value Time** - The time at which the ramp reaches it's final value.
- **Ramp Function Type** Combo Box - Select one of these entries to choose between a voltage or current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.2.3 The Dual-slope Ramp Function

The dual-slope ramp function starts at zero and reaches it's peak amplitude in to seconds (specified by the user), and then ramps down past some value (tail-value - again specified by the user) at instant to. To select the dual-slope ramp function, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the Dual-slope Ramp Function Details Screen will appear (see *figure 7.3*).

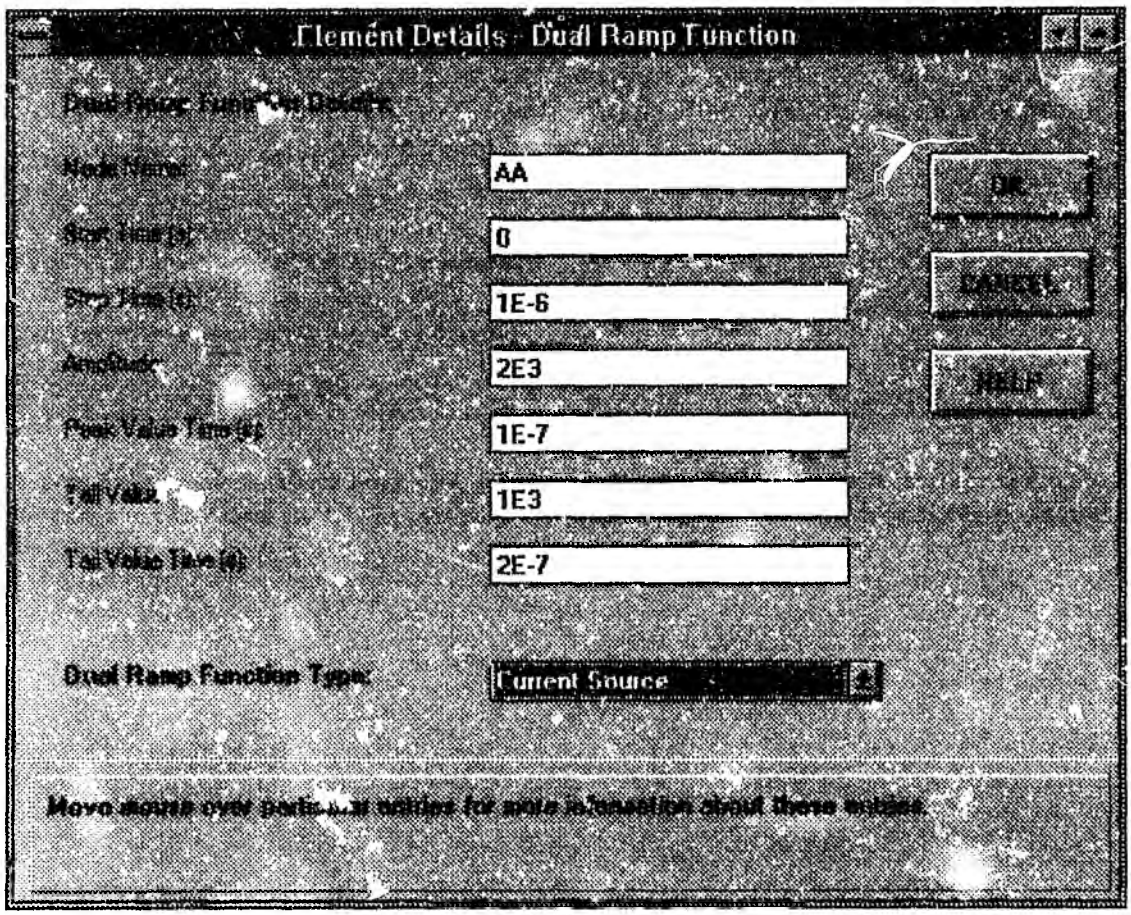


Figure 7.3 - The Dual Slope Ramp Function Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.
- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. if left blank the source will continue indefinitely.
- **Amplitude** - The peak amplitude in volts or amps.
- **Peak Value Time** - Time at which the ramp reaches it's peak value.
- **Tail Value** - The particular value on function tail in volts or amps.
- **Tail Value Time** - The time at which the ramp reaches it's final value.
- **Dual-slope Ramp Function Type** Combo Box - Select one of these entries to choose between a voltage or current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.2.4 The Double Exponential Function

This function is described by the equation:

$$f(t) = A[e^{Bt} - e^{Ct}] \qquad \textbf{(7.1)}$$

To select the double exponential function, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the Double Exponential Function Details Screen will appear (see *figure 7.4*).

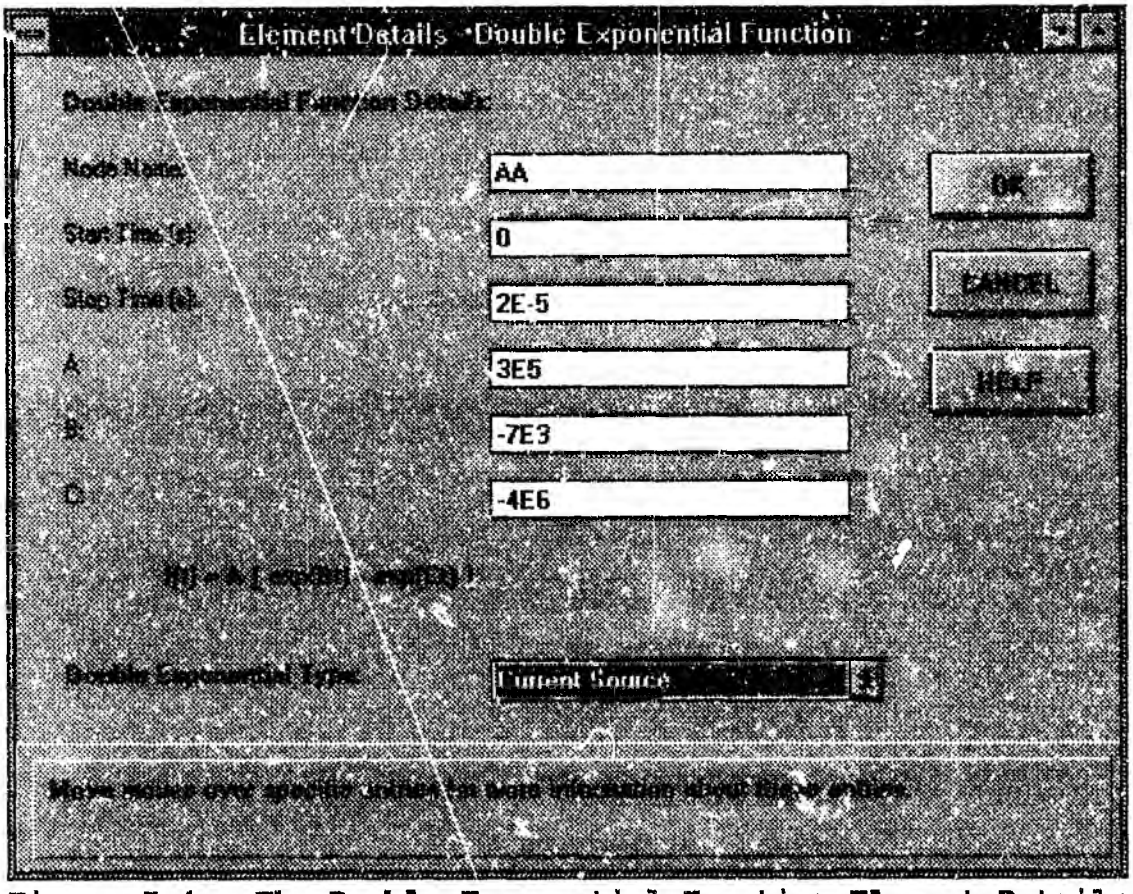


Figure 7.4 - The Double Exponential Function Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.
- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. If left blank the source will continue indefinitely.
- **A** - See equation 7.1 above.
- **B** - See equation 7.1 above.
- **C** - See equation 7.1 above.
- **Step Function Type** Combo Box - Select one of these entries to choose between a voltage or current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.2.5 The Sinusoidal Source

The sinusoidal source is a standard sinusoidal current or voltage source. To select the sinusoidal function, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the Sinusoidal Function Details Screen will appear (see *figure 7.5*).

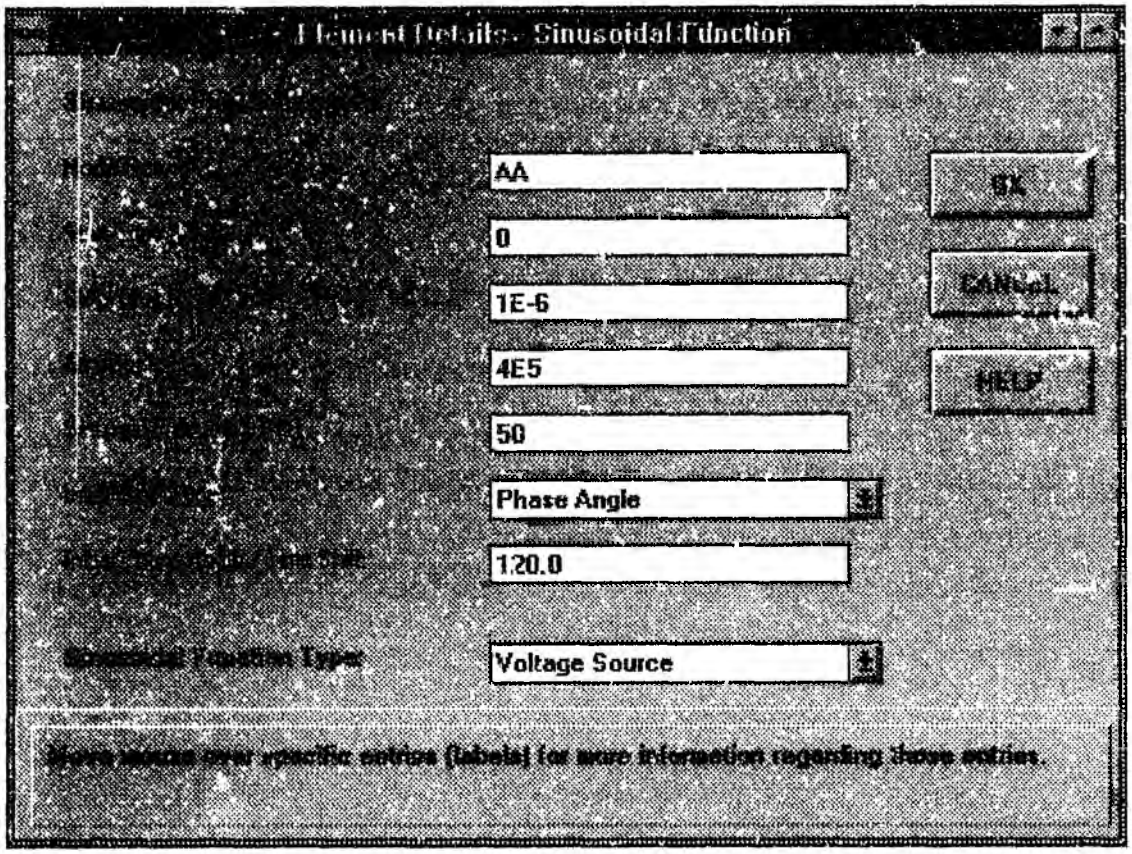


Figure 7.5 - The Sinusoidal Function Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.

- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. If left blank the source will continue indefinitely.
- **Amplitude** - This is the peak value of the sinusoid in volts or amps.
- **Frequency** - This is the frequency of the sinusoid in Hz.
- **Initial Phase Angle** and **Time Shift** combo box - One of these two buttons must be selected. The selection will determine whether the initial condition of the sinusoid is given in phase angle, or time shift.
- **Initial Phase Angle/Time Shift** - The initial position of the sinusoid in degrees or seconds depending on which radio button was selected (see previous entry).
- **Double Exponential Function Type** Combo Box - Select one of these entries to choose between a voltage or current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.2.6 The User Defined Voltage or Current Source

The User-defined source is defined point by point with a value for each simulation time step. To select the user-defined source, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of source, the User-defined Voltage/Current Source Details Screen will appear (see figure 7.6).

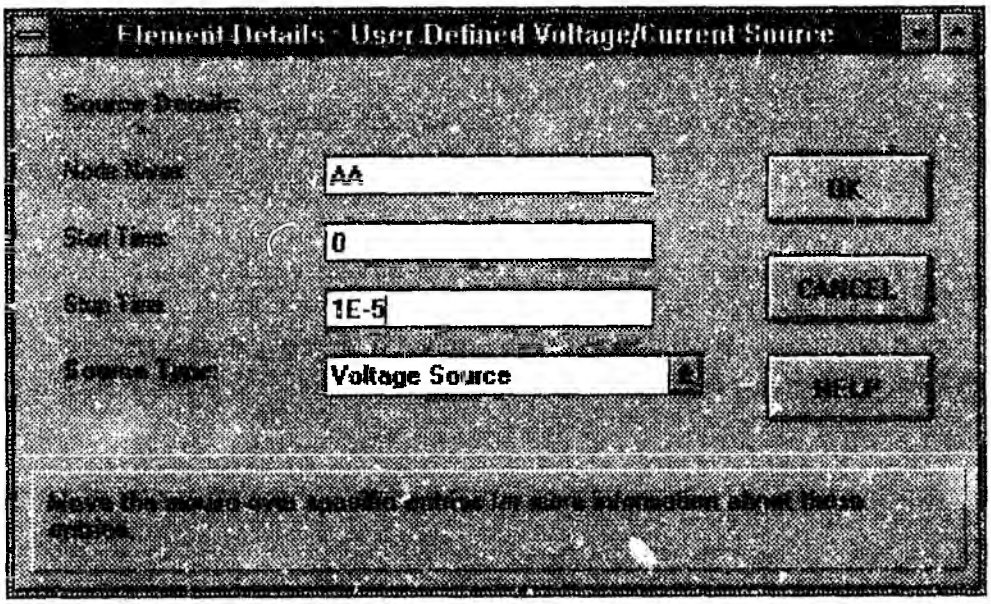


Figure 7.6 - The User Defined Source Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Node Name** - The name of the node to which the source is attached (the source is placed between this node and the local ground).
- **Start Time** - The delay before the source is activated in seconds.
- **Stop Time** - The time at which the source is terminated - source value is returned to zero, in seconds. If left blank the source will continue indefinitely.
- **Number of Time Steps** - The number of time steps for which the source will be defined.
- **Time Step Values** - This button will produce the screen which allow the user to enter the time step values of the user-defined source (see *figure 7.3*).
- **Source Type** combo box - Select whether the source is a voltage or current source.

To enter the time step values, for each time step, enter the time in seconds, followed by a comma, and then the voltage / current in Volts or Amps. To enter the next time step, press enter, and type in the next step.

Once all the entries in both of these screens have been completed, select the *OK* button to exit each screen. To cancel the description of the element, select the *CANCEL* button at any

time. To obtain on-line Help, select the *HELP* button at any time. **For information on the TACS source, see chapter 9.**

7.3 Branch Card Type Elements

7.3.1 The Simple RLC Branch

This type of branch allows the modelling of a series connection of resistance, inductance and capacitance (not all the elements need be present). To select the simple RLC branch, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of element, the Simple RLC Branch Details Screen will appear (see *figure 7.7*).

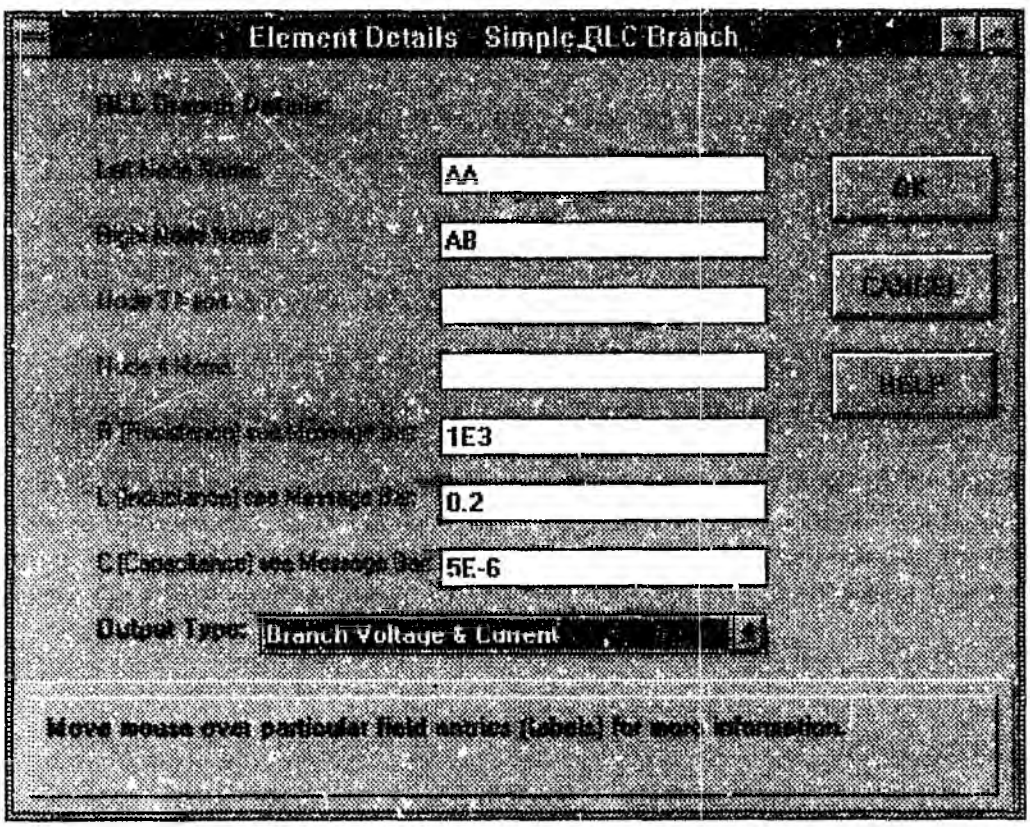


Figure 7.6 - The Simple RLC Branch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Left Node Name** - The name of the node to the left of the RLC branch (If this node is earth, leave the name blank).

- **Right Node Name** - The name of the node to the right of the RLC branch (if this node is earth, leave the name blank).
- **Node3 Name** - The name of the left node of an identical RLC branch (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)
- **Node4 Name** - The name of the right node of an identical RLC branch (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)
- **R** - The resistance (if any) specified in **Ω**.
- **L** - The inductance (if any) specified in either mH or **Ω**, depending on which radio button is selected (see L and C radio buttons below).
- **C** - The capacitance (if any) specified in either μF or μmhos depending on which radio button is selected (see L and C radio buttons below).
- **L** and **C** (mh or **Ω**, and μF or $\mu\Omega$) combo boxes - One of the two units must be selected for each of the Capacitance and Inductance.
- **Output Type** Combo Box - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.3.2 Distributed Parameter RLC Branch / Single Transmission Line Conductor

This type of branch allows the modelling of a distributed line in terms of it's modal distributed parameters. To select the distributed parameter RLC branch, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of element, the Distributed Parameter RLC Branch Details Screen will appear (see *figure 7.8*).

