

Temporal Expert System Shell

by

Sharad Sachdev
B.E. (Computer Science), 1994,
National Institute of Engineering, Mysore, India.

Thesis

submitted in partial fulfillment of the requirement
for the degree of Master of Science (Computer Science)

Acadia University,
Fall Convocation 1998

© by Sharad Sachdev 1998



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-33826-6

Abstract

By accommodating users with diverse needs and backgrounds, user interfaces are revolutionizing the application of computers. Intuitive interfaces allow the user to perform complex tasks with little knowledge of the underlying logic. A popular approach in Artificial Intelligence for temporal knowledge representation and reasoning is to use first order logic. In order to use logic the user must understand its syntax and semantics. This thesis presents a logic independent graphical user interface (GUI) and discusses the principles of user interface design. The GUI allows the user to enter, query, and receive temporal information using color-coded symbols. The user does not have to be familiar with the particular logic used by the implementation.

Acknowledgements

I would like to express my heartfelt thanks to Dr. André Trudel for his guidance, help and support throughout my M.Sc. studies.

I would like to extend my thanks to Dr. Carolyn Watters and Dr. Eric Neufeld, for spending their time and being my internal and external examiners.

Finally, I would like to express special thanks to my parents for their blessings, care and encouragement.

Table of Contents:

1. INTRODUCTION.....	1
2. TEMPORAL REASONING AND CLP.....	4
2.1 Constraints.....	4
2.1.1 Temporal Constraints.....	6
2.2 Logic and Temporal Reasoning.....	7
2.3 Temporal information.....	8
2.4 Constraint Logic Programming (CLP).....	9
2.4.1 Programming with constraints.....	11
2.4.2 CLP Vs logic programming.....	13
2.5 ECLiPS ^e (CLP).....	16
3. GRAPHICAL USER INTERFACE.....	18
3.1 Introduction.....	18
3.2 What is Correctness ?.....	19
3.3 Significance of HCI in designing GUI.....	21
3.4 Tools and techniques for designing a GUI.....	25
3.4.1 Color.....	25
3.4.2 Graphics.....	28
3.4.3 Icons.....	28
3.5 Other expert system shells.....	29
4. SYSTEM OVERVIEW.....	31
4.1 Tcl/Tk.....	32
4.2 Conceptual Model.....	33
4.3 Graphical User Interface.....	34
4.4 System Database.....	53

5. INFERENCE ENGINE AND KNOWLEDGE BASE.....	57
5.1 Inference engine.....	57
5.2 Knowledge base.....	62
6. EVALUATION OF THE GUI.....	65
6.1 User Evaluation.....	68
6.1.1 Results of the Evaluation.....	73
6.2 User Satisfaction Evaluation.....	77
6.2.1 Results of the user satisfaction questionnaire.....	81
6.3 Conclusions from the observations.....	82
6.4 Conclusions from the questionnaire.....	83
6.5 Modifications to the GUI.....	84
7. CONCLUSION.....	85
7.1 Future Development.....	87
7.2 Applications.....	89
BIBLIOGRAPHY.....	90
A. USER MANUAL.....	95
B. SOURCE CODE.....	118

List of Figures:

Figure 2.4.1: ECLIPSe code for point and integral.....	12
Figure 2.4.2.2: PROLOG code & its execution.....	14
Figure 4.2: Conceptual Model.....	33
Figure 4.3.1: Screen for Entering Information.....	35
Figure 4.3.2: Color Window.....	36
Figure 4.3.3: Point Event.....	39
Figure 4.3.4: Limitless Event.....	40
Figure 4.3.5: Fixed Event.....	41
Figure 4.3.6: FixedLeft Event.....	42
Figure 4.3.7: FixedRight Event.....	43
Figure 4.3.8: Point Events and Fixed Event.....	44
Figure 4.3.9:Screen for Querying the System.....	46
Figure 4.3.10: Query Screen with dropdown list.....	47
Figure 4.3.11: Querying Point Event.....	48
Figure 4.3.12: Querying Limitless Event.....	50
Figure 4.3.13: Querying using What's True.....	51
Figure 4.3.14: Point Event and Fixed Event (closed interval).....	52
Figure 4.3.15: Point Event and Fixed Event (open interval).....	53
Figure 4.4.1: Main application window.....	54
Figure 4.4.2: Event Details.....	54
Figure 4.4.3: Colors Used.....	55
Figure 4.4.4: Symbols.....	56
Figure 5.1: Point relations.....	59
Figure 5.1.1: Integral relations.....	60
Figure 5.2: Fact representation in the Knowledge Base.....	63
Figure 6.1: Evaluation Tasks.....	70
Figure 6.2: Observation Form.....	72

Figure 6.3: User population for the Evaluation.....	74
Figure 6.4: User Satisfaction Questionnaire.....	78
Figure 6.5: Results of the User Satisfaction Questionnaire.....	81

List of Tables:

Table 3.4.1: Color Combinations for Graphical User Interfaces.....	27
--	----

Chapter 1

Introduction

“As natural selection works solely by and for the good of each being, all corporeal and mental endowments will tend to progress towards perfection.”

Charles Darwin, Origin of Species

Every interesting real world problem has a temporal component. For example, if we want to simulate a telephone switch in software, the software must be able to deal with the representation of simultaneous calls of varying duration. From our everyday life, we are aware that the symptoms of a disease change over time. Another example from the internet would be an intelligent dynamic web site that monitors the time, duration and type of hits, and modifies its presentation over a 24 hour time period in order to best serve its clients. Therefore, if we are to use a computer to solve real problems, time must be explicitly represented. A popular approach in Artificial Intelligence for temporal knowledge representation and reasoning is to use first order logic.

CHAPTER 1. INTRODUCTION

One drawback with using logic is that the user must understand its syntax and semantics in order to interact with the implementation. This thesis presents a logic independent graphical user interface (GUI). The GUI allows the user to enter, query, and receive temporal information using color-coded symbols. The user does not have to be familiar with the particular logic used by the implementation.

The main research challenge is the definition of the GUI. The GUI should be designed and implemented in such a way that it is easy to learn, use, efficient and effective. It should allow the user to enter and query temporal information using icons and color-coded symbols. For example, using a pull down menu the user could choose “John is sleeping”. The user then places a dot on a horizontal time line at 2 a.m. to specify that John is sleeping at this time. The user can alternatively place a line to represent a time interval. Precise quantitative point and interval based information is easy to represent graphically. Challenging types of information, which are difficult to represent, are “I’ll meet you later”, “It rained throughout the day today”, and “Bob played squash and tennis for equal amounts of time”.

CHAPTER 1. INTRODUCTION

Constraint logic programming and temporal reasoning is introduced in the next chapter.

Chapter three discusses features of effective, efficient and user-friendly GUI's.

Chapter four gives a system overview. This chapter concentrates on the processes of representing time, entering and querying information.

Chapter five presents the inference engine, which is the backbone of the system and the knowledge base.

Chapter six discusses evaluation of the user interface.

Chapter seven presents directions for future work and conclusions.

Chapter 2

Temporal reasoning and constraint logic programming (CLP)

“Any sufficiently advanced technology is indistinguishable from magic.”
-- Arthur C. Clarke

This chapter introduces constraint properties and temporal constraints, followed by a discussion on logic and temporal reasoning. Constraint logic programming is explained in detail with the help of examples. Finally, we present ECLiPS^e and its features.

2.1 Constraints

Constraints have many properties that make them unique and flexible for our use in AI and computer science applications [Magh95]:

- Constraints and multi-object relationships: Constraints can involve relationships between multiple objects:

John and Steve must live in separate towns. (1)

CHAPTER 2. TEMPORAL REASONING AND CLP

The relationship “live in separate towns” treats John and Steve equally. It gives the same amount of information about both. It does not give information of where each one lives, but it gives specific information about a relationship between the two.

- Adirectional property: There are no computational directions or flow of information represented in a constraint. In (1), there is no specification of which one is first, and there is no priority. Knowing one of John’s or Steve’s residence, it gives us information about the other.
- Partial knowledge representation: Constraints allow the specification of partial knowledge. We do not need to know everything about the domain of discourse or the relationships between individuals when writing constraints. For example:

John has a table in his living room.

is acceptable by itself. It is not necessary to specify all the constraints that apply to John’s house.

- Constraint representation: A constraint usually does not have a unique representation. For example:

The Bar is open between 8:00 a.m. and 12:00 p.m.

and,

The Bar is closed between 12:00 p.m. and 8:00 a.m.

both have the same meaning but are represented in different ways. The particular representation chosen usually depends on the problem domain and implementation.

- Soft constraints: An example of a soft constraint is:

John's table can be red.

This does not mean that John's table is red, but implies that red is a possible color or an expected color of John's table. These soft constraints give expectations or predictions.

2.1.1 Temporal constraints

Constraints can be atemporal:

John eats dinner.

This constraint does not specify when John eats his dinner. If we introduce time:

John eats dinner at 6:00 p.m.

We know what John is doing at 6:00 p.m. Time can give constraints a temporal limit, which makes them more accurate and precise. For example:

Mary works from 9:00 a.m. to 6:00 p.m. (2)

specifies an occupation for Mary between the time limit (9:00 a.m. – 6:00 p.m.). If in addition we have:

Mary works at the Library.

then using (2), we know that Mary is in the Library between 9:00 a.m. – 6:00 p.m. These two constraints influence any solution sought or decision to be taken concerning Mary. If for example, something happened at 10:00 a.m. at the Halifax Shopping Mall, we can conclude that Mary was not involved in it.

2.2 Logic and Temporal Reasoning

Logic consists of a formal system for describing states of affairs, using the syntax and semantics of the language. Syntax describes how to write valid sentences and semantics state how sentences relate to states of affairs. There are various kinds of logic such as propositional, first order, and fuzzy logic [Russ95].

We concentrate on a variant of first order logic for temporal reasoning, which assumes that the world is ordered by a set of time points and/or intervals, and includes built in mechanisms for reasoning about time.

Formulas in a first order temporal logic denote statements whose truth-value may change over time. Examples of such statements are:

- It is presently raining.
- John was on the phone between 2 and 4 p.m.
- Mary came in at 3 p.m. and left later.

2.3 Temporal information

There are four types of temporal information:

1. Point based qualitative information.
2. Interval based qualitative information.
3. Point based quantitative information.
4. Interval based quantitative information.

An example of point based qualitative information is:

John is working at time t_1 .

An example of point based quantitative information is:

John is walking at a speed of 5 km/hr at time t_1 .

An example of interval based qualitative information is:

John was working from time t_1 to t_2 .

An example of interval based quantitative information is:

John worked for 1 hour from time t_1 to t_2 .

2.4 Constraint Logic Programming (CLP)

Constraint Logic Programming [Fruh93] is a new class of programming languages combining the declarativity of logic programming with the efficiency of constraint solving. New application areas, amongst them many different classes of combinatorial search problems such as scheduling, planning or resource allocation can now be solved, which were intractable for logic programming. The most important advantage that these languages offer for certain problems is short development time while exhibiting an efficiency comparable to imperative languages.

Constraint Logic Programming adds richer data structures to a logic programming system thus allowing semantic objects (e.g., arithmetic expressions) to be directly expressed and manipulated. Logic programming has a uniform but simple computation rule, a depth-first search procedure, resulting in a generate and test procedure with its well-known performance problems for large search applications [Fruh93]. Constraint logic programming overcomes this problem by its active use of constraints, pruning the search tree in an *a priori* way. The key aspect is the tight integration between constraint evaluation and search.

CHAPTER 2. TEMPORAL REASONING AND CLP

Constraint solving has been used in many different application areas such as engineering, planning and graphics [Fruh93]. Problems such as scheduling, allocation, layout, fault diagnosis and hardware design are typical examples of constrained search problems. A reason for the success of CLP in recent applications has been the choice of constraint systems integrated into the different implementations. The selection of new constraint domains needs to satisfy both technical and practical criteria [Jaff87]:

- The expressive power of the computation domain,
- The existence of a complete and efficient constraint solver,
- Its relevance in applications.

The constraint solver is complete if it is able to decide the satisfiability of any set of constraints of the computational domain. To achieve efficiency the constraint solver needs to be incremental, i.e. when adding a new constraint C to an already solved set of constraints S , the constraint solver should not start solving the new set $S \cup \{ C \}$ from scratch.

2.4.1 Programming with constraints

The amalgamation of logic programming and constraints is called constraint logic programming (CLP). A CLP interpreter must have two components: an inference engine which deals with resolution, and a domain-specific constraint engine which maintains the constraint store in a standard form, and upon a request from the inference engine, is able to inform it whether the constraints it suggests can be consistently added to the store.

Examples of CLP systems are CHIP [Dinc88], CLP(R) [Hein92] and ECLⁱPS^e [Abde95]. An example of ECLⁱPS^e code is shown in figure 2.4.1. Interval based information is represented using the integral relation. $\text{integral}(a, b, f, x)$ is true if and only if the integral of f from a to b is x . We use a limited version of the integral relation which can be viewed as the measure of the duration of truth of f over the interval (a, b) .

For example, “running” is true throughout the interval $(0, 10)$ is written as $\text{integral}(0, 10, \text{running}, 10)$. and “running” for half the time is $\text{integral}(0, 10, \text{running}, 5)$.

CHAPTER 2. TEMPORAL REASONING AND CLP

Information that is true at an isolated point is represented using the point relation. For example, “running” is true at time 5 is written as point(5,running) .

```
point(T,F) :-
    A #<= T,
    B #>= T,
    C #= B-A,
    integral(A,B,F,C) .
integral(0,10,running,10) .
```

Figure 2.4.1 ECLⁱPS^e code for point and integral

To determine if F is true at an isolated point T, we can look for an interval (A,B) which contains T and over which F is true throughout. This strategy is captured by the first rule in figure 2.4.1.

Computation begins with a goal and an empty set of constraints. An arithmetic constraint or an atom is selected with the usual left-right atom selection rule at each stage. When an atom is selected, the set of rules is searched in the usual top-down fashion, each time matching that atom with the head of some rule. Execution proceeds as in PROLOG until a constraint is encountered. When a constraint is selected it is added to the set of collected constraints, and it is determined whether the resulting set has a solution. If no solution is found, backtracking occurs. At every point in the derivation, the set of constraints is satisfiable.

CHAPTER 2. TEMPORAL REASONING AND CLP

Let us return to the program in figure 2.4.1, and examine a successful execution path:

```
? - point(X,running).  
{ } ? - point(X,running).  
{T=X, F=running} ? - A#<=T, B#>=T, C#=B-A, integral(A,B,running,C).  
{T=X, F=running, A#<=T} ? - B#>=T, C#=B-A, integral(A,B,running,C).  
{T=X, F=running, A#<=T, B#>=T} ? - C#=B-A, integral(A,B,running,C).  
{T=X, F=running, A#<=T, B#>=T, C#=B-A} ? - integral(A,B,running,C).  
{T=X, F=running, A#<=T, B#>=T, C#=B-A, A=0,B=10,C=10} ? -
```

At each step, the set of constraints is shown on the left of “?-”. Note that at the first step, the set is empty. In the penultimate step, $integral(A,B,F,C)$ unifies with $integral(0,10,running,10)$ ECLⁱPS^e solves the final set of constraints and returns the answer:

$$X=0..10$$

This means $point(X,running)$ is true whenever X has a value between 0 and 10.

2.4.2 CLP vs. logic programming

Constraint logic programming is a generalization of logic programming. Program execution in ECLⁱPS^e is similar to PROLOG's, but unification is more general. Executing the code shown in figure 2.4.1 in

CHAPTER 2. TEMPORAL REASONING AND CLP

ECLiPS^e and the PROLOG equivalent code at the top of figure 2.4.2.2 gives the results shown in figure 2.4.2.1.

<u>Query</u>	<u>PROLOG</u>	<u>ECLiPS^e</u>
point(5, running)	NO	YES
point(X, running)	NO	YES X = 0..10

Figure 2.4.2.1 Comparison between PROLOG and ECLiPS^e

PROLOG Code:

```
point(T,F):-
    A =< T,
    B >= T,
    C = B-A,
    integral(A,B,F,C).

integral(0,10,running,10).
```

PROLOG execution:

```
?- point(5, running).
|
Integer required
|
No

?- point(X, running).
|
Integer required
|
No
```

PROLOG code and its execution for point(5, running) and point(X, running).

Figure 2.4.2.2

CHAPTER 2. TEMPORAL REASONING AND CLP

Due to the presence of the arithmetic constraint $A \leq T$, PROLOG is unsuccessful with the unification. The PROLOG code and its execution for `point(5, running)` and `point(X, running)` are shown in figure 2.4.2.2. The proof trees are as follows:

?- point(5,running).

?- A =< 5, B >= 5, C = B-A, integral(A,B,running,C).

Fails. (Cannot associate any value with A in the goal A =< 5.)

?- point(X,running).

?- A =< X, B >= X, C = B-A, integral(A,B,running,C).

Fails. (Cannot associate any value with A in the goal A =< X.)

In the proof tree for `point(5,running)`, PROLOG tries to solve $A \leq 5$ but is not able to unify any value with A and fails. Similarly, in `point(X,running)`, PROLOG is unable to unify any value with A.

As shown above, PROLOG does not deal with constraints. Therefore, to handle constraint based problems for temporal reasoning we need a more powerful constraint-solving environment, which is provided by CLP.

2.5 ECLiPS^e (CLP)

The CLP system we use in our implementation is ECLiPS^e. ECLiPS^e (ECRC Common Logic Programming System) [Abde95] is a development environment for constraint programming applications. It contains several constraint solver libraries, which use extended PROLOG technology with persistent knowledge base and constraint handling features.

Features of ECLiPS^e:

- Incremental Compiler: ECLiPS^e is based on an incremental interactive compiler. ECLiPS^e programs are both fast and flexible.
- Source Variable Names: ECLiPS^e is able to remember the source names of variables so that debugging programs becomes easier.
- Flexibility: ECLiPS^e enables the user to modify most of the system features, build separate applications or include new features.

CHAPTER 2. TEMPORAL REASONING AND CLP

- Memory: All ECLⁱPS^e memory areas are automatically extended when necessary. There are no limits (other than the available memory) to the size of atoms or strings or their number, the length of integers and there is no limit on the complexity of compiled clauses.
- Strings: ECLⁱPS^e has the data type string whose representation is compact and compatible with strings in C and Tcl/Tk.
- Complete Search Rule: Most PROLOG systems implement logic programming incompletely because they use the depth-first search, but ECLⁱPS^e also supports depth-first iterative search along with DFS.
- Modules: ECLⁱPS^e has a sophisticated modular concept that makes it possible to build large applications, avoid name clashes and to hide information from unauthorized access.
- Stream I/O: The ECLⁱPS^e I/O is based on the concept of *streams* that are mapped on the I/O channels of the underlying operating system.
- On-line Documentation: The PROLOG Built-in Predicate Reference Manual is available on-line. Calling `help(PredSpec)` will display the appropriate manual page.

Chapter 3

Graphical User Interface.

“The hope is that, in not too many years, human brains and computing machines will be coupled together very tightly and that the resulting partnership will think as no human brain has ever thought and process data in a way not approached by the information-handling machines we know today.” J.C.R Licklider, 1960 outline of “Man-Computer Symbiosis”

3.1 Introduction

A user interface is the boundary between a computer system, comprising of hardware and software, and the human user. User interface software is a significant component of contemporary computer systems and graphical user interfaces (GUI's), are now ubiquitous.

The main challenge in designing a temporal expert system shell is developing a graphical user interface with the following characteristics [Norm95]:

CHAPTER 3. GRAPHICAL USER INTERFACE

- Easy to use,
- Easy to learn,
- Effective and efficient,
- Completely represents the situation,
- Logic system independent,
- Incorporates semantic graphic symbols and icons with meaningful colors.

This chapter begins by considering notions of correctness and means of achieving it. Then, it discusses the significance of Human Computer Interaction (HCI). Finally, this chapter discusses the tools and techniques for graphical user interface design focussing on the significance of color, graphics and icons.

3.2 What is Correctness?

The IEEE Standard Glossary of Software Engineering Terminology [Ises94] defines *correctness* in terms of freedom from faults, meeting of specified requirements, and adherence to user needs and expectations. Software correctness is more commonly described using the terms *validation* and *verification*:

CHAPTER 3. GRAPHICAL USER INTERFACE

Validation asks

Are we building correct software?

Verification asks

Are we building the software correctly?

Both validation and verification are important. Validation failure constitutes a breach of contract between the developer and the client for whom the software is being produced. Verification failure results in software containing potential faults or flaws. Clearly, neither is desirable.

Although correctness is important for software in general, it is particularly important for graphical user interfaces. The graphical user interface represents the aspect of the software that is directly perceived by a user. If the user interface is incorrect, the software will be perceived as incorrect or inadequate, regardless of the correctness of the underlying functionality.

Confidence in the correctness of a graphical user interface is usually achieved by prototyping or by testing. Prototyping can be used to validate and to verify that the interface meets usability requirements.

3.3 Significance of HCI in designing GUI

Human computer interaction is a discipline concerned with the design, evaluation and implementation of interfaces for interactive computing systems for human use and the study of major phenomena surrounding them.

Donald Norman [Norm95] suggests three key principles that help to ensure good HCI:

- Visibility,
- Affordance,
- Feedback.

Controls need to be *visible* with good mapping with their effects and their design should also suggest their functionality. *Affordance* refers to the properties of objects, i.e., the operations and manipulations that can be done to a particular object. For example, doors can afford opening, and chairs afford support. The concept of *feedback* is straightforward, there should be a response to actions. This gives a sense of closure and certainty that the command has been received or the operation has been successful.

In order to produce computer systems with good usability, it is essential to keep in mind the following:

CHAPTER 3. GRAPHICAL USER INTERFACE

1. The user interface is a large and complex component of a software system. Needs of the user are very critical. Designers should survey the intended users of the system before making decisions, since incorrect assumptions about the users may lead to inappropriate design decisions.
2. A new system's interface should be compatible with other interfaces, taking into consideration technology advances. This principle is demonstrated in software for Microsoft Windows and for the Apple Macintosh. Most software for these environments have the same basic interface techniques.
3. When a designer designs an interface it is a mistake to provide too much functionality. This flaw can result in an interface that is complex and difficult for novice users to navigate. However, Mayhew [Mayh92] suggests that it is possible to provide an interface that is rich in functionality, but easy to use.
4. The user interface should be organized so that users can perform more than one task at a time and switch easily between tasks. This usability feature is demonstrated in the Microsoft Windows multitasking environment. A modal dialog window requires that the user interact with it before interacting with any other window, e.g., password dialog box.

CHAPTER 3. GRAPHICAL USER INTERFACE

5. A system should always respond to a user's input. The user should be kept abreast with the internal processes that are being executed by the computer. Messages like "Please Wait ..." or "Working ..." should be used to let the user know that the system is executing a process.
6. Facilities for direct manipulation with the interface. The user should be able to perform actions on visual objects instead of using a command interface. This is a typical feature of all window environments. The interface usually gives the user point and click access. Direct manipulation provides a direct and easy to use interface.
7. Information pertaining to the internal functions of the system should not be presented to the user, since this type of information can be confusing. If this is unavoidable, then the information should be presented in a simple and convenient manner.
8. Flexibility is an important principle, since it accommodates variations in the user's skills and preferences. This could be one of the difficult features to incorporate.
9. A system should tolerate errors made by the user. System crashes should be minimized and simple recovery measures presented, such as an UNDO button. An over sensitive system will inhibit the user's ability to learn the system and their productivity by making them work slower to avoid errors.

CHAPTER 3. GRAPHICAL USER INTERFACE

Underlying all HCI research and design is the belief that the people using a computer system should come first. Their needs, capabilities and preferences for performing various activities should drive system design and implementation. People should not have to change radically to fit in with the system, the system should be designed to match their requirements.

Our ability to attend to one event from among competing stimuli in an environment has been psychologically termed as Focused Attention [Norm95]. The streams of information we choose to attend to, will tend to be relevant to the activities and intentions that we have at that time. When we attempt to attend to more than one thing at a time, it is called Divided Attention. The means for guiding attention within the context of a GUI are:

- Important information or information that needs immediate attention should be displayed in a prominent place to catch the user's eye.
- Screens should be structured for easy navigation.
- Information should be grouped and ordered into meaningful parts.
- Alerting techniques such as error dialog box, reverse video and auditory warnings should be used.

CHAPTER 3. GRAPHICAL USER INTERFACE

- Windows should be used to partition the computer screen into discrete or overlapping sections.

3.4 Tools and techniques for designing a GUI

People interact with their world through the mental models that they have developed. Specifically, the ideas and the abilities they bring to the job are based on the mental models that they develop about that job. As interface designers we need to help the user in developing mental models of the system that will aid him or her to understand the job and perform the task.

The proper use of color, graphics, symbols and icons communicates facts and ideas more quickly and aesthetically to the user. In the following sections, color, graphics and icons are discussed in further detail.

3.4.1 Color

Color provides an effective way to structure information as well as makes the environment pleasant and enjoyable to look at. Excessive use of colors, however, results in color pollution. Many cognitive tests have been performed to find out the relevance of colors and the main findings are:

CHAPTER 3. GRAPHICAL USER INTERFACE

- **Segmentation:** Color is a very powerful way of dividing a display into separate regions. Color can be used to emphasize as well as categorize data. It can also facilitate searching through information.
- **Simplicity:** It is more difficult to use color effectively than it is to use it ineffectively. Simplicity is important in the design of color interfaces. Do not attach more than one meaning to a color.
- **Restrict number of colors with meaning:** The magic number [Mill56] for short-term memory is seven plus or minus two.
- **Consistency:** The intuitive ordering of color can help establish intuitive consistency in the design. The spectral and perceptual order red, green, yellow, blue can guide the order of concepts attached to colors. Red is first in the spectral order and focuses in foreground, green and yellow focus in the middle, while blue in the background.
- **Prominence:** Color should be used to make features prominent. For example, currently active files could be shown in a different color. Standardized interface colors should be established and used across the development. For example, red is a good color to alert the user for an error. Yellow is appropriate for a warning message, and green to show positive progress.

CHAPTER 3. GRAPHICAL USER INTERFACE

Table 3.4.1 shows good and bad color combinations [Panc95]. As explained in the table below, for a white background, the best text color would be black or blue and the worst color would be cyan and yellow.

Background	Best Colors	Worst Colors
White	Black, Blue	Cyan, Yellow
Black	Yellow, White	Blue
Red	Black	Blue, Magenta
Green	Black, Red	Cyan
Blue	Red, White, Yellow	Black
Cyan	Blue, Red	Green, White, Yellow
Magenta	Black, Blue	Cyan, Green

Table 3.4.1 Color Combinations for Graphical User Interfaces.

Marcus [Marc90] has given some effective suggestions for creating good interfaces:

- Use dark colors for background.
- Opposite colors go well together.
- Avoid the simultaneous display of highly saturated, spectrally extreme colors. For example, bright violet with red.
- Avoid using adjacent colors that differ only in the amount of pure blue.
- Use bright colors for danger or for getting the user's attention.

3.4.2 Graphics

Graphics help to present objects in a pictorial rather than verbal form. It is true that a picture is worth a thousand words and a chart is worth a dozen tables of numbers. The human ability to extract information from visual scenes is more fundamental than the ability to manipulate data arithmetically. Graphics can be used for the following [Marc90]:

- To display complex relationships,
- To show component relationships (e.g. mimic display of car),
- For dynamic data,
- Map display for geographic data,
- To display trends/projections (e.g. stock market data),
- For quick interpolation.

3.4.3 Icons

Icons are small pictorial images that are used to represent system objects, application tools such as those for drawing, utilities and commands. For example, as part of a desktop metaphor, objects associated with working at an office desk such as phone diaries,

CHAPTER 3. GRAPHICAL USER INTERFACE

schedulers are depicted as icons. These icons reduce the complexity of the system, making it easier to learn and use.

Advantages to using icons:

- Icons are easy to understand.
- Icons save space.
- Icons can permit international use.
- Icons are effective for tasks that require a diversity of manipulative operations to be performed (e.g., range of drawing and painting techniques.)
- Icons act as mnemonic tags for tasks where large amounts of information have to be readily identified.

3.5 Other expert system shells

Some of the expert system shells are:

ACQUIRE: ACQUIRE [Acqi97] is a knowledge acquisition system and an expert system shell. It is a complete development environment for building and maintaining knowledge-based applications. It provides a step-by-step methodology for knowledge engineering that allows the domain experts themselves to be directly involved in structuring and encoding the knowledge. Features include a structured approach to knowledge acquisition, a model of knowledge acquisition based on

CHAPTER 3. GRAPHICAL USER INTERFACE

production rules and decision tables, handling uncertainty by qualitative, non-numerical procedures.

FOCL: FOCL [Focl92] is an expert system shell written in common LISP.

It learns Horn Clause programs from examples and background knowledge. The expert system includes a backward-chaining rule interpreter and a graphical interface to the rule and fact base.

FLEX: FLEX [Flex94] is a hybrid expert system shell available across a wide range of different hardware platforms which offers frames, procedures and rules integrated within a logic programming environment.

BABYLON: BABYLON [Baby96] is a development environment for expert system. It includes frames, constraints, prolog-like logic formalism, and a description language for diagnostic applications. It is implemented in common LISP and has been ported to a wide range of hardware platforms.

In summary, user interface software is complex, highly interactive, modeless, concurrent, graphical, and has user-based real-time requirements. Emerging user interface technologies such as multi-modal interaction, multi-media and intelligent agents will give it a prominent position in any software development process.

Chapter 4

System overview

“To every thing there is a season, and a time to every purpose under the heaven; A time to be born and a time to die; a time to plant, and a time to pluck up that which is planted; A time to kill, and a time to heal; a time to break down, and a time to build up.”

- The Bible, Ecclesiastes, 3

Tcl/Tk is a development toolkit for building GUI's. The Tool Command Language (Tcl) is the embedded scripting language and Toolkit (Tk) builds widgets on top of Tcl.

This chapter begins with an introduction to Tcl/Tk. The model, illustrating the various phases involved in building a temporal expert system shell, is explained. Then, this chapter talks about system specific details including entering, representing and querying information.

4.1 Tcl/Tk

Tcl is a scripting language for controlling and extending applications. It provides generic programming facilities such as variables, loops and procedures that are useful for a variety of applications.

Tk is the graphical user interface extension for applications created using Tcl. Tk extends core Tcl facilities with commands for building user interfaces. Tk is used to create widgets, arrange widgets, and bind events to Tcl commands.

Various advantages [John97] of using Tcl/Tk for developing graphical user interfaces are:

- Scripting language: While there are a few compilers, the vast majority of Tcl users run their programs as scripts. Scripts are easier to develop and faster to execute than full-fledged C or C++ programs because there is no need to compile and link the program. The Tcl user just executes the Tcl interpreter and runs the script.
- Easy to learn: Unlike other programming languages, Tcl/Tk is easy to learn and use.
- Works on many different platforms: Tcl works well in the Unix environment, particularly because developing graphical applications on Unix under Motif tends to be a troublesome task. This is one of the main reasons for the success of Tcl/Tk. On the Windows side, versions of Tcl exist for Windows 3.1, Windows 95 and Windows NT. The latest version of Tcl also runs on Apple's Macintosh platform.

CHAPTER 4. SYSTEM OVERVIEW

- Extension and modification: One of the main uses of Tcl is that it is possible to add commands to Tcl, thereby, extending the language. The Tcl interpreter is a C function that can be linked with various applications.
- Tcl/Tk is free: Tcl/Tk is available for free, on the Internet. This feature of the language makes it popular amongst the student community.
- Works well with the Internet: Tcl includes a number of built-in features that make working with World Wide Web pages easier. The text widget, for example, supports tags that help in creating hypertext links in the text. Tcl is good for CGI scripts as well.

4.2 Conceptual Model

We use Tcl/Tk to implement the GUI. The GUI is one component of our system. The block diagram in figure 4.2 gives an overview of the system.

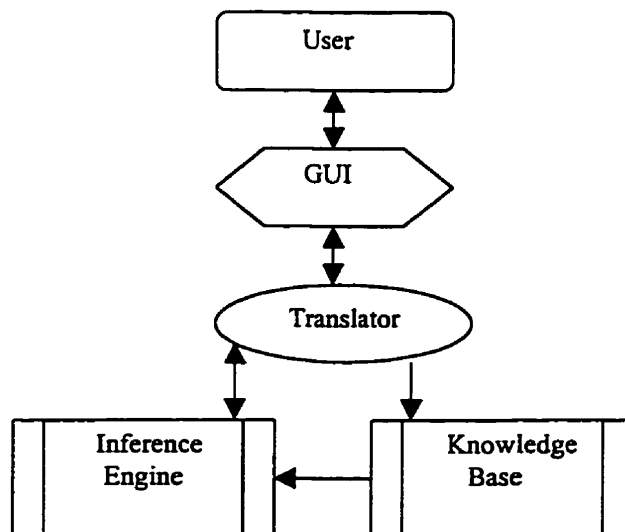


Figure 4.2 Conceptual Model

CHAPTER 4. SYSTEM OVERVIEW

The GUI provides the environment for representing, entering and querying information. The translator is a set of Tcl instructions, which provides a link between the graphical user interface and the inference engine. It maps the information entered by the user onto the logical form and stores it as facts in the knowledge base. Every fact contains the following information:

- Unique color that represents an event.
- Temporal information associated with that event.

The inference engine is the backbone of the temporal expert system shell. It interacts with the knowledge base, which contains facts to solve the queries requested by the user. The result of the query is sent back to the translator, where it is mapped from logical to user understandable graphical form. The inference engine and knowledge base are discussed in chapter 5. The graphical user interface and translator are discussed in the remainder of this chapter.

4.3 Graphical User Interface

- **Entering Information:**

The screen shown in figure 4.3.1 is used to enter information. An event is defined as a single item of temporal information. Every event has a name, description and a temporal component. For example:

- John had a meeting at 4 p.m.

CHAPTER 4. SYSTEM OVERVIEW

The event name for this event is “Meeting”, the event description is “John had a meeting at 4 p.m.” and the temporal component is that it is true at 4 p.m. Another example is:

- I went jogging between 5 p.m. and 5:30 p.m.

The event name for this event is “Jogging”, the event description is “I went jogging between 5 p.m. and 5:30 p.m.” and the temporal component is that it is true between 5 p.m. and 5:30 p.m.

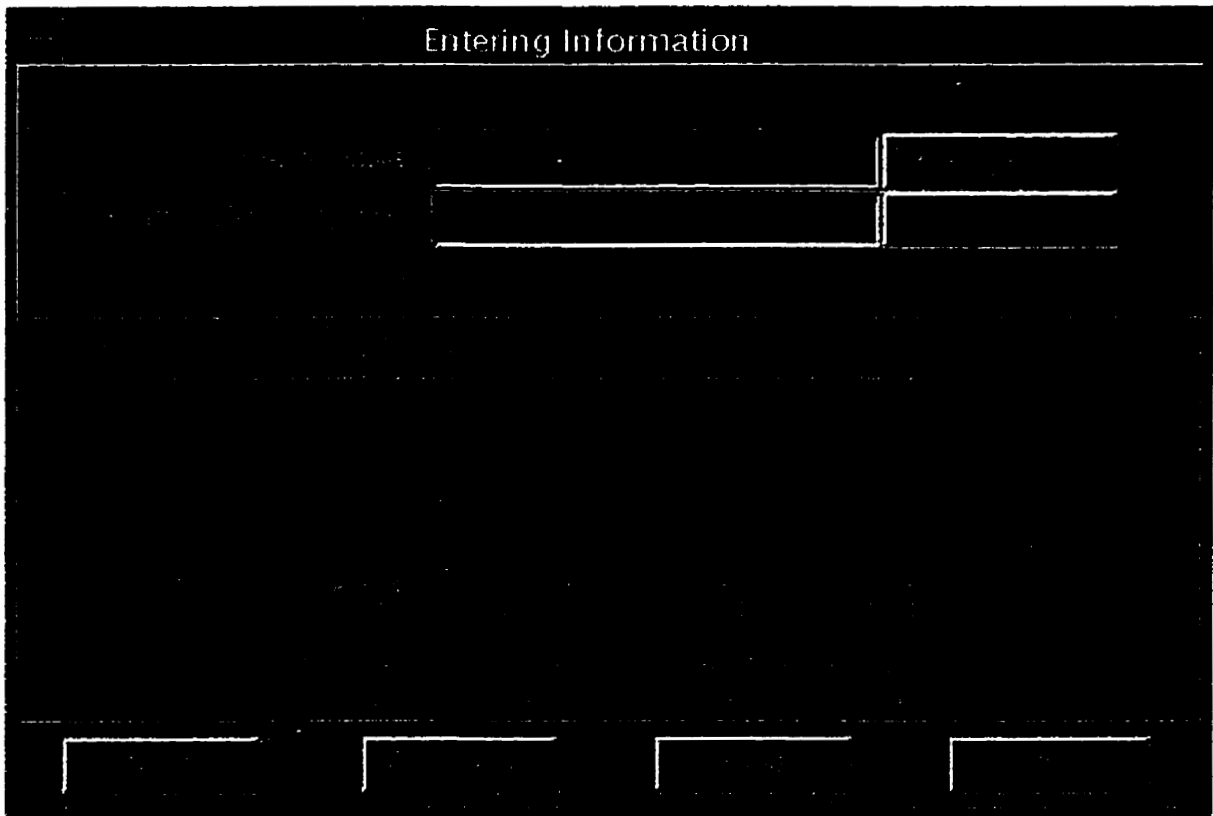
The image shows a screenshot of a computer screen with a black background and white text and lines. At the top center, the text "Entering Information" is displayed. Below this title, there is a large rectangular area containing several input fields. On the left side of this area, there is a vertical list of labels, though they are mostly illegible due to the low resolution. To the right of these labels are several horizontal input boxes. One box is notably taller than the others. At the bottom of the screen, there are four small, separate rectangular input boxes arranged horizontally.

Figure 4.3.1 Screen for Entering Information

The text entry box labeled “Event Name” is used for entering the name of an event. The “Event description” text box provides a location

CHAPTER 4. SYSTEM OVERVIEW

for adding event details. The user can move from one data field to another either by tabbing to, or by clicking in the desired field. The system prompts the user with an error message, if any of these text boxes are left empty.

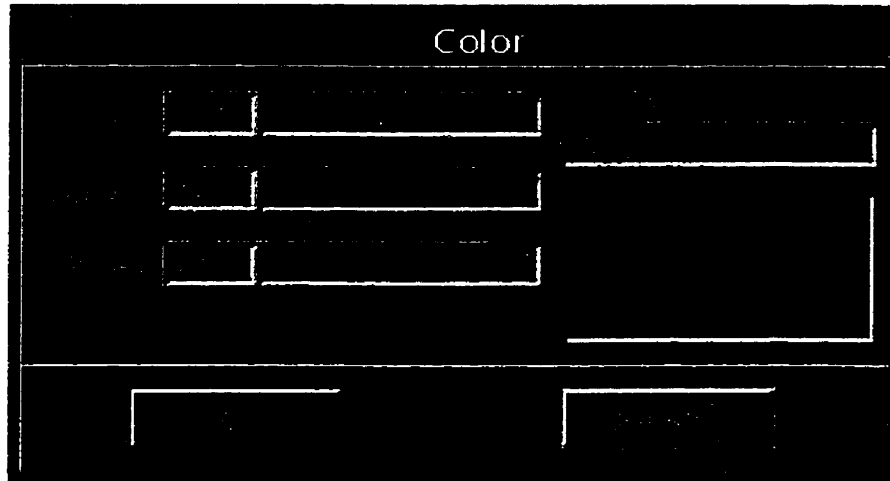


Figure 4.3.2 Color Window

The next step is to pick a color. The “Pick Color” button in figure 4.3.1 invokes the color window shown in figure 4.3.2. The user can select a color for an event by filling in the numerical values for red, green and blue. Numerical values for each of these primary colors range between 0 and 254. It is also possible to select a color by sliding the triangular tab on the color scale for each color. The selection text box and the color display window (they both appear in the right hand side of the window in figure 4.3.2) change their color values as the user slides the triangular tab on the color scale. The user confirms his/her selection by clicking on the “OK” button.

CHAPTER 4. SYSTEM OVERVIEW

Every event is represented by a unique color. This unique color is used as an index for the event. We can represent a total of $255 \times 255 \times 255$ events.

The unique color associated with each event is used in querying information from ECLIPSe and in maintaining database integrity. It should be noted that if the user attempts to pick the same color for more than one event, the system flags an error message and prompts the user to pick another color. As a visual cue, symbolic icons used later will have the same color as the event. The user can change or modify any information entered up to this stage.

There are different categories of events that can be represented. Depending upon the category of an event, the user chooses an icon from one of the boxes labeled "Point Event", "Limitless Event", etc. in figure 4.3.1 and appropriately places it on the time scale. The time scale appears near the center of figure 4.3.1. The user can modify or change the position of the icon/icons on the time scale. The scale represents a 12-hour time period between 12 a.m. and 12 p.m. The minimum unit of time on the scale is 15 minutes, which corresponds to 0.25 centimeters on the ruler. The smallest unit of time that can be represented on the scale is called a grid interval.

In order to select an icon, the user points and clicks on the desired icon, which can then be dragged along with the mouse pointer to any position on the time scale. The icon may be positioned at any of the grid

CHAPTER 4. SYSTEM OVERVIEW

intervals. The user may modify the position of the icon on the time scale by pointing and clicking on the icon and dragging it along with the mouse pointer to a new position.

It is important to note that an icon can only be positioned on the time scale and it becomes invisible if the user tries to place it at any other location on the canvas. As a visual cue, clicking on an icon will change its color to red, which indicates that it is currently selected by the user. Icons are drawn to resemble the temporal extent that they represent.

After entering the current event, the user clicks on the “Continue” button to add another event. Finally, clicking on the “Done” button closes the current window, saving the details to a file and control is transferred back to the main application window. As the name suggests, the “Cancel” button aborts the current operation, clears the screen and takes the control back to the “Event Name” text box. The “Help” button provides an online help facility for the current screen and guides the user during the processing sequence. This helps the user to learn the interface, and lessens the probability of undesired results.

The following explains the various categories of information that can be represented using the symbolic icons.

- Point event: It represents those events which occur at precise points.

For example,

John called at 2:30 a.m., 4:45 a.m., 6:15 a.m. and 12 p.m. (1)

CHAPTER 4. SYSTEM OVERVIEW

This can be represented using the “Point Event” by placing the icon at the appropriate time points on the scale, as shown in figure 4.3.3. It is important to note that the user selects the “Point Event” icon four times in order to position it at four different locations – 2:30 a.m., 4:45 a.m., 6:15 a.m. and 12:00 p.m.

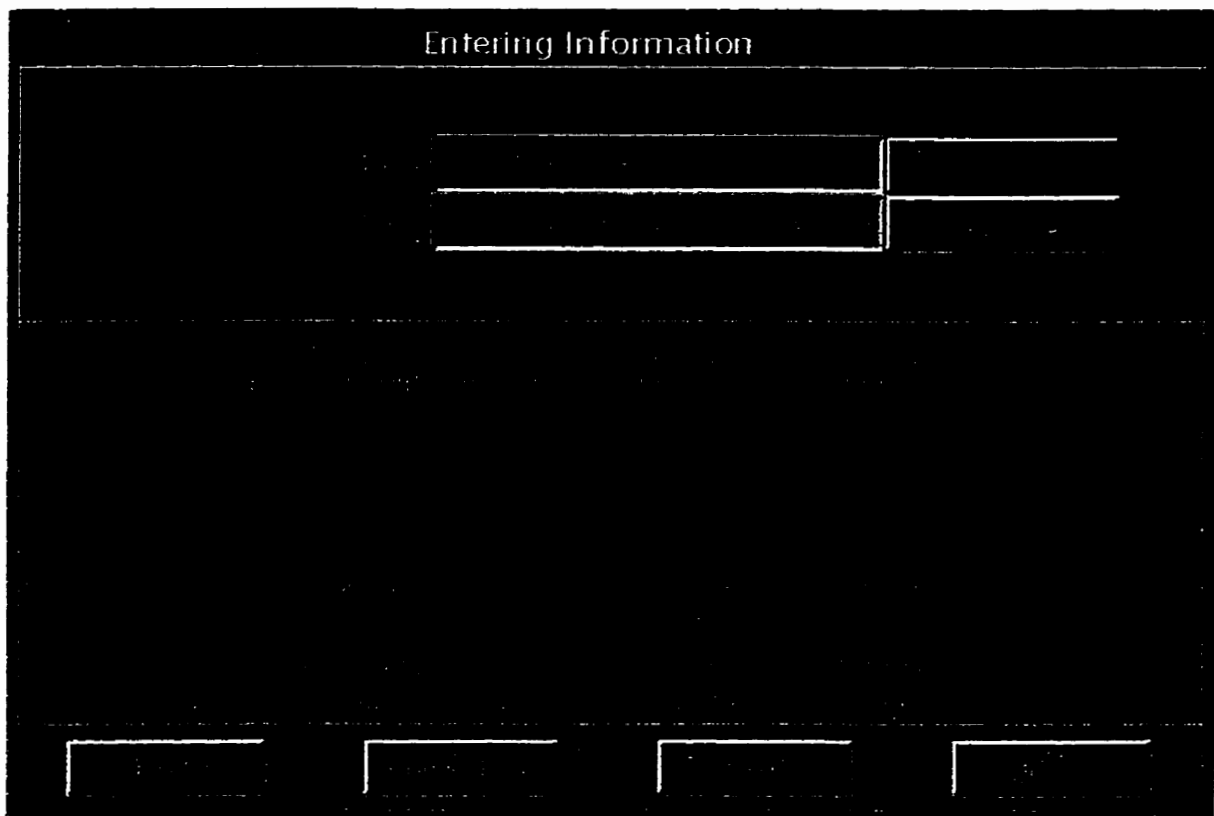


Figure 4.3.3 Point Event.

- **Limitless Event:** This icon represents an event that started at some unknown time in the past and continues until an unknown time in the future. For example,

It was raining throughout the day. (2)

(Note that in this example, “day” represent a 24 hour time period.)

CHAPTER 4. SYSTEM OVERVIEW

This event possibly started before 12 a.m. and ended after 12 p.m. but was definitely true between 12 a.m. and 12 p.m. When the user places the "Limitless Event" icon on the time scale, it automatically configures itself to cover the entire scale, as shown in figure 4.3.4.

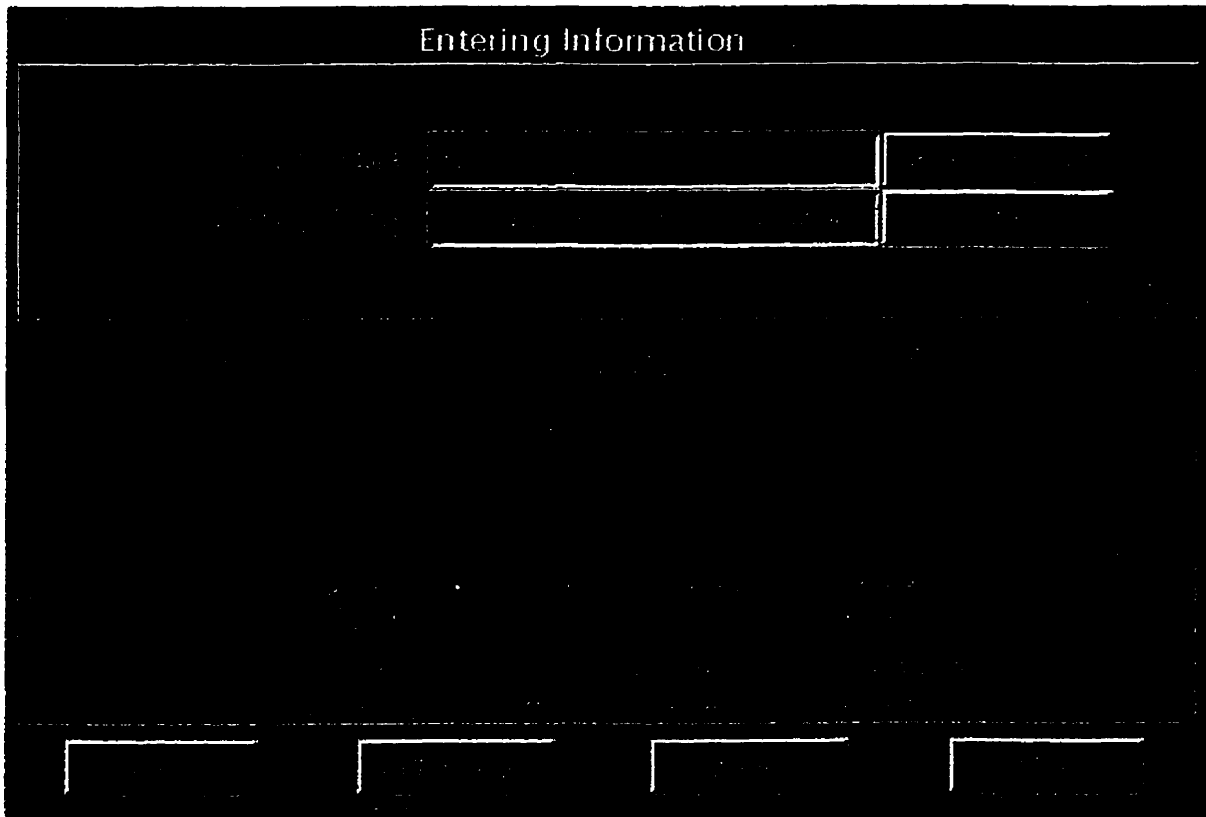


Figure 4.3.4 Limitless Event

- Fixed Event: It represents those events, which have a precise starting and ending point. For example,

I went jogging between 6 a.m. and 8 a.m. (3)

CHAPTER 4. SYSTEM OVERVIEW

“Jogging” has a precise starting and ending point and is represented using the “Fixed Event” icon. The above event is true between 6 a.m. and 8 a.m. To position it on the time scale, the user points and clicks on the “Fixed Event” icon and drags it to the appropriate position on the time scale.

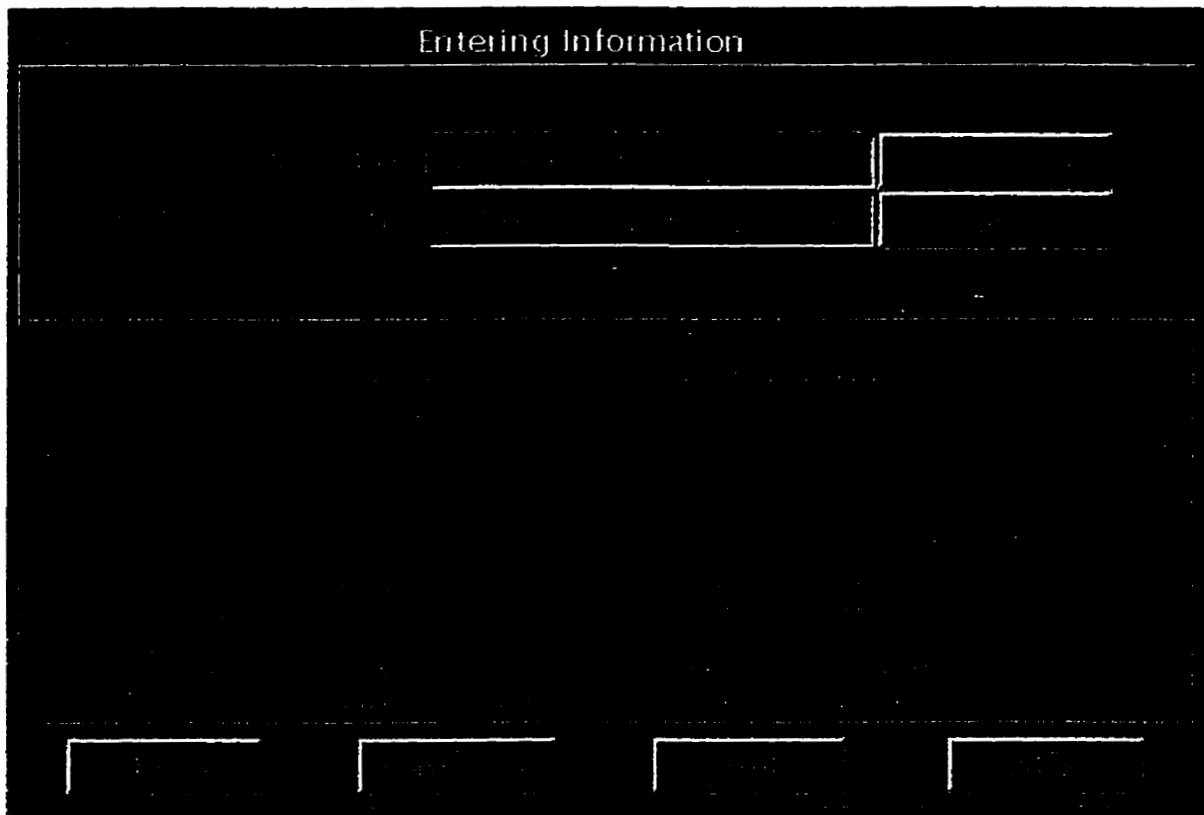


Figure 4.3.5 Fixed Event

The user can modify the length of this icon on the time scale by clicking on a small square attached to the icon (one is shown at 8 a.m. in figure 4.3.5) and dragging it with the mouse. This configures the “Fixed Event” icon to a desired length. As a visual cue, the small square attached to this icon changes its color to red, when the user

CHAPTER 4. SYSTEM OVERVIEW

brings the mouse pointer over it. The minimum time interval that this icon can represent is 15 minutes.

- FixedLeft Event: This icon represents an event which starts at a known fixed point and continues into the future. For example, The basketball game started at 9 a.m. and finished in the afternoon.

(4)

This event is definitely true between 9 a.m. and 12 p.m. and it ends after 12 p.m. The user places the left end of the “FixedLeft Event” icon at 9 a.m. on the time scale and the right end of the icon automatically covers the rest of the scale indicating that the event continues into the future as shown in figure 4.3.6.

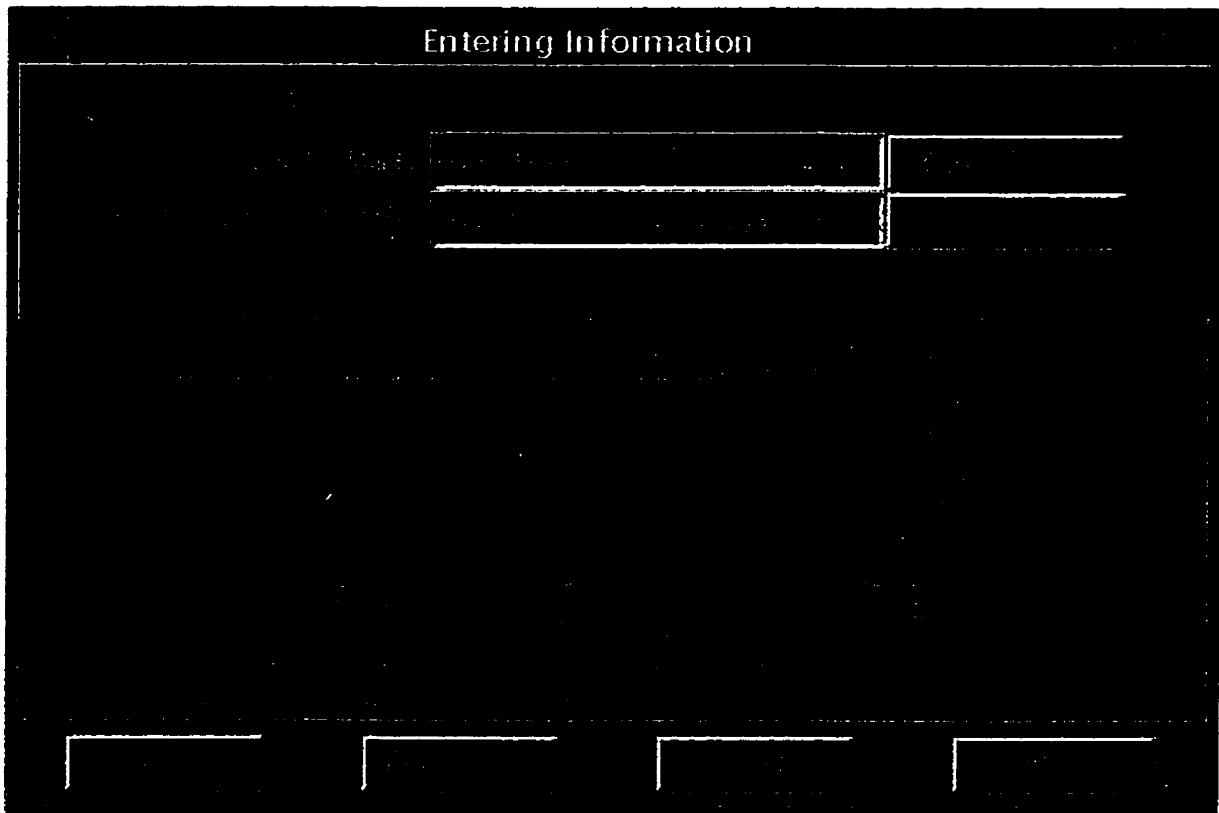


Figure 4.3.6 FixedLeft event

CHAPTER 4. SYSTEM OVERVIEW

- FixedRight Event: An event that started at some unknown time in the past and has a fixed ending point is represented by this icon. For example,

The Snow storm started last night and ended today at 10 a.m. (5)

It is important to note that the event started at some point before 12 a.m. and continued till 10 a.m. Event "Snow storm" was definitely true between midnight and 10 a.m.