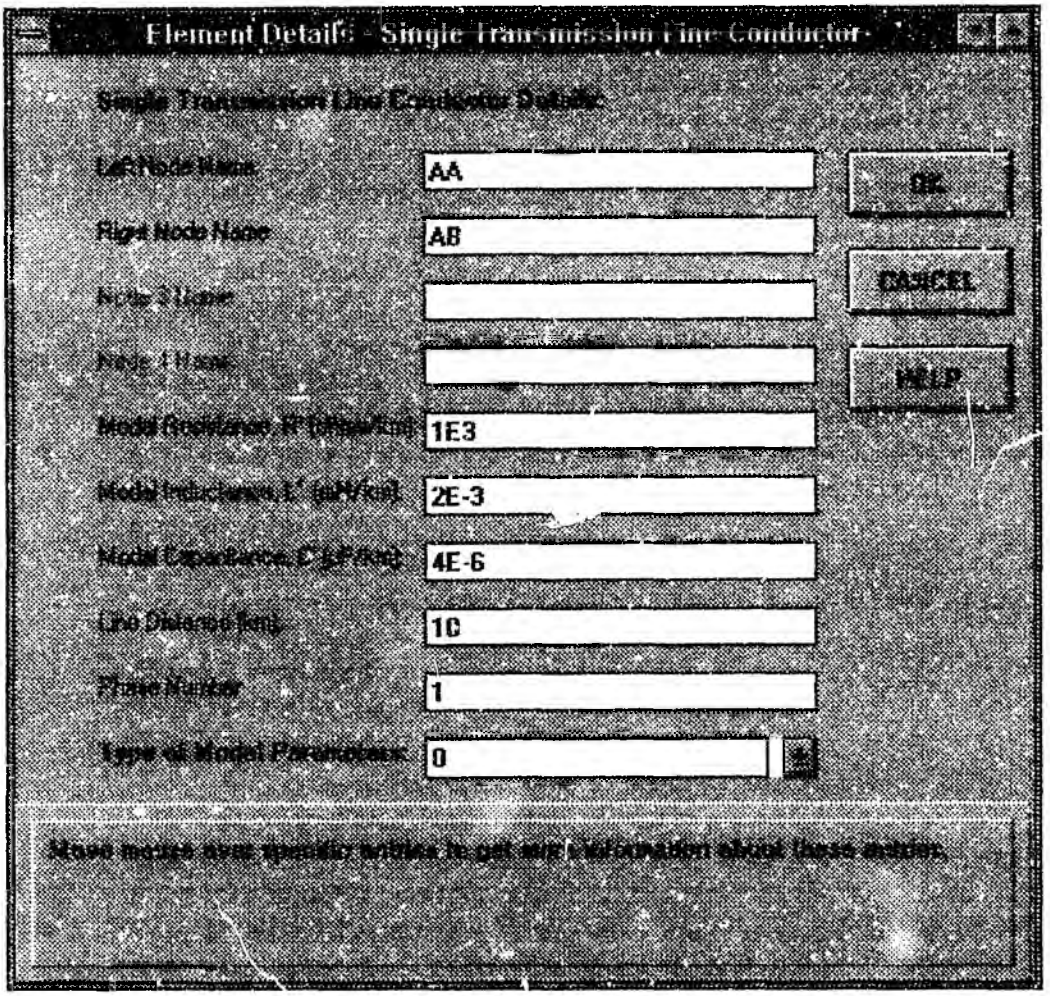


Figure 7.8 - The Single Transmission Line Conductor Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Left Node Name** - The name of the node to the left of the RLC branch (If this node is earth, leave the name blank).
- Right Node Name- The name of the node to the right of the RLC branch (if this node is earth, leave the name blank).
- **Node3 Name** - The name of the left node of an identical RLC branch (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)
- **Node4 Name** - The name of the right node of an identical RLC branch (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)

- **R** - R is the modal distributed resistance (if any) specified in Ω/km.

- **L** - This entry has one of a number of possibilities depending on which radio button was selected for the type of modal parameters specified. If type 1 is selected:
  - L is the modal distributed inductance (if any) specified in either mH/km or Ω/km, depending on which radio button is selected (see L and C radio buttons below).

  If type 2 is selected:
  - L is the modal surge impedance of the line in Ω.

  If type 3 is selected:
  - L is the modal surge impedance of the line in Ω.
- **C** - This entry has one of a number of possibilities depending on which radio button was selected for the type of modal parameters specified. If type 1 is selected:
  - C is the modal distributed capacitance (if any) specified in either μF/km or μmhos/km depending on which choice is selected (see L and C combo boxes below).

  If type 2 is selected:
  - C is the modal propagation velocity in km/sec.

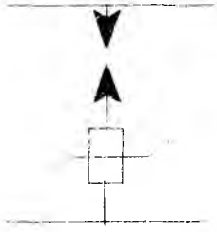
  If type 3 is selected:
  - C is the modal travel time in sec (must exceed the simulation time step (see Setup Simulation).
- **Line Distance** - The length of the line in km. Must be entered as a negative number to signal an untransposed line.
- **Type of Modal Parameters** combo box - Flag for specifying which type of modal parameters is required from type 1, 2, or 3.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.3.3 The Piecewise-linear model Surge Arrestor

The method used by ATP for the solution of circuits containing piecewise-linear models precludes it's use where the waveform is not well behaved (eg. where a non-linear resistor is connected in parallel with a linear inductor - the voltage across the linear inductor is proportional to the rate of change of the inductor current which can be discontinuous).

If we consider a non-linear resistive element with a series spark gap



The parameters of this branch include:

- The branch voltage at which breakdown of the spark gap occurs (zero if no spark gap actually present)
- the v-i characteristic of the non-linear resistive element
- the arc extinguishing behaviour of the spark gap (extinguishing at a current zero, or (optionally) if the branch voltage (voltage across both the arc and the resistive element) drops below some specified value)

To select the Piecewise-linear model surge arrestor, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of element, the Piecewise-linear Spark Gap Details Screen will appear (see *figure 7.9*).

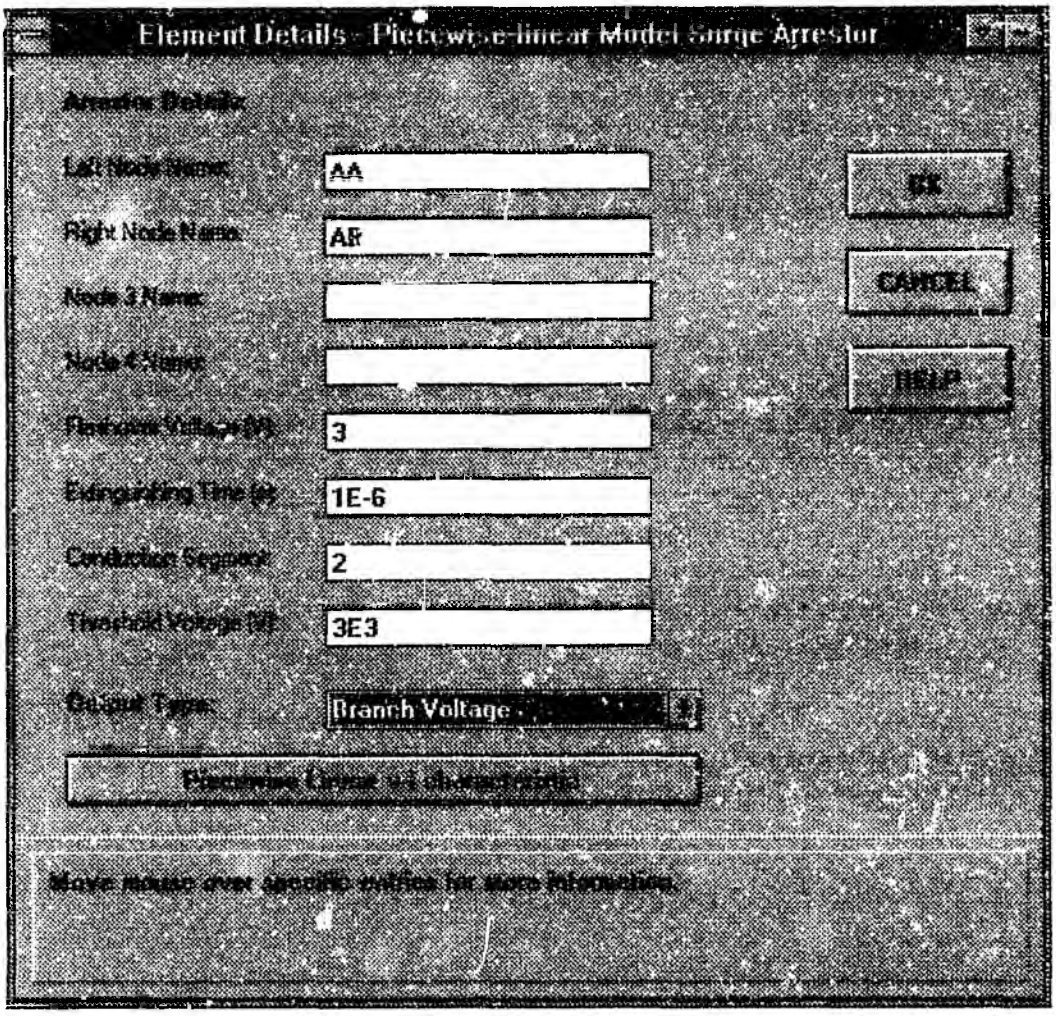


Figure 7.9 - The Piecewise-linear Surge Arrestor Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Left Node Name** - The name of the node to the left of the Surge Arrestor (If this node is earth, leave the name blank).
- **Right Node Name**- The name of the node to the right of Surge Arrestor (if this node is earth, leave the name blank).
- **Node3 Name** - The name of the left node of an identical surge arrestor (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)

- **Node4 Name** - The name of the right node of an identical surge arrestor (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)
- **Flashover Voltage** - Voltage at which breakdown of the spark gap occurs in volts.
- **Extinguishing Time** - The earliest time after flashover that the arc can extinguish at a current zero in seconds.
- **Conduction Segment** - Segment of the piecewise-linear v-i characteristic at which conduction begins after flashover.
- **Threshold Voltage** - The threshold value of branch voltage for arc extinction, in volts. If no threshold value exists, then leave blank.
- **Output Type** combo box - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.
- **Current Voltage Characteristics** - This command button opens a screen which allows for the input of **i(k)** and **v(k)** (see *figure 7.10*) - the k'th point on the v-i characteristic in amps and volts of the **ZnO arrestor.**

Once all the entries in this screen as well as in the *Current Voltage Characteristics* screen, have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

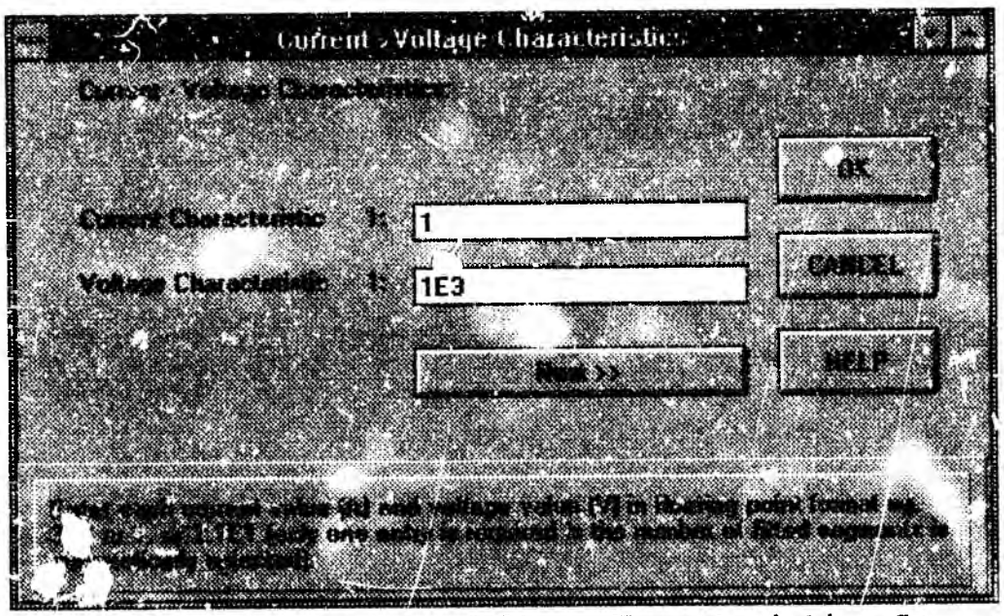
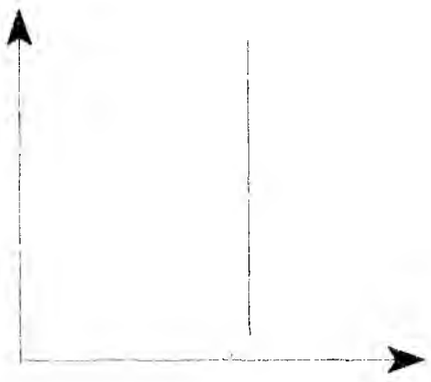


Figure 7.10 - The Current Voltage Characteristics Screen

7.3.4 The Exponential Model Surge Arrestor

The exponentially modelled v-i characteristic is as follows:



with defining function

$$i = \text{MULT}\,[(V)/(VREF)]^{EXPON} \qquad (7.2)$$

A **Zinc-Oxide** (ZnO) surge arrestor can be modelled using contiguous exponential v-i characteristic segments. To select the exponentially modelled surge arrestor, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of element, the exponentially modelled Spark Gap Details Screen will appear (see figure 7.11).

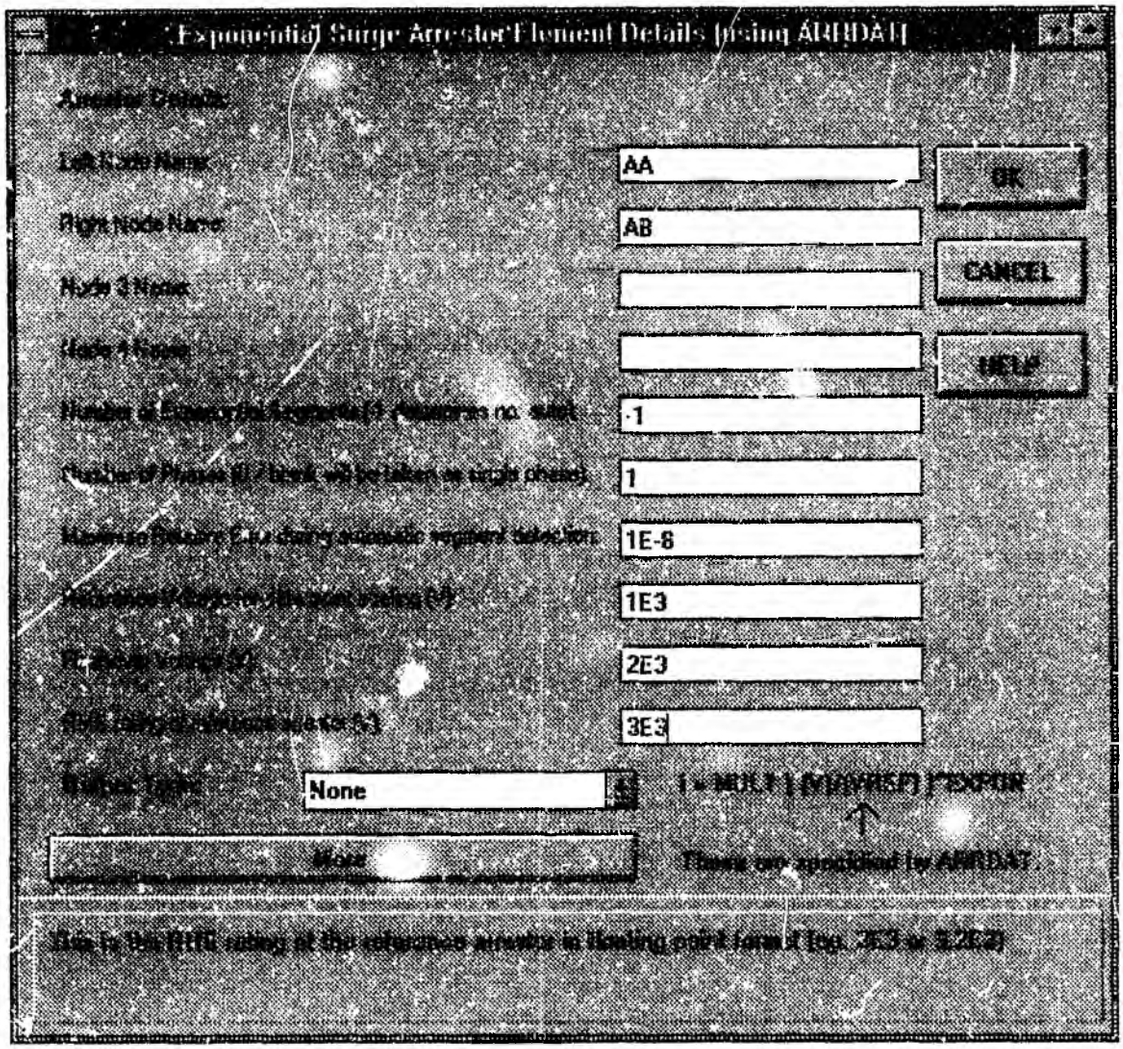


Figure 7.11 - The Exponentially Modelled Surge Arrestor

The user is required to complete every entry in the screen. These entries are the following:

- **Left Node Name** - The name of the node to the left of the Surge Arrestor (If this node is earth, leave the name blank).
- **Right Node Name**- The name of the node to the right of Surge Arrestor (if this node is earth, leave the name blank).
- **Node3 Name** - The name of the left node of an identical surge arrestor (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)
- **Node4 Name** - The name of the right node of an identical surge arrestor (if one exists, entering the node names allows the user to ignore the rest of the entries for this branch since they are the same as the other branch described)

- **Flashover Voltage** - Normalised voltage at which breakdown of the spark gap occurs in volts. For a gapless arrestor, enter any negative number such as -1.0.
- **VREF** - The value of VREF in the defining function see equation 7.2 above.
- **Number of Segments** - The number of segments of the piecewise-linear spark gap.
- **Output Type** combo box - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.
- **More** button - Selecting this button calls the ARRDAT form. This function calculates the MULT-EXPON values of ZnO surge arrestors (see *figure 7.12*). This is the only option offered in ATP for Windows (ie. you must use ARRDAT for the exponentially modelled surge arrestor.

Once all the entries in this screen as well as in the *More* screen, have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

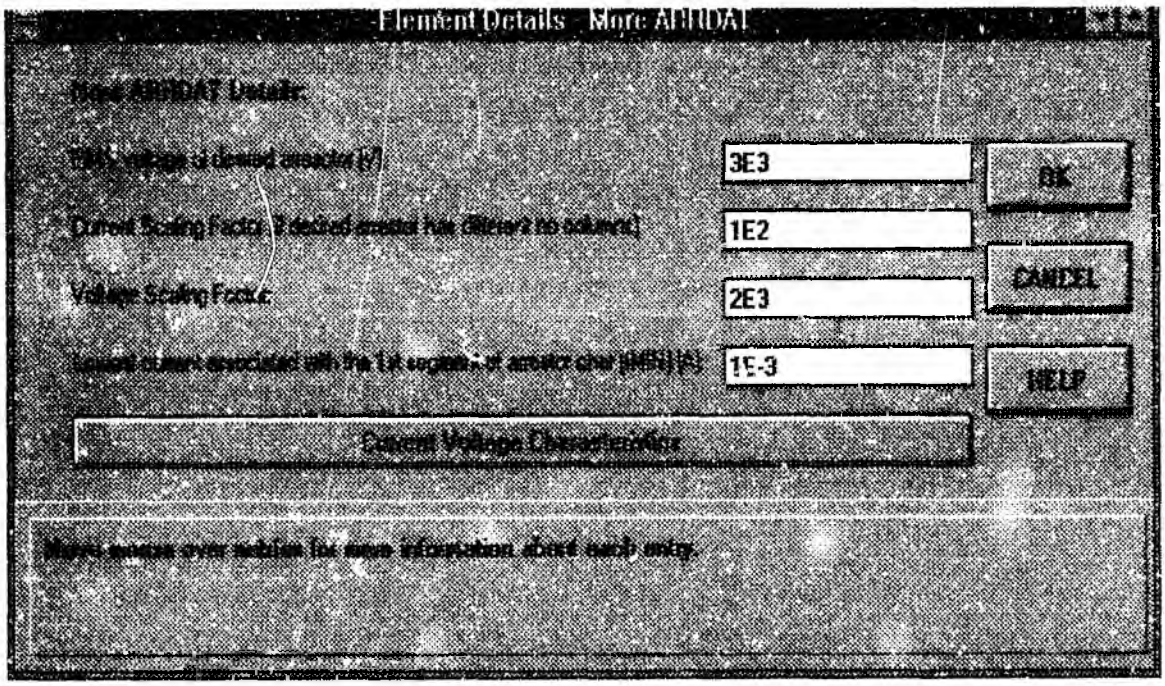


Figure 7.12 - The ARRDAT Element Details Screen

ARRDAT is an ATP support program which fits exponential curves to a set of v-i data points. The fitting is performed in the log-log plane using a least-squares algorithm. There are two options for the number of exponential segments:

- ARRDAT automatically chooses the number of segments and the boundaries of the segments according to a user specifies maximum permissible relative error
- user specifies both the number of segments and the boundaries of the segments.