The user places the right end of the icon at 10 a.m. on the time scale and the left end automatically covers the rest of the scale. It indicates that the event started sometime before 12 a.m. and continued till 10 a.m. as shown in figure 4.3.7.

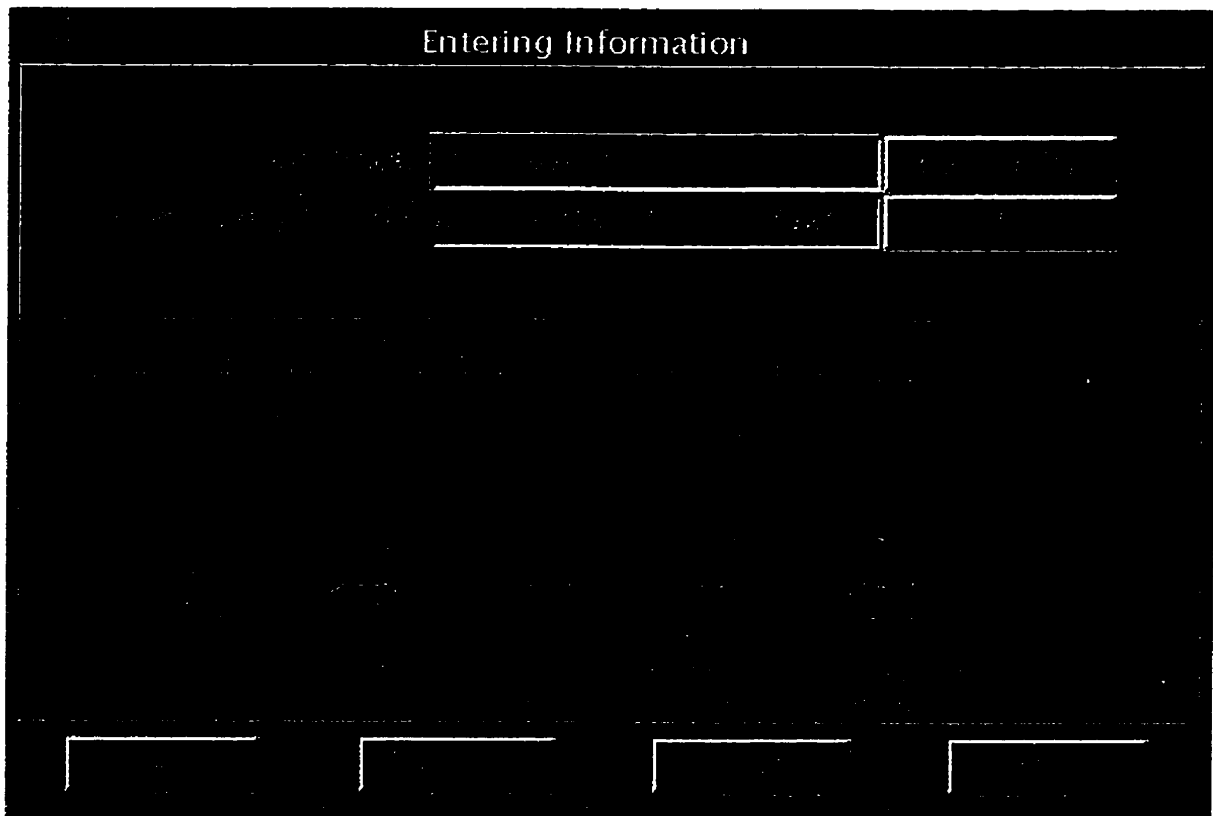


Figure 4.3.7 FixedRight Event

CHAPTER 4. SYSTEM OVERVIEW

It is possible to represent events that require the use of more than one type of symbolic icon. For example,

The telephone switch had its peak load at 4 a.m., 6 a.m., 10 a.m.
11:30 a.m. and between 10:00 till 11:30 a.m. (6)

Example (6) requires the use of "Point Event" and "Fixed Event" to capture the event details as shown in figure 4.3.8.

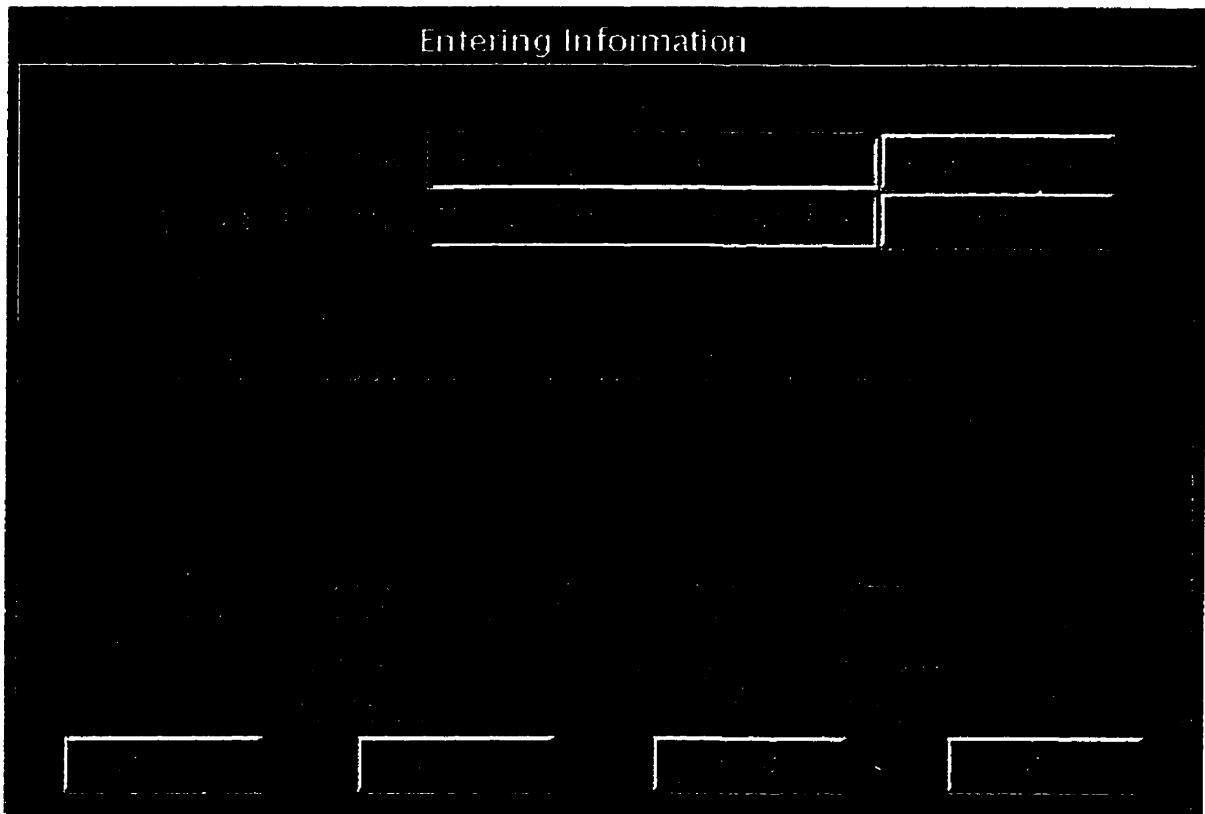


Figure 4.3.8 Point Events and Fixed Event

The user selects the "Point Event" icon four times in order to position it at four different locations: 4 a.m., 6 a.m., 10 a.m., 11:30 a.m.

CHAPTER 4. SYSTEM OVERVIEW

The “Fixed Event” icon is selected once and positioned between 10 a.m. and 11:30 a.m.

Example (6) represents the case where the event occurred between 10 a.m. and 11:30 a.m. including the end points. This is a closed interval. Figure 4.3.5 represents the event “I went jogging between 6 a.m. and 8 a.m.” which is definitely true between 6 a.m. and 8 a.m. No commitment is made at the end points. We represent this event with an open interval.

● Querying the system:

The user can perform the following queries:

1. Is an event true at the given time ?
2. What is true about an event ?

The screen shown in figure 4.3.9 is used to query information. The layout of the query window is similar to the enter information window. Providing consistency among different windows is essential to the design of the graphical user interface.

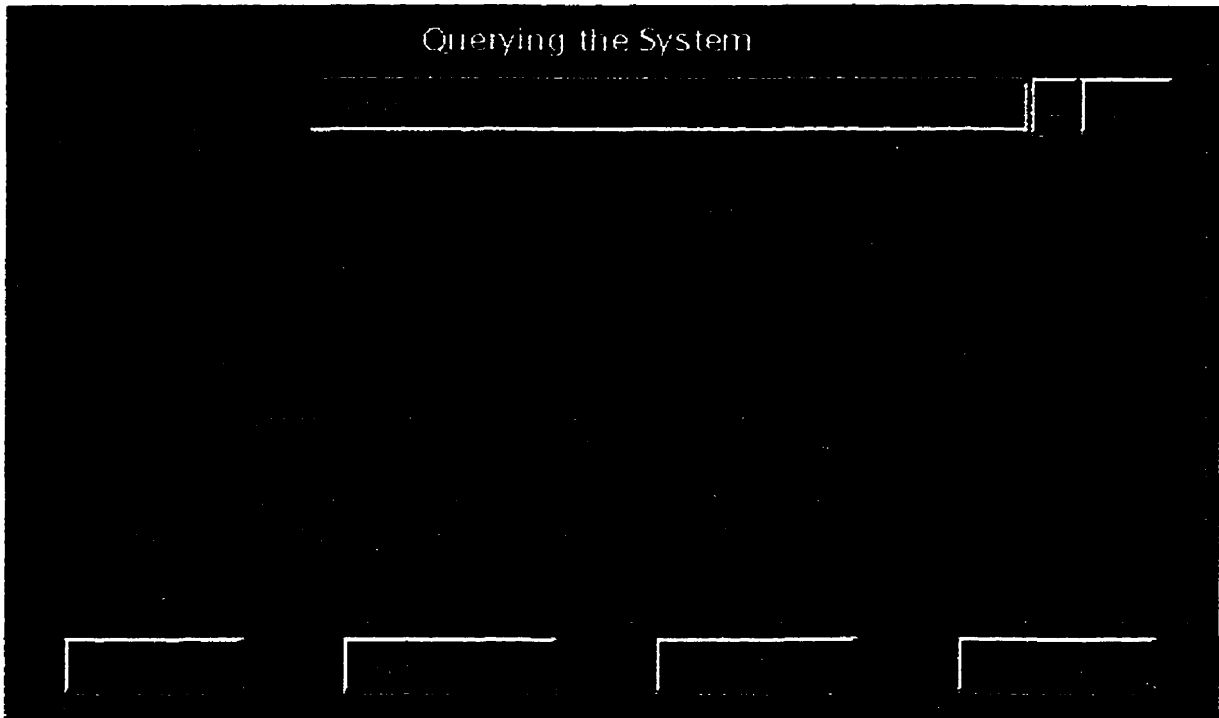


Figure 4.3.9 Screen for Querying the System

“Select Event” is the label for the dropdown event list box shown by a button with a downward arrow sign. The user can view the list of events by clicking on this button as shown in figure 4.3.10. An event is selected from the list by pointing and clicking on it. The user confirms his/her selection by clicking on the “OK” button.

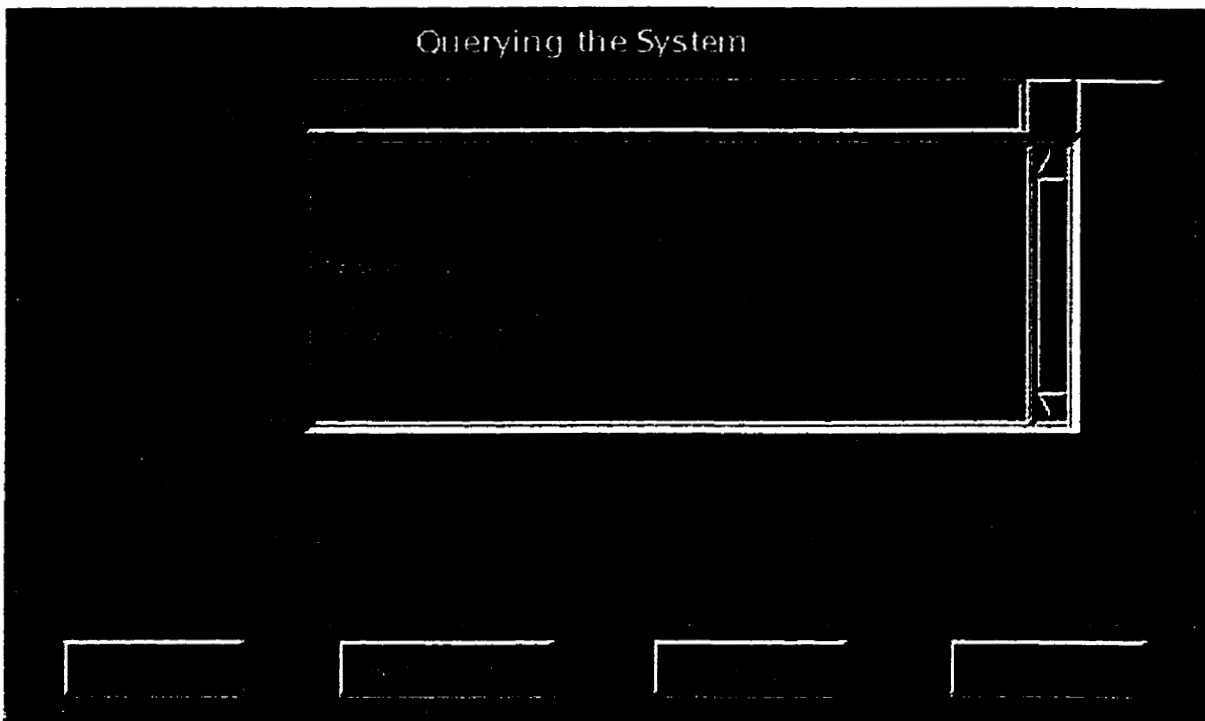


Figure 4.3.10 Query Screen with dropdown list

To query the system about an event, the user selects and positions the appropriate icon on the time scale and clicks on the “Query” button. The system displays the result(s) of the query in a separate dialog window. As the name suggests the “New Query” button allows the user to perform a new query by clearing the screen and returning control to the “Select Event” dropdown list box. The “Exit” button closes the current window and transfers control back to the main application window. The “Help” button provides an online help for the current screen and guides the user during the processing sequence.

When the user clicks on the “Query” button, Tcl opens a pipe. This pipe is used to send properly formatted the query commands to ECL^{PS}^e. The ECL^{PS}^e inference engine interacts with the knowledge base, which is

CHAPTER 4. SYSTEM OVERVIEW

the repository of facts, to solve the queries requested by the user. Output from the inference engine is written to a file, which is parsed by the translator and the results are displayed in a dialog window.

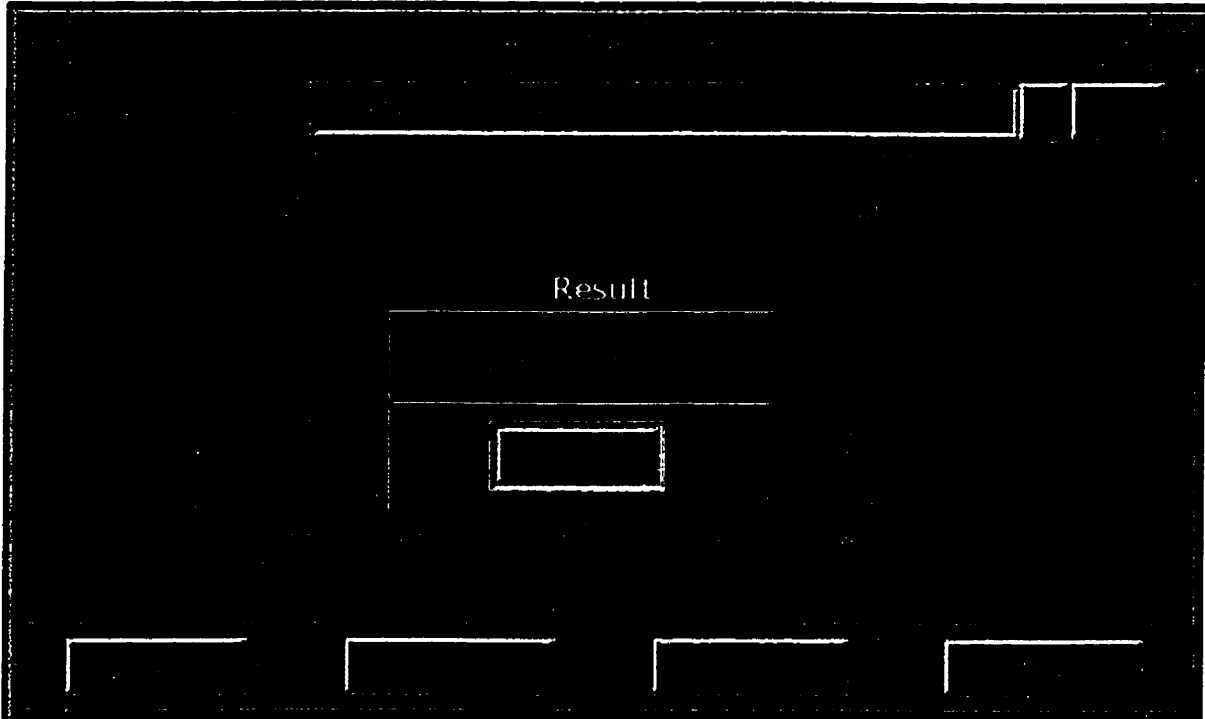


Figure 4.3.11 Querying Point Event

We conclude with a few query examples. The first one is:

- John called at 2:30 a.m., 4:45 a.m., 6:15 a.m. and 12 p.m.

The event name for this event is “Phone calls” and is true at 2:30 a.m., 4:45 a.m., 6:15 a.m. and 12 p.m. The user selects this event from the dropdown event list box by clicking on it and confirms his/her selection by clicking on the “OK” button. The user selects the “Point Event” icon four times and places it at the following four positions on the time scale: 2:30 a.m., 4:45 a.m., 6:15 a.m. and 11 a.m.

CHAPTER 4. SYSTEM OVERVIEW

Clicking on the “Query” button initiates the query procedure and the system displays the result of the query in a dialog window as shown in figure 4.3.11. This query results in “No”. The user should place the “Point Event” icon at 12 p.m. instead of 11 a.m. for the query result to be true.

Figure 4.3.12 shows another query example.

- It was raining throughout the day.

The event name for this event is “Rain”, the event description is “It was raining throughout the day”. This event possibly started before 12 a.m. and ended after 12 p.m., but was definitely true between 12 a.m. and 12 p.m.

The user confirms his/her selection by clicking on the “OK” button and then positions the “Fixed End” icon between 4 a.m. and 7 a.m. The query procedure is initiated by clicking on the “Query” button.

As shown in the figure 4.3.12 the query results in “True”. It means the event “Rain” is true between 4 a.m. and 7 a.m.

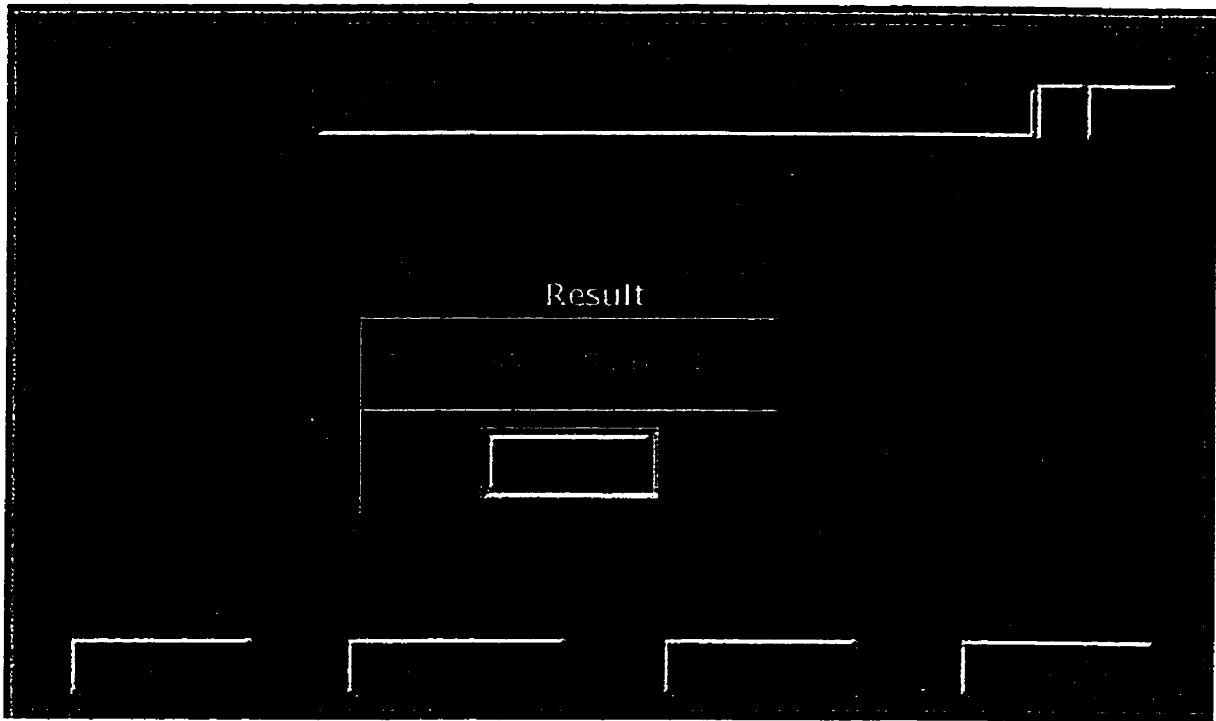


Figure 4.3.12 Querying Limitless Event

The user can find out what is true about any event by selecting an event from the dropdown list box and confirming his/her selection by clicking on the “OK” button. Finally, clicking on the “What’s True” icon, displays a dialog window showing the time at which the event is true.

For example, the user selects the event “Phone calls” from the dropdown list of events and clicks on the “OK” button to confirm the selection. The user then clicks on the “What’s True” icon and the system displays the results of the query using the dialog window shown in figure 4.3.13. The event “Phone calls” is true at points 2:30 a.m., 4:45 a.m., 6:15 a.m. and 12 p.m.

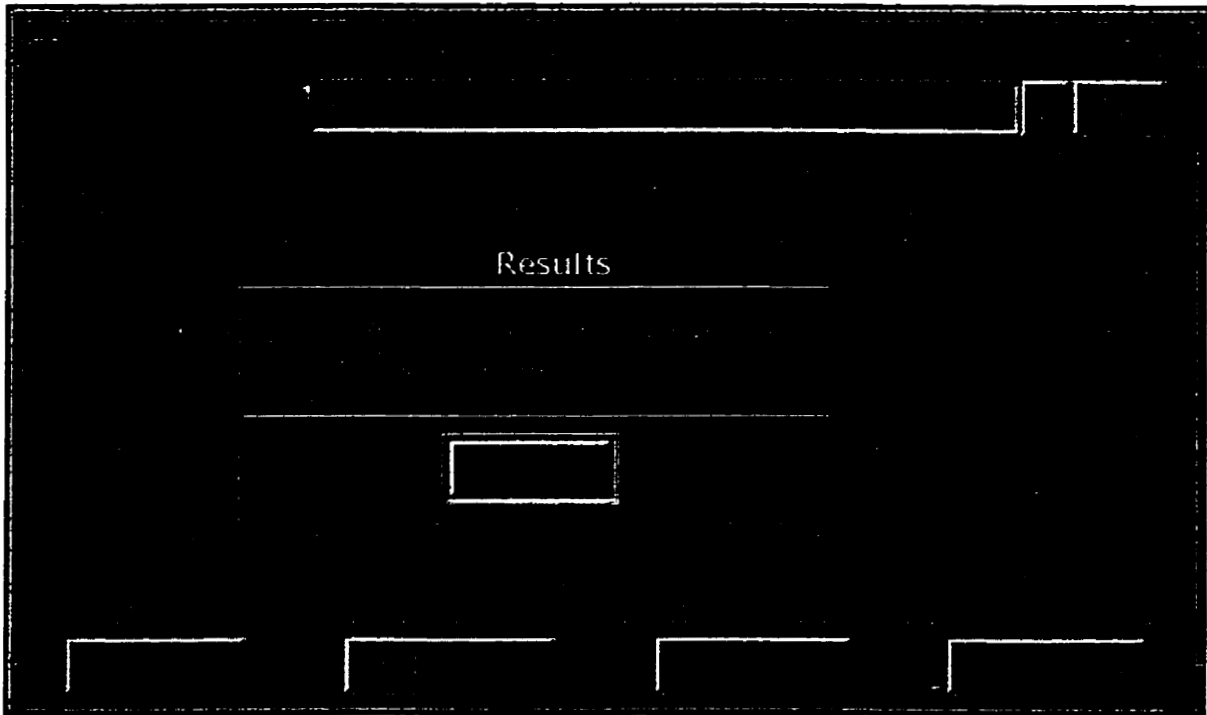


Figure 4.3.13 Query using What's True

We can also do open and closed interval queries. Recall example (3):

- I went jogging between 6 a.m. and 8 a.m.

The event name for this event is “Jogging” and is true over the open interval between 6 a.m. and 8 a.m. The “Fixed Event” icon is positioned between 6 a.m. and 8 a.m. on the time scale along with the “Point Event” icons at 6 a.m. and 8 a.m. respectively, as shown in figure 4.3.14. This makes the event “Jogging” true in a closed interval. The user initiates the query procedure by clicking on the “Query” button. The query results in “No” because the event “Jogging” is not true in a closed interval between 6 a.m. and 8 a.m.

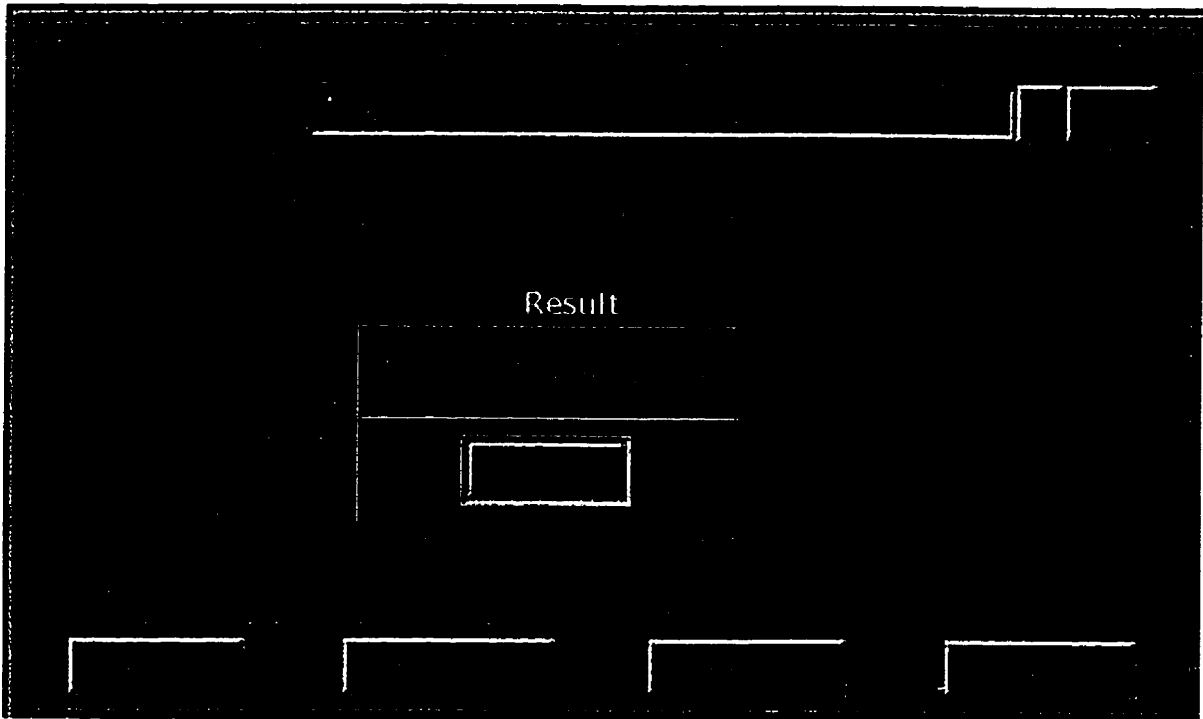


Figure 4.3.14 Point Event and Fixed Event (closed interval)

Let's perform the same query again over the open interval between 6 a.m. and 8 a.m. The "Fixed Event" icon is positioned between 6 a.m. and 8 a.m. on the time scale as shown in figure 4.3.15.

The user initiates the query procedure by clicking on the "Query" button. The query results in "True" because the event "Jogging" is true between 6 a.m. and 8 a.m.

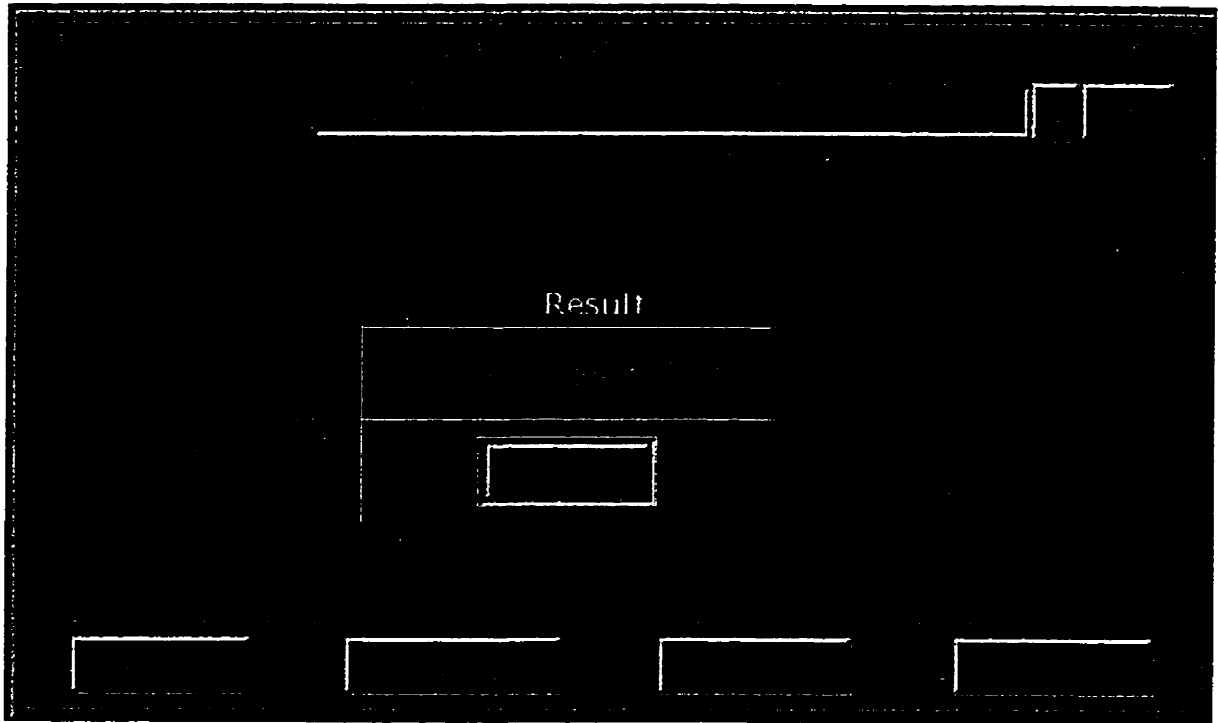


Figure 4.3.15 Point Event and Fixed Event (open interval)

4.4 System Database

Information about the events is stored in the database file and can be viewed by the user. The main application window is shown in figure 4.4.1 with its “Options” menu pulled down.

The “Options” menu provides the following choices: “View Database”, “Colors Used” and “Symbols”.

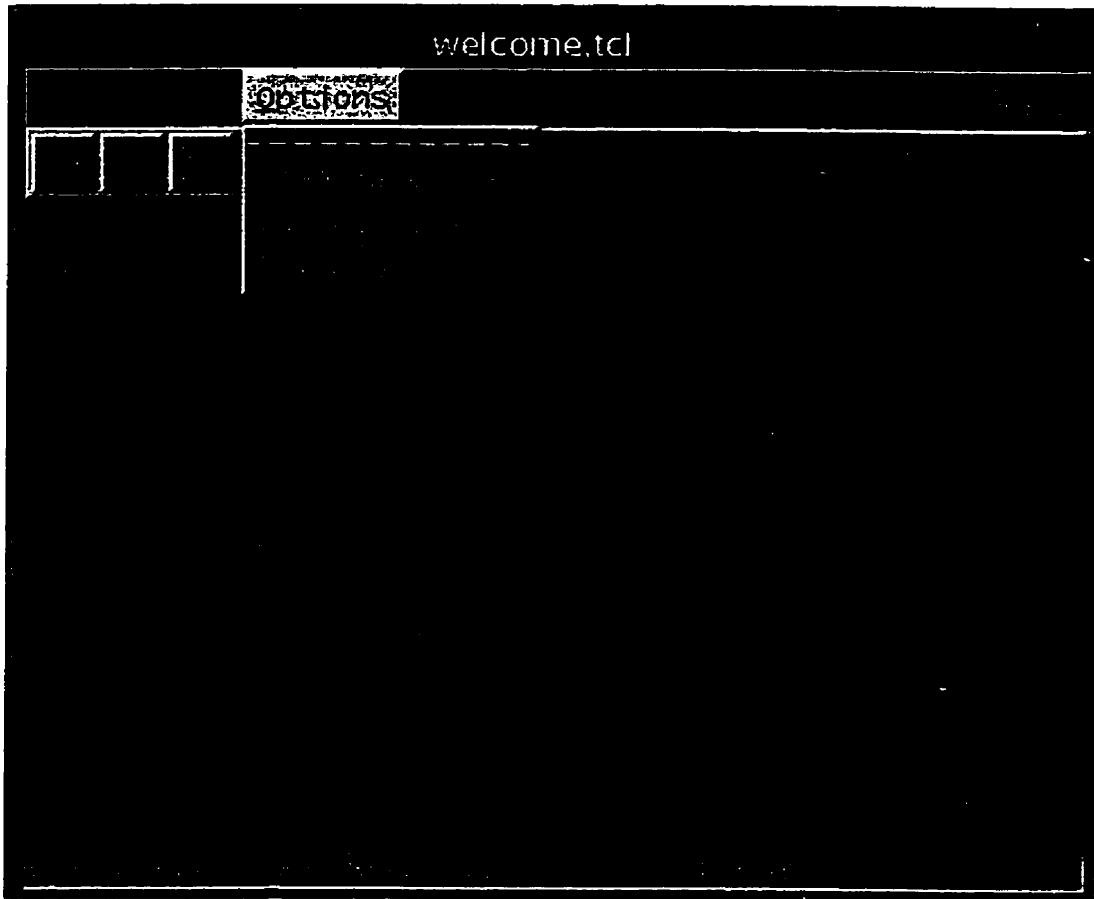


Figure 4.4.1 Main application window

At any point during the processing sequence, the user may access the database.

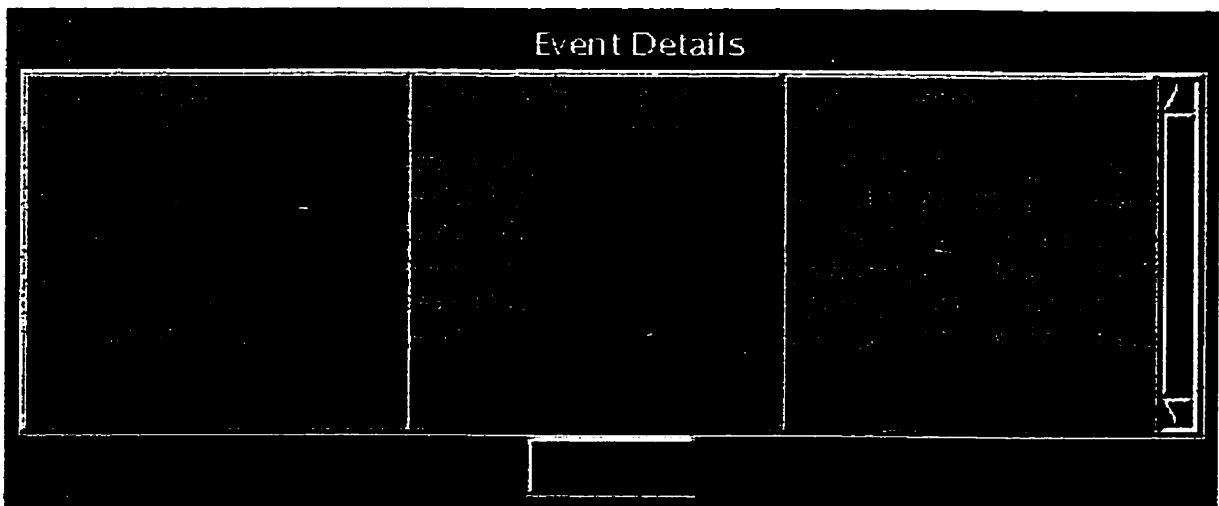


Figure 4.4.2 Event Details

CHAPTER 4. SYSTEM OVERVIEW

Choosing "View Database" from the "Options" menu opens a dialog window showing "Event Name", "Associated Color" and "Event description" in a tabular form as shown in figure 4.4.2.

Similarly, choosing "Colors Used" from the "Options" menu opens a dialog window showing the colors currently in use. The user can scan the colors along with their hexadecimal values. The color window has a vertical and horizontal scroll bar for easy navigation as shown in figure 4.4.3

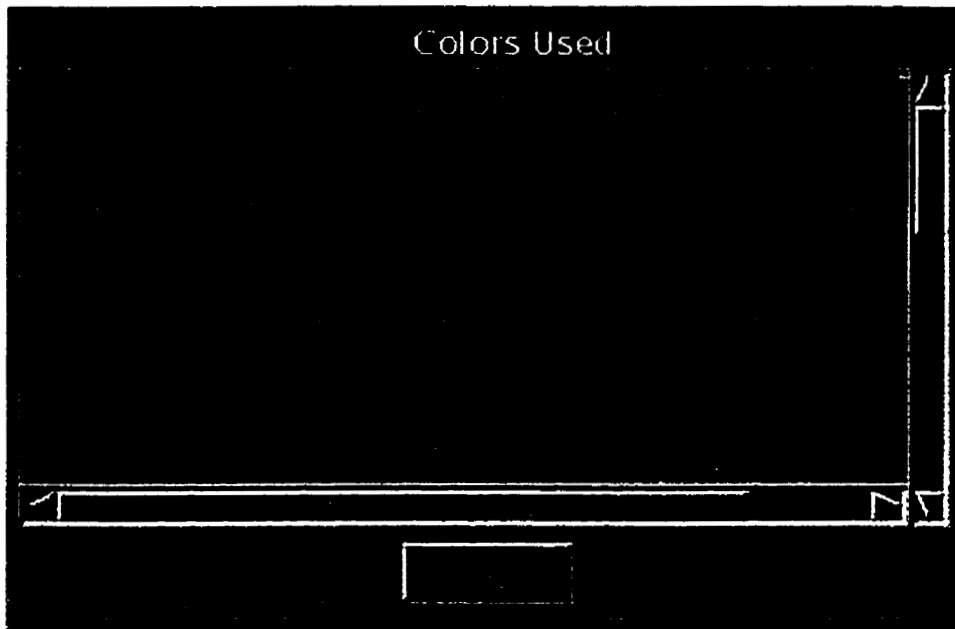


Figure 4.4.3 Colors Used

Clicking on the "Symbols" under the "Options" menu opens a window that displays a list of symbols used. Each symbol is displayed along with its name and meaning as shown in figure 4.4.4.

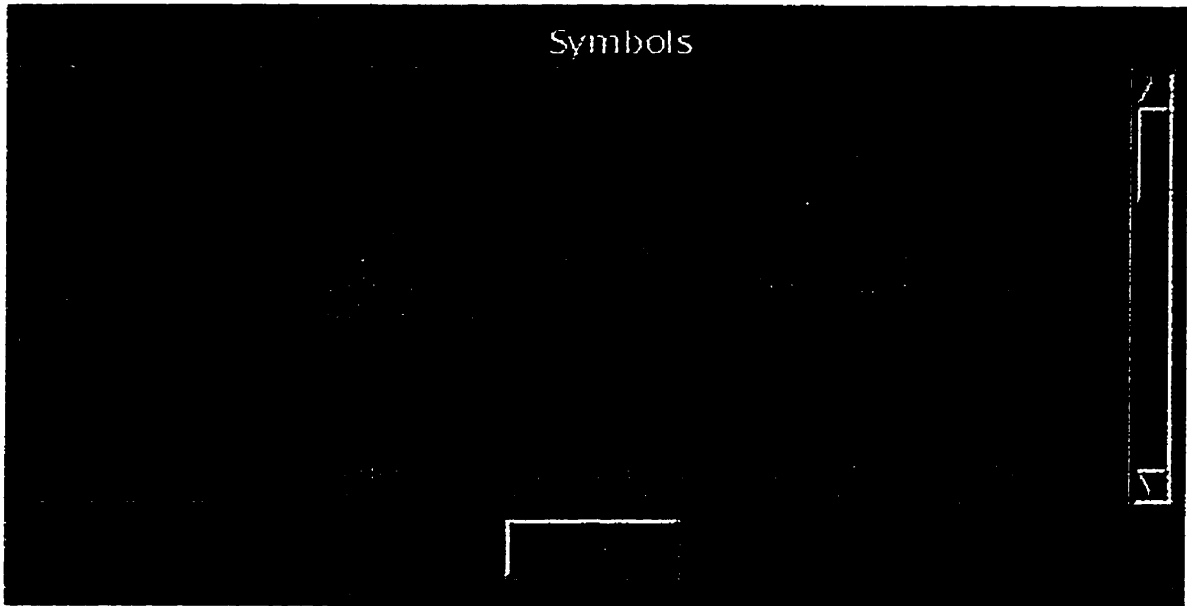


Figure 4.4.4 Symbols

These dialog windows help keep the user updated about the state of the system.

There is an on-line tutorial that explains the complete working of the graphical user interface for the temporal expert system shell. It helps the user to learn the interface, and lessens the probability of undesired results.

Chapter 5

Inference engine and Knowledge base

Again I saw that under the sun the race is not to the swift, nor the battle to the strong, nor bread to the wise, nor riches to the intelligent, nor favor to the man of skill; but time and chance happen to them all.

- The Bible, Ecclesiastes, 9

This chapter discusses the inference engine and knowledge base. The inference engine, written in ECLⁱPS^e, is the backbone of the system. The inference engine interacts with the knowledge base, which is the repository of facts, to solve the queries requested by the user.

5.1 Inference engine

Our inference engine consists of ECLⁱPS^e and rules for solving queries about interval and point based information [Good92].

CHAPTER 5. INFERENCE ENGINE AND KNOWLEDGE BASE

Interval based information is represented using the integral rule. $\text{integral}(a, b, f, x)$ is true if and only if the integral of f from a to b is x . We use a limited version of the integral rule which can be viewed as the measure of the duration of truth of f over the interval (a, b) . For example, "jogging" is true throughout the interval $(0, 10)$ is written as $\text{integral}(0, 10, \text{jogging}, 10)$ and "jogging" is true for half the time is $\text{integral}(0, 10, \text{jogging}, 5)$. Note that in the latter case, "jogging" may not have been true over a single sub-interval of length 5.

Information true at an isolated point is represented using the point rule. $\text{point}(t, f, x)$ is true if and only if $f(t)=x$. For example, "running" is true at time 5 is written as $\text{point}(5, \text{running}, 1)$. The "1" represents "true".

For the examples that follow in this section, assume the knowledge base consists of the following facts:

$\text{point}(5, \text{running}, 1)$.

$\text{integral}(0, 10, \text{jogging}, 10)$.

We conclude this section with a description of the rules used to implement the point and integral relations.

CHAPTER 5. INFERENCE ENGINE AND KNOWLEDGE BASE

Point relations:

`pt(T,F,X) :-` (1)
`point(T,F,X).`

`pt(T,F,1) :-` (2)
`A #< T,`
`B #> T,`
`C #= B-A,`
`integral(A,B,F,C).`

Figure 5.1 Point relations

The `pt(T,F,X)` rule shown in figure 5.1 (1), is true if there is a corresponding fact `point(T,F,X)` in the knowledge base. For example, to find out if the event “running” is true at 5, the inference engine checks the knowledge base for the fact `point(5,running,1)`. On finding the above fact, it successfully satisfies rule (1) and returns with “Yes”.

To determine if `F` is true at an isolated point `T`, we can also look for an interval `(A,B)` which contains `T` and over which `F` is true throughout. This strategy is captured by the second rule in figure 5.1. For example, to find out if the event “jogging” is true at 5, the inference engine checks the knowledge base and finds `integral(0,10,jogging,10)`. `integral(A,B,jogging,C)` is unified with `integral(0,10,jogging,10)`. At the given point `T=5`, the ECLiPS^e inference

CHAPTER 5. INFERENCE ENGINE AND KNOWLEDGE BASE

engine successfully satisfies rule (2) and therefore the event “jogging” is true at point 5.

Integral relations:

$$\begin{aligned} \text{int}(A,B,F,X) &:- & (3) \\ &\quad \text{integral}(A,B,F,X), !. \end{aligned}$$

$$\begin{aligned} \text{int}(A,B,F,C) &:- & (4) \\ &\quad X \#<= A, \\ &\quad Y \#>= B, \\ &\quad Z \# = Y-X, \\ &\quad \text{integral}(X,Y,F,Z), \\ &\quad C \# = B-A. \end{aligned}$$

$$\begin{aligned} \text{int}(A,B,F,C) &:- & (5) \\ &\quad X \#<= A, \\ &\quad Y \#> A, \\ &\quad Y \#< B, \\ &\quad Z \# = Y-X, \\ &\quad \text{integral}(X,Y,F,Z), \\ &\quad \text{int}(Y,B,F,Temp), \\ &\quad C \# = Y-A+Temp. \end{aligned}$$

Figure 5.1.1 Integral relations

Interval based information is solved using the integral rules in figure 5.1.1. The $\text{int}(A,B,F,X)$ rule shown in (3), is true if there is a corresponding fact $\text{integral}(A,B,F,X)$ in the knowledge base. For

CHAPTER 5. INFERENCE ENGINE AND KNOWLEDGE BASE

example, to find out if the event “jogging” is true between the interval 0 and 10, the inference engine checks the knowledge base for the fact $\text{integral}(0,10,\text{jogging},10)$. On finding the above fact, it matches it with the goal in (3) and returns with “Yes”.

The inference engine uses the rule $\text{int}(A,B,F,C)$ shown in (4), to find out if an event F is true over a super-interval (X,Y) of (A,B) . It is important to note that this integral rule checks the truth-value of an event F over a closed interval. An event F is true over the interval (A,B) , if there exists an $\text{integral}(X,Y,F,Z)$, such that its lower limit X is less than or equal to A , its upper limit Y is greater than or equal to B , the difference of X and Y is Z and the difference of A and B is C .

For example, event “jogging” is true throughout the interval $(0,10)$. To find out if “jogging” is true between the interval $(5,8)$, the inference engine checks the knowledge base for $\text{integral}(0,10,\text{jogging},10)$. For the given interval $(5,8)$, the ECLIPSE inference engine successfully satisfies rule (4) and therefore the event “jogging” is true between $(5,8)$.

The rule $\text{int}(A,B,F,C)$ shown in (5), implements the following additive property of integrals:

$$\int_a^b f(x) dx = \int_a^c f(x) dx + \int_c^b f(x) dx.$$

$\text{integral}(10,11,\text{jogging},1)$.

For example, the event “jogging” is true between $[0,10]$ and $[10,11]$. The inference engine uses the rule $\text{int}(A,B,F,C)$ in (5) to prove that

“jogging” is true between [0,11]. Note that we use “int” and “integral” in the code in order to distinguish between the rules and the facts in the knowledge base. This helps to avoid infinite loops. Similarly for “pt” and “point”

5.2 Knowledge base

The point and the interval rules are static. On the other hand, the knowledge base is dynamic because facts are added as a result of user interaction.

The user enters the event name, event description, color and temporal information for every event into the system. The translator maps the information entered by the user to a logical form and stores it as facts in the knowledge base. The unique color that represents each event is used to index that event. Every fact in the knowledge base contains the following information:

- Unique color that represents the event.
- Temporal information associated with that event.

The following example illustrates what gets stored in the knowledge base as a result of the translation from the graphical to the logical form.

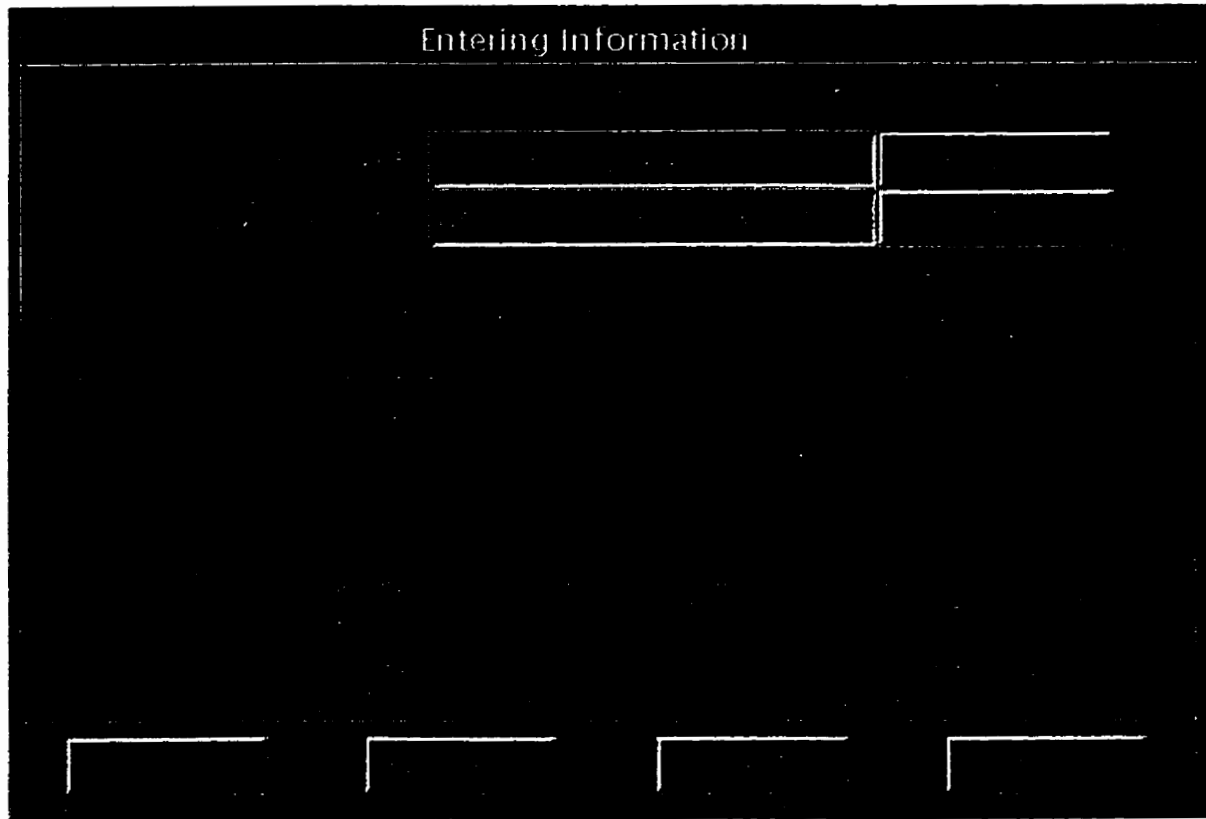


Figure 5.2 Fact representation in the knowledge base

- Mary will be in her office at 8 a.m. and between 10 a.m. and 11 a.m.

This event is represented by positioning a "Point Event" icon at 8 a.m. and "Fixed Event" icon between 10 a.m. and 11 a.m. as shown in figure 5.2.

The translator maps the event details entered by the user into the logical form and stores it in the knowledge base as the following facts:

CHAPTER 5. INFERENCE ENGINE AND KNOWLEDGE BASE

```
point(8.0,bc7a60,1).  
integral(10.0,11.0,bc7a60,1).
```

Note that “bc7a60” is the code for the color associated with the event. Every time the user enters information, new facts are added to the knowledge base.

The inference engine and the knowledge base constitute the back end of the system and are transparent to the user. Due to the loose coupling between the implementation and the interface, we can change the underlying implementation without changing the interface. This feature makes the design more robust. New features can be easily added to the existing system.

Chapter 6

Evaluation of the GUI

It were not best that we should think alike; it is difference of opinion that makes horse races.

-Mark Twain

The designer is faced with the task of estimating the effectiveness of the user interface. A complete evaluation of the user interface design is an expensive process. It requires the support of cognitive scientists and trained technicians in graphical design. It involves designing and performing a number of statistically significant experiments with typical users. This type of experiment may be economically feasible for large projects, but for comparatively small development projects, this type of experiment is unrealistic. However, user interface evaluation is an important part of the design process and it should not be ignored, but should be scaled to suit the particular design. These types of evaluations may be less reliable but they serve as a measure of user preferences and to detect particular error prone operations.

CHAPTER 6. EVALUATION OF THE GUI

An evaluation technique that can be used is an observation process. The user is assigned specific tasks to carry out, and the results gathered by observing their interaction with the interface. This is a form of diagnostic analysis and is a subjective method of evaluation. Laurel [Laub90] proposes several steps to follow when doing this type of evaluation:

- Set up the observation: This involves preparing the questions or tasks that the user will be doing in the observation. These tasks should emphasize all the important parts of the interface being evaluated. Create a realistic situation for the observation; that is, set up the environment similar to one, which would be used normally by the user.
- Recruit users for the evaluation: In recruiting users, make sure that the subjects have approximately the same experience level as typical users of the system. Also they should not be familiar with or have pre-conceived notions about the product.
- Describe the purpose of the observation: Set the users at ease by stressing that they are involved in a design process and emphasize that it is the product that is being tested, and not the skills of the users.
- User's claim: Tell the user that he/she can quit anytime.

CHAPTER 6. EVALUATION OF THE GUI

- Explain how to think aloud: Ask users to say what comes to mind when they work. Laurel [Laub90] suggests that by listening to users talk and plan, the administrator will be able to examine the user's expectations of the product as well as their intentions and problem solving strategies.
- Demonstrate the equipment to be used: Perform a demonstration of the equipment that the user will use during the evaluation. This allows the user to become familiar with the equipment.
- Explain that help will not be provided: It is important that the administrator allow users to work with the product without interference or extra help. This is the best way to observe how users really interact with the product. If there is on-line help available, then point this out to the user, if it has not already been noticed.
- Describe the tasks and introduce the product: Describe to the user the overall function of the product and explain what each task requires. If a demonstration of the product is required beforehand, make sure that something to be tested is not demonstrated.
- Ask the user to voice questions before the evaluation begins: The user should be given time to browse through the interface for a set amount of time and ask general questions before the evaluation begins.
- Conclude the observation: After the observation is over, re-explain to the user the reason for the observation and answer any remaining

questions. Also provide some form of user satisfaction questionnaire to obtain the users' overall opinion of the product.

6.1 User Evaluation

The evaluation of the temporal expert system shell's user interface is based on the observation process outlined above. The user is given tasks to perform using the functions of the user interface. These tasks, presented in figure 6.1, cover all the important aspects of the user interface: entering information, querying the system, and general tasks.

When entering information, the user enters the event details, picks a color for the event and adds temporal information for the event by positioning the required icon on the time scale. When querying the system, the user performs queries using two methods:

1. Is an event true at the given time ?
2. What is true about an event ?

During the general tasks section of the evaluation, the user verifies the database for the total number of events and prints the active database file.