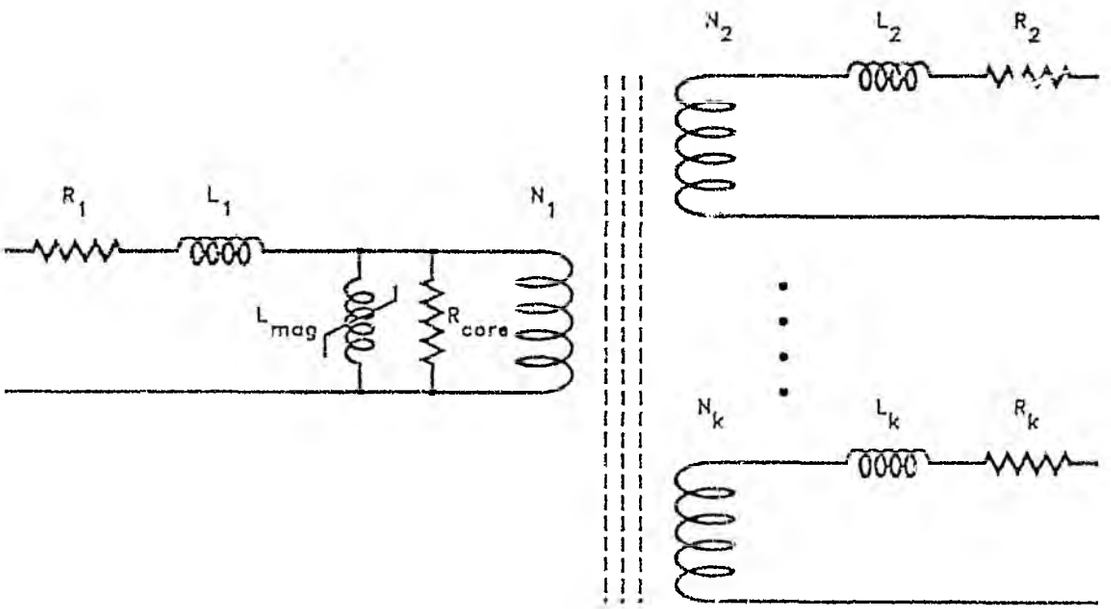
The user is required to complete every entry in the screen. These entries are the following:

- **RMS rating of reference arrestor** - This is the RMS voltage rating of the desired arrestor.
  Current Scaling Factor - This is the current scaling factor (if the desired arrestor has a different number of columns compared with the reference arrestor), in floating point format, in Amps..
- **Voltage Scaling Factor** - Any additional voltage scaling factor, in floating point format. in volts (usually enter 1.0).
- **Lowest Current** - The lowest current associated with the first segment of the arrestor characteristic - cannot be zero since the log of 0 is not defined, in floating point format, in amps. Set to a small value such as 0.001.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.3.5 Transformers

ATP allows the modelling of multi-winding single phase transformers which may exhibit non-linearity in their flux versus magnetizing current characteristics.
The equivalent circuit is



The magnetisation branch L_{mag}, is assumed to be linear during steady state initialisation ($t < 0$) and is specified as a single point on the flux versus magnetisation current characteristic. Symmetry is assumed to apply for the first and third quadrants.

The magnetisation branch, L_{mag}, during the simulation ($t > 0$) is specified as a piecewise

linear flux versus magnetisation current characteristic where the characteristic is assumed to pass through the origin. Symmetry is assumed to apply for the first and third quadrants.

The first segment of the piecewise linear characteristic ($t > 0$) must have the same slope as that for the linear characteristic used for steady state initialisation ($t < 0$). The phase angles of the source voltages may have to be adjusted to avoid a sudden jump to the second segment at $t = 0$ (all fluxes must have values within the first segment of the piecewise linear characteristic immediately prior to $t = 0$).

If a three phase transformer were constructed from the single phase transformers then the previous model would be used. However the more common situation is where the three phase transformer is constructed on a single core. While the windings are independent for positive and negative sequence mmfs and hence can be analysed as three independent single-phase transformers; where there is a zero sequence mmf, the associated core reluctance is a function of the core type.

To select an ATP transformer, follow the instructions given in 7.1. Once the *Element Details* menu has been selected for this type of element, the Transformer Details Screen will appear depending on which element you selected (ie. single phase or three phase - see *figures 7.13 and 7.15*).

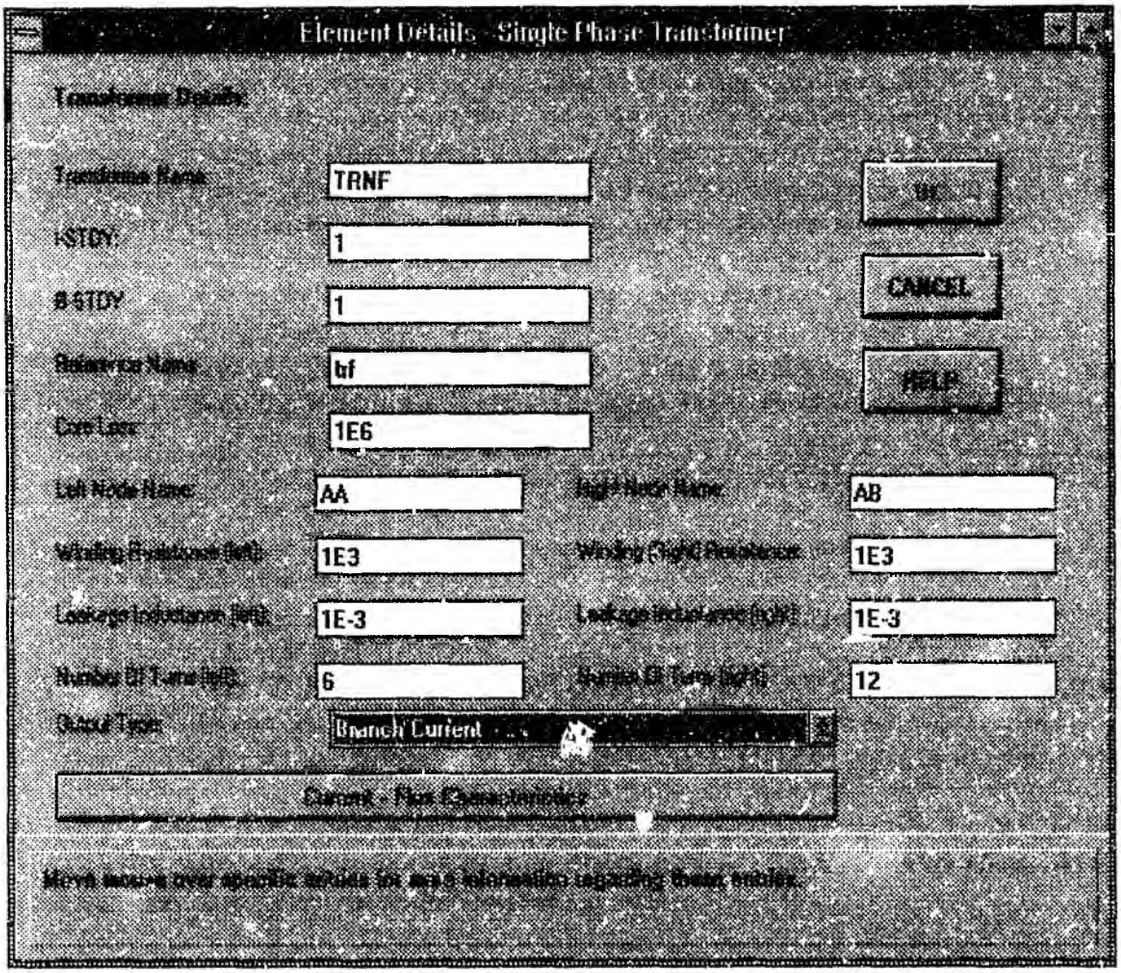


Figure 7.13 - The Single Phase Transformer Element Details Screen

To create a new single phase transformer, the following steps are required:

- Fill in the transformer details of the following:
  - **Ref Name** - The reference name of an identical transformer (name entered in the *Transformer Name* field of that transformer) whose parameters are to be duplicated for this transformer.
  - **ISTDY** - The current component of point on linear characteristic of the flux versus magnetising current characteristic to be used in the steady state initialisation, in amperes.
  - **ΨSTDY** - The flux component of point on linear characteristic of the flux versus magnetising current characteristic to be used in the steady state initialisation, in webers.
  - **Transformer Name** - The transformer name (must be unique). This must

be no more than 6 alpha-numeric characters.

■ **Core Loss** - The core loss expressed as a linear resistance in parallel with the magnetisation inductance in Ω. The default value is ∞.

- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch
- **Left Node Name** - This is the left node name of the winding on the left hand side of the transformer (the right node is earthed).
- **Right Node Name** - This is the right node name of the winding on the right hand side of the transformer (the left node is earthed).
- **Winding Resistance (Left)** - this is the resistance of the left hand side winding in ohms.
- **Winding Resistance (Right)** - this is the resistance of the right hand side winding in ohms.
- **Leakage Inductance (Left)** - this is the leakage inductance of the left hand side winding in mH.
- **Leakage Inductance (Right)** - this is the leakage inductance of the right hand side winding in mH.
- **Current Flux Characteristics** command button - this command button calls the current flux characteristics screen, which allows the user to enter the current flux characteristic of this transformer point by point (see figure 7.14)

Clicking on the *Cancel* option at any time, cancels this operation, while clicking on *OK* saves all the changes made to this element, and exits from the *Circuit Element Menu*. To obtain on-line Help, select the *Help* button at any time.

The current flux characteristics screen is as follows:

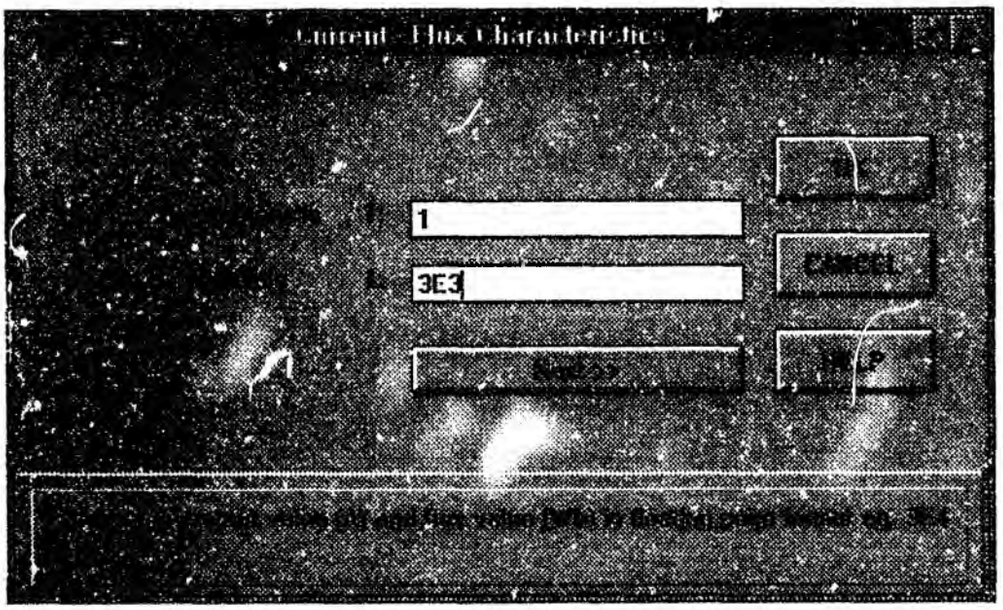


Figure 7.14 - the Current Flux Characteristics Screen

The user is required to enter every current-flux characteristic in the Current Characteristic and Flux Characteristic text boxes, and then click on the Next button to move to the next entry. When all the entries have been completed, click on OK.

7.3.6 The Three Phase Transformer

For the shell type transformer, the zero sequence mmfs see the same reluctance as the positive and negative sequence mmfs. This means that it can be modelled as three independent single phase transformers (see 7.3.15). The entries for the transformer are assumed to be the same for every phase of the transformer. The zero sequence mmfs for a Core-type transformer see a much larger reluctance as the zero-sequence flux must return outside the core. This transformer can therefore be modelled as three single phase transformers, but with a different reluctance of the zero-sequence flux path. This means that if the core-type transformer is selected, the zero-sequence flux path entry will be available whereas with the shell-type, this entry will be greyed out. The screen for the three phase transformer is as follows:

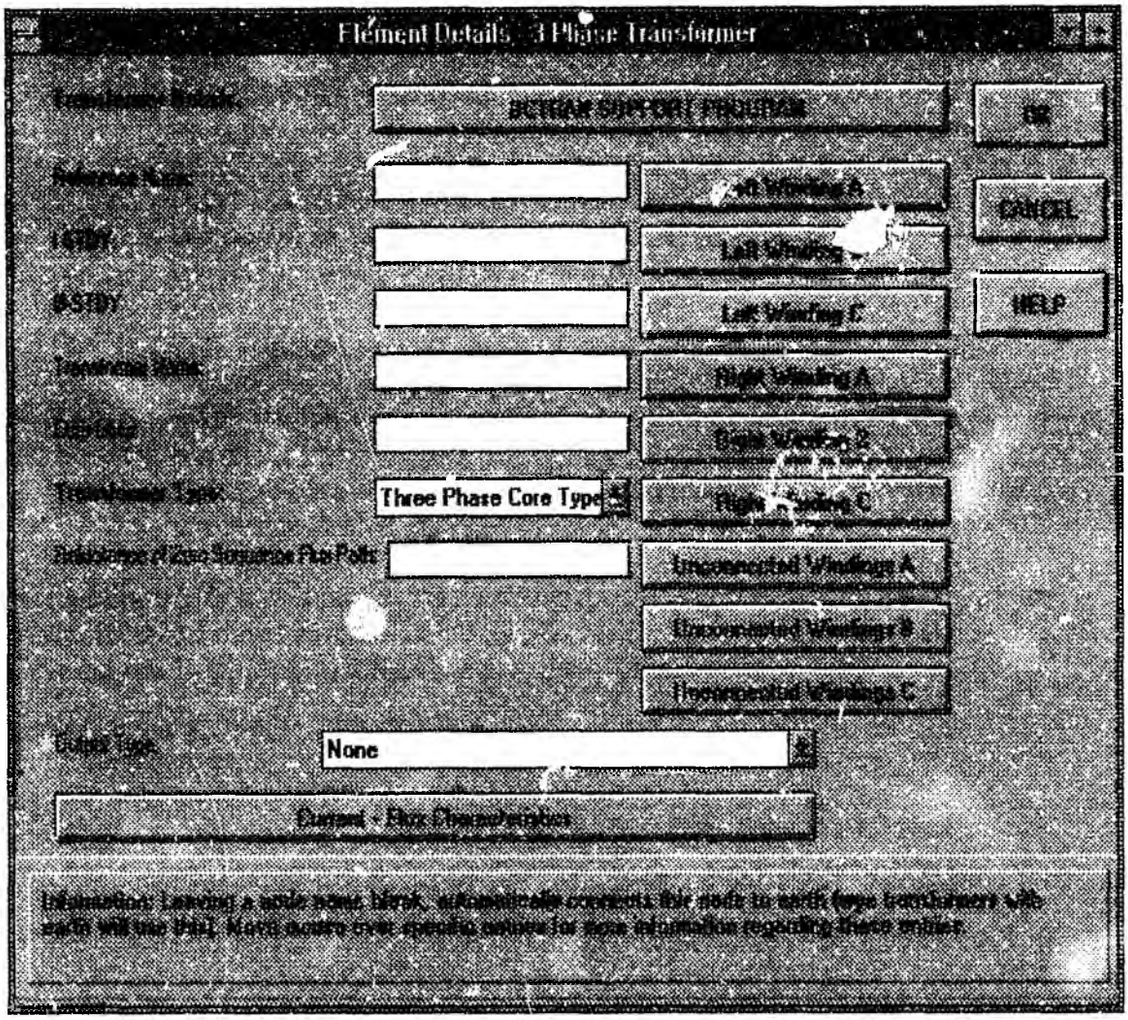


Figure 7.15 - The Three Phase Transformer Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Ref Name** - The reference name of an identical transformer (name entered in the *Transformer Name* field of that transformer) whose parameters are to be duplicated for this transformer.
- **ISTDY** - The current component of point on linear characteristic of the flux versus magnetising current characteristic to be used in the steady state initialisation, in amperes.
- **ΨSTDY** - The flux component of point on linear characteristic of the flux versus magnetising current characteristic to be used in the steady state initialisation, in webers.
- **Transformer Name** - The transformer name (must be unique). This must be

no more than 6 alpha-numeric characters.

- **Core Loss** - The core loss expressed as a linear resistance in parallel with the magnetisation inductance in Ω. The default value is ∞.
- **Reluctance of zero-sequence flux path** - (only valid for the core type transformer) - self explanatory (see explanation of at beginning of three phase transformer section.
- **Output Type** combo boxes - This is a flag to indicate which type of output is required from the primary windings. The options are *none*, and *primary current*.
- **Left Winding (A - C)** command buttons - these command buttons will produce the winding details screen for each of these windings (see *figure 7.16*). the left side is generally used as the primary side, however this is not essential.
- **Current-Flux characteristics** command button - this command button calls the current flux characteristics screen (see *figure 7.14*)

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain On-line help, select the *HELP* button at any time.

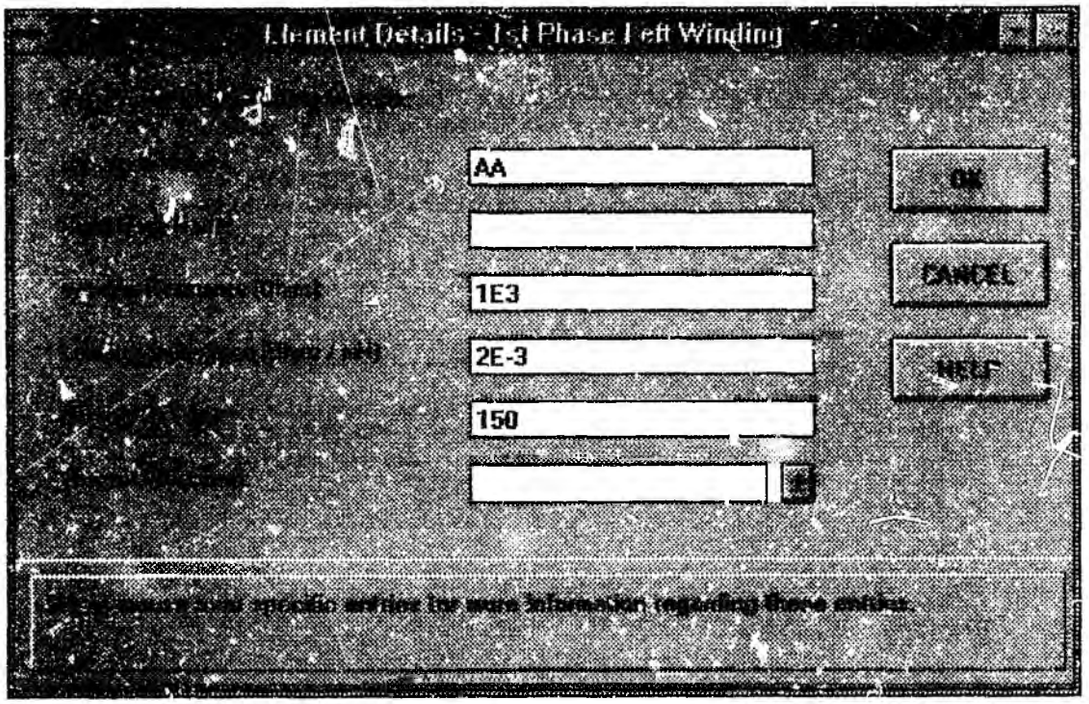


Figure 7.16 - The Winding Element Details Screen

- **Left Node Name** - The name of the left node of the relevant winding of the transformer.
- **Right Node Name** - The name of the right node of the relevant winding of the transformer
- **Winding Resistance** - The Winding resistance of the (n)th winding of the transformer in Ω.
- **Leakage Inductance(n)** - The winding leakage inductance of winding (n) of the transformer in mH or Ω (see Setup Simulation - Inductance units).
- **Number of turns(n)** - The number of turns of winding (n)
- **Output Required** - this is used to indicate if any output is required from this particular winding.