As the user is performing these tasks the evaluator uses an observation form (figure 6.2) to record the results of the evaluation. To categorize the user's skills, information is gathered pertaining to a user's

CHAPTER 6. EVALUATION OF THE GUI

experience with computers and windows environment. For each task performed, the following information is recorded:

- The time it took to perform a task: This information is used to determine the rate at which the user learnt the user interface procedures.
- The type of help required by the user: This information determines the ease with which the user learns the interface, and whether or not on-line help is being used.
- Problems the user encounters: This information is used to determine the user's understanding of the interface and whether the functions of the user interface are properly defined.
- The user's comments: This information gives the user's satisfaction with the interface functions, and features that they think should be included in the interface.
- General observations: These are observations made by the evaluator about the user's interaction with the interface.

Evaluation Tasks

Part 1: Entering the Information

Enter the following event details:

1. **Event Name:** Meal Schedule.
 Event Description: Tim had breakfast at 8 a.m. and lunch at 12 p.m.
 Pick Color: Red=60, Green=120, Blue=85.
 Place one "Point Event" icon at 8 a.m. and another "Point Event" icon at 12 p.m.

2. **Event Name:** Homework.
 Event Description: Henry did his homework between 7 a.m. and 9 a.m.
 Pick Color: Red=50, Green=100, Blue=50.
 Place the icon "Fixed Event" between 7 a.m. and 9 a.m. on the time scale.

3. **Event Name:** Snow.
 Event Description: It was snowing throughout the day.
 Pick Color: Red=95, Green=200, Blue=175.
 Place the icon "Limitless Event" on the time scale.

Part 2: Querying the System

4. Find out the time points at which the event "Meal Schedule" is true?

5. Select the event "Jogging" from the event list and find out if this event is true between 10 a.m. and 11 a.m.? Also, check if the event is true at point 10 a.m. and point 11 a.m. on the time scale? (Note: Event details for "Jogging" already exist in the Knowledge base).

Figure 6.1 Evaluation Tasks

6. Find out at what time the event "Phone calls" is true and verify the results.
What are the steps involved in verification?

7. Is the event "Snow" true between 5 a.m. and 8 a.m.?

Part 3: General tasks

8. How many events are currently stored in the database?

9. Print the active database file.

10. Is "Black" one of the colors used to represent an event.

Note: Black has a hexadecimal value of #000000.

Figure 6.1 Evaluation Tasks cont'd

Observation Form

User Information

1. Experience with Computers.

Novice Intermediate Expert

2. Experience with Windows.

Novice Intermediate Expert

3. Experience with Unix.

Novice Intermediate Expert

Task Information

4. Task Number: _____

4. Time taken to perform the task: _____

5. Did user require help: Yes No

If yes, what kind of help: _____

6. Problems with interface: _____

7. Comments user made: _____

8. General Observations: _____

Figure 6.2 Observation Form

6.1.1 Results of the Evaluation

There were ten participants in the evaluation, and their computer expertise was categorized as novice, intermediate or expert. The number of participants in the individual categories is represented in figure 6.3. This characterization was based on the users' knowledge and experience with computers and the windowing environment. This user population was chosen because the interface is targeted towards novice and intermediate users. The expert users were chosen to obtain the opinion of individuals who have had experience with similar products and could give informed ideas about the interface.

Each participant in the evaluation was introduced to the interface, and the novice users were given a demonstration on how to use the window environment on the workstation, and a brief summary on how to manipulate windows in the interface and the widgets using the mouse.

The following general steps were given in the introduction along with the steps that were outlined for this evaluation process:

- Explain how to use on-line help.
- Explain the structure of the interface. The user was instructed that all the major functions could be reached through the main window, and that each function performed in the interface is done through a specific window.

CHAPTER 6. EVALUATION OF THE GUI

- Explain that the only help given will be to interpret the tasks and inform them of the type of function required to perform the task.

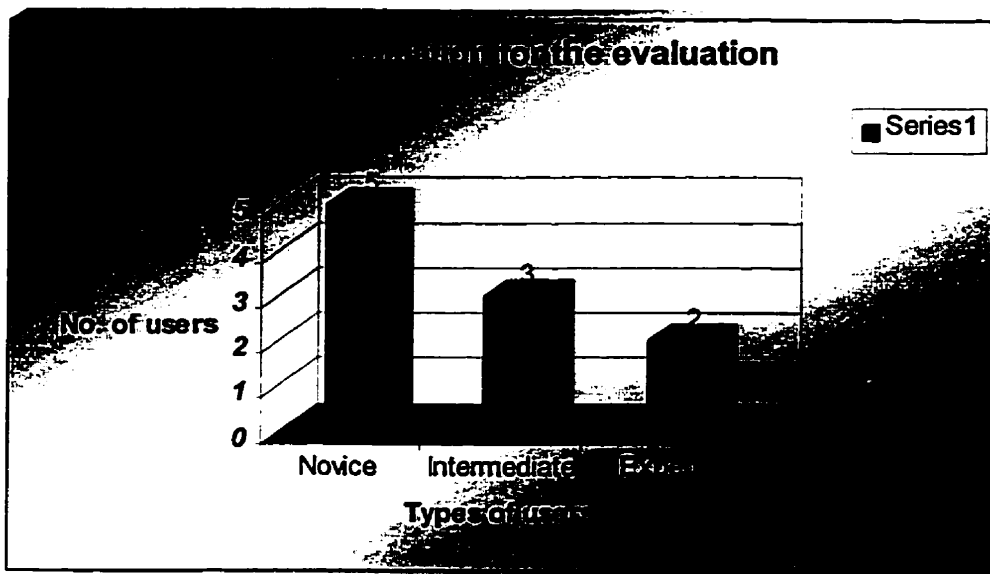


Figure 6.3 User population for the evaluation

6.1.1.1 Observations

Task 1: Enter the event details for the event "Meal Schedule".

As it was the first task, 70% of the users had to be informed that to position an icon on the time scale, the user has to single click on the icon and with the mouse button-1 depressed, drag the icon to a specified position. The expert users were easily able to pick the color for an event by filling in the numerical values for red, green and blue. Novice users

CHAPTER 6. EVALUATION OF THE GUI

took more time to select the color as they used the mouse to point and select the red, green and blue values from the color scales.

This task took the longest for all users to perform, since they were learning to navigate the interface.

Task 2: Enter event details for the event “Homework”.

90% of the users understood how to enter the event name, event description and pick a color. The main complaint that the experienced users had was that tabbing from one text field to another was not functioning. Hence, mistakes were made in typing the values into the correct field. Also the process of selecting, positioning and configuring the “Fixed Event” icon on the time scale was not obvious to the users.

Task 3: Enter event details for the event “Snow”.

80% of the users were able to complete this task easily. The novice users took more time to select and position the “Limitless Event” icon on the time scale. Users complained that there was no way to delete/change the information once entered into the database file.

Task 4: Querying the system for the event “Meal Schedule”.

Once the use of the “What’s True” query icon was explained to the users, all of them were able to find out the time points at which the event “Meal Schedule” is true.

CHAPTER 6. EVALUATION OF THE GUI

Task 5: Querying the system for the event “Jogging”.

60% of the users were able to execute the query. However, 40% of the users had to be told that it is possible to position one type of icon over the other.

Task 6: Querying the system for the event “Phone calls”.

All the users were able to find out the time points at which the event “Phone calls” is true. 80% of the users had to be explained the meaning of the task in detail. Users found the verification task to be the most difficult of all the queries.

Task 7: Querying the system for event “Snow”.

This query was easily performed. However, one user inquired whether he could place “Point Event” icons at all points between the time interval 5 a.m. and 8 a.m. to check if the event is true. The meaning of “Fixed event” icon was explained later to the user.

Task 8: Finding the total number of events in the database.

All the users were able to accomplish this task. 40% of the users found an alternative method of doing this task by choosing “Show database” under the “Options” menu.

CHAPTER 6. EVALUATION OF THE GUI

Task 9: Printing the database file.

Users had to be explained that the print option by default prints the active database file. Everyone found it relatively easy to print the file after the explanation.

Task 10: Finding out if “Black” was one of the colors used for an event.

All the users easily performed this task.

6.2 User Satisfaction Evaluation

Chin et al [Chap90] mention that user acceptance of a system is a critical measure of the system’s success. Although a system may be evaluated favorably on every performance measure, the system may not be used very much, because of the user’s dissatisfaction with the system and its interface.

The questionnaire that will be used to perform this part of the user evaluation will be based on the QUIS 5.0¹(shown in figure 6.4).

¹ Questionnaire for User Interface Satisfaction developed by the Human-Computer Interaction Laboratory at the University of Maryland [Chap90].

User Satisfaction Questionnaire

1. Overall reactions to the user interface.

a) terrible wonderful
0 1 2 3 4 5 6 7 8 9

b) difficult easy
0 1 2 3 4 5 6 7 8 9

c) rigid flexible
0 1 2 3 4 5 6 7 8 9

2. Organization of information.

confusing very clear
0 1 2 3 4 5 6 7 8 9

3. Sequence of screens.

confusing very clear
0 1 2 3 4 5 6 7 8 9

4. Use of terms throughout system.

inconsistent consistent
0 1 2 3 4 5 6 7 8 9

5. Computer terminology (icons) related to task you are doing.

never always
0 1 2 3 4 5 6 7 8 9

Figure 6.4 User Satisfaction Questionnaire

CHAPTER 6. EVALUATION OF THE GUI

6. Messages on screen which prompt user for input.
confusing clear
0 1 2 3 4 5 6 7 8 9
7. Computer keeps you informed about what it is doing.
never always
0 1 2 3 4 5 6 7 8 9
8. Error Messages.
unhelpful helpful
0 1 2 3 4 5 6 7 8 9
9. Learning to operate the user interface.
difficult easy
0 1 2 3 4 5 6 7 8 9
10. Exploring features by trial and error.
difficult easy
0 1 2 3 4 5 6 7 8 9
11. Remembering use of commands.
difficult easy
0 1 2 3 4 5 6 7 8 9

Figure 6.4 User Satisfaction Questionnaire cont'd

13. Tasks can be performed in an obvious manner.

never

always

0 1 2 3 4 5 6 7 8 9

14. Correcting mistakes.

difficult

easy

0 1 2 3 4 5 6 7 8 9

15. Experienced and inexperienced users needs are taken into consideration.

never

always

0 1 2 3 4 5 6 7 8 9

Figure 6.4 User Satisfaction Questionnaire cont'd

6.2.1 Results of the user satisfaction questionnaire

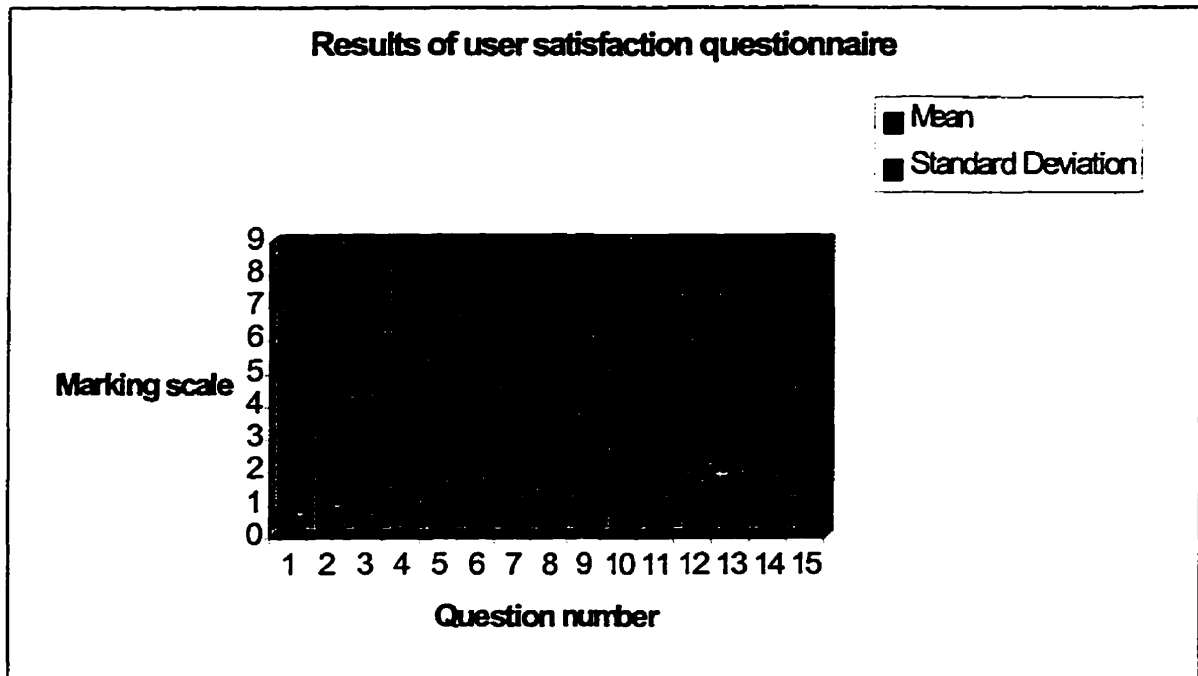


Figure 6.5 Results of the User Satisfaction Questionnaire

Figure 6.5 represents the results of the user satisfaction questionnaire shown in figure 6.4. The users responded to the questionnaire after completing the evaluation tasks. From figure 6.5 we see that the means vary by a small amount across questions and the standard deviations are low.

The two highest standard deviations occurred in question number 5, which dealt with recognizing icons, and question number 14, which dealt with correcting mistakes. In question number 5, a possible reason for this is that the user either likes working with the icons, or thinks it is inconvenient to remember the functionality of each icon. In question

number 14, a possible reason is that experienced users make more mistakes in typing, therefore they have more mistakes to correct, while novices are much more careful in checking that the data was correct before adding it to the database file.

6.3 Conclusions from the observations

The conclusions that can be made from these observations are:

- There should be tabbing provided between fields, since users find it more natural when typing to use the tab key instead of having to move from the keyboard to the mouse.
- There should be either an up front description of icons, or some facility other than on-line help that notifies the user of the functions of icons. However, icons help users learn the functions of the interface more easily, since they do not have to remember commands, but can instead use recognition.
- On-line help is very useful in explaining processes that are not obvious to users.
- The consistency in the layout of the windows and the similarity between “Entering Information” and “Querying the system” screens helped users to rapidly learn the functions of the interface.
- There should be immediate feedback to the user about the latest action performed. It helps in deciding the next step.

CHAPTER 6. EVALUATION OF THE GUI

- Changes to data before writing to the database file should be easier in the window where the data is being added, so that the user does not have to delete/change the database file. There should be a means for modifying the data already entered into the database file.
- User actions are unpredictable, it is useful to provide a method of undoing an action or an “UNDO” button.

6.4 Conclusions from the questionnaire

The conclusions drawn from the results of the user satisfaction questionnaire are:

- The general reaction to the interface was favorable, although the flexibility was not highly regarded.
- Icons are only appreciated by some users, hence an alternative should be provided.
- The error messages are adequate and useful in helping the users to perform the correct procedures.
- Learning to operate the interface is simple and straightforward. Understanding the functionality takes time.
- The on-line help is beneficial to the user.
- The interface can be used by both novices and experienced users efficiently.
- Correcting mistakes should be as easy as possible.

6.5 Modifications to the GUI

Following the results of the user evaluation, several changes were made to the user interface to reflect user preferences and to improve the structure:

- In the “Entering Information” window, a tabbing function has been provided for easy movement from one text box to another. Now, the user can either use the keyboard or mouse pointer to move from one field to another.
- Labels have been added to all the icons used for entering temporal information. This will help the user in identifying the icons.
- Whenever the user points and selects an icon, the icon changes its color to red, giving a visual cue to the user that the icon is currently selected. After the user positions the icon on the time scale, the icon takes the color of the event that it represents.
- In “Querying the System”, the “What’s True” icon changes its color to blue when the user clicks on it. This gives feedback to the user that the query operation has been initiated.
- A window explaining the use and functionality of all the icons used to represent temporal information has been added to the main application window. This will help the user in selecting the right icon for an event.

Chapter 7

Conclusion

Never remind someone of a kindness or act of generosity you have shown him or her. Bestow a favor and then forget it.

-Little Book on Wisdom.

The aim of this thesis was to present a logic independent graphical user interface (GUI). The main research challenge was the definition of the GUI. The GUI is developed using the principles of user interface design and implemented in a way that it is easy to learn, use, efficient and effective. The GUI allows the user to enter, query, and represent temporal information using color-coded symbols. The user does not have to be familiar with the particular logic used by the implementation.

Following the design of the interface and the background research that was required the following conclusions were drawn:

CHAPTER 7. CONCLUSION

- **The user interface design is crucial to the success of a product. The user interface defines the quality of a product and the ease of use and learning of the system.**

- **The three most important principles in user interface design are:**
 1. **The interface should be designed to suit the needs and the abilities of the anticipated user. Users should not be forced to adapt to an interface because it is convenient to implement or because it is suited to the system's designer.**
 2. **User interfaces must be consistent. Consistency should be maintained within a system and across subsystems.**
 3. **User interfaces should have on-line help. Help should be accessible and context sensitive.**

- **Object-oriented concepts have made a huge impact on user interfaces. Designers are applying object-oriented concepts to the design presentation and the integration of user interfaces.**

- **The GUI to the temporal expert system allows the user to successfully enter and query temporal information using the icons and symbols.**

- **The temporal expert system shell uses various icons such as "Point Event", "Limitless Event", etc. to capture temporal information. The**

CHAPTER 7. CONCLUSION

model has been successful with the small subset of the temporal information.

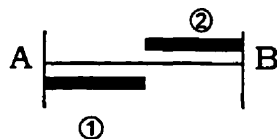
- Meaningful icons can be added to the system to handle more categories of events. Logic used is more powerful than the GUI.
- The inference engine and knowledge base of the system are kept transparent to the user.
- The loose coupling between the implementation and interface makes the design more robust. One can change the underlying implementation without changing the graphical user interface.
- The evaluation of a user interface is a valuable process in making the design decisions that improve user satisfaction and quality of the interface.
- The user does not have to familiar with the particular logic used by the implementation. It is easy to use GUI for entering, representing and querying information compared to using Eclipse and the logic directly.

7.1 Future Development

The functionality of the GUI can be improved in several areas:

- Enhance the capabilities of the system to handle more categories of events. For example, include icons to handle the following events:
 - ◆ John played squash and tennis for equal amounts of time between 5 p.m. and 7 p.m.

The following icon could represent this event:



The length of the line AB represents the time between 5 p.m. and 7 p.m. The horizontal bars ① and ② represent the two activities squash and tennis. The horizontal bars ① and ② have the same length because squash and tennis were played for the same amounts of time. Making these bars oscillate between A and B represents that these activities occurred between 5 p.m. and 7 p.m. The above icon becomes complicated when there are more than 3 or 4 activities. We conclude that it is not an obvious problem. One solution could be mixing graphics and text to capture the event details. The underlying ECL^{PS}^e inference engine has the capability of handling such events.

Another example is:

CHAPTER 7. CONCLUSION

- ◆ Paul ran 1 hour between 12 p.m. and 6 p.m.
- Extend the capability of the system to handle true, false and unknown information for the events like:
 - ◆ Henry was in Halifax yesterday but he will not be in Halifax today. We do not know where he will be tomorrow.
- Extend the capability of the system to handle conjunction and disjunction for the events like:
 - ◆ I was either at Tim Hortons or the Coffee Merchant for lunch.
- Enhance the capability of the system to handle implications for the events like:
 - ◆ My dog sits whenever I whistle.
 $\forall t$, if whistle blows at time t , then dog sits at time t .
 $\forall t$ point(t , whistle, 1) \rightarrow point(t , dog_sits, 1).
- Add a feature to delete or modify the events already entered into the system.

7.2 Applications

The GUI could be used in scheduling applications. The user enters the time at which a person is available during the day and system stores the details in the knowledge base. This information can be used later in setting up appointments.

BIBLIOGRAPHY

Bibliography:

[Alle91] James F. Allen. *Temporal Reasoning and planning*, In Reasoning about plans. Morgan Kaufmann publishers Inc., 1: 1-68, 1991.

[Acqi96] <http://vww.com/ai/acquire/acquire.html>

[Abde95] *Eclipse 3.5 ECRC*, Common Logic Programming System, User Manual Dec 1995.

[Baby96] <http://www.gmd.de>

[Chap90] Chapians, A. and Burdurka, W. *Specifying Human-Computer Interface requirements*. Behavior and Information Technology, Vol 9, 1990, 479-492.

[Dinc88] M. Dincbas, P.Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. *The constraint logic programming CHIP*, In proceedings of the International Conference on Fifth Generation Computer Systems, Tokyo, Japan, December 1988, 693-702, 1988.

[Debr89] *The SB-Prolog System Version 3.1*, A User Manual, Department of Computer Science, University of Arizona, Tucson, Arizona, Dec '89.

BIBLIOGRAPHY

[Fruh93] *Constraint logic programming – An Informal Introduction*, technical report ECRC-93-5

[Fole96] Foley, J.D. Van Dam, A., Feiner, S. K. and Hughes, J.F. *Computer Graphics: - Principles and Practice*. Addison-Wesley Publishing Company, Inc., 1996.

[Focl92] “*The role of prior knowledge in inductive learning*”, *Machine learning* 9:54-97, 1992.

[Flex94] <http://www.lpa.co.uk>

[Gabb95] Gabbay, Dov M., Hogger C. J. and Robinson J. A. *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol-4, Epistemic and Temporal Reasoning 1995. Pages 175-247.

[Good92] S.D.Goodwin, E.Neufeld and A.Trudel (1992). *Temporal reasoning with real valued functions*. Pacific Rim International Conference on Artificial Intelligence (PRICAI '92), Sept 23-25, Seoul Korea P. 1266-1271.

BIBLIOGRAPHY

[Hein92] Neim C. Heintze, Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H.C. *The CLP(R) programming manual*, Sep '92.

[Ises94] *IEEE Standard Glossary of Software Engineering Terminology*. In IEEE Software Engineering Standard Collection. IEEE, 1994. Std 610.12-190.

[Jaff87] Joxan Jaffar and Jean-Louis Lassez. *Constraint logic programming*. In Proceeding of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany, pages 111-119. ACM Jan '87.

[John97] Eric Foster-Johnson, *Graphical applications with Tcl and Tk*, second edition, M&T Books, 1997.

[Laub90] Laurel, B. *The art of Human-Computer Interface Design*, Addison-Wesley, 1990.

[Magh95] Kamel Maghur, *A constraint based multi-agent planner*, 1995.

[Monu74] U. Montanari. *Networks of constraints: Fundamental properties and application to picture processing*. Information Science, 7(2):95-132, 1974.

BIBLIOGRAPHY

[Marc90] Marcus, A. *Designing Graphical User Interface*. UnixWorld (Oct '90), 135-138.

[Mayh92] Mayhew, D. *Principles and Guidelines in Software User Interface Design*, Prentice-Hall 1992.

[Mill56] Miller, G. The Magical Number Seven Plus or Minus Two: *Some Limits on Our Capacity for Processing Information*. The Psychological Review 63, (Mar. 1956), 81-97.

[Norm95] Norman, D. *“Turn Signals Are the Facial Expressions of Automobiles”*, Addison-Wesley Publishing Company, 1995.

[Panc95] Pancake, C. M. *Principles of Color Use for Software Developers*. Tutorial M1 from Supercomputing '95, 1995.

[Russ95] Norvig, Russell, *Artificial Intelligence, A Modern Approach*, Prentice Hall, Inc., 1995.

[Suth63] I.E. Sutherland. *Sketchpad: A man-machine graphical communication system*, In Proceeding of the AFIPS Spring Joint Computer Conference, Detroit, MI, USA, 329-46, 1963.

BIBLIOGRAPHY

[Tyug70] Enn Tyugu. *Solving problems on computation models*, J. Computational Mathematics and Math. Phys., 10:716-33, 1970.

Appendix A

User Manual

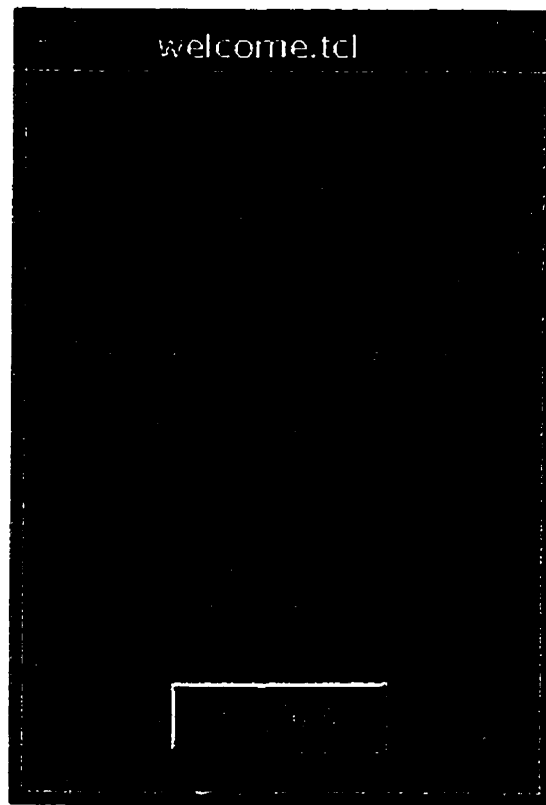


Figure A.1: Start Window

To start the temporal expert system shell, type in the command:

welcome.tcl

APPENDIX A. USER MANUAL

The execution of this command opens a welcome window as shown in figure A.1. By clicking on the “Continue button”, a window opens which gives access to the user interface and the tutorial (figure A.2).



Figure A.2: Interface Window

To start the interface:

Click on the “Start Shell” button to open the main application window as shown in figure A.3.

To start the tutorial:

Click on the “Start Tutorial” button to give the user access to the tutorial functions which are discussed later.

A.1 Using the main window:

The main window acts as a control point for the functions of the interface.

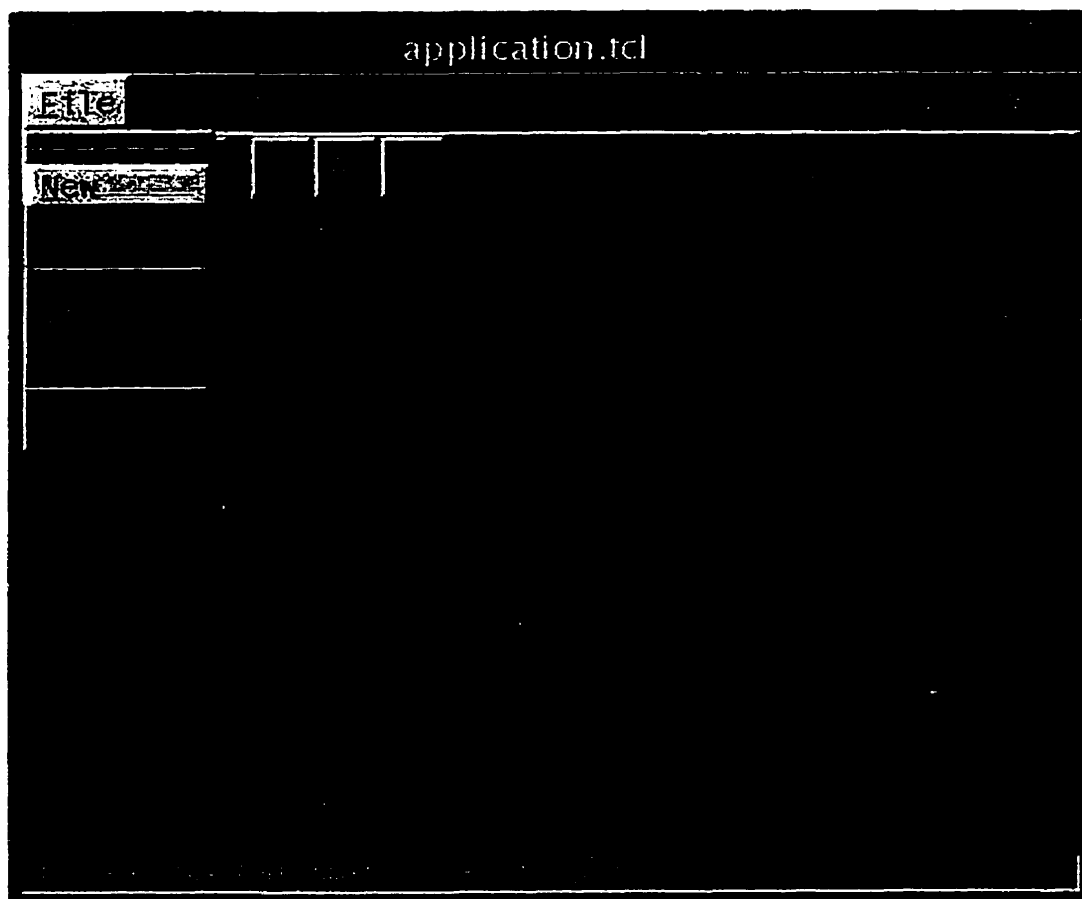


Figure A.3: Main application window

A.1.1 File menu:

The File menu provides the following functions:

Create a new file:

1. Click on the "New" button to open the dialog window as shown in figure A.4. This allows the user to create a new file to save event details.
2. Enter the name of the file as shown in figure A.4.
3. User clicks on the "OK" button to confirm the file name. The "Cancel" button aborts the operation.