7.3.7 Black Box Element

ATP for Windows allows for the use of a Black Box element. This element places an *$Include* file in the data case, that includes elements which have already been previously designed for ATP. To select Black Box element, drag and drop the Black Box element from the Circuit Element Panel. Right click on this element, and select the Element Details menu option. This will produce the screen as seen in *figure 7.17.*

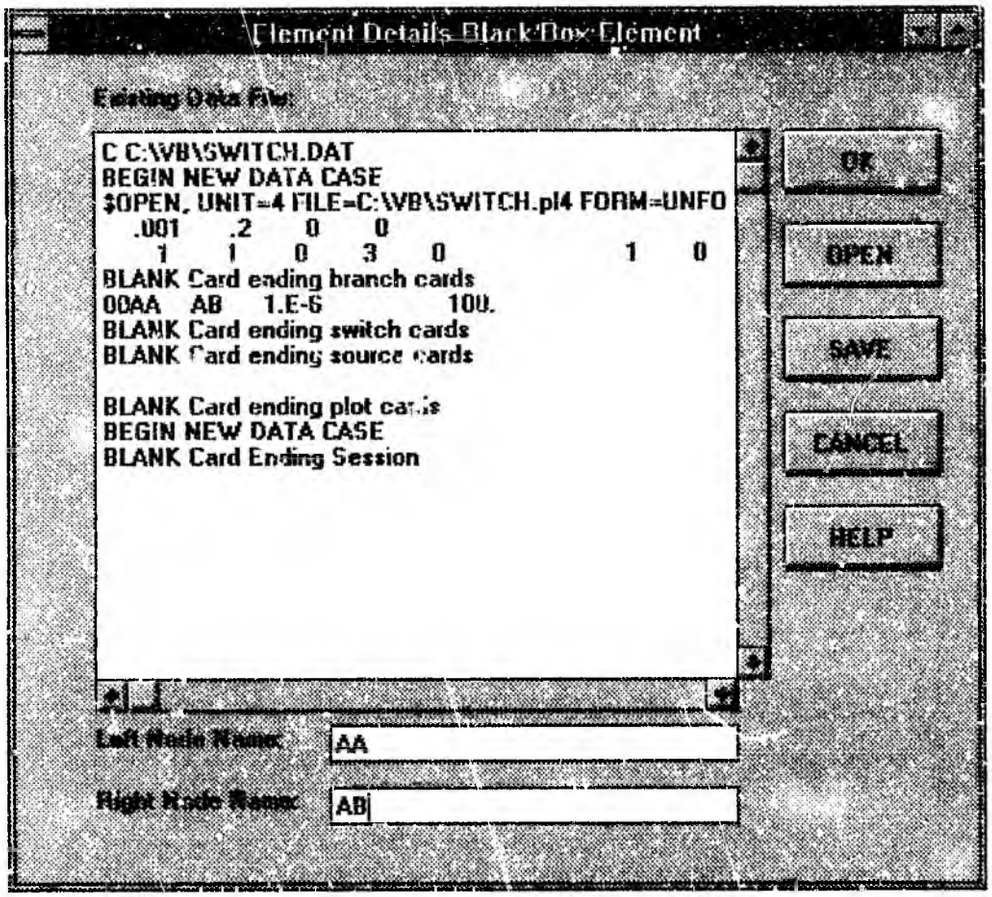


Figure 7.17 - The Black Box Element Details Screen

To open an existing ATP data file, select the *OPEN* button. This will produce a standard *File Open* screen, and allow the user to select a text / data file. Once this file has been selected, click on the *OK* button to continue. This will automatically cause the selected ATP file to be included in the simulation. For on-line Help, select the *HELP* button at any time.

7.4 Switch Card Elements

7.4.1 Switches

Invariably transient waveforms are influenced by the instant in the cycle at which the switch (or switches) operate. To select any switch, select the switch icon from the *Circuit Element Panel*, place it in the *Circuit Region*, and right click on the element. Selecting the Element Details option will produce the Switch Element Details screen (see figure 7.18).

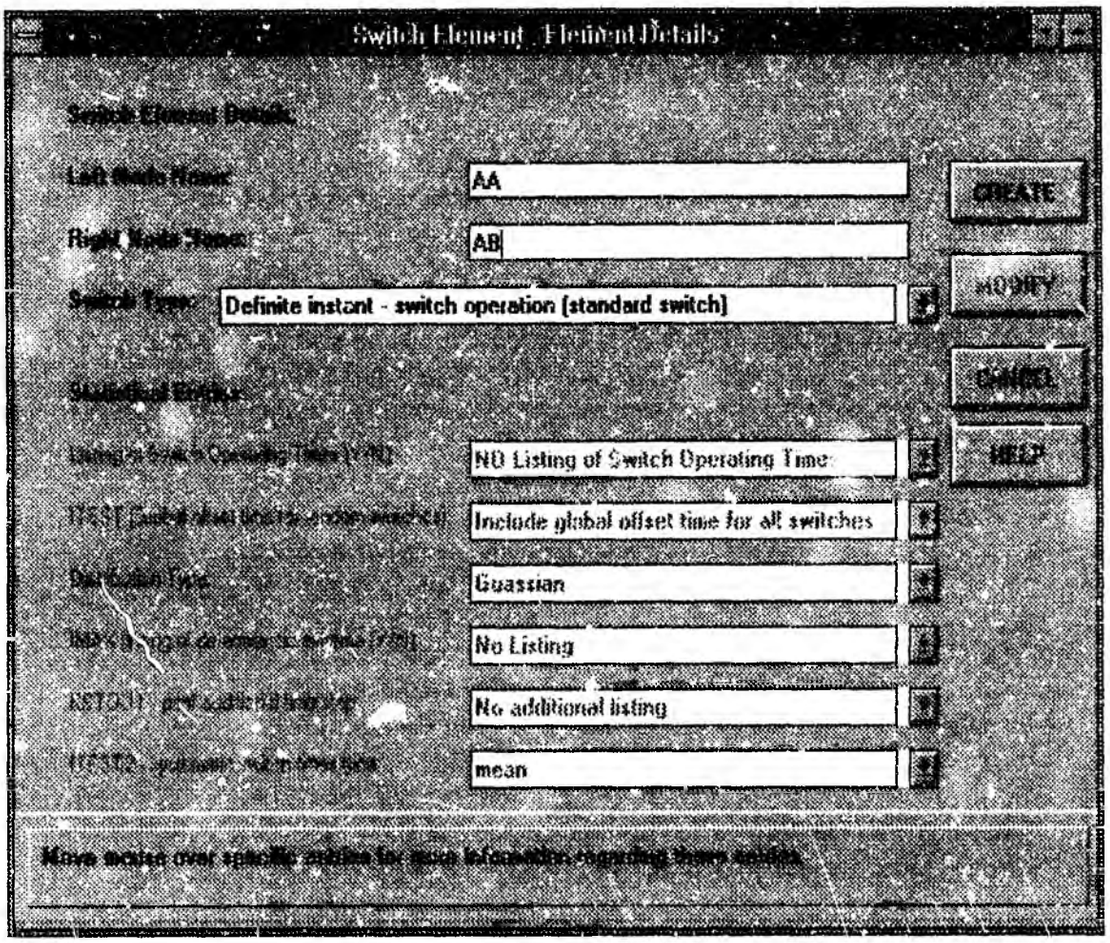


Figure 7.18 - The Switch Element Details Screen

To create a new switch element, the following steps are required:

- Now name the nodes to the left and right of the element by clicking on the space provided for the names of these nodes, and editing the default names automatically assigned by **ATP for Windows**, or the names previously given to these nodes if this editing is necessary. If one or both of these nodes have been named in the description of a touching element, there is no need to rename them.
- lect the type of switch required from the *Element Type* '' t by clicking on the combo box corresponding to the required type. The types available are:
  - Definite instant - switch operation (standard switch)
  - Random instant - switch operation -> closing only
  - Random instant - switch operation -> opening only

- Slave Closing Switch
- Slave Opening Switch
- Systematically increasing instant of operation -> closing switch
- Constant Delay Slave Switch

- There are a number of flag options which must be selected in this screen either before or after creating a switch. All of these are used for specific switches, and need not be filled in for every switch. The default values are automatically used if none of the combo boxes are selected. These are the following (see *Type of Element and setup flags* box):
  - **Listing of Switch Operating Times/No Listing of Switch Operating Times** - This is a flag for controlling the listing of switch operating times. Selecting the *Listing of Switch Operating Times* option will list these while selecting the *No Listing of Switch Operating Times* (default) will suppress this listing.
  - **ITEST** - This is a flag for controlling the global offset times for random switches. The options are:
    - Include global offset time for all switches (default).
    - No global offset time
    - Include global offset time only for closing switches
    - Include global offset time only for opening switches.
  - **Distribution Type** - This is a flag for controlling the distribution for the random switch operating times. The options are *Gaussian* (default) and *uniform*.
  - **IMAX** - This is a flag for controlling the listing of deterministic extrema for each energisation. The options are *no listing* (default) and *listing*.
  - **KSTOUT** - This is a flag for controlling the printout of additional time step data for each energisation. The options are *no additional listing* (default) or *additional listing*.
  - **ITEST2** - This is a flag for controlling whether systematically decreasing switch times T are to be the median or minimum times (systematic and const delay slave switches). The options are *mean* (default) or *minimum*.

- Now click on the *Create* button on the right of the screen. This will automatically open the Modify/Create Switch screen corresponding to the type of switch chosen.

It the element already exists, and you want to change some of the parameters of the element, select the *Modify* option. This will open the *Modify/Create Switch screen* corresponding to the type of switch chosen.

Clicking on the *Cancel* option at any time, cancels this operation, while clicking on *OK* saves all the changes made to this element, and exits from the *Circuit Element Menu*. To obtain on-line Help, select the *Help* button at any time.

7.4.1.1 The Standard Switch (Definite instant - switch operation)

To select the standard switch, follow the instructions given in 7.4.1 Once the *Element Details* menu has been selected for this type of switch, the Standard Switch Details Screen will appear (see figure 7.19)

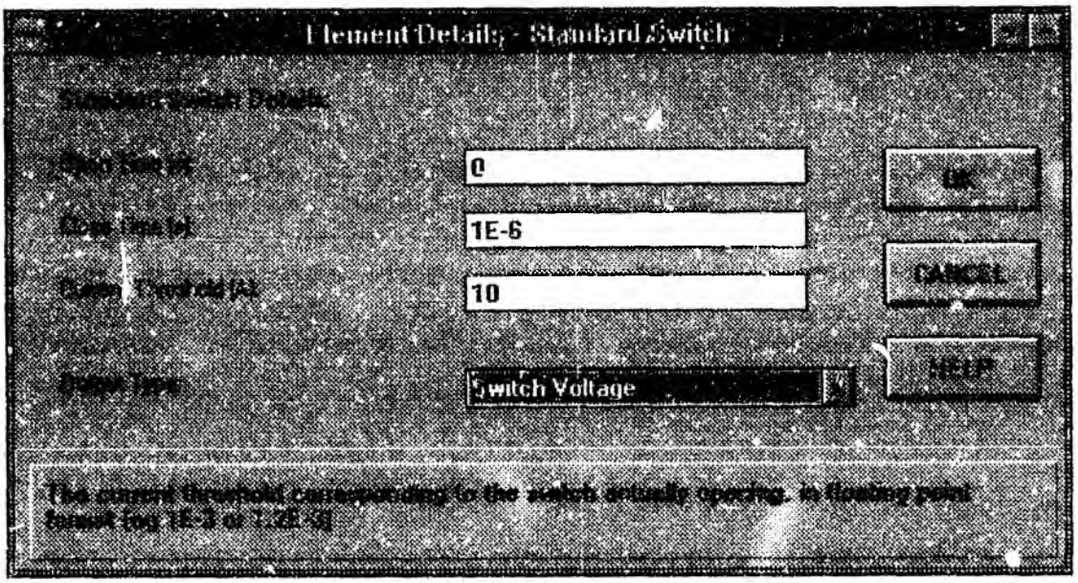


Figure 7.19 - The Standard Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Close Time** - The switch closing time in seconds. Entering -1.0 requests that the switch be closed during the initial establishment of steady state conditions.
- **Open Time** - The start of attempted switch opening in seconds. Entering 0.0 (or blank) requests the switch to open at the start of the simulation (must have been previously closed by setting the Close Time to -1.0).
- **Current Threshold** - The current threshold corresponding to the switch actually opening in amperes.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

To simulate arc extinction at current zero (or below some threshold value) the switch actually opens:

- when the current crosses zero after Open Time
- when the current drops below some threshold value (specified by the Current Threshold variable) after the Open Time

Obviously the current waveform after opening time would be determined by the external circuit behaviour

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.2 Closing Switch (Random instant - switch operation -> closing only)

A commonly occurring situation is where a mechanical circuit breaker opens or closes without regard for the position within the voltage cycle (for Monte Carlo analysis we repeat the simulation a large number of times with a spread of switch operating instants to get a representative spread of switch operating instants to get a representative spread of switching surge amplitudes).

First we give particular switch a random jitter about its own average closing time and then we give all switches a common (global) random offset.

To select the closing switch, follow the instructions given in 7.3.13. Once the *Element Details* menu has been selected for this type of switch, the Closing Switch Details Screen will appear (see *figure 7.20*).

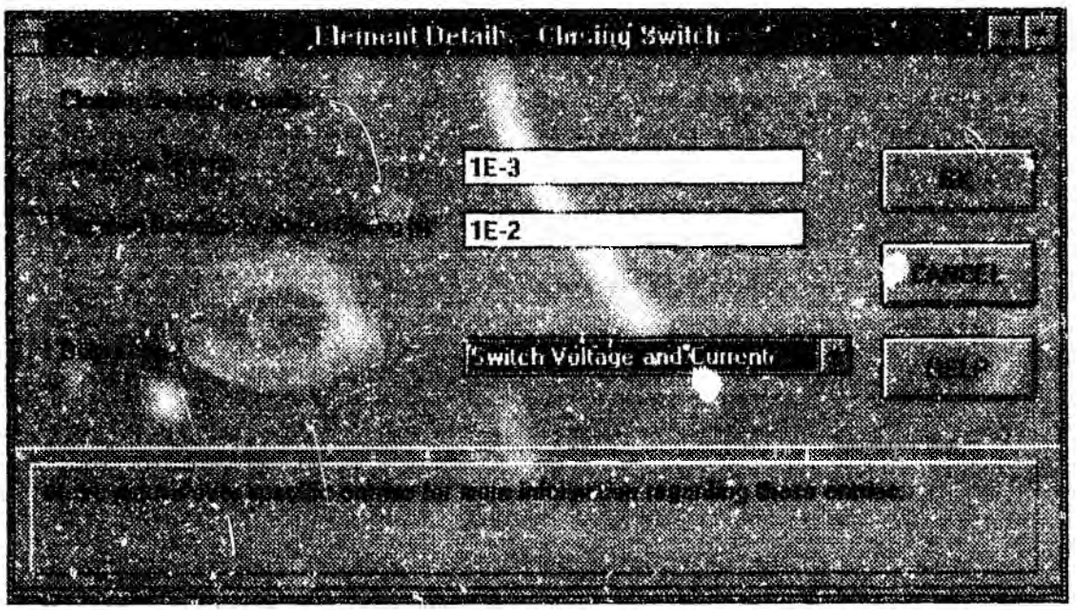


Figure 7.20 - The Closing Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Ave Close Time** - The average closing of the switch in seconds.
- **Jitter Deviation** - The standard deviation the jitter in the switch closing time in seconds.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

The random jitter about the average closing time can have a Gaussian or uniform distribution. The choice of Gaussian or uniform distribution can be made via the variable *Distribution Type* in the main *Switch Element Details Screen.*

7.4.1.3 Opening Switch (Random instant - switch operation -> opening only)

To select the opening switch, follow the instructions given in 7.3.13. Once the *Element*

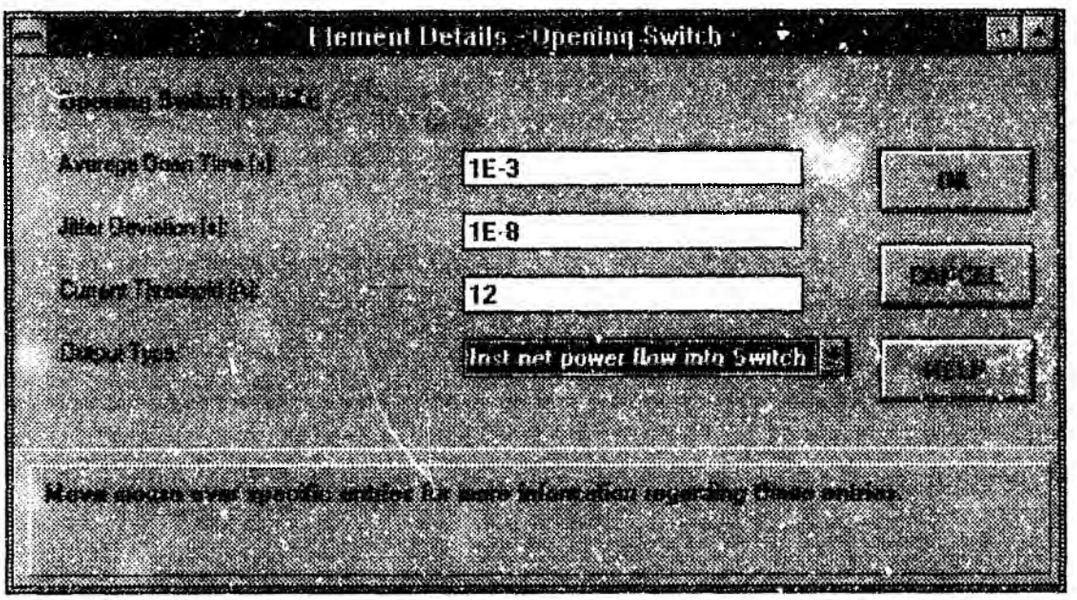


Figure 7.21 - The Opening Switch Element Details Screen

Details menu has been selected for this type of switch, the Opening Switch Details Screen will appear (see figure 7.21). The user is required to complete every entry in the screen. These entries are the following:

- **Ave Open Time** - The average opening time of the switch in seconds.
- **Jitter Deviation** - The standard deviation of the jitter in the switch opening time in seconds.
- **Current Threshold** - The current threshold corresponding to the switch **actually** opening, in seconds.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.4 The Slave Closing Switch

When multiple switches are involved a Gaussian or uniform delay characteristic can be modelled between the operating instants of slave switches and the operating instants of master switches (can be used for modelling circuit breakers having closing resistors).

The actual closing instant of a slave switch can be delayed with respect to the actual closing instant of a master switch according to the relationship

$$TCLOSE_{(slave)} = TCLOSE_{(master)} + TDELAY$$

The mean and standard deviation of TDELAY, and the fact that a switch is a slave of another switch (master switch) is indicated by having a separate slave switch for this purpose.

To select the slave closing switch, follow the instructions given in 7.3.13. Once the *Element Details* menu has been selected for this type of switch, the Slave Closing Switch Details Screen will appear (see figure 7.22).

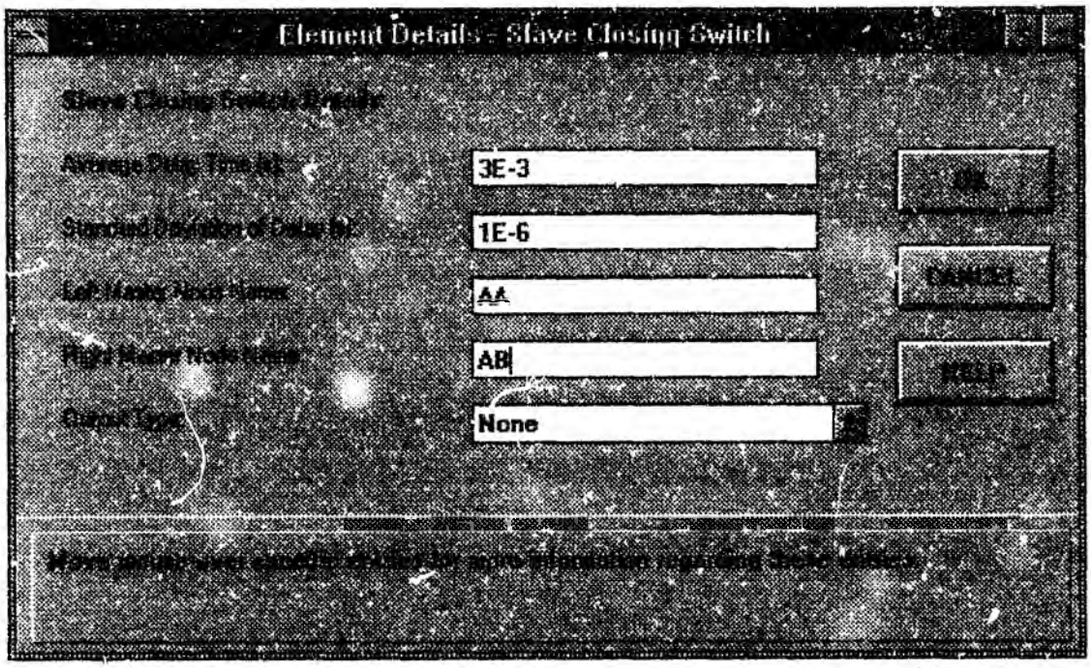


Figure 7.22 - The Slave Closing Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Ave Delay Time** - The average delay in closing of this slave switch with respect to the master switch, in seconds. The average time delay can be negative, but the resultant actual operating time may never be negative.
- **Standard Deviation of Delay** - This is the standard deviation of the delay in closing this slave switch with respect to the master switch in floating point format, in seconds.
- **Master Left Node Name** - The name of the left node of the master switch.
- **Master Right Node Name** - The name of the right node of the master switch.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.5 The Slave Opening Switch

This is the same as the slave closing switch, except that it opens instead of closing. The element details screen for this element can be seen in figure 23 below.

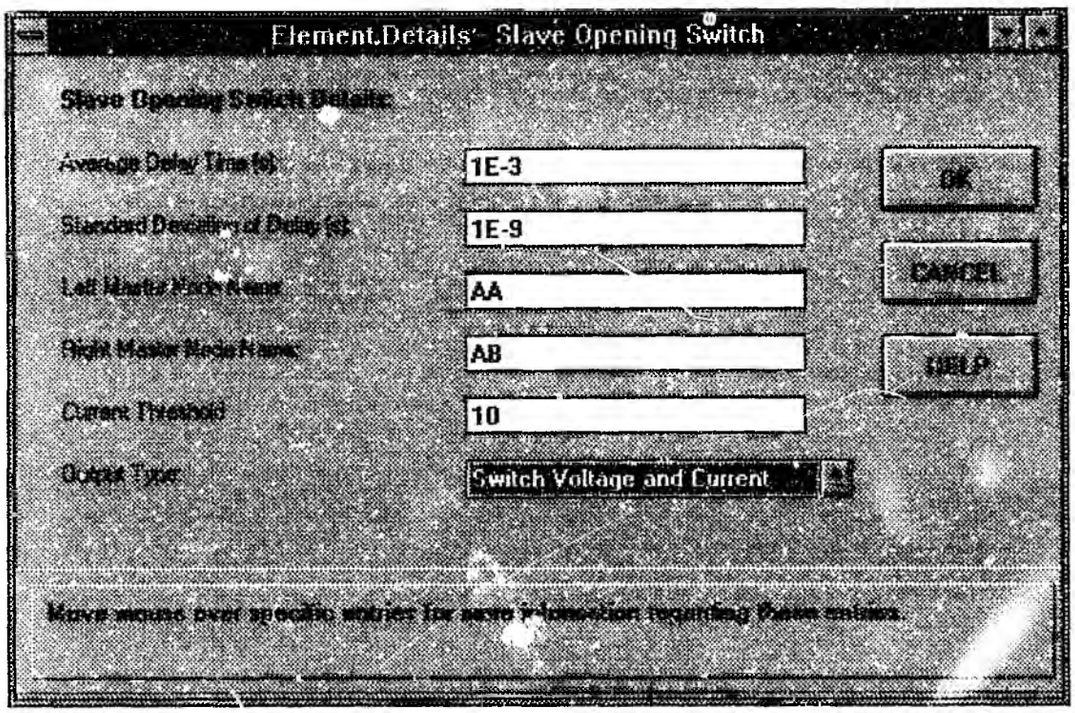


Figure 7.23 - The Slave Opening Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Ave Delay Time** - The average delay in closing of this slave switch with respect to the master switch, in seconds. The average time delay can be negative, but the resultant actual operating time may never be negative.
- **Standard Deviation of Delay** - This is the standard deviation of the delay in closing this slave switch with respect to the master switch in floating point format, in seconds.
- **Master Left Node Name** - The name of the left node of the master switch.
- **Master Right Node Name** - The name of the right node of the master switch.
- **Current Threshold** - This is the current threshold corresponding to the switch **actually** opening, in floating point format in amperes.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and

instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.6 Systematically increasing instant of operation -> closing switch

To investigate the effect of the instant of closure during the cycle, the switch operating time can be systematically increased by a constant value over a particular interval. See figure 7.24 below

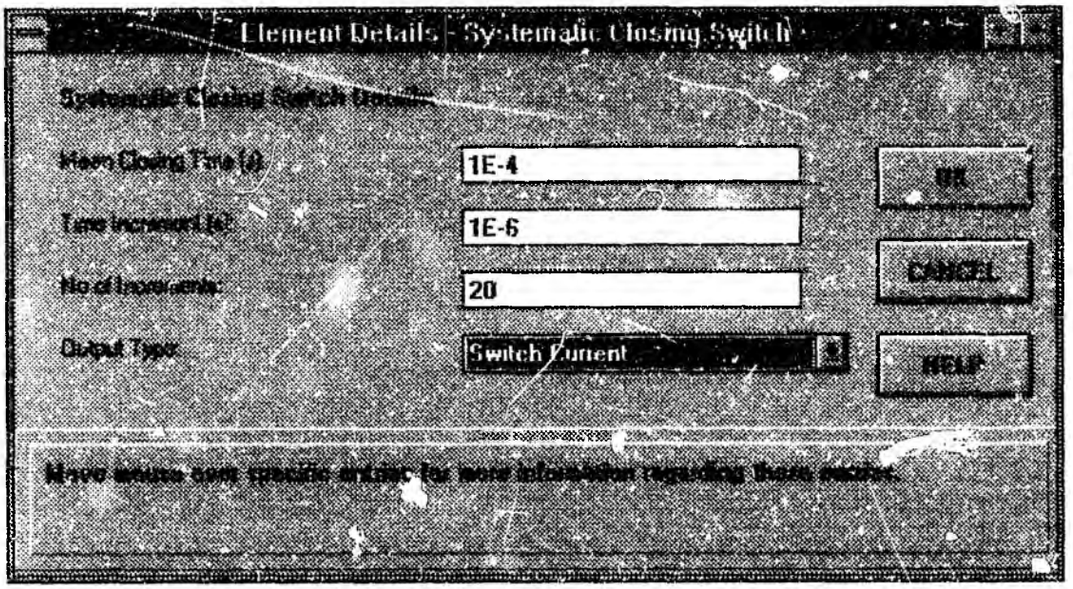


Figure 7.24 - The Systematically Closing Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Mean Closing Time** - This is the mean closing time (ITEST2 = mean), or the initial closing time (ITEST2 = minimum), in floating point format, in seconds (ITEST2 is an entry in the general Switch Element screen).
- **Time Increment(s)** - This is the time increment for the systematic increase of closing time in floating point format, in seconds.
- **Number Of Increments** - the number of increments in floating point format.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.7 The Constant Delay Slave Switch

Where multiple switches are involved, a constant delay can be modelled between the closing instants of switches and the closing instants of master switches. This can be used

for modelling circuit breakers having closing resistors.

$$TCLOSE_{(slave)} = TCLOSE_{(master)} + CONST$$

The Element Details screen for this element is as seen in ure 7.25 below.

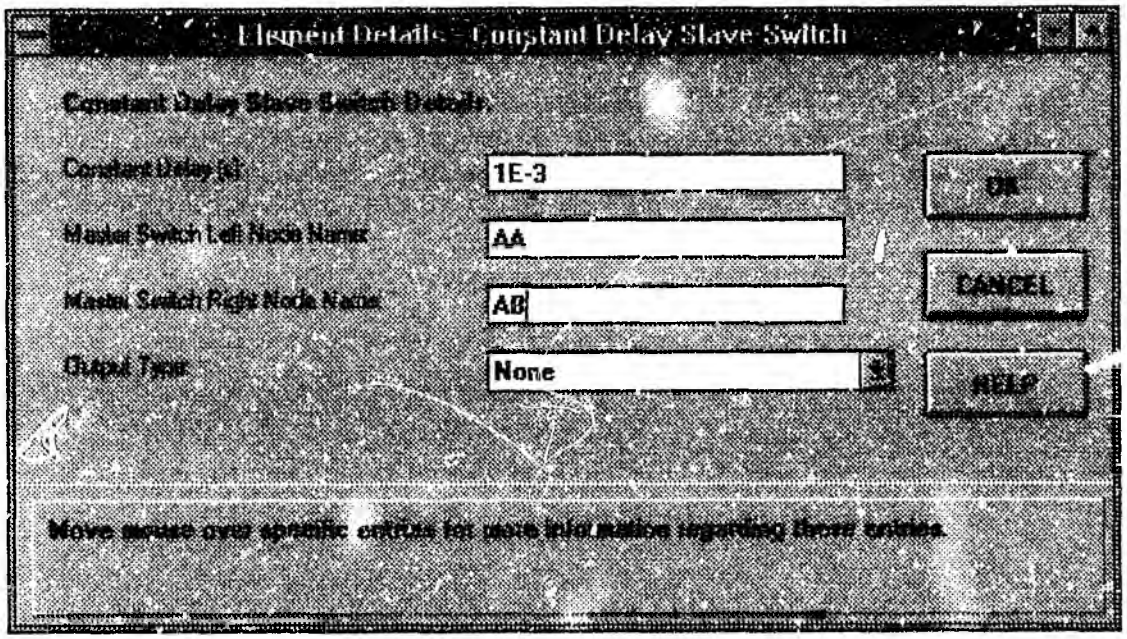


Figure 7.25 - The Constant Delay Slave Switch Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Constant Delay**- This is the constant delay in closing of this switch with respect to the master switch in seconds (can be negative).
- **Master Switch Left Node Name** - This is the name of the left node of the master switch.
- **Master Switch Right Node Name** - This is the name of the right node of the master switch.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

7.4.1.8 The Diode/SCR Element

The ATP diode starts to conduct when the forward voltage exceeds the value VIG (forward threshold voltage), and stops conducting when the forward current drops below the value IHOLD (holding current). THE ATP SCR (Silicon Controlled Rectifier) starts to conduct

when the forward voltage exceeds the value VIG, and when the TACS gate signal GRID becomes greater than zero (see Chapter 8 - using TACS elements), and stops conducting when the forward current drops below the value IHOLD. If the ATP SCR is forward biased again before the expiry of the recovery time (Recovery Time entry), following commutation, then it conducts again regardless of the value of the gate signal, GRID.

To select an ATP diode or SCR, follow the instructions given in section 7.1. This will produce the Element Details screen for this element (see figure 7.26).

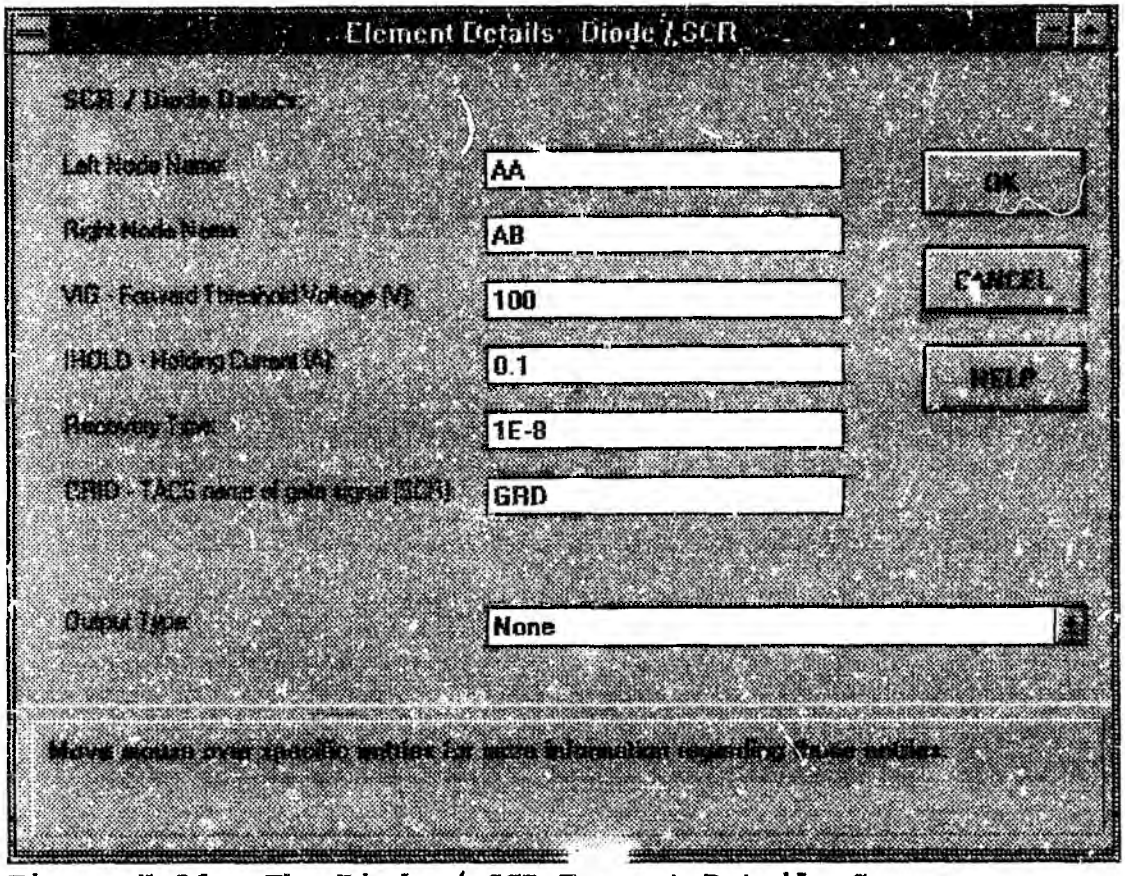


Figure 7.26 - The Diode / SCR Element Details Screen

The user is required to complete every entry in the screen. These entries are the following:

- **Left Node Name** - The name of the left node of the SCR or diode branch.
- **Node6 Name** - The name of the right node of the SCR or diode branch.
- **VIG** - The forward threshold voltage specified in volts. The default value is 0.0.
- **IHOLD** - The holding current specified in amperes. Default value is 0.0.
- **Recovery Time** - The recovery time specified in seconds. The default value is 0.0
- **GRID** - The TACS name of the gate signal for an SCR (see Chapter 7 Using TACS Elements). Leave blank for diode.
- **Output Type** combo boxes - Flag for specifying which type of output if any is required from the following: None, branch current, branch voltage (between terminal nodes), branch current and branch voltage, and instantaneous power into branch and net energy flow into branch.

Once all the entries in this screen have been completed, select the *OK* button to exit the

screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

8 USING TACS (TRANSIENT ANALYSIS OF CONTROL SYSTEMS) ELEMENTS

One of the functions offered by ATP, is the use of Control System parameters in High Voltage transient simulations. The TACS module attached to ATP allows the user to:

- express parameter relationships using FORTRAN statements
- express parameter relationships using controller blocks
- express parameter relationships using miscellaneous special devices, and
- create TACS controlled components such as resistors and inductors

FORTRAN statements, controller blocks and special devices can be readily interconnected.

All TACS controlled elements can be selected from the *Circuit Element Panel*.

8.1 How To Use It

Select the icon marked *TACS* with a **resistor** underneath the TACS writing, from the *Circuit Element Menu*, and place the element anywhere in the circuit presently in the *Circuit Region*, or select the *Select Element* option from the *Run* sub-menu, select the *TACS Element Library*, and then select TACS Element and move it to the required spot in the *Circuit Region* (see section 5.4 - Select Element). Once this has been done, right clicking on the TACS element, will produce the normal *Circuit Element Menu* associated with any circuit element in **ATP for Windows**. The *Element Details* option of a TACS element will automatically produce the TACS Element Details screen, but the user will be given the option of selecting the type of TACS element required (FORTRAN relationship, Controller block, or special device), (see figure 8.1 - The TACS Element Details Screen). To select the type of TACS element required, click on the combo box corresponding to the required type.

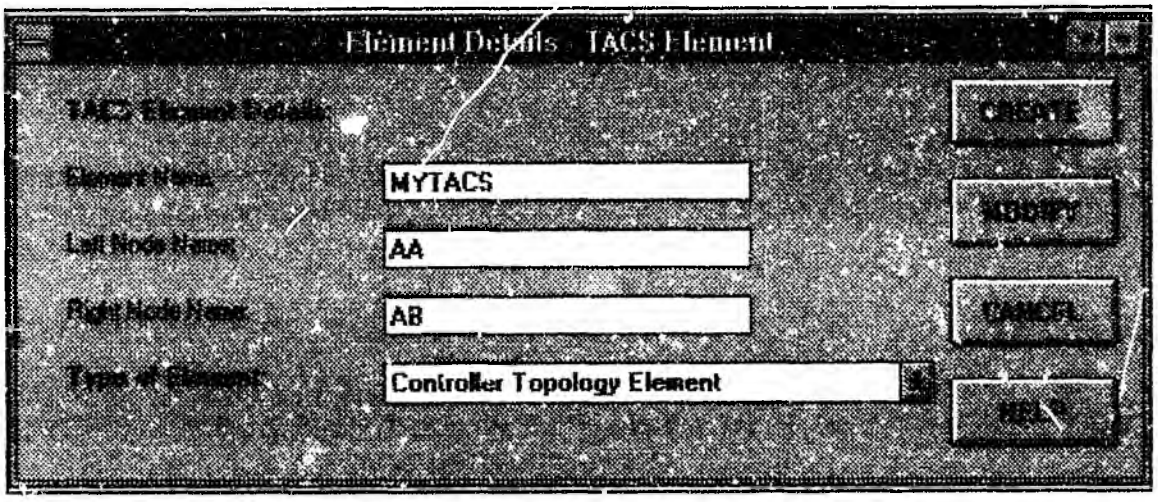


Figure 8.1 - The TACS Element Details Screen

To create a new TACS element, the following steps are required:

- name the element by clicking on the name space provided, and editing the name automatically given to the TACS Element by **ATP for Windows.**
- now name the nodes to the left and right of the element by clicking on the space provided for the names of these nodes, and editing the default names automatically assigned by **ATP for Windows**, or the names previously given to these nodes if this editing is necessary. If one or both of these nodes have been named in the description of a touching element, there is no need to rename them.
- Select the type of TACS element required as described above.
- Now click on the *Create* button on the right of the screen. This will automatically open the Modify/Create TACS element screen corresponding to the type of TACS element chosen (see figures 8.2 - 8.5).

If the element already exists, and you want to change some of the parameters of the element, select the *Modify* option. This will open the Modify/Create TACS element screen corresponding to the type of TACS element chosen (see figures 8.2 - 8.12).

Clicking on the *Cancel* option at any time, cancels this operation, while clicking on *OK* saves all the changes made to this element, and exits from the *Circuit Element Menu.* To obtain on-line Help, select the *Help* button at any time.

8.2 Creating a FORTRAN relationship TACS element

8.2.1 Function

Creating a FORTRAN relationship TACS element allows the user to use elements such as time varying resistors, and other elements which aid in the modelling of situations that can not be simulated directly using ATP for Windows.

8.2.2 How To Use It

Follow the instructions for creating a new TACS element in section 8.1, and select the *FORTRAN relationship* combo box in the TACS Element Details Screen. This will produce the FORTRAN TACS element screen (see figure 8.2).

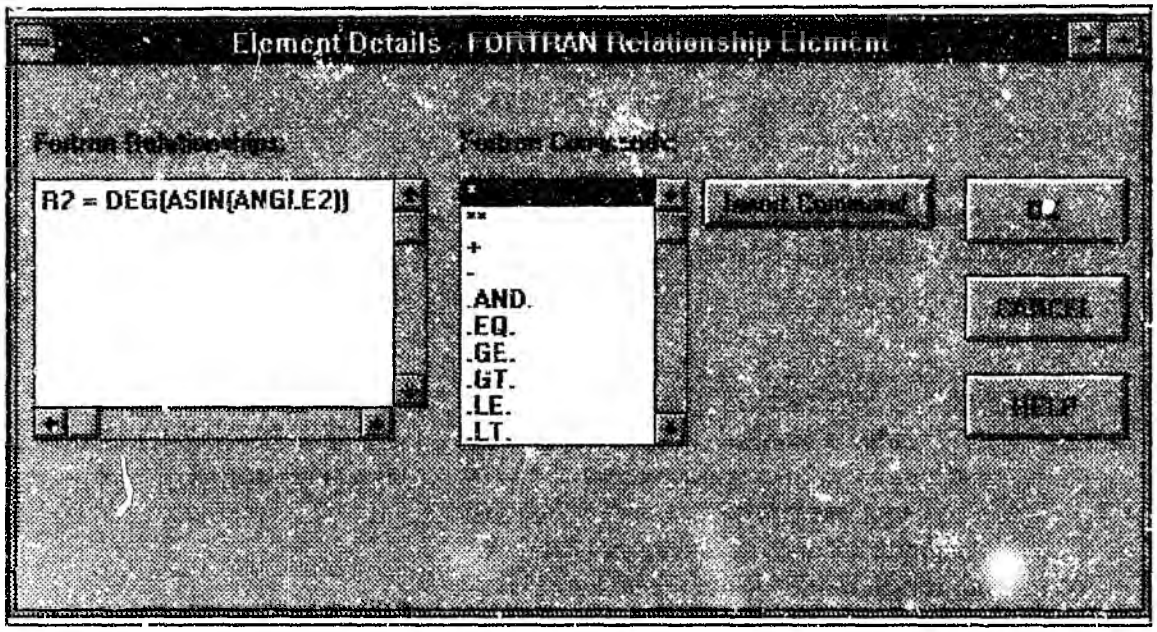


Figure 8.2 - The FORTRAN Relationship TACS Element Details Screen

To describe the TACS element using FORTRAN statements, simply type the FORTRAN statement required in the *FORTRAN relationships box*. To use a FORTRAN command, type it out, or select it from the list of FORTRAN commands on the right hand side of the screen, and then select the *Insert* option, or double click on the required command, or drag the command from the list into the Fortran Relationships list.