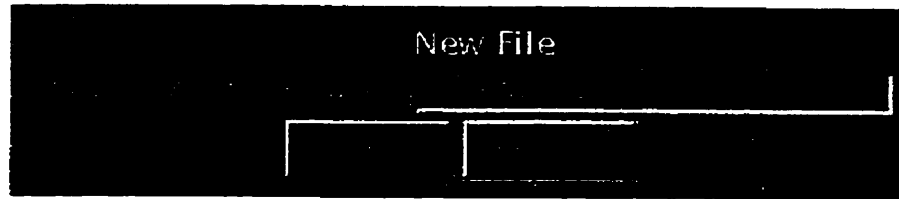
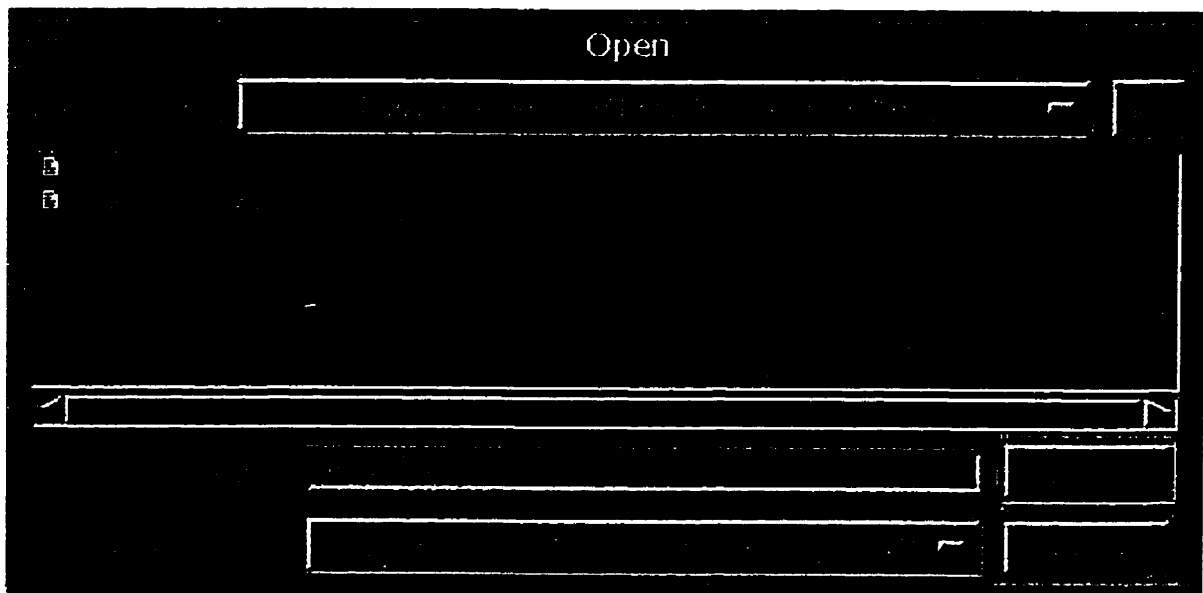


Figure A.4: New file window

Open an existing file:

1. Click on the "Open" button to open the dialog window shown in figure A.5. This allows the user to open an existing file, which will be used to save event details.
2. Select the type of the file and file name to open.
3. The "Cancel" button aborts the operation. The Directory options menu allows the user to choose any directory. The user gives his/her



confirmation by clicking on the "Open" button.

Figure A.5: File open window

Save a file:

As the name suggests, clicking on this button saves the current file.

Print a file:

1. The Print button allows the user to print the active database file.

Clicking on the print button opens up a confirm window as shown in figure A.6.

2. The user can confirm by clicking on the “OK” button. The “Cancel” button aborts the print operation.

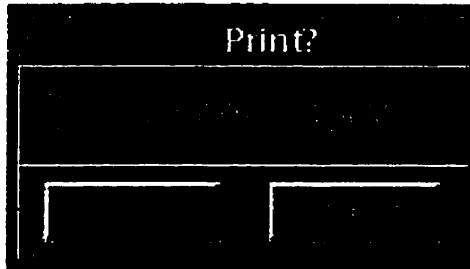


Figure A.6: Print confirm window.

Exit:

The “Exit” command closes all the windows and ends the application.

A.1.2 Edit menu:

The Edit menu provides the following functions:

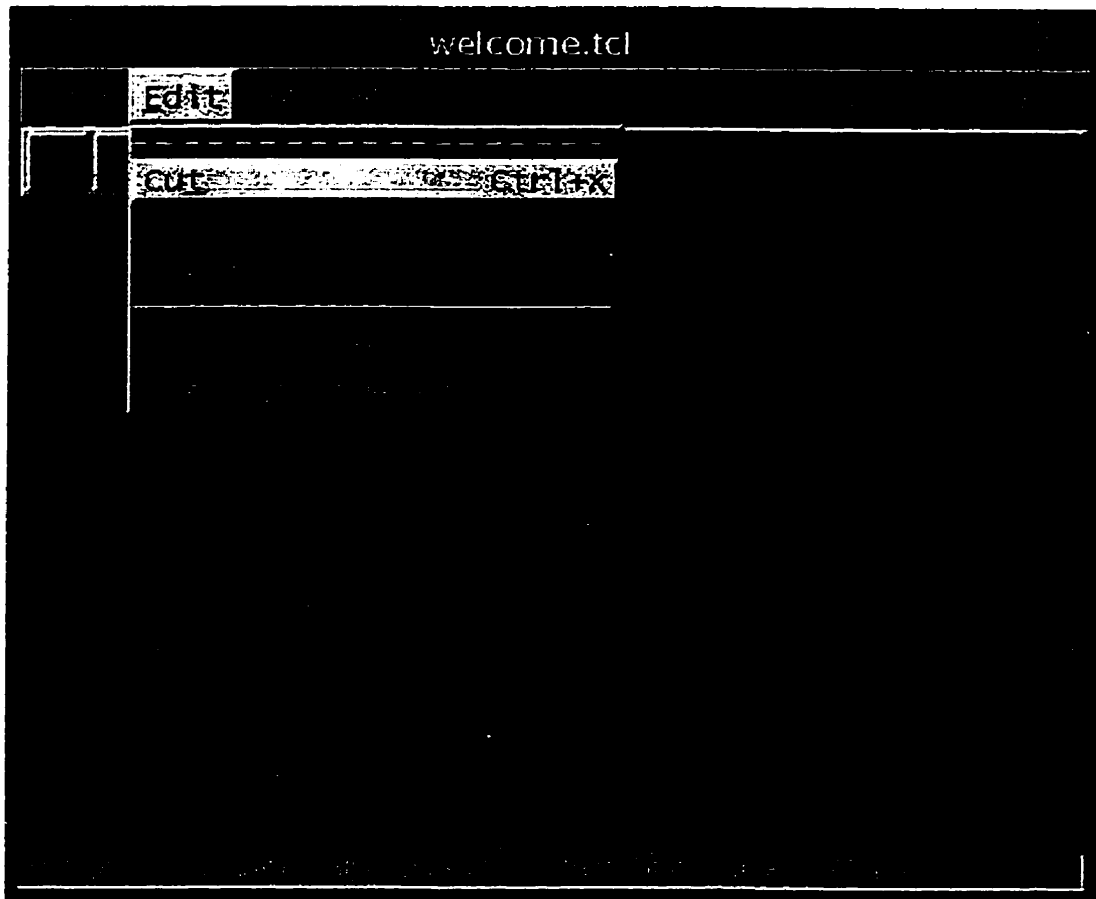


Figure A.7: Main window (Edit menu)

Cut, Copy and Paste operations:

They function as expected on text.

Entering information:

Click on "Enter Information" menu option under "Edit" menu to open a window as shown in figure A.8. This window is used to enter information.

To enter a "Point Event":

Consider the following example:

- Mary has a meeting at 10 a.m. (1)

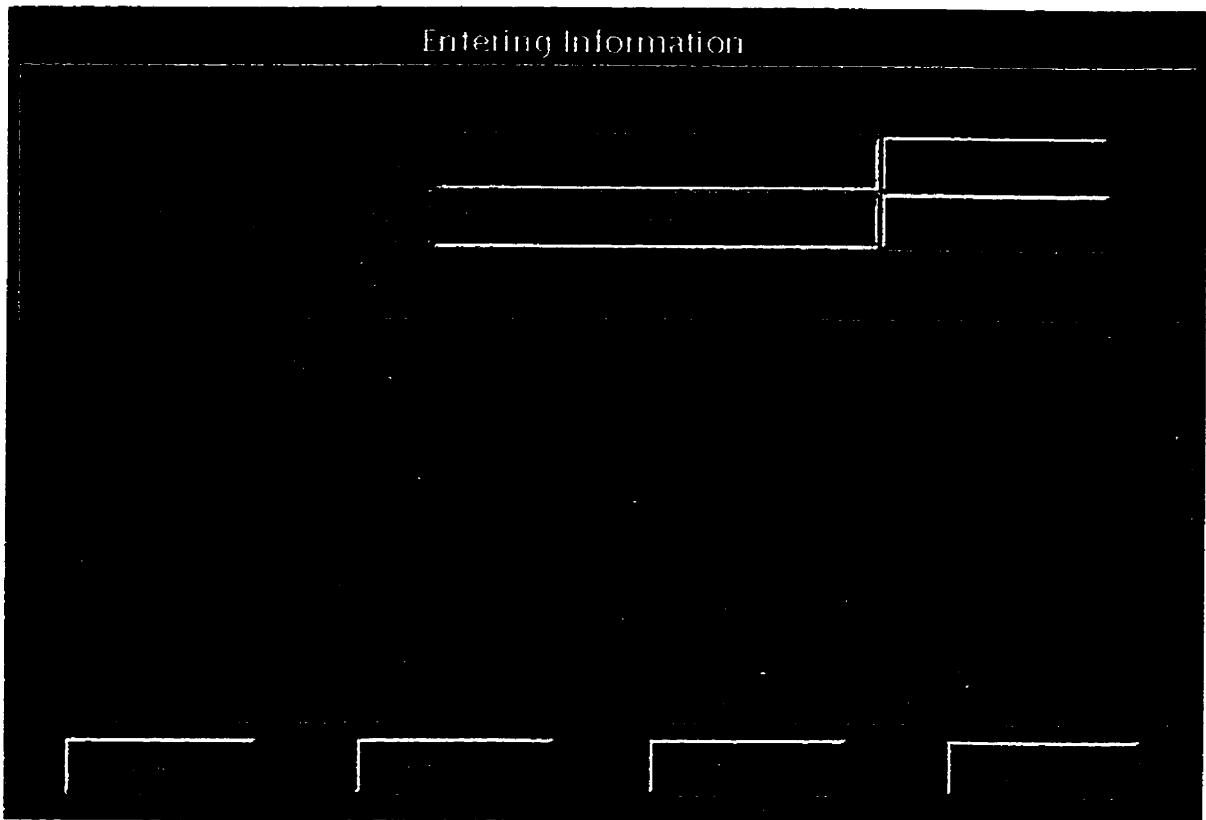


Figure A.8: Entering information

The event name for this event is "Meeting", the event description is "Mary had a meeting at 10 a.m." and the temporal component is that it is true at 10 a.m.

The steps to enter event (1) are:

1. Enter the event name "Meeting" for this event as shown in figure A.8.
2. Enter the event description in the specified text box.
3. Pick a color by clicking on the "Pick Color" button. This opens a color window as shown in figure A.9.

APPENDIX A. USER MANUAL

4. The user can select a color for the event by filling in the numerical values for red, green and blue. It is also possible to select a color by sliding the triangular tab on the color scale for each color.
5. Click on the "OK" button to confirm the selection.
6. Select the "Point Event" icon in figure A.8 by pointing and clicking on it. Then drag the icon along with the mouse pointer to position 10 a.m. on the time scale.
7. Click on the "Continue" button to add another event.

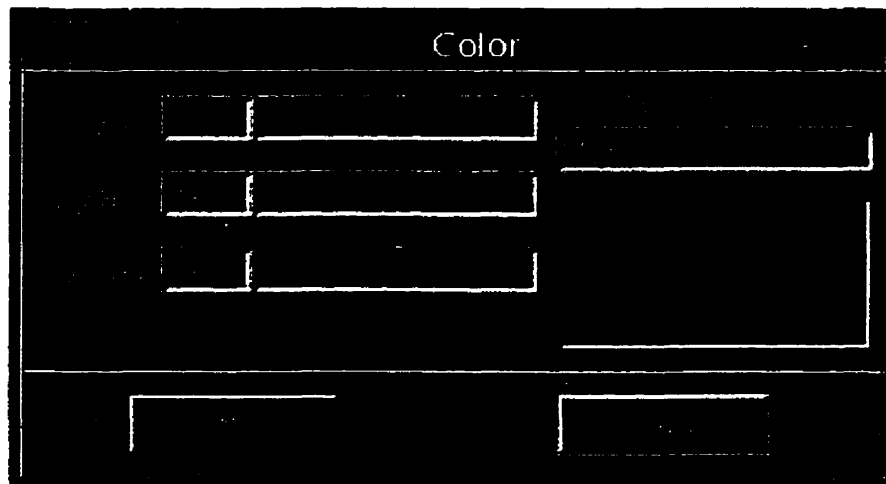


Figure A.9 Color window

8. The "Cancel" button aborts the operation and takes the control to the "Event Name" text box.
9. The "Help" button provides help on the "Enter information" screen.
10. Clicking on the "Done" button will close the current screen and will take the control back to the main window.

To enter a "Limitless Event":

Consider the following example:

APPENDIX A. USER MANUAL

- The water pump was working throughout the day. (2)

The event name for this event is “Water pump”, the event description is “The water pump was working throughout the day.” and the temporal component is that this event possibly started before 12 a.m. and ended after 12 p.m.

Steps followed to enter event (2) are:

1. Enter the event name “Water pump” for this event as shown in figure A.10.
2. Enter the event description in the specified text box.
3. Pick a color by clicking on the “Pick Color” button. This operation is explained above.
4. Click on the “OK” button to confirm the selection.
5. Select the “Limitless Event” icon by pointing and clicking on it. Then drag the icon along with the mouse pointer to position it on the time scale. When the user places the “Limitless Event” icon on the time scale, it automatically configures itself to cover the entire scale as shown in figure A.10.
6. Click on the “Continue” button to add the another event.

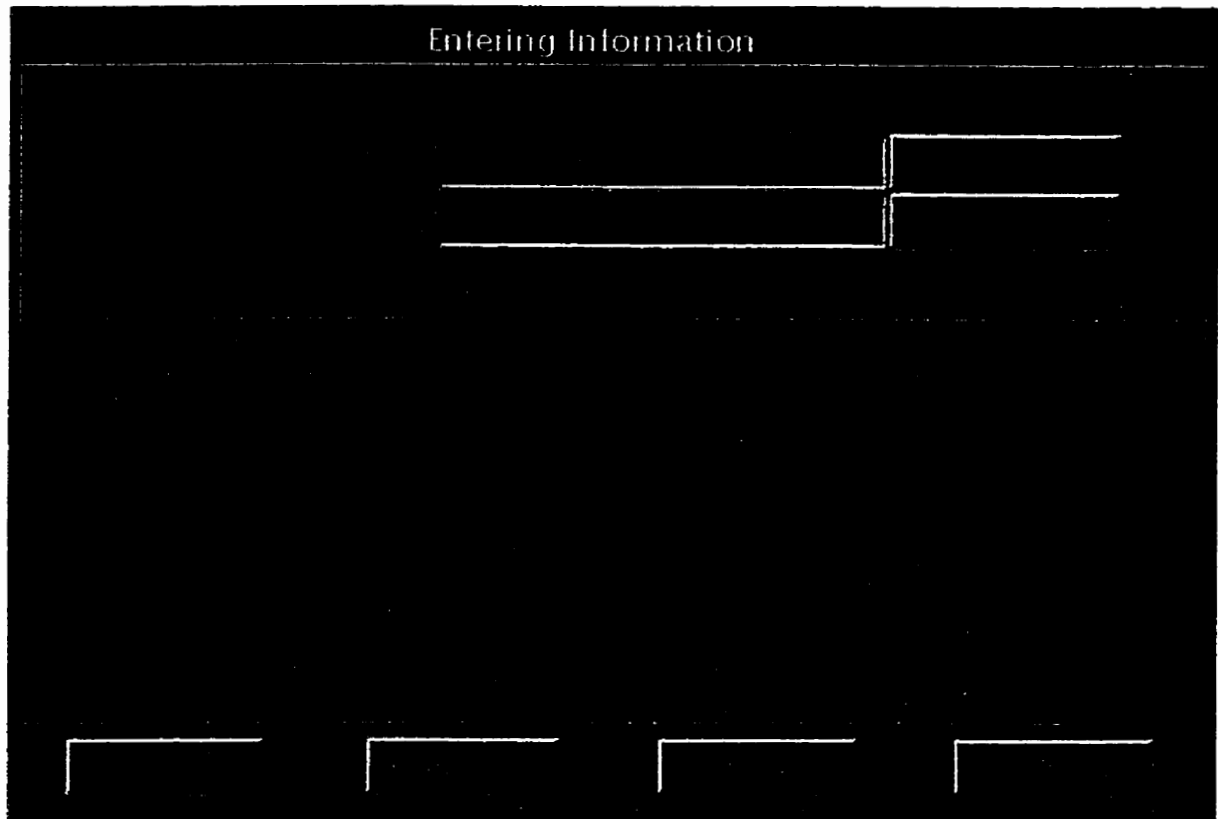


Figure A.10 Limitless Event

To enter a "Fixed Event":

Consider the following example:

- Henry was watching TV between 8:30 a.m. and 10 a.m. (3)

The event name for this event is "Watching TV", the event description is "Henry was watching TV between 8:30 a.m. and 10 a.m." and the temporal component is that the event is true between 8:30 a.m. and 10 a.m.

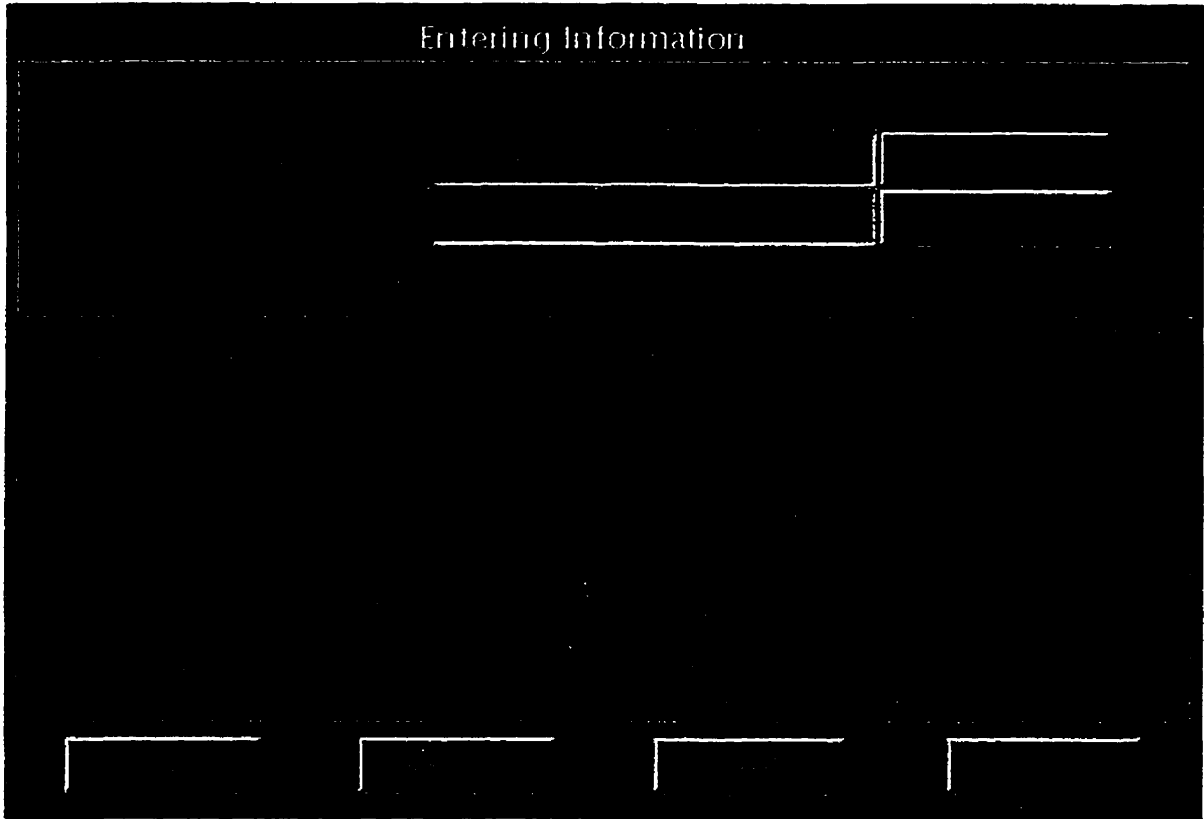


Figure A.11 Fixed Event

Steps followed to enter event (3) are:

1. Enter the event name "Watching TV" for this event in the "Event Name" text box.
2. Enter the event description in the specified text box.
3. Pick a color by clicking on the "Pick Color" button.
4. Click on the "OK" button to confirm the selection.

APPENDIX A. USER MANUAL

5. Select the “Fixed Event” icon by pointing and clicking on it and then drag it to the appropriate position on the time scale. (In this case, between 8:30 a.m. and 10 a.m.)
6. The user can modify the length of this icon on the time scale by clicking on a small square attached to the icon and dragging it with the mouse. This configures the “Fixed Event” icon to a desired length as shown in figure A.11.
7. Click on the “Continue” button to add another event.

To enter a “FixedLeft Event”:

Consider the following example:

- Sonia started the painting at 10 a.m. and worked on it till late afternoon. (4)

The event name is “Painting” and the event description is “Sonia started the painting at 10 a.m. and worked on it till late afternoon.” This event is definitely true between 10 a.m. and 12 p.m.

We use the “FixedLeft Event” icon to represent event (4) and the steps followed are:

1. Enter the event name “Painting” for this event as shown in figure A.12
2. Enter the event description in the specified text box.
3. Pick a color by clicking on the “Pick Color” button as explained in the previous examples.
4. Click on the “OK” button to confirm the selection.

APPENDIX A. USER MANUAL

5. Select the "FixedLeft Event" icon by pointing and clicking on it and then drag it to the appropriate position on the time scale.
6. The user places the left end of the "FixedLeft event" icon at 10 a.m. on the time scale and the right end of the icon automatically covers the rest of the scale as shown in the figure.
7. Click on the "Continue" button to add another event.

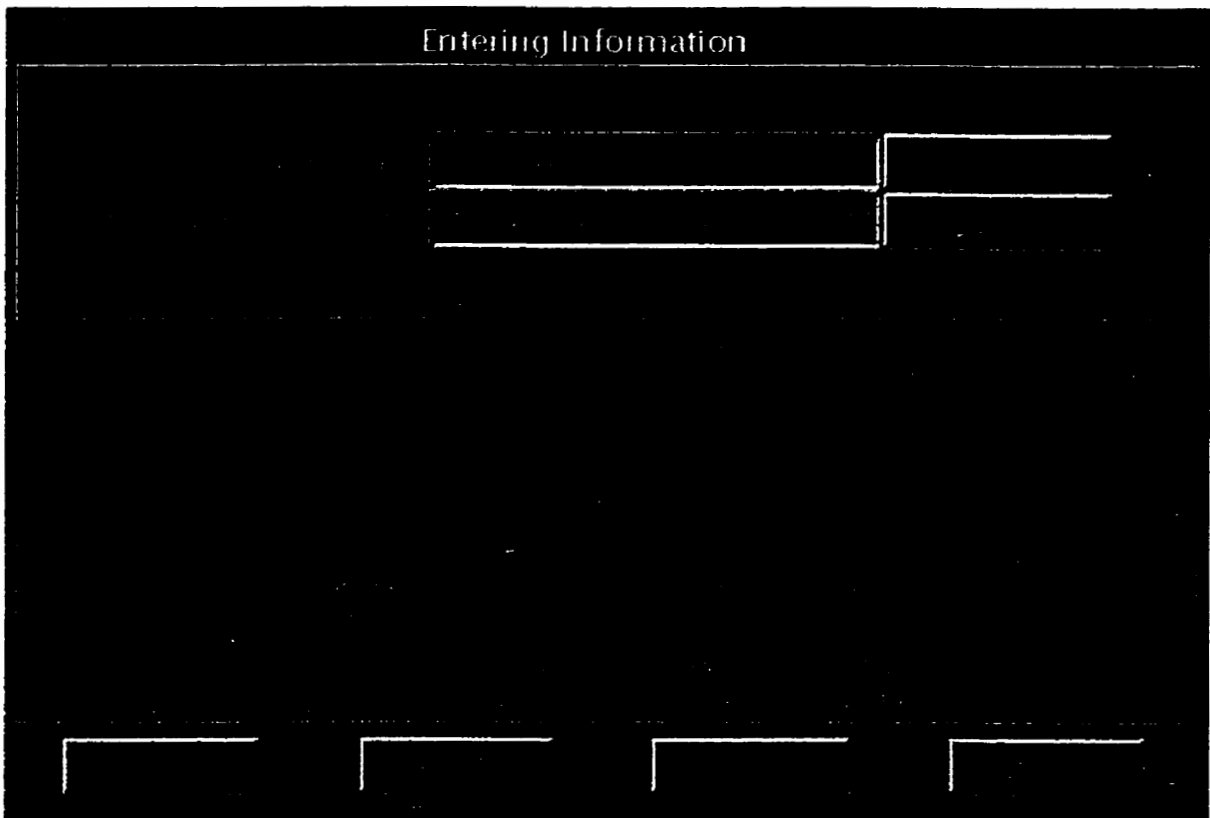


Figure A.12 FixedLeft Event

To enter a "FixedRight Event":

Consider the following example:

- The snow storm started last night and ended today at 10 a.m. (5)

APPENDIX A. USER MANUAL

It is important to note that the event started at some point before 12 a.m. and continued till 10 a.m. Event "Snow storm" was definitely true between midnight and 10 a.m. and therefore we use the "FixedRight Event" icon to represent it.

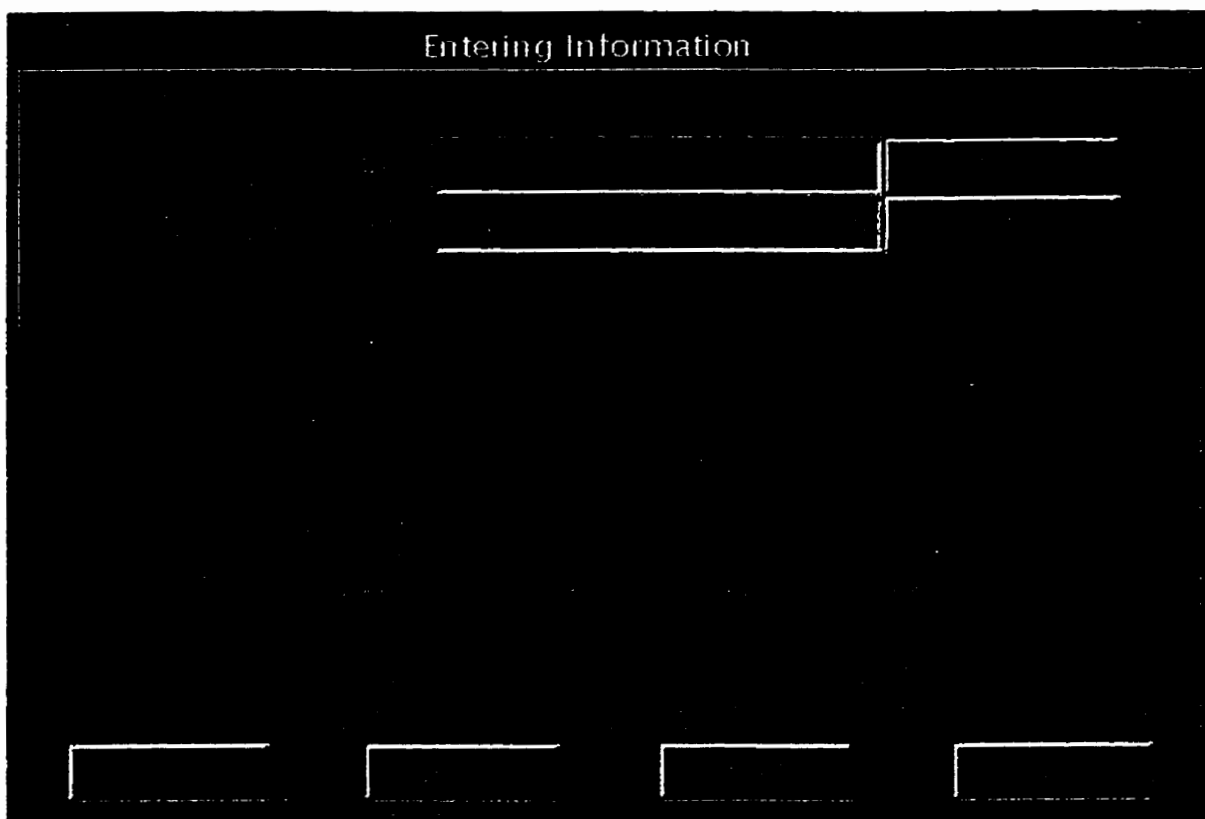


Figure A.13 FixedRight Event

After entering the event name, description and color, the user positions the "FixedRight Event" icon on the time scale. The user places the right end of the icon at 10 a.m. on the time scale and the left end automatically covers the rest of the scale as shown in figure A.13.

Querying the system:

Clicking on the “Query Information” option under “Edit” menu opens a window shown in figure A.14. This screen is used to query the system.

The user can perform the following queries:

1. Is an event true at a given time ? (A)
2. What is true about an event ? (B)

The following example (3) will be used to perform the query:

- Henry was watching TV between 8:30 a.m. and 10 a.m.

To perform a query using (A):

1. Select the event “Watching TV” from the dropdown list box by clicking on the down arrow button next to the “OK” button (figure A.14)
2. User confirms his/her selection by clicking on the “OK” button.
3. Configure the “Fixed Event” icon on the time scale between 8:30 a.m. and 10 a.m.
4. Click on the “Query” button to execute the query. Figure A.14 displays the result for the query.

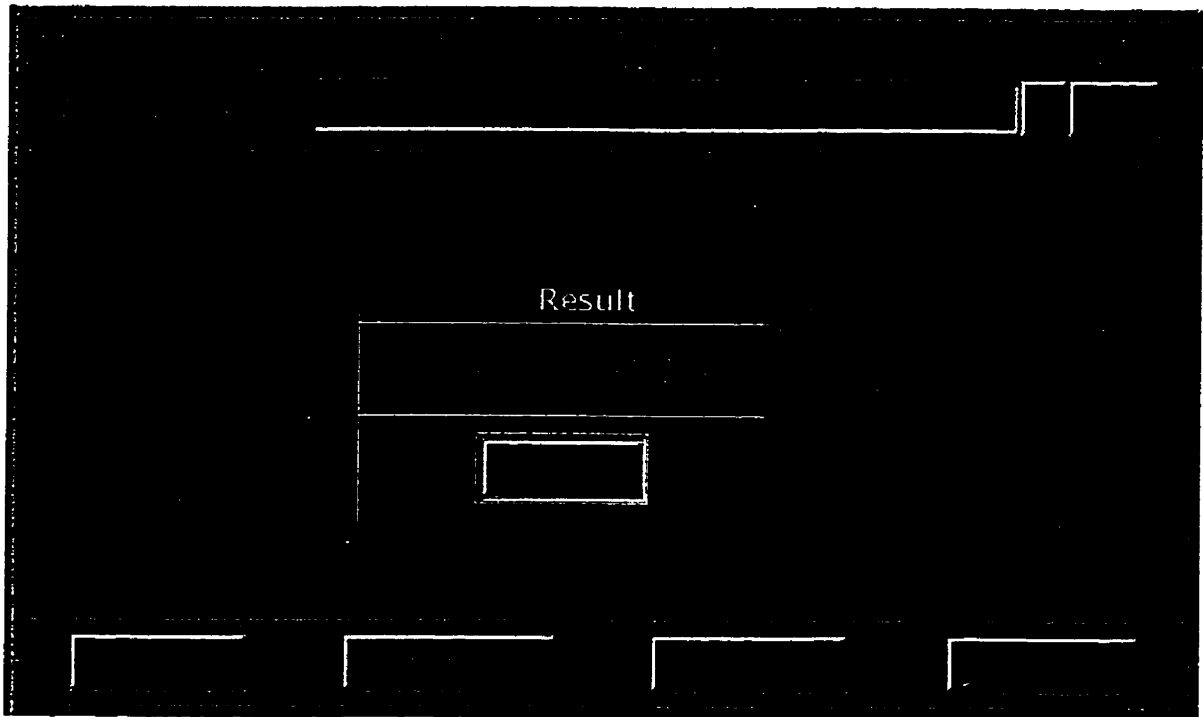


Figure A.14 Query window for Fixed Event

To perform a query using (B):

1. Select an event "Watching TV" from the dropdown list box by clicking on it.
2. Click on the "OK" button to confirm the selection.
3. User then points and clicks on the "What's True" icon and the system displays the results of the query as shown in figure A.15.

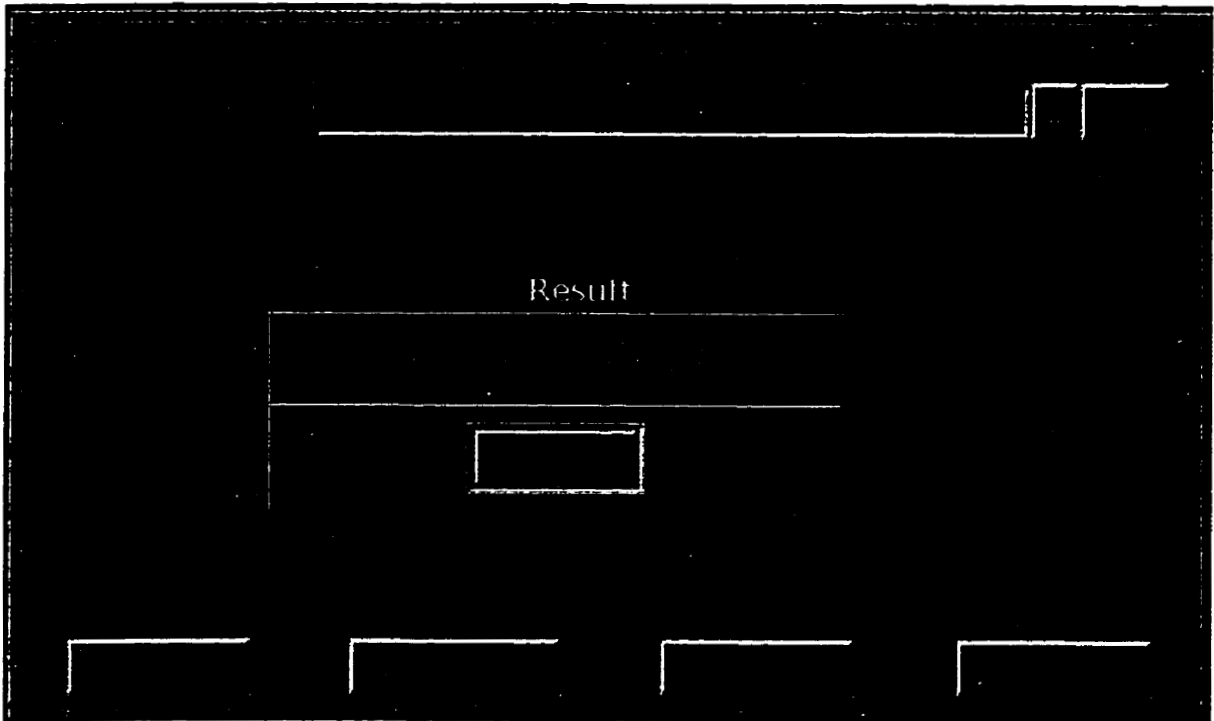


Figure A.15 Query using What's True

A.1.3 Options menu:

The "Option" menu as shown in figure A.16 provides the following functions:

View Database:

1. Click on the "View Database" button under the "Options" menu to open a window that displays event name, event description and color used for all the events in the database as shown in figure A.17.
2. Click on the "OK" button to close this window.

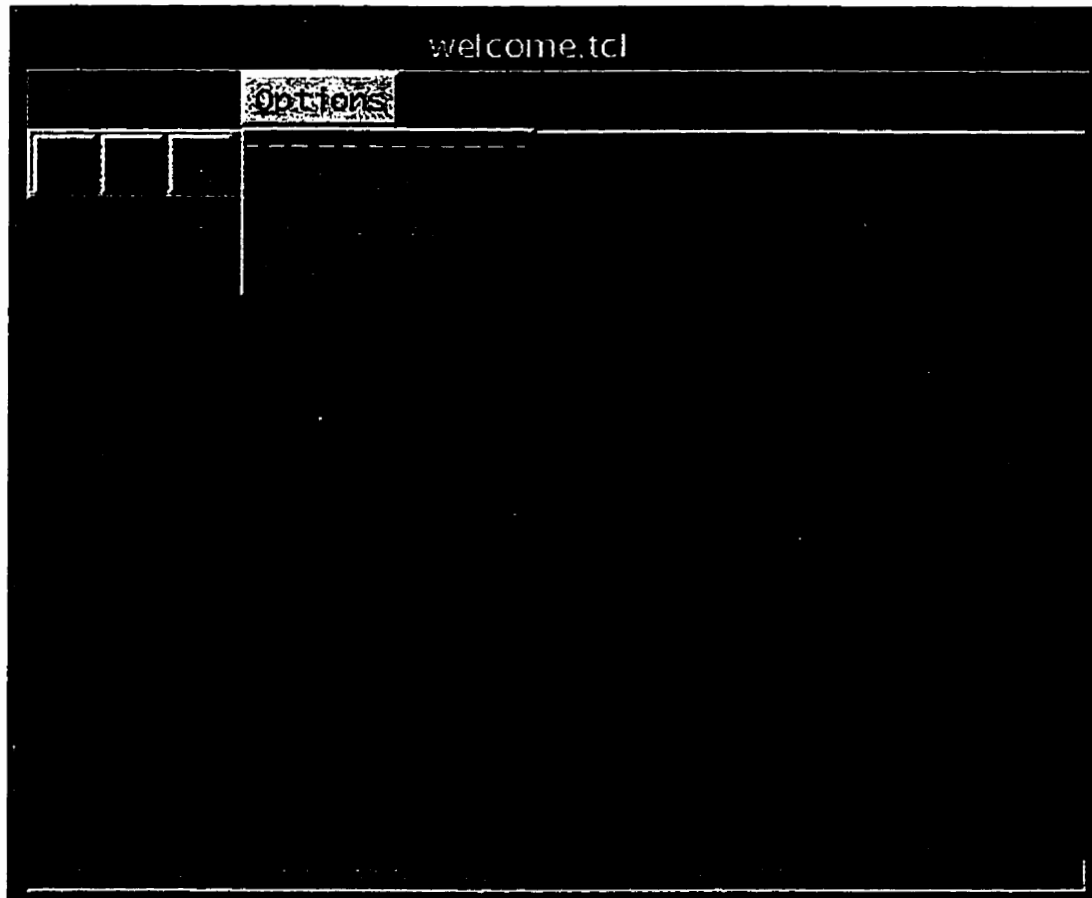


Figure A.16 Main window (Options menu).

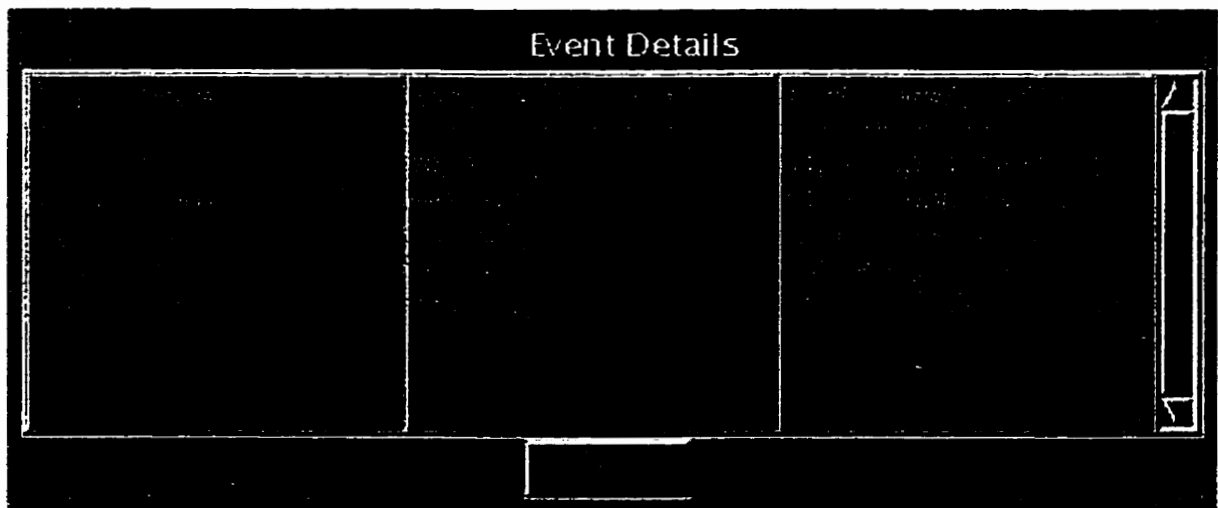


Figure A.17 View Database

Colors Used:

1. Click on the “Colors Used” button under the “Options” menu to open a window that displays all the colors in use.
2. Every color is displayed along with its hexadecimal value as shown in figure A.18. Note that every color in figure A.17 also appears in figure A.18.
3. Click on the “OK” button to close this window.

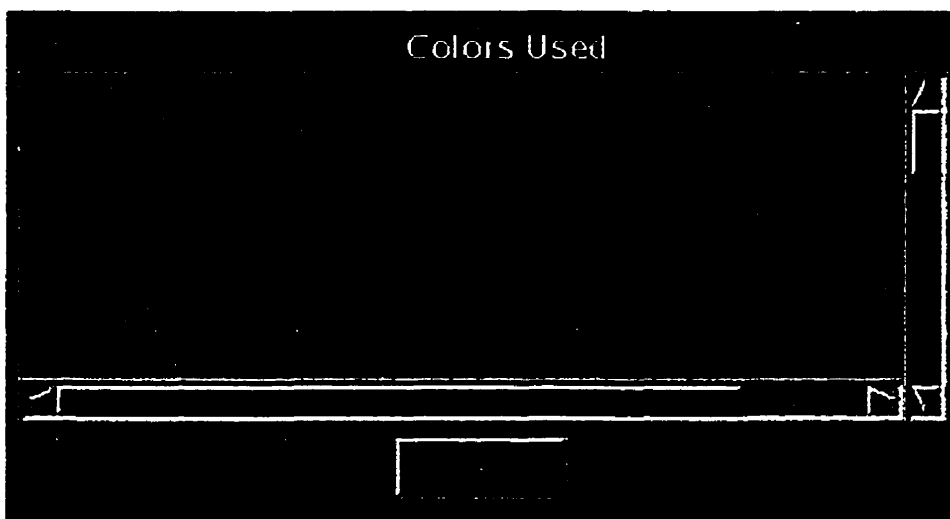


Figure A.18 Colors Used

Symbols:

1. Click on the “Symbols” button on the “Options” menu to open a window that displays a list of symbols used.
2. Each symbol is displayed along with its name and meaning as shown in figure A.19.

APPENDIX A. USER MANUAL

3. Click on the "OK" button to close this window.

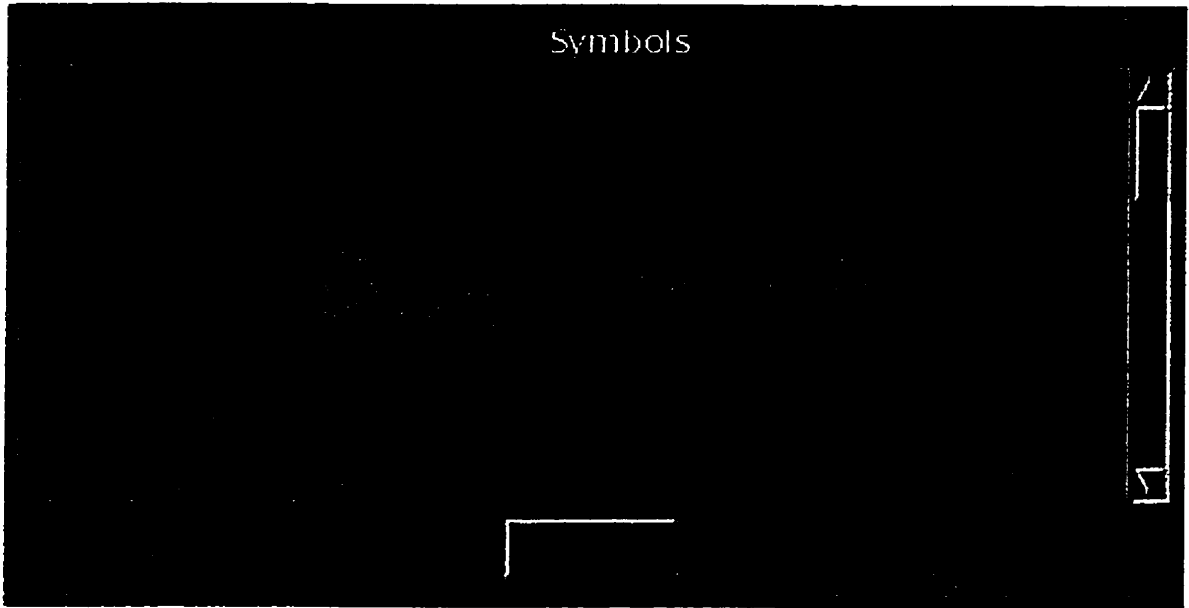


Figure A.19 Symbols

A.2 Tutorial

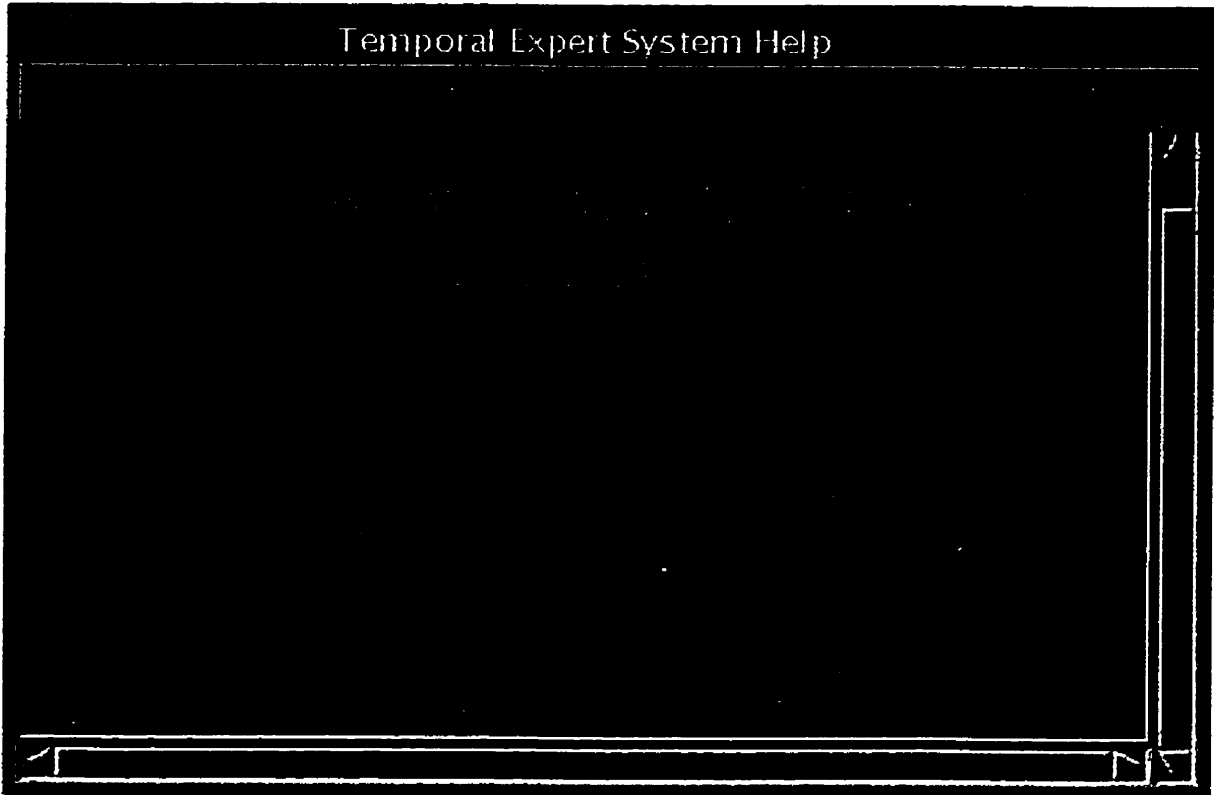


Figure A.20 Tutorial window

To start the tutorial:

1. Click on the start tutorial button in the Interface Window (figure A.2) to open the Tutorial Window (figure A.20).
2. This window provides help on various topics such as "Overview of the System", "Entering the Information" etc., as shown in figure A.20.
3. The help topics shown above have a "hypertext" behavior.
4. Clicking on the appropriate topic opens the help for that topic in the same window as shown in figure A.21.

To get help on Overview of the System:

1. Click on the "Overview of the System" to get help on the overview of the system as shown in figure A.21.
2. Click on the "Return to Main Menu" to get back to the main menu.
The user can also get back to the main menu by clicking on the "Original Topics" from the "File" menu. (Figure A.21).
3. Click on the "Close" button to terminate the Tutorial.
4. The Edit menu provides a "copy" function for copying any text from the Help screen.

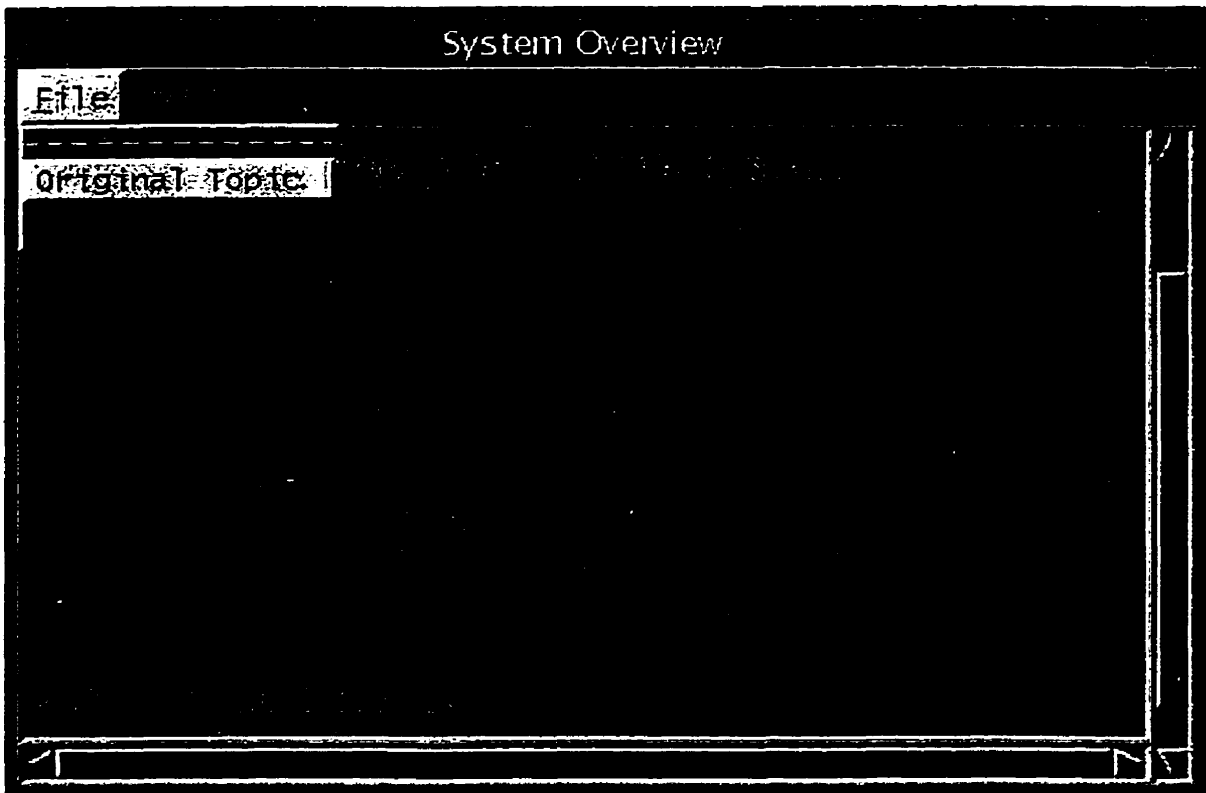


Figure A.21 Overview of the system

Appendix B

APPENDIX B. SOURCE CODE

```
#!/usr/local/bin/wish

#
# Canvas widget with the welcome information
#

proc call_main {} {
    source interface.tcl
    destroy .can
}

# Creates the required font.

font create textfont -family Helvetica -size 27 -weight bold \
    -slant italic
set bigf textfont
font create textfont1 -family Courier -size 16 -weight bold \
    -slant italic
set medf textfont1
font create textfont2 -family Times -size 16 -slant italic \
    -weight bold
set namef textfont2

# Canvas with various canvas items.

canvas .can -width 7c -height 10c
.can create rectangle .25c .25c 6.75c 9.75c -outline black \
    -width 2

.can create text 3.25c 1.5c -font $bigf \
    -text "WELCOME" -fill red
.can create line 1c 2c 5.5c 2c -fill black -width 2

.can create text 3.5c 3c -font $medf \
    -text "To" -fill blue
.can create text 3.5c 4c -font $medf \
    -text "The Temporal Expert " -fill blue
.can create text 3.25c 5c -font $medf \
    -text "System Shell" -fill blue

.can create bitmap 3c 6c -bitmap gray12 -foreground black
.can create bitmap 3.5c 6c -bitmap gray25 -foreground black

.can create text 3.5c 7c -font $medf \
    -text "Designer" -fill maroon
.can create text 3.25c 7.5c -font $medf \
    -text "Sharad Sachdev" -fill maroon

button .can.b -text "Continue" -font $namef \
    -command {call_main}
.can create window 3.5c 9c -window .can.b

pack .can
```


APPENDIX B. SOURCE CODE

```
#!/usr/local/bin/wish

#
# Frame widgets within the interface window
#
proc s_shell { } {
    source application.tcl
    destroy .fl
}

# Creates the required font

font create txtfont -family Times -size 18 -slant italic\
    -weight bold
set namef txtfont

# Frames with buttons.

frame .fl -relief raised -borderwidth 1
button .fl.b1 -text "-- Start Shell --" -font $namef \
    -foreground maroon -borderwidth 5 -command { s_shell}
button .fl.b2 -text "-- Start Tutorial --" -font $namef \
    -foreground maroon -borderwidth 5 -command { source helpdlg.tcl}
button .fl.b3 -text "-- Close --" -font $namef \
    -foreground maroon -borderwidth 5 -command { exit }
pack .fl.b1 .fl.b2 .fl.b3 -padx 5 -pady 1 -ipadx 4 -ipady 6 -fill both

pack .fl

#!/usr/local/bin/wish

#Creating the application menu

#setting global variable for storing file name.
global nfname
set nfname ""

#
# Create images for toolbar bitmaps.
#
image create bitmap tool_new -file ../toolbar/new.xbm
image create bitmap tool_open -file ../toolbar/open.xbm
image create bitmap tool_save -file ../toolbar/save.xbm
image create bitmap tool_cut -file ../toolbar/cut.xbm
image create bitmap tool_copy -file ../toolbar/copy.xbm
image create bitmap tool_paste -file ../toolbar/paste.xbm

# Old way to create a menubar

frame .menubar -relief raised -borderwidth 1

#File menu

menubutton .menubar.file -text "File" \
    -menu .menubar.file.menu -underline 0
pack .menubar.file -side left
```