NB. All arguments for trigonometric functions must be given in RADIANS, and all answers to inverse trigonometric functions will be in RADIANS.

The following is a list of all the FORTRAN functions available, and an explanation of their meanings:

- algebraic operators: +, -, *, /, **
- logical operators: **.AND.**, **.OR.**, **.NOT.**
- relational operators: **.EQ.** - equal to,
  .NE. - not equal to,
  .LT. - less than,
  .LE. - less than or equal to,
  .GT. - greater than,
  .GE. - greater than or equal to.
- ordinary functions: **SIN** (argument in radians)
  COS (argument in radians)
  TAN (argument in radians)
  COT (argument in radians)
  SINH (argument in radians)
  COSH (argument in radians)
  TANH (argument in radians)
  ASIN (argument) with answer in radians
  ACOS (argument) with answer in radians
  ATAN (argument) with answer in radians
  EXP (argument)
  LOG (argument)
  LOG10 (argument)
  SQRT (argument) - square root
  ABS (argument) - absolute value
- special functions: **TRUNC** (argument) gives the integer part
  MINUS (argument) inverts the sign
  INVERS (argument) inverts the argument
  RAD (argument) converts from deg to rad
  DEG (argument) converts from rad to deg
  SIGN (argument) gives -1 (if < 0) or 1 (if >=0)
  NOT (argument) gives 0 (>0) or +1 (<=0)
  SEQ6 (argument) = INT(modulo -6(arg))
  RAN (argument) assigns a random number to the argument

Valid arguments are TACS parameter names, numeric constants (integer or real), and other FORTRAN functions.

The name of an argument in FORTRAN statements may not be longer than 6 alpha-numeric characters, may not include any of the following characters:

+, -, *, /,), (, and the SPACE character,
and output parameters may not start with a numeric character.
An example of a valid FORTRAN statement is:

ANGLE = DEG(ATAN(CNTRL - BIAS2)) + 36.2

A number of special parameters are available for use in FORTRAN statements. These are:

- **TIMEX** - simulation time, in seconds (= 0 during the initial establishment of steady state conditions).
- **ISTEP** - simulation time step number.
- **DELTAT** - simulation step size, in seconds.
- **FREQHZ** - power system frequency (of first sinusoidal source specified in EMTP data case), in Hz.
- **OMEGAR** - 2π * FREQHZ, in radians.
- **ZERO** - 0.0.
- **MINUS1** - -1.0
- **PLUS1** - +1.0
- **INFNTY** - $+\infty$ (very large number that still fits computer system)
- **PI** - π

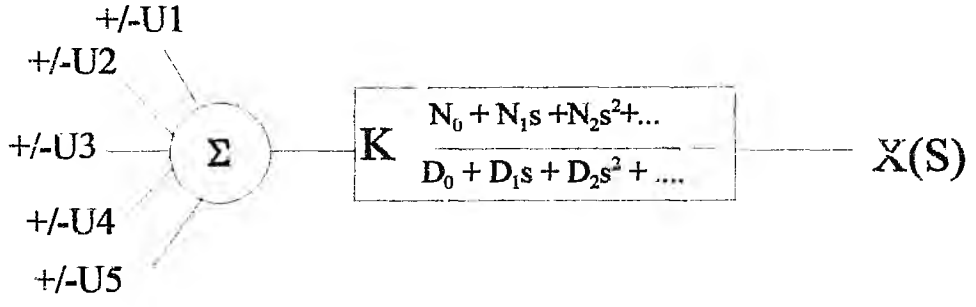
8.3 Creating a TACS element which uses control topologies

8.3.1 Function

Creating a TACS element which uses control topologies allows the user to use feedback control and transfer function blocks for a single ATP element.

8.3.2 How To Use It

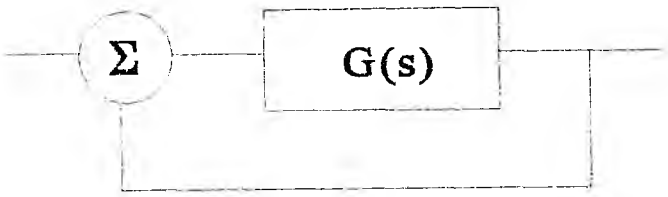
A controller topology generally has a form such as the simple example below:



The features of the controller are:

- feedback
- transfer function blocks (where the transfer function is expressed in terms of the Laplace variable, s, which in turn can be related back to a differential equation expressing the relationship between the single input parameter of the block, and single output parameter)

Controllers can be reduced to the following fundamental building block:



with the following transfer function:

$$X(s) = K\frac{N_0 + N_1 s + N_2 s^2 + \ldots}{D_0 + D_1 s + D_2 s^2 + \ldots}[\pm U_1 \pm U_2 \pm U_3 \pm \ldots \pm U_5] \quad (8.1)$$

The input parameters, U_n, can include:

- ATP node voltages
- ATP voltage and current sources
- ATP switch currents
- ATP switch statuses (open or closed)
- TACS internal waveform generators (sources)
- output parameters from other FORTRAN statements, controller blocks or special devices

The output parameter, X(s), can be:

- a TACS-controlled ATP component
- the input parameter of another FORTRAN statement, controller block or special device

Input and output parameters must have names with a maximum of six alphanumeric characters.

To create a TACS element which uses controller topologies, follow the instructions for creating a new TACS element in section 8.1, and select the *Controller Topology* combo box in the *TACS Element Details Screen*. This will produce the *Controller topology TACS Element Screen* (see figure 8.3).

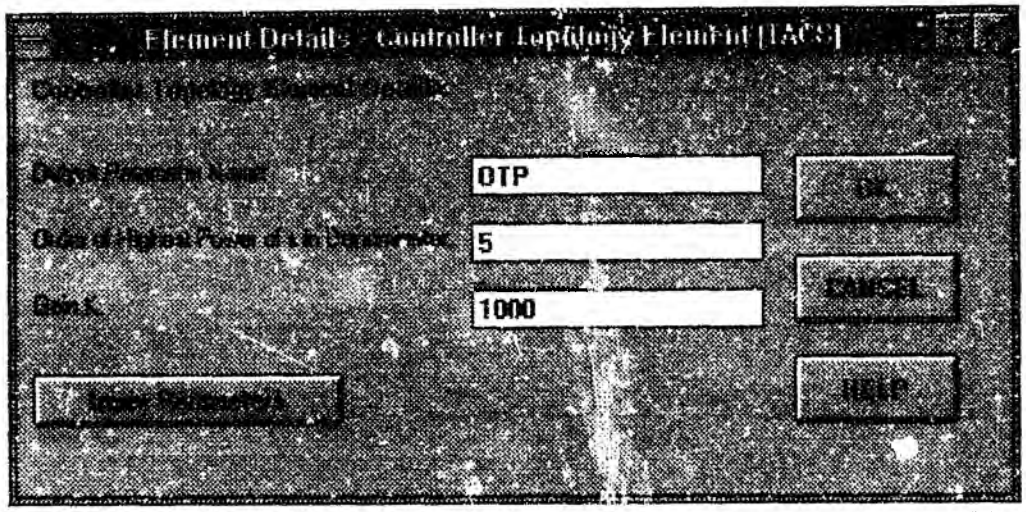


Figure 8.3 - The Controller Topology Element Details Screen

The user is required to complete every entry in the Element Controller details box. These entries are the following:

- **The Order of the Highest Power of s in the Denom** - The order of the highest power of s in the denominator (the order of the numerator must be less than or equal to the order of the denominator).
- **Output Parameter Name** - The name of the output parameter (no more than 6 alphanumeric characters).
- **K** - The gain of the controller.
- Input Parameters command button - This command button will open the Input Parameters Screen, which allows the user to enter the input parameters (U1-U5), and the numerators and denominators (N0-N7, and D0-D7) see figure 8.4 below.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

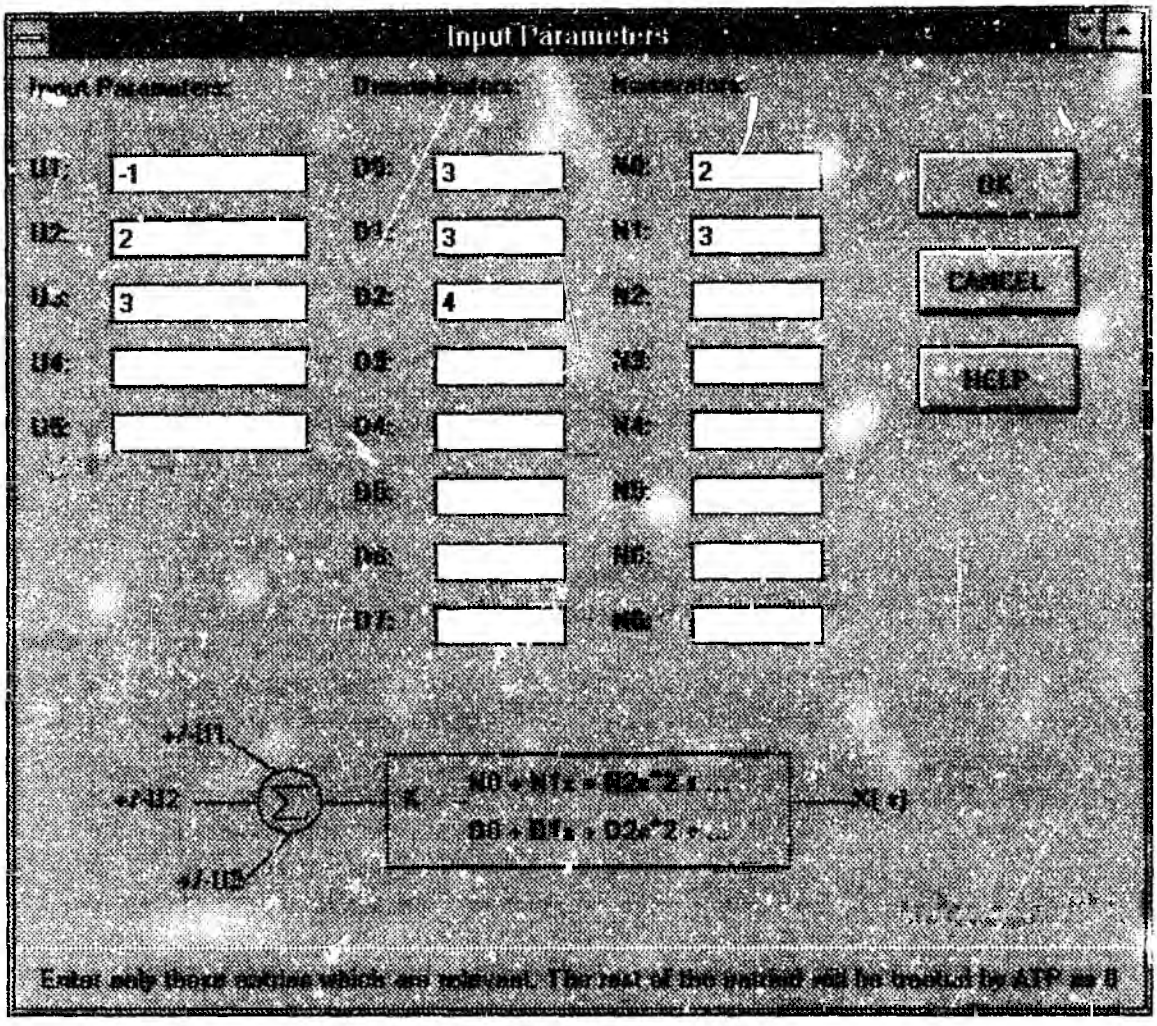


Figure 8.4 - The Controller Topology Element Input Parameters Screen

The user is required to enter those entries which are relevant to this controller topology element:

- N_0 - N_7 - The numerators coefficients in the transfer function (see equation 8.1 above).
- D_0 - D_7 - The denominator coefficients in the transfer function (see equation 8.1 above).
- **Input Parameters U1-U5** and +/- - The names of the input parameters (1-5), and the polarity of these parameters (no more than 6 alphanumeric characters for the name). If no name is specified, it is assumed that the input

does not exist. If no sign is specified for the input, the default sign is +.

8.4 Creating a TACS element which is a special device

8.4.1 Function

There are 16 special devices which can be created using TACS elements in ATP, these are:

- The frequency sensor
- The relay-operated switch
- The level-triggered switch
- The transport delay
- The pulse transport delay
- The digitizer
- Point-by-point user-defined non-linearity
- The multi-operation time-sequenced switch
- The controlled integrator
- The simple derivative
- The input-IF component
- The signal selector
- The sample and track
- Instantaneous min/max
- Min/max tracking
- The accumulator and counter

ATP for Windows only offers the use of the Relay-operated switch, the Level-triggered switch, the controlled integrator, and the simple differentiator, although the other elements will be added to **ATP for Windows** at a later stage.

8.4.2 How To Use It

To create a special device, follow the instructions of section 8.1, and select the combo box marked *Special Devices*. This will produce the *Special Devices Screen* (see figure 8.5)

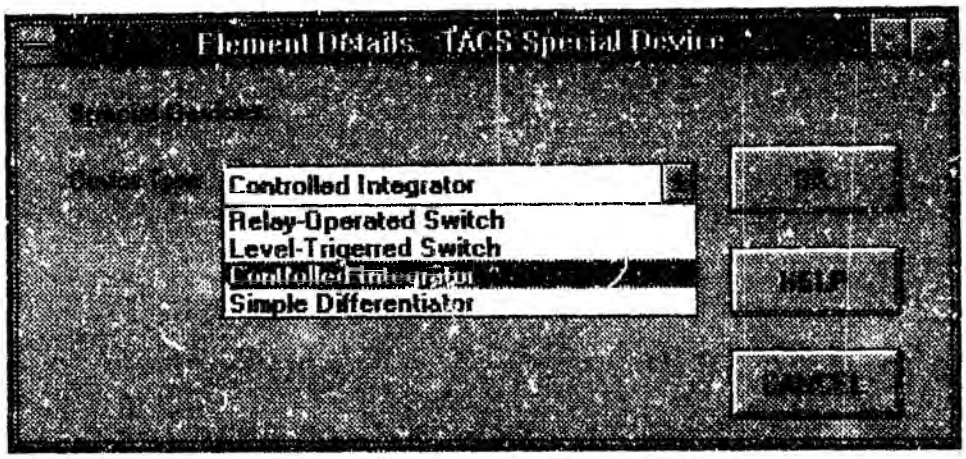


Figure 8.5 - The TACS Special Device Element Details Screen

To select one of the special devices scroll through the list of special devices, and select the combo box corresponding to the required special device. Once this has been done, select the *OK* button. To cancel the operation at any time (and return to the main screen), select the *CANCEL* button, and for on-line Help select the *HELP* button at any time.

8.4.3 The Relay-operated Switch

A normally open switch closes when the *absolute value* of the DRIVING SIGNAL equals or exceeds the sum of the NAMED THRESHOLD and the FIXED THRESHOLD.

A normally closed switch opens when the *absolute value* of the DRIVING signal is less than the sum of the NAMED THRESHOLD and the FIXED THRESHOLD.

The output is 0.0 if the switch is open (for more details see block diagram on the actual Relay Operated Switch Element Details Screen. Selecting the Relay operated switch will produce the Relay Operated Switch screen (see figure 8.5).

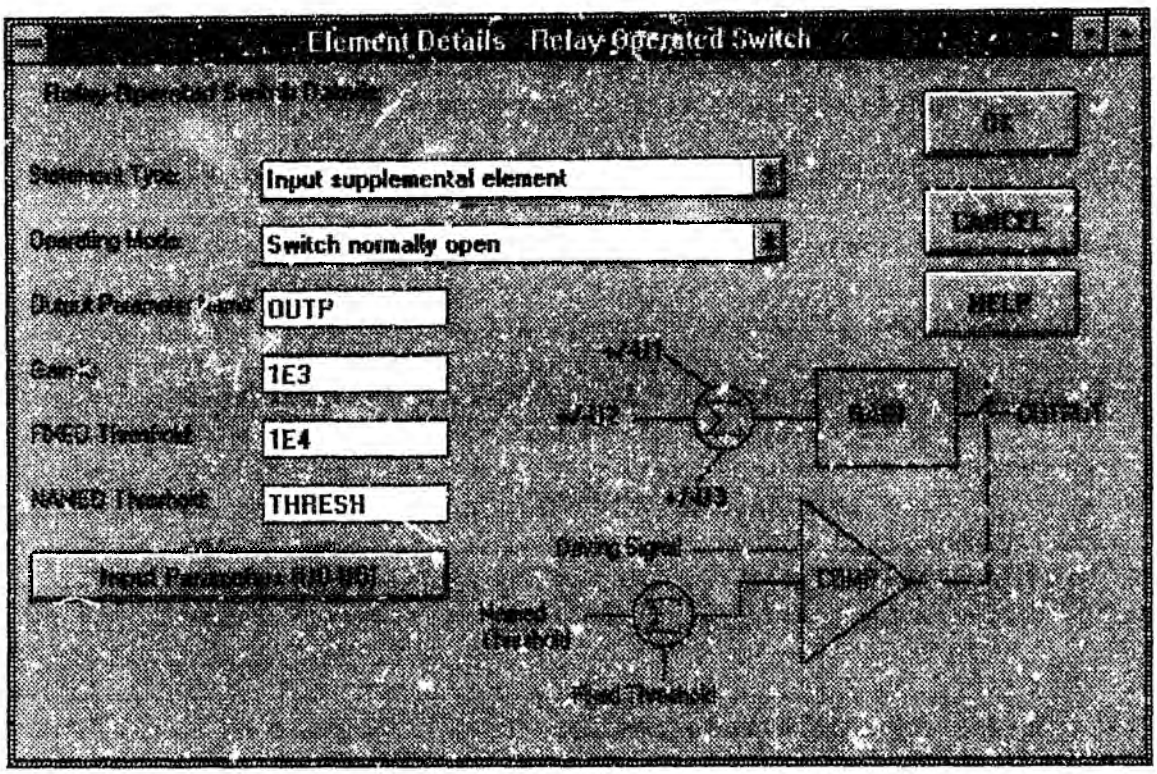


Figure 8.6 - The Relay Operated Switch Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Parameter Name** - The name of the output parameter (no more than 6 alphanumeric characters).
- **K** - The gain (default is 1.0)
- **FIXED THRESHOLD** - The fixed threshold of the switch.
- **Statement Type** combo boxes - The type of statement - determines how this TACS statement fits in with other TACS statements. The options here are:
  - 'Input' supplemental element - The inputs of 'input' supplemental elements can only be TACS sources or the outputs of other 'input' supplemental elements. The outputs of 'input' supplemental elements can be connected to the inputs of controller blocks or any type of supplemental element.
  - 'Output' supplemental elements - The inputs of 'output' supplemental elements can be the outputs of controller blocks, or the outputs of any supplemental element. The outputs of 'output' supplemental elements can be connected to the inputs of other 'output' supplemental elements or interfaced

to ATP.
- 'Inside' supplemental elements - these are left-over of the above two groups, they are used for example between two controller blocks.

- **Operating Mode** of the switch - select the combo box corresponding to the correct mode:
  - Switch normally open
  - Switch normally closed
  - Switch normally open, but closed during the initial establishment of steady state conditions
  - Switch normally closed, and closed during the initial establishment of steady state conditions.
- **NAMED THRESHOLD Name** - This is the name of the TACS parameter which will supply the NAMED THRESHOLD level (optional).
- **DRIVING SIGNAL Name** - This is the name of the TACS parameter that will supply the valuer of the DRIVING SIGNAL level.
- **Input Parameters** command button - this opens the Input Parameters screen relating to the time controlled switch (allows for the entry of U1-U5) see figure 8.7 be

Once all the entries in this screen are completed, select the *OK* button to exit the screen. To cancel the description entered select the *CANCEL* button at any time. To obtain on-line Help, select the HELP button at any time.

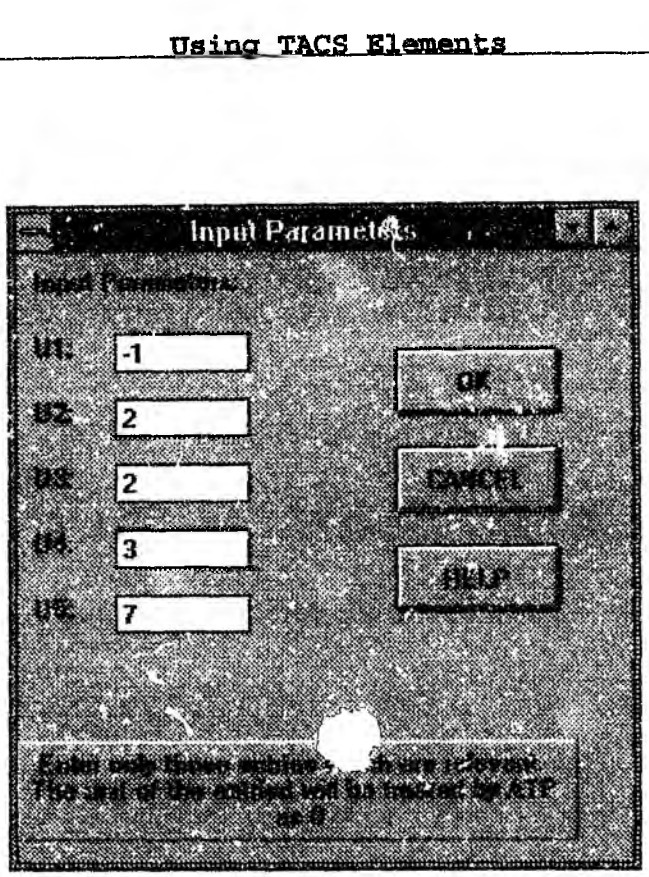


Figure 8.7 - The Input Parameters Screen

The user is required to fill in the relevant entries in this screen:

- **Input parameters 1-5 and +/-** - The names of input parameters (1-5) (no more than 6 alphanumeric characters for the name), and the polarity of these inputs. If no name is specified, it is assumed that the input does not exist. If no sign is specified for the input, the default sign is +.

8.4.4 The Level-triggered Switch

A normally open switch closes when the *actual value* of the DRIVING SIGNAL equals or exceeds the sum of the NAMED THRESHOLD and the FIXED THRESHOLD.

A normally closed switch opens when the *actual value* of the DRIVING signal is less than the sum of the NAMED THRESHOLD and the FIXED THRESHOLD.

The output is 0.0 if the switch is open (for more details see the block diagram on the actual Level Triggered Switch Element Details Screen. Selecting the Level-triggered switch will produce the Level-triggered Switch screen (see figure 8.8).

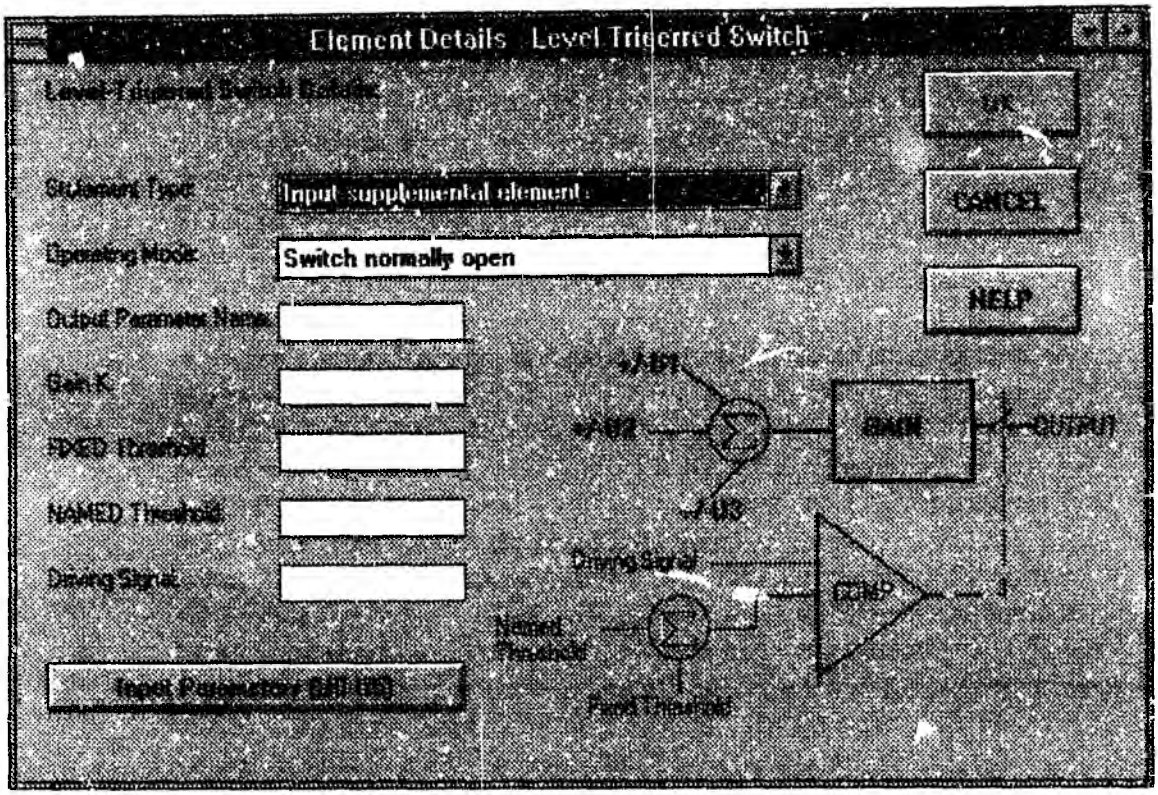


Figure 8.8 - The Level Triggered Switch Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Parameter Name** - The name of the output parameter (no more than 6 alphanumeric characters).
- **K** - The gain (default is 1.0)
- **FIXED THRESHOLD** - The fixed threshold of the switch.
- **Statement Type** combo boxes - The type of statement - determines how this TACS statement fits in with other TACS statements. The options here are:
  - 'Input' supplemental element - The inputs of 'input' supplemental elements can only be TACS sources or the outputs of other 'input' supplemental elements. The outputs of 'input' supplemental elements can be connected to the inputs of controller blocks or any type of supplemental element.
  - 'Output' supplemental elements - The inputs of 'output' supplemental elements can be the outputs of controller blocks, or the outputs of any supplemental element. The outputs of 'output' supplemental elements can be connected to the inputs of other 'output' supplemental elements or interfaced

to ATP.
- 'Inside' supplemental elements - these are left-over of the above two groups, they are used for example between two controller blocks.

- **Operating Mode** of the switch - select the combo box corresponding to the correct mode:
  - Switch normally open
  - Switch normally closed
  - Switch normally open, but closed during the initial establishment of steady state conditions
  - Switch normally closed, and closed during the initial establishment of steady state conditions.
- **NAMED THRESHOLD Name** - This is the name of the TACS parameter which will supply the NAMED THRESHOLD level (optional).
- **DRIVING SIGNAL Name** - This is the name of the TACS parameter that will supply the value of the DRIVING SIGNAL level.
- **Input Parameters** command button - this will open the Input Parameters screen (see figure 8.7), which will allow the user to enter the following:
- **Input parameters 1-5 and +/-** - The names of input parameters (1-5) (no more than 6 alphanumeric characters for the name), and the polarity of these inputs. If no name is specified, it is assumed that the input does not exist. If no sign is specified for the input, the default sign is +.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help select the *HELP* button at any time.

8.4.5 The Controlled Integrator

The integration is performed (starting with the value supplied by the RESET parameter) when the value supplied by the CONTROL parameter is greater than 0.0 (or no CONTROL parameter is specified).

$$\text{OUTPUT}(T) = \text{OUTPUT}(t - \Delta t) \int_{t-\Delta t}^{t} U \, dt \qquad \textbf{(8.1)}$$

If the value supplied by the CONTROL parameter is less than or equal to 0.0 then the output is equal to the value supplied of the RESET parameter (or 0.0 if no RESET parameter is specified) (for more information see block diagram on the actual Controlled Integrator Element Details Screen). Selecting the controlled Integrator provides the Controlled Integrator screen (see figure 8.9).

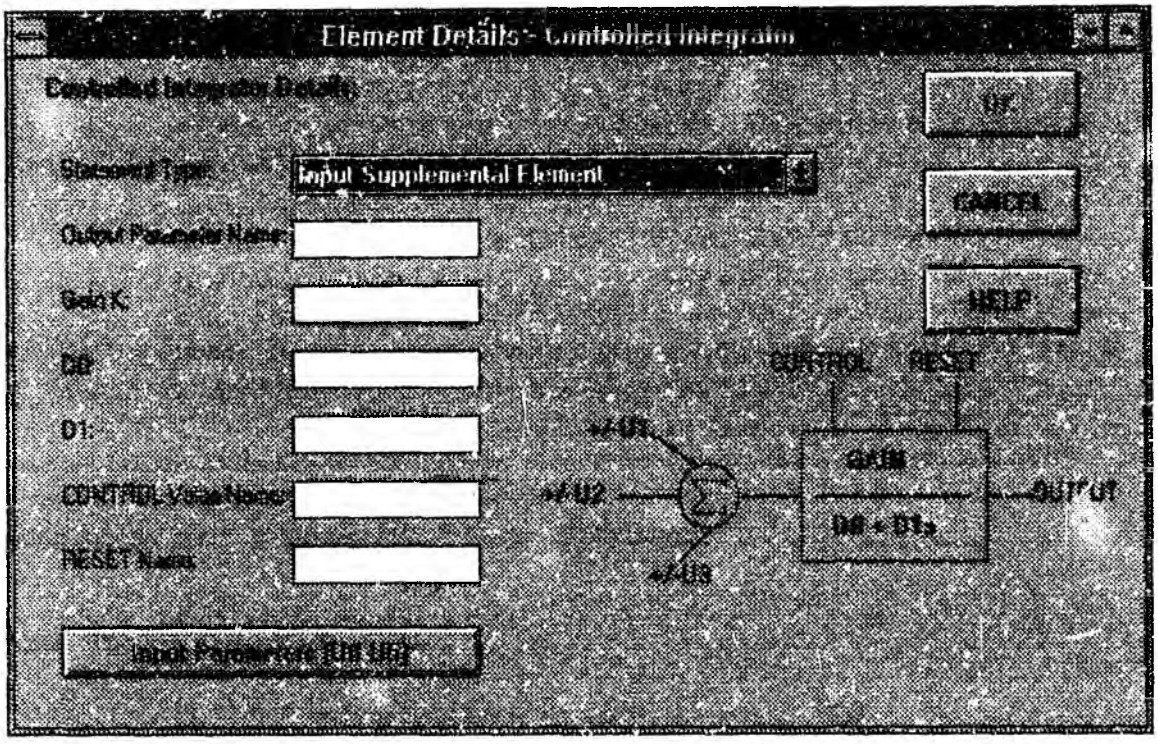


Figure 8.9 - The Controlled Integrator Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Parameter Name** - The name of the output parameter (no more than 6 alphanumeric characters).
- **K** - The gain (default is 1.0)
- **Statement Type** combo boxes - The type of statement - determines how this TACS statement fits in with other TACS statements. The options here are:
  - 'Input' supplemental element - The inputs of 'input' supplemental elements can only be TACS sources or the outputs of other 'input' supplemental elements. The outputs of 'input' supplemental elements can be connected to the inputs of controller blocks or any type of supplemental element.
  - 'Output' supplemental elements - The inputs of 'output' supplemental elements can be the outputs of controller blocks, or the outputs of any supplemental element. The outputs of 'output' supplemental elements can be connected to the inputs of other 'output' supplemental elements or interfaced to ATP.
  - 'Inside' supplemental elements - these are left-over of the above two

groups, they are used for example between two controller blocks.

- **D_0 and D_1** - The values of the denominators D_0 and D_1.
- **CONTROL VALUE Name** - This is the name of the TACS parameter that will supply the CONTROL value (or leave blank).
- **RESET Name** - This is the name of the TACS parameter that will supply the RESET value (or leave blank).
- **Input Parameters** command button - This opens the Input Parameters screen (see figure 8.7) which allows the user to enter the following:
- **Input parameters 1-5 and +/-** - The names of input parameters (1-5) (no more than 6 alphanumeric characters for the name), and the polarity of these inputs. If no name is specified, it is assumed that the input does not exist. If no sign is specified for the input, the default sign is +.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

8.4.6 The Simple Differentiator

$$\text{OUTPUT} = \text{GAIN}*[U(t) - U(t - \Delta t)] / [\Delta t] \qquad \textbf{(8.2)}$$

For more details see the block diagram on the actual Simple Differentiator Element Details Screen below. Selecting the simple differentiator provides the Simple Differentiator screen (see figure 8.10).

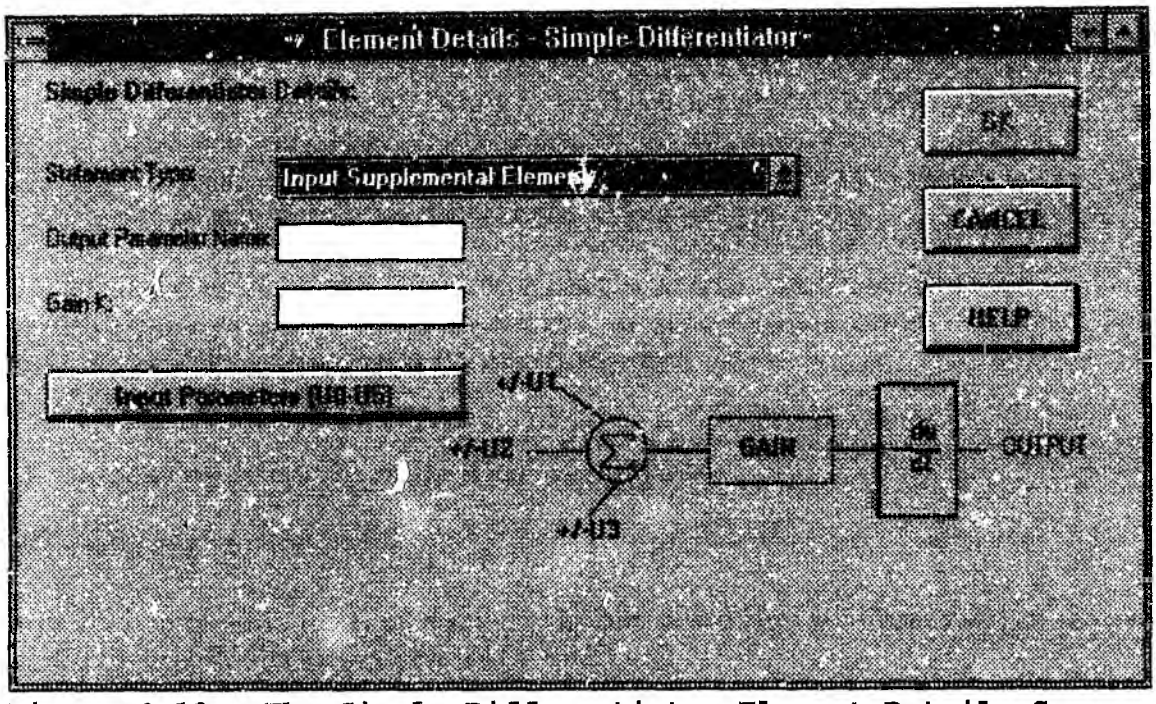


Figure 8.10 - The Simple Differentiator Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Parameter Name** - The name of the output parameter (no more than 6 alphanumeric characters).
- **K** - The gain (default is 1.0)
- **Statement Type** combo boxes - The type of statement - determines how this TACS statement fits in with other TACS statements. The options here are:
  - 'Input' supplemental element - The inputs of 'input' supplemental elements can only be TACS sources or the outputs of other 'input' supplemental elements. The outputs of 'input' supplemental elements can be connected to the inputs of controller blocks or any type of supplemental element.
  - 'Output' supplemental elements - The inputs of 'output' supplemental elements can be the outputs of controller blocks, or the outputs of any supplemental element. The outputs of 'output' supplemental elements can be connected to the inputs of other 'output' supplemental elements or interfaced to ATP.
  - 'Inside' supplemental elements - these are left-over of the above two groups, they are used for example between two controller blocks.
- **Input Parameters** command button - This command button opens the Input

Parameters screen (see figure 8.7) and allows the user to enter the following details:

- **Input parameters 1-5 and +/-** - The names of input parameters (1-5) (no more than 6 alphanumeric characters for the name), and the polarity of these inputs. If no name is specified, it is assumed that the input does not exist. If no sign is specified for the input, the default sign is +.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

8.5 Creating a TACS controlled component

8.5.1 Function

There are 2 components (other than voltage and current sources - see Chapter 9) that can be specifically modelled using available TACS functions:

- TACS controlled resistors
- TACS controlled ATP switches

8.5.2 How To Use It

To create a TACS controlled component, follow the instructions for creating a new TACS element in section 8.1, and select the *TACS controlled component* combo box in the *TACS Element Details Screen*. This will produce the *TACS Controlled Component Screen* (see figure 8.11).

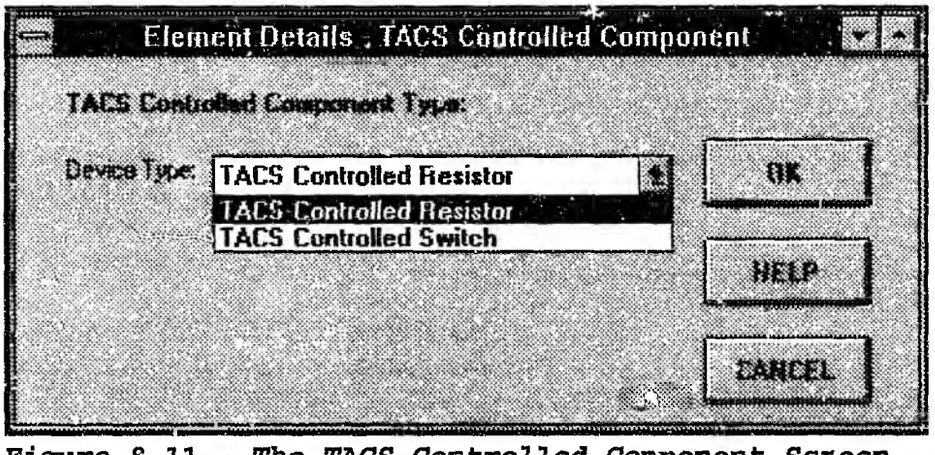


Figure 8.11 - The TACS Controlled Component Screen

To select one of the TACS controlled components, choose the combo box corresponding to the required component. Once this has been done, select the *OK* button on the right of the screen. To cancel the operation at any time (and return to the main screen), select the *CANCEL* button, and for on-line Help, select the *HELP* button at any time.

8.5.3 The TACS controlled resistor

The TACS controlled resistor allows the user to use a resistor who's value is determined by a TACS parameter outside of the resistor's description. Once this component is selected, the TACS Controlled Resistor Screen will appear (see figure 8.12).

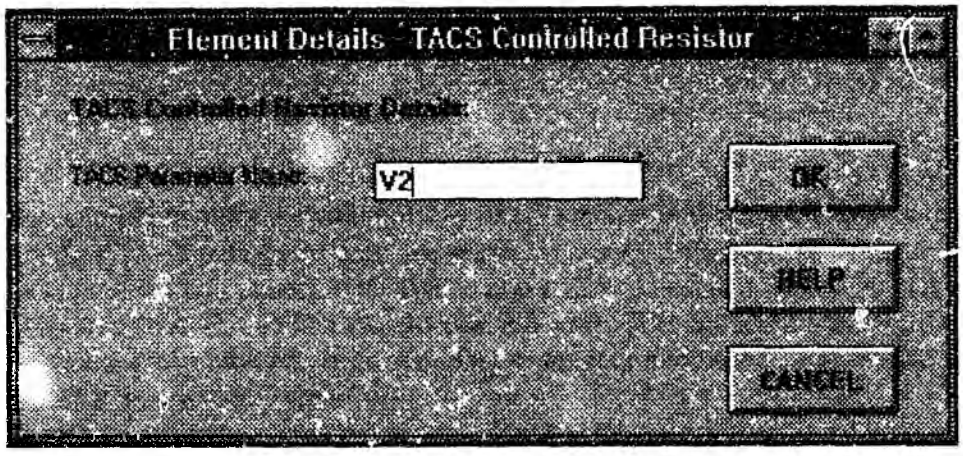


Figure 8.12 - The TACS Controlled Resistor Screen

The user is required to enter the name of the TACS parameter which will supply the value of the TACS controlled resistor. Note that if this TACS parameter does not appear in any of the other TACS elements descriptions during run-time, an error will occur. For on-line Help, select the *HELP* button at any time.

8.5.4 The TACS controlled switch

The logic of a switch is

CONTRL < = 0	:	switch open
CONTRL > = 0	:	switch closed

Once this component is selected, the TACS Controlled Switch Screen will appear (see

figure 8.13).

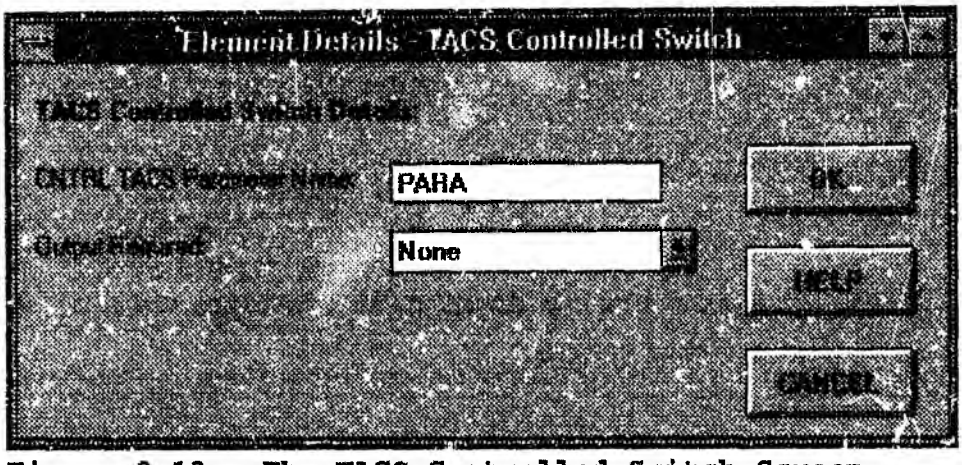


Figure 8.13 - The TACS Controlled Switch Screen

The user is required to enter the name of the TACS parameter which will control the switch (note that if this TACS parameter does not appear in any of the other TACS elements descriptions, an error will occur). The user is also required to select which type (if any) of output is required from this switch (see section 8.3.13.1 - The Standard Switch). This is done by selecting the combo box corresponding to the required output. For on-line Help, select the *HELP* button at any time.

9 USING TACS (TRANSIENT ANALYSIS OF CONTROL SYSTEMS) SOURCES AND WAVEFORMS

Another possible use of TACS is for voltage and current sources. Using the ATP built in TACS functionality, it is possible to define:

- DC levels or single pulses
- Sinusoidal waveform generators
- Repetitive pulses
- Repetitive ramps
- TACS controlled ATP voltage and current sources

TACS controlled sources or waveforms can be selected from the *Circuit Element Panel.*

9.1 How To Use It

Select the icon marked *TACS* with a **waveform** underneath the TACS writing, from the *Circuit Element Menu*, and place the element anywhere in the circuit presently in the *Circuit Region* or select the *Select Element* option from the *Run* menu, then select the *TACS Sources Library*, select the TACS Source and move it to the required spot in the *Circuit Region* (see section 5.4 - Select Element). Once this has been done, right clicking on the TACS element, will produce the normal *Circuit Element Menu* associated with any circuit element in **ATP for Windows**. The *Element Details* option of a TACS element will automatically produce the Element Details screen, but the user will be given the option of selecting the type of TACS source or waveform required (see figure 9.1 - The TACS Source/Waveform Details Screen). To select the type of TACS source or waveform required, click on the combo box corresponding to the required type.

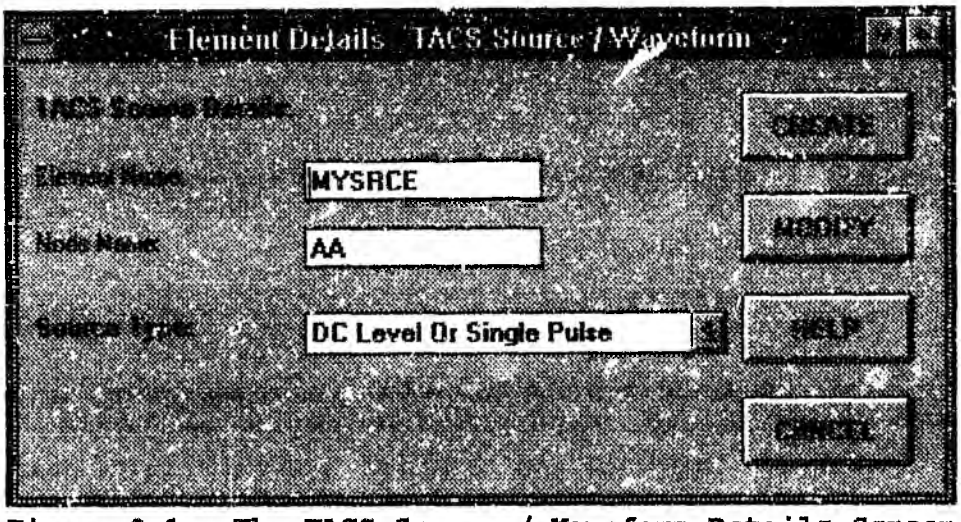


Figure 9.1 - The TACS Source / Waveform Details Screen

To create a new TACS element, the following steps are required:

- name the source by clicking on the name space provided, and editing the name automatically given to the TACS source by **ATP for Windows.**
- Name the node touching the source. The source will then be connected between this node and the local ground. If the source is a TACS generator which is not connected to the circuit, but has an output used in the circuit, leave the name box empty.
- Select the type of TACS source required as described above.
- Now click on the *Create* button on the right of the screen. This will automatically open the Modify/Create TACS source screen corresponding to the type of TACS source chosen (see figures 9.2 - 9.5).

If the source already exists, and you want to change some of the parameters of the source, select the *Modify* option. This will open the *Modify/Create TACS source screen* corresponding to the type of TACS source chosen (see figures 9.2 - 9.6).

Clicking on the *Cancel* option at any time, cancels this operation, while clicking on *OK* saves all the changes made to this source, and exits from the *Circuit Element Menu*. To obtain on-line Help, select the *HELP* button at any time.

9.2 Creating a DC level or single pulse

9.2.1 Function

This allows the user to create a DC level starting at a selected time and stopping at a selected time.

9.2.2 How To Use It

To create a DC level or single pulse, follow the instructions for creating a new TACS source in section 9.1, and select the *DC Level or Single Pulse* combo box in the TACS Source Details Screen. This will produce the DC level screen (see figure 9.2).

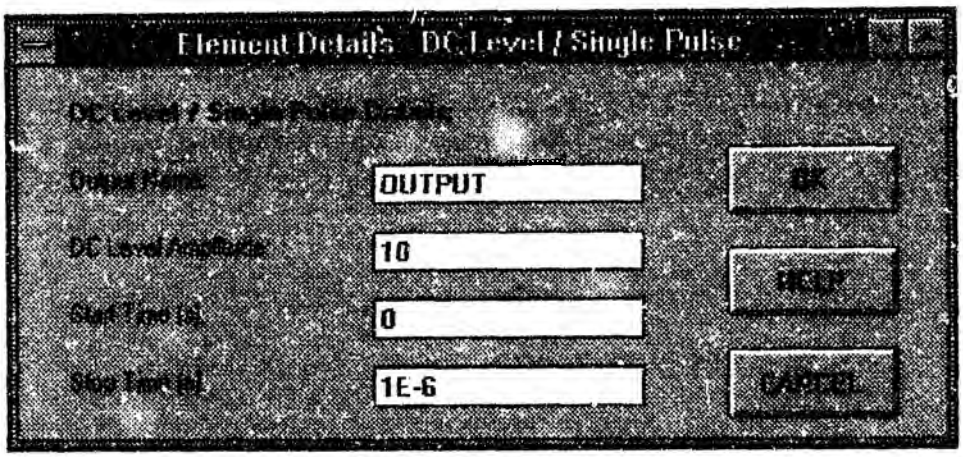


Figure 9.2 - The DV Level / Single Pulse Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Name** - The name assigned to the output of this waveform generator.
- **DC Level Amplitude** - The amplitude of the DC level or single pulse in volts.
- **Start Time** - Delay before source is activated in seconds. The source value will be zero during this delay.
- **Stop Time** - Time at which the source is terminated - source is returned to zero, in seconds. If left blank, the source will continue indefinitely.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

9.3 Creating a Sinusoidal Waveform Generator

9.3.1 Function

This allows the user to create a sinusoidal source starting at a selected time and stopping at a selected time, with an initial phase angle and a selected amplitude.

9.3.2 How To Use It

To create a sinusoidal source, follow the instructions for creating a new TACS source in section 9.1, and select the *Sinusoidal Source* combo box in the TACS Source Details Screen. This will produce the Sinusoidal Source screen (see figure 9.3).

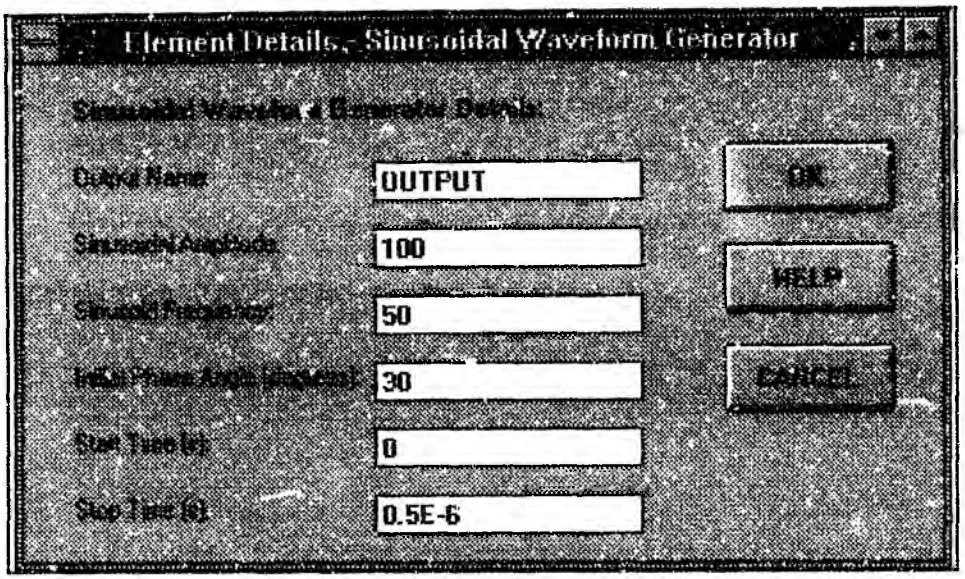


Figure 9.3 - The Sinusoidal Waveform Generator Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Name** - The name assigned to the output of this waveform generator.
- **Sinusoidal Amplitude** - The amplitude of the sinusoid in volts.
- **Sinusoid Frequency** - The frequency of the source in Hz.
- **Initial Phase Angle** - The initial phase angle of the source in degrees.
- **Start Time** - Delay before source is activated in seconds. The source value will be zero during this delay.
- **Stop Time** - Time at which the source is terminated - source is returned to zero, in seconds. If left blank, the source will continue indefinitely.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

9.4 Creating a Repetitive Pulse

9.4.1 Function

This allows the user to create a repetitive pulse starting at a selected time and stopping at a selected time, with a selected amplitude and period as well as pulse width.

9.4.2 How To Use It

To create a repetitive pulse, follow the instructions for creating a new TACS source in section 9.1, and select the *Repetitive Pulse* combo box in the TACS Source Details Screen. This will produce the Repetitive Pulse screen (see figure 9.4).

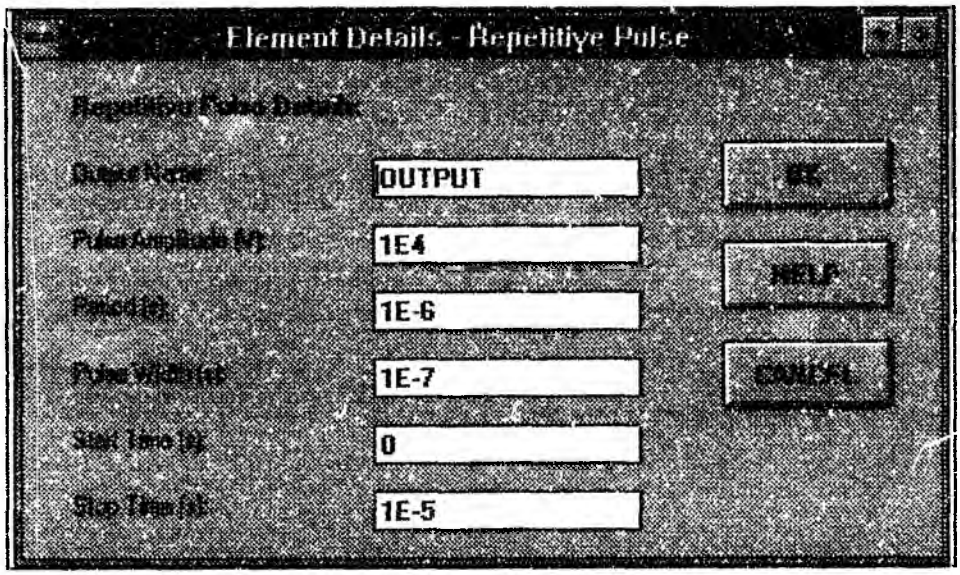


Figure 9.4 - The Repetitive Pulse Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Name** - The name assigned to the output of this waveform generator.
- **Pulse Amplitude** - The amplitude of the repetitive pulse in volts.
- **Period** - The period of the pulses in seconds.
- **Pulse Width** - The pulse width in seconds.
- **Start Time** - Delay before source is activated in seconds. The source value will be zero during this delay.
- **Stop Time** - Time at which the source is terminated - source is returned to zero, in seconds. If left blank, the source will continue indefinitely.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

9.5 Creating a Repetitive Ramp

9.5.1 Function

This allows the user to create a repetitive ramp starting at a selected time and stopping at a

selected time, with a selected peak amplitude and period.

9.5.2 How To Use It

To create a repetitive ramp, follow the instructions for creating a new TACS source in section 9.1, and select the *Repetitive Ramp* combo box in the TACS Source Details Screen. This will produce the Repetitive Ramp screen (see figure 9.5).

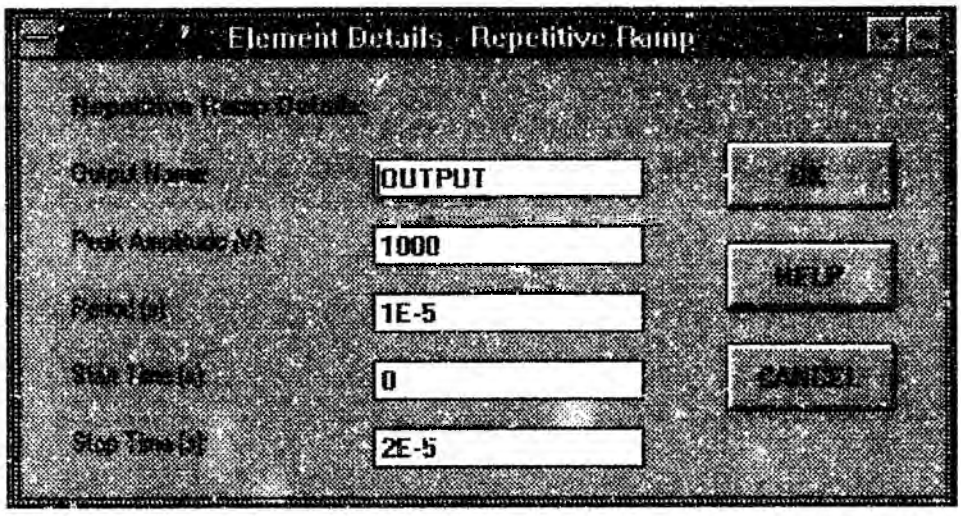


Figure 9.5 - The Repetitive Ramp Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **Output Name** - The name assigned to the output of this waveform generator.
- **Peak Amplitude** - The peak amplitude of the repetitive ramp in volts.
- **Period** - The period of the ramps in seconds.
- **Start Time** - Delay before source is activated in seconds. The source value will be zero during this delay.
- **Stop Time** - Time at which the source is terminated - source is returned to zero, in seconds. If left blank, the source will continue indefinitely.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

9.6 Creating a TACS controlled voltage or current source

9.6.1 Function

This allows the user to create voltage or current source with an amplitude which is directly supplied by a TACS parameter.

9.6.2 How To Use It

To create a TACS controlled voltage or current source, follow the instructions for creating a new TACS source in section 9.1, and select the *TACS Controlled Voltage or Current Source* combo box in the TACS Source Details Screen. This will produce the TACS Controlled Source screen (see figure 9.6).

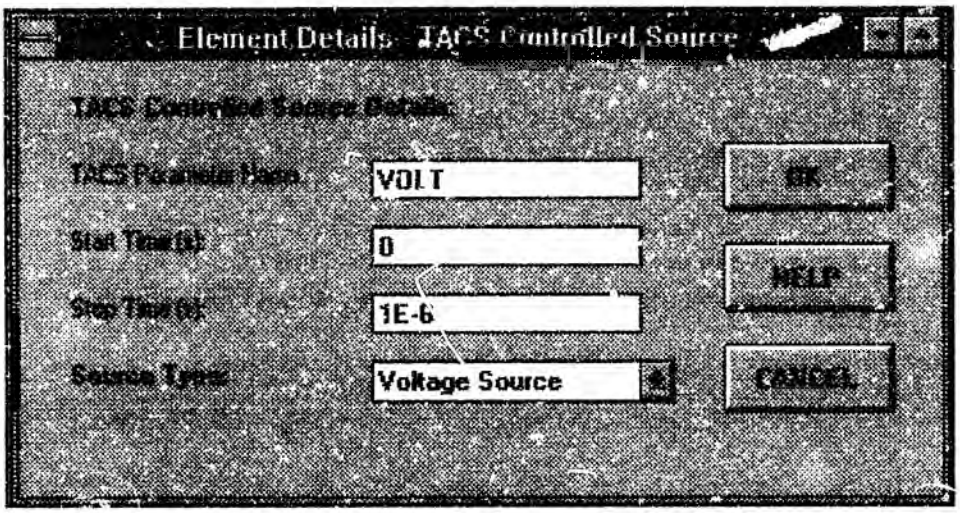


Figure 9.6 - The TACS Controlled Source Element Details Screen

The user is required to complete every entry in the Element Details box. These entries are the following:

- **TACS Parameter Name** - The name of the TACS parameter which determines the amplitude of the source.
- **Start Time** - Delay before source is activated in seconds. The source value will be zero during this delay.
- **Stop Time** - Time at which the source is terminated - source is returned to zero, in seconds. If left blank, the source will continue indefinitely.
- **Voltage Source Combo box** - Select this combo box if the source is a voltage source.
- **Current Source Combo box** - Select this combo box if the source is a current source.

Once all the entries in this screen have been completed, select the *OK* button to exit the screen. To cancel the description of the element, select the *CANCEL* button at any time. To obtain on-line Help, select the *HELP* button at any time.

10 MODELLING TRANSMISSION LINES ON ATP FOR WINDOWS

Another option in ATP for Windows is to model transmission lines using any of the following models:

- the NOMINAL-PI model of a short transmission line
- the EQUIVALENT-PI model for a transmission line of any length, and
- the JMARTI modal parameters model for transmission lines

10.1 How To Use It

Select the pylon icon from the *Circuit Element Menu*, and place the element anywhere in the circuit in the *Circuit Region* or select the *Select Element* option from the *Run* sub-menu, then select the *Transmission Lines Library*, and select the *Transmission Lines Element* and move it to the required spot in the *Circuit Region* (See section 5.4 - Select Element). Once this has been done, right clicking on the pylon element, will produce the normal *Circuit Element Menu* associated with any circuit element in **ATP for Windows**. The *Element Details* option of a pylon element will automatically produce the *Element Details* screen, but the user will be given the option of entering the dimensions of the pylon, and the transmission line associated with this pylon. Every transmission line stage in the circuit is represented by one pylon who's details are entered in the *Element Details* screen for the pylon element (see figure 10.1 - the *Element Details* screen for the pylon element). Note that the user is required to select one of three combo boxes to signify which type of model is required.