APPENDIX B. SOURCE CODE

```
menu .menubar.file.menu

.menubar.file.menu add command -label "New" \
    -underline 0 -command { source file_new.tcl }
.menubar.file.menu add command -label "Open..." \
    -underline 0 -command { source filedlg.tcl }
.menubar.file.menu add separator
.menubar.file.menu add command -label "Save" \
    -underline 0 -command { proc_save }
.menubar.file.menu add command -label "Print" \
    -underline 0 -command { proc_print }
.menubar.file.menu add separator
.menubar.file.menu add command -label "Exit" \
    -underline 0 -command { destroy . }

#Edit menu

menubutton .menubar.edit -text "Edit" \
    -menu .menubar.edit.menu -underline 0
pack .menubar.edit -side left

menu .menubar.edit.menu

.menubar.edit.menu add command -label "Cut" \
    -underline 2 -accelerator "Ctrl+x" -command "edit_cut all"
.menubar.edit.menu add command -label "Copy" \
    -underline 0 -accelerator "Ctrl+c" -command "edit_copy all"
.menubar.edit.menu add command -label "Paste" \
    -underline 0 -accelerator "Ctrl+v" -command "edit_paste all"
.menubar.edit.menu add separator
.menubar.edit.menu add command -label "Enter Information" \
    -underline 1 -command { source Aruler.tcl }
.menubar.edit.menu add command -label "Query Information" \
    -underline 5 -command { source lquery.tcl }

#Options menu

menubutton .menubar.options -text "Options" \
    -menu .menubar.options.menu -underline 0
pack .menubar.options -side left

menu .menubar.options.menu

.menubar.options.menu add command -label "View Database" \
    -underline 0 -command {source show_fact_color.tcl }
.menubar.options.menu add command -label "Colors Used" \
    -underline 0 -command {source show_color.tcl }
.menubar.options.menu add command -label "Symbols" \
    -underline 0 -command {source show_symbol.tcl}

#Help menu

menubutton .menubar.help -text "Help" \
    -menu .menubar.help.menu -underline 0
pack .menubar.help -side right
```

APPENDIX B. SOURCE CODE

```
menu .menubar.help.menu

.menubar.help.menu add command -label "About..." \
    -underline 0 -command "help_about .menubar"

#Create a faked main area.

label .main -text ""

pack .menubar -side top -fill x -expand true

#####
# Toolbar
#####

frame .toolbar -bd 2 -relief raised

    # Group for new, open, save.
    set frm .toolbar.file_group
    frame $frm -bd 0

    button $frm.new -image tool_new -command {puts new}
    button $frm.open -image tool_open -command {puts open}
    button $frm.save -image tool_save -command {puts save}
    pack $frm.new $frm.open $frm.save -side left
    pack $frm -side left

    # Set up short help.
    bind $frm.new <Enter> \
        ".status configure -text {New file}"
    bind $frm.new <Leave> \
        ".status configure -text { }"

    bind $frm.open <Enter> \
        ".status configure -text {Open file}"
    bind $frm.open <Leave> \
        ".status configure -text { }"

    bind $frm.save <Enter> \
        ".status configure -text {Save file}"
    bind $frm.save <Leave> \
        ".status configure -text { }"

    # Group for cut, copy, paste.
    set twidget 1
    set frm .toolbar.clip_group
    frame $frm -bd 0

    button $frm.cut -image tool_cut -command {edit_cut .}
    button $frm.copy -image tool_copy -command {edit_copy .}
    button $frm.paste -image tool_paste

    pack $frm.cut $frm.copy $frm.paste -side left
```

APPENDIX B. SOURCE CODE

```
bind all <Button-2> { beforePaste %W}

proc beforePaste {textw} {
    global twidget
    set twidget $textw
}

bind $frm.paste <Button> {edit_paste $twidget}

bind $frm.cut <Enter> \
    ".status configure -text {Cut}"
bind $frm.cut <Leave> \
    ".status configure -text { }"

bind $frm.copy <Enter> \
    ".status configure -text {Copy}"
bind $frm.copy <Leave> \
    ".status configure -text { }"

bind $frm.paste <Enter> \
    ".status configure -text {Paste}"
bind $frm.paste <Leave> \
    ".status configure -text { }"

# Pack second group with X padding to space out.
pack $frm -side left -padx 10

pack .toolbar -side top -fill x

pack .main -ipady 150 -ipadx 250 -expand true -side top

# Status area

label .status -relief sunken -anchor w -borderwidth 1 \
    -text "Status"
pack .status -fill x -side bottom

bind .menubar.file <Enter> \
    ".status configure -text {Operations:New,Open,Save,Print,Exit}"
bind .menubar.file <Leave> \
    ".status configure -text { }"

bind .menubar.edit <Enter> \
    ".status configure -text \
        {Operations:Cut,Copy,Paste,Enter Info,Query Info}"
bind .menubar.edit <Leave> \
    ".status configure -text { }"

bind .menubar.options <Enter> \
    ".status configure -text {Operations:View Database,Colors
Used,Symbols}"
bind .menubar.options <Leave> \
    ".status configure -text { }"
```

APPENDIX B. SOURCE CODE

```
bind .menubar.help <Enter> \  
".status configure -text {Help}"  
bind .menubar.help <Leave> \  
".status configure -text { }"  
  
#  
# Implementing the cut action for a text widget.  
#  
proc edit_cut { textwidget } {  
  
# Check if any text is selected in textwidget.  
set owner [selection own]  
  
# clear clipboard  
clipboard clear  
  
catch {  
set text [selection get]  
clipboard append $text  
# Delete selected text.  
$owner delete sel.first sel.last  
}  
puts "Calling from cut"  
puts "textwidget is $textwidget"  
  
}  
  
#  
# Implementing the copy action for a text widget.  
#  
proc edit_copy { textwidget } {  
  
# Check if any text is selected in textwidget  
set owner [selection own]  
  
# clear clipboard  
clipboard clear  
  
catch {  
clipboard append [selection get]  
}  
puts "Calling from copy"  
puts "textwidget is $textwidget"  
  
}  
  
#  
# Implementing the paste action for a text widget.  
#  
proc edit_paste {textwidget} {  
  
puts "Called from edit_paste"  
set owner [selection own]
```

APPENDIX B. SOURCE CODE

```
puts "textwidget is $textwidget and owner is $owner"
catch {
    set clip [selection get -selection CLIPBOARD]
}

set idx [$textwidget index insert]
catch {
    $textwidget insert $idx $clip
}
}

bind . <Control-Key-x> "edit_cut . ;break"
bind . <Control-Key-c> "edit_copy . ;break"
bind . <Control-Key-v> "edit_paste . ;break"

proc help_about {toplevel} {
    tk_messageBox -default ok -icon info -message \
    "GUI for Temporal Expert System,
    - by Sharad Sachdev" \
    -parent $toplevel -title "About System" -type ok
}

# Printing options

proc proc_print {} {
    global nfname
    if {$nfname != ""} {
        set pmess [tk_messageBox -parent .menubar\
        -title {Print?} -type okcancel -icon warning\
        -message\
        "Confirm file print."]
        if {$pmess == "ok"} {
            set command "lpr -P cs-lw4 $nfname"
            eval exec $command
        } elseif {$pmess == "cancel"} {}
    } else {
        set pmess [tk_messageBox -parent .menubar\
        -title {Error} -type ok -icon error\
        -message\
        "No file to print. Select a file before printing."]
    }
}

#!/usr/local/bin/wish

# This file reads the file which stores the facts,descrip,color
# information and displays the color in the canvas widget.
#
#
# Procedure file_read given below stores the fact,descrip,
# color information in the global variable list_of_events,
# list_of_descrip variables.
#
```

APPENDIX B. SOURCE CODE

```
set g .colorwindow
oplevel $g -class Dialog
wm title $g "Colors Used"
wm transient $g .
wm geometry $g +300+300

set filename event_details.dat

global list_of_events
global list_of_descrip
global list_of_colors

set data ""
set list_of_events ""
set list_of_descrip ""
set list_of_colors ""
if {[file readable $filename]} {
    set fileid [open $filename "r"]
    while {[eof $fileid] != 1} {
        gets $fileid data
        set list_of_events [linsert $list_of_events end $data]
        gets $fileid data
        set list_of_colors [linsert $list_of_colors end $data]
        gets $fileid data
        set list_of_descrip [linsert $list_of_descrip end
$data]
    }
    close $fileid
}

canvas $g.clr_can -width 12c -height 6c -yscrollcommand \
"$g.v_scroll set" -xscrollcommand "$g.h_scroll set" \
-scrollregion { 0 0 500 600 }

scrollbar $g.v_scroll -command "$g.clr_can yview"
scrollbar $g.h_scroll -command "$g.clr_can xview" -orient horizontal

#
#Create the heading
#
font create txtfont1 -family Times -size 18 -slant italic \
-weight bold
set namef txtfont1

font create txtfont2 -family Courier -size 10 -slant italic
set namef2 txtfont2

$g.clr_can create text 6c .5c -font $namef -text \
"- Following colors are in use -" -fill blue
#
#Fill in canvas.
#

#Postion paramters x and y
set x1 1; set y1 2
```

APPENDIX B. SOURCE CODE

```
set len [llength $list_of_colors]
set len [expr $len-1]
for {set i 0} {$i < $len} {incr i} {
    set tmp [lindex $list_of_colors $i]
    if { $x1 == 13} {
        set y1 [expr $y1 + 2.5]
        set x1 1
    }
    set x2 [expr $x1 + 1]
    set y2 [expr $y1 + 1]

    $g.clr_can create rectangle ${x1}c ${y1}c ${x2}c ${y2}c \
        -outline black -fill $tmp -width 1.25
    set ytext_pos [expr $y2 + .5]
    set xtext_pos [expr $x1 + .5]
    $g.clr_can create text ${xtext_pos}c ${ytext_pos}c \
        -font $namef2 -text $tmp -fill maroon
    set x1 [expr $x1 + 2]
}

frame $g.frame -relief raised
pack $g.frame -side bottom -fill x -pady 2m
button $g.frame.b -text " OK " -command {destroy $g}
pack $g.frame.b -side left -expand 1

pack $g.v_scroll -side right -fill y
pack $g.h_scroll -side bottom -fill x

pack $g.clr_can -side left -expand 1

font delete txtfont1 txtfont2
#!/usr/local/bin/wish

# This file reads the file which stores the facts,descrip,color
# information and displays the same in form of MultiColumn list.
#
#
# Procedure file_read given below stores the fact,descrip,
# color information in the global variable list_of_events,
# list_of_descrip variables.
#

set t .multilist
toplevel $t -class Dialog
wm title $t "Event Details"
wm transient $t .
wm geometry $t +300+300

set filename event_details.dat

global list_of_events
global list_of_descrip
global list_of_colors
```


APPENDIX B. SOURCE CODE

```
set data ""
set list_of_events ""
set list_of_descrip ""
set list_of_colors ""
if {[file readable $filename]} {
    set fileid [open $filename "r"]
    while {[eof $fileid] != 1} {
        gets $fileid data
        set list_of_events [linsert $list_of_events end $data]
        gets $fileid data
        set list_of_colors [linsert $list_of_colors end $data]
        gets $fileid data
        set list_of_descrip [linsert $list_of_descrip end
$data]
    }
    close $fileid
}

#
# Tcl script that creates multiple listboxes
# with one scrollbar.
#
#
# This proc scrolls a number of listboxes all together
# from one scrollbar.
#
# The scroll_list holds a list of the widgets
# to scroll. This must be a list. The args
# hold all the remaining arguments, which
# come from the scrollbar. All these are
# passed to each widget in the scroll_list.
#
proc multi_scroll { scroll_list args } {
    global t
    # Get info on list of listboxes.
    set len [llength $scroll_list]

    for {set i 0} {$i < $len} {incr i} {
        set temp_list [lindex $scroll_list $i]
        eval $temp_list yview $args
    }
}

#
# Fill in list with various data.
#
proc FillList1 { listvar } {
    global list_of_events
    set len [llength $list_of_events]
    set tmp "Event Name"
    eval $listvar insert end {$tmp}
    set tmp "-----"
```

APPENDIX B. SOURCE CODE

```
        eval $listvar insert end {$tmp}

        for {set i 0} { $i <= $len} {incr i} {
            set tmp [lindex $list_of_events $i]
            eval $listvar insert end {$tmp}
        }
    }

proc FillList2 { listvar } {
    global list_of_colors
    set len [llength $list_of_colors]
    set tmp "Associated Color"
    eval $listvar insert end {$tmp}
    set tmp "-----"
    eval $listvar insert end {$tmp}

    for {set i 0} { $i <= $len} {incr i} {
        set tmp [lindex $list_of_colors $i]
        eval $listvar insert end {$tmp}
    }
}

proc FillList3 { listvar } {
    global list_of_descrip
    set len [llength $list_of_descrip]
    set tmp "Event Description"
    eval $listvar insert end {$tmp}
    set tmp "-----"
    eval $listvar insert end {$tmp}

    for {set i 0} { $i <= $len} {incr i} {
        set tmp [lindex $list_of_descrip $i]
        eval $listvar insert end {$tmp}
    }
}

# Use a frame around all lists.

frame $t.frame -relief groove -borderwidth 3
button $t.button -text " OK " -command { destroy $t}

listbox $t.frame.list1 \
    -borderwidth 1 \
    -relief raised \
    -selectmode single \
    -yscrollcommand "$t.frame.scroll set" \

listbox $t.frame.list2 \
    -borderwidth 1 \
    -relief raised \
    -selectmode single \
    -yscrollcommand "$t.frame.scroll set" \

listbox $t.frame.list3 \
    -borderwidth 1 \
    -relief raised \
```

APPENDIX B. SOURCE CODE

```
-selectmode single \  
-yscrollcommand "$t.frame.scroll set" \  
  
# Fill lists with data.  
FillList1 $t.frame.list1  
FillList2 $t.frame.list2  
FillList3 $t.frame.list3  
  
scrollbar $t.frame.scroll \  
-command \  
{ multi_scroll {$t.frame.list1 $t.frame.list2 $t.frame.list3} }  
  
pack $t.frame.scroll -side right -fill y  
  
pack $t.frame.list1 $t.frame.list2 \  
$t.frame.list3 -side left  
pack $t.frame  
pack $t.button  
  
#!/usr/local/bin/wish  
  
#  
#This file shows the various symbols which are used in  
#the program. This includes symbols for entering and  
#querying information.  
#  
  
set u .symbolwindow  
toplevel $u -class Dialog  
wm title $u "Symbols"  
wm transient $u .  
wm geometry $u +300+300  
  
canvas $u.sym_can -width 15c -height 6c -scrollregion { 0 0 600 800} \  
-yscrollcommand "$u.v_scroll set"  
  
scrollbar $u.v_scroll -command "$u.sym_can yview"  
  
#  
#Create the heading  
#  
font create txtft1 -family Times -size 18 -slant italic \  
-weight bold  
set namef1 txtft1  
  
font create txtft2 -family Courier -size 10 -slant italic \  
-weight bold  
set namef2 txtft2  
  
font create txtft3 -family Courier -size 10  
set namef3 txtft3  
  
$u.sym_can create text 8c 1c -font $namef1 -text \  
"- Symbols and their meanings -" -fill blue
```

APPENDIX B. SOURCE CODE

```
set c $u.sym_can

#
# The following procedures create the symbols on the canvas.
#

# rulerMkTab --
# This procedure creates a new circular polygon in a canvas to
# represent a point event.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the tab stop.
#

proc rulerMkTab {c x y} {
    set v1 [wininfo fpixels $c .3c]
    $c create oval [expr $x-$v1/2] [expr $y-$v1/2] [expr $x+$v1/2] \
        [expr $y+$v1/2] -fill black
}

# rulerMkTabIL --
# This procedure creates a new Infinite line item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the IL item.
#

proc rulerMkTabIL {c x y} {
    set v1 [wininfo fpixels $c 1c]
    $c create line $x $y [expr $x+$v1] $y \
        -arrow both -fill black -width 8
}

# rulerMkTabFE --
# This procedure creates a new Fixed End line item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#

proc rulerMkTabFE {c x y} {
    set v1 [wininfo fpixels $c 1c]
    $c create line $x $y [expr $x+$v1] \
        $y -fill black -width 8
}

# rulerMkTabFL
# This procedure creates a new Fixed Left item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
```

APPENDIX B. SOURCE CODE

```
#

proc rulerMkTabFL {c x y} {
    set vl [winfo fpixels $c 1c]
    $c create line $x $y [expr $x+$vl] \
        $y -fill black -width 8 -arrow last
}

# rulerMkTabFR
# This procedure creates a new Fixed Right item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#
# vl plays an important role in positioning the item.

proc rulerMkTabFR {c x y} {
    set vl [winfo fpixels $c 1c]
    $c create line $x $y [expr $x+$vl] $y \
        -fill black -width 8 -arrow first
}

# rulerMkTabQues
# This procedure creates a new Question item in the canvas.
# Clicking on this tells everything about the selected event

proc rulerMkTabQues { c x y} {
    $c create bitmap 2c 18c -bitmap questhead
}

#
#Fill in the canvas.
#Creates the symbol and associated message window
#

$c addtag well withtag [$c create rect 1.5c 3.5c 2.5c 2.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well withtag [rulerMkTab $c [winfo pixels $c 2c] \
    [winfo pixels $c 3c]]
$c create text 2c 4c -text "Point Event" \
    -font $namef2 -fill maroon

message $c.ml -width 10c -text "Used for events which occur at precise\
points over the given time scale. E.g., Phone rang at 2:00 pm."
$c create window 9c 3c -window $c.ml

$c addtag well withtag [$c create rect 1.5c 6.5c 2.5c 5.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well withtag [rulerMkTabIL $c [winfo pixels $c 1.5c] \
    [winfo pixels $c 6c]]
$c create text 2c 7.15c -text "Limitless Event" \
    -font $namef2 -fill maroon
```

APPENDIX B. SOURCE CODE

```
message $c.m2 -width 10c -text "Event that occurred at some unknown\  
time in past and continues in future.E.g., It has been raining today."  
$c create window 9c 6c -window $c.m2
```

```
$c addtag well withtag [$c create rect 1.5c 9.5c 2.5c 8.5c \  
-outline black -fill [lindex [$c config -bg] 4]]  
$c addtag well withtag [rulerMkTabFE $c [wininfo pixels $c 1.5c] \  
[wininfo pixels $c 9.05c]]  
$c create text 2c 10.15c -text "Fixed Event" \  
-font $namef2 -fill maroon
```

```
message $c.m3 -width 10c -text "Event that occurred between two points\  
on the given time scale. Activity has precise start and end points\  
E.g., We had lunch between 2 & 3 pm."  
$c create window 9c 9c -window $c.m3
```

```
$c addtag well withtag [$c create rect 1.5c 12.5c 2.5c 11.5c \  
-outline black -fill [lindex [$c config -bg] 4]]  
$c addtag well withtag [rulerMkTabFL $c [wininfo pixels $c 1.5c] \  
[wininfo pixels $c 12.05c]]  
$c create text 2c 13.15c -text "FixedLeft Event" \  
-font $namef2 -fill maroon
```

```
message $c.m4 -width 10c -text "Event that starts at a known fixed\  
point and continues in future. E.g., Basketball game started at 6 pm\  
and went on till late evening"  
$c create window 9c 12c -window $c.m4
```

```
$c addtag well withtag [$c create rect 1.5c 15.5c 2.5c 14.5c \  
-outline black -fill [lindex [$c config -bg] 4]]  
$c addtag well withtag [rulerMkTabFR $c [wininfo pixels $c 1.5c] \  
[wininfo pixels $c 15.05c]]  
$c create text 2c 16.15c -text "FixedRight Event" \  
-font $namef2 -fill maroon
```

```
message $c.m5 -width 10c -text "Event which started at some unknown\  
point in past and has fixed end point. Complement of FixedLeft Event\  
E.g., After a long sleep, I woke up at 12:00 pm."  
$c create window 9c 15c -window $c.m5
```

```
$c addtag well withtag [$c create rect 1.5c 18.5c 2.5c 17.5c \  
-outline black -fill [lindex [$c config -bg] 4]]  
$c addtag well withtag [rulerMkTabQues $c [wininfo pixels $c 1.5c] \  
[wininfo pixels $c 18.05c]]  
$c create text 2c 19.15c -text "What's True" \  
-font $namef2 -fill maroon
```

```
message $c.m6 -width 10c -text "This symbol is used for querying the \  
system. It tells, all what is true for a selected event"  
$c create window 9c 18c -window $c.m6
```

```
frame $u.frame -relief raised  
pack $u.frame -side bottom -fill x -pady 2m  
button $u.frame.b -text " OK " -command {destroy $u}
```

APPENDIX B. SOURCE CODE

```
pack $u.frame.b -side left -expand 1

pack $u.v_scroll -side right -fill y

pack $u.sym_can -side left -expand 1

font delete txtft1 txtft2 txtft3
#!/usr/local/bin/wish

#
# ruler.tcl --
#
# This script creates a canvas widget that displays a ruler
# with tab stops that can be set, moved, and deleted.
#
# A list of all global variables
# user_filename is the name of the file entered by the user.

set user_filename event_details.dat

# proc positionWindow to sets the position of the Window on the screen.

proc positionWindow w {
    wm geometry $w +300+300
}

#Font variable
set font {Courier 12}

#Color for the widget items
set color #2230f0

# rulerMkTab --
# This procedure creates a new circular polygon in a canvas to
# represent a point event.
#
# Arguments:
# c - The canvas window.
# x, y - Coordinates at which to create the tab stop.
#

proc rulerMkTab {c x y} {
    set vl [winfo fpixels $c .3c]
    $c create oval [expr $x-$vl/2] [expr $y-$vl/2] [expr $x+$vl/2] \
        [expr $y+$vl/2] -fill black
}

# rulerMkTabIL --
# This procedure creates a new Infinite line item in a canvas.
#
# Arguments:
# c - The canvas window.
# x, y - Coordinates at which to create the IL item.
#
# Arguments:
```

APPENDIX B. SOURCE CODE

```
proc rulerMkTabIL {c x y} {
    set v1 [winfo fpixels $c 1c]
    $c create line $x $y [expr $x+$v1] $y \
        -arrow both -fill black -width 8
}

# rulerMkTabIL2 --
# This procedure creates a new Infinite line item in a canvas.
# Behaves similar to the above but covers the entire length.

proc rulerMkTabIL2 {c x y} {
    global color
    set v1 [winfo fpixels $c 12c]
    $c create line $x $y [expr $x+$v1] $y -arrow both -fill $color \
        -width 8
}

# Set no. of fixed objects as numFE
set numFE 0
# rulerMkTabFE --
# This procedure creates a new Fixed End line item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#

proc rulerMkTabFE {c x y} {
    set v1 [winfo fpixels $c 1c]
    $c create line $x $y [expr $x+$v1] \
        $y -fill black -width 8
}

# rulerMkTabFL
# This procedure creates a new Fixed Left item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#

proc rulerMkTabFL {c x y} {
    set v1 [winfo fpixels $c 1c]
    $c create line $x $y [expr $x+$v1] \
        $y -fill black -width 8 -arrow last
}

# rulerMkTabFL2
# This procedure creates a new Fixed left item in a canvas.
# Behaves similar to the above but covers the right length.

proc rulerMkTabFL2 {c x y} {
    upvar #0 demo_rulerInfo v
```


APPENDIX B. SOURCE CODE

```
    global color
    set vl [wininfo fpixels $c lc]
    $c create line $x $y $v(right) \
        $y -fill $color -width 8 -arrow last
}

# rulerMkTabFR
# This procedure creates a new Fixed Right item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
# vl plays an important role in positioning the item.

proc rulerMkTabFR {c x y} {
    set vl [wininfo fpixels $c lc]
    $c create line $x $y [expr $x+$vl] $y \
        -fill black -width 8 -arrow first
}

# rulerMkTabFR2
# This procedure creates a new Fixed Right item in a canvas.
# Behaves similar to the above but covers the left length.

proc rulerMkTabFR2 { c x y} {
    upvar #0 demo_rulerInfo v
    global color
    set vl [wininfo fpixels $c lc]
    $c create line $v(left) $y $x $y -fill black -width 8 \
        -fill $color -width 8 -arrow first
}

# setting up the window manager options.
set w .ruler
global tk_library
catch {destroy $w}
toplevel $w
wm title $w "Entering Information"
wm iconname $w "ruler"
positionWindow $w
set c $w.c

# Creating the first section of the entering information screen.
frame $w.entry -borderwidth 1 -relief raised
label $w.entry.event_name -text "Event Name:"
entry $w.entry.event_entry -width 25 -textvariable event

label $w.entry.fact_name -text "Event Description:"
entry $w.entry.fact_entry -width 25 -textvariable descrip

button $w.entry.color -text "Pick Color" \
    -command { source colordlg.tcl }
button $w.entry.ok -text "OK" -command {
    puts "Event Name:  $event"
    puts "Fact Description: $descrip"
```

APPENDIX B. SOURCE CODE

```
        puts "Color selected for this fact: $color"

#       Important file manipulation steps ...

    if { $event == "" || $descrip == "" } {
        set result [tk_messageBox -parent .ruler \
            -title Error -type ok -icon error \
            -message \
            "Missing Event Name or Event description !! "]
        focus $w.entry.event_entry
    } else {
        file_write event_details.temp $event $color $descrip
    }
}

# Using grid manager to place the objects.
grid config $w.entry.event_name -column 0 -row 0 -sticky e
grid config $w.entry.event_entry -column 1 -row 0 -sticky snw
grid config $w.entry.color -column 2 -row 0 -sticky snw

grid config $w.entry.fact_name -column 0 -row 1 -sticky e
grid config $w.entry.fact_entry -column 1 -row 1 -sticky snw
grid config $w.entry.ok -column 2 -row 1 -sticky snw

pack $w.entry -side top -fill x -ipady 1c -ipadx 1c

frame $w.buttons
pack $w.buttons -side bottom -fill x -pady 2m
button $w.buttons.cancel -text " Cancel " -command { cancel $w }
button $w.buttons.continue -text "Continue" \
    -command {continue_proc $w knowledge_base $color}

button $w.buttons.done -text " Done " -command {
    done $w knowledge_base $color
}
button $w.buttons.help -text " Help " -command { source
enterHelp.tcl}
pack $w.buttons.cancel $w.buttons.continue $w.buttons.done \
    $w.buttons.help -side left -expand 1

focus $w.entry.event_entry

#File for storing event_details.
proc file_write { filename event color descrip } {
    return [catch {
        set fileid [open $filename "a+"]
        puts $fileid $event
        puts $fileid $color
        puts $fileid $descrip
        puts "Data written to file $filename"
        close $fileid
    }]
}

#Cancel procedure, invoked when user clicks on the Cancel button.
proc cancel { w } {
    puts "Procedure cancel called "
```

APPENDIX B. SOURCE CODE

```
    upvar #0 point_Obj po
    upvar #0 infinite_Obj ie
    upvar #0 finitel_Obj fl
    upvar #0 finiter_Obj fr
    upvar #0 finite_Obj f
    global count countIL countFL
    global countFR countFE
    global eventNameEntered
    global user_filename
    $w.entry.event_entry delete 0 end
    $w.entry.fact_entry delete 0 end
    set color #2230f0
    $w.c delete box1 tab tab1 tab2 tab3 tab4 active active1 \
active2 active3 active4 box fend1 fixed

    #Copy the $user_filename event_details.temp

    if {[file exists $user_filename]} {
        file copy -force $user_filename event_details.temp
    } else {
        file delete event_details.temp
    }

    # Unsetting the arrays
    if {[array exists po]} { unset po }
    if {[array exists ie]} { unset ie }
    if {[array exists fl]} { unset fl }
    if {[array exists fr]} { unset fr }
    if {[array exists f]} { unset f }

    # Reinitializing the variables used arrays
    set count -1
    set countIL -1
    set countFL -1
    set countFR -1
    set countFE -1

    update
    focus $w.entry.event_entry
}

#Procedure invoked when user clicks on the "Continue" button.
proc continue_proc {w filename color} {
    puts "Procedure continue called "
    upvar #0 point_Obj po
    upvar #0 infinite_Obj ie
    upvar #0 finiter_Obj fr
    upvar #0 finitel_Obj fl
    upvar #0 finite_Obj f

    global event_details.temp
    global user_filename nfname
    global count countIL countFL
    global countFR countFE
    global numFE
    set numFE $countFE
```

APPENDIX B. SOURCE CODE

```
$w.entry.event_entry delete 0 end
$w.entry.fact_entry delete 0 end
$w.c delete box1 tab tab1 tab2 tab3 tab4 active active1 \
active2 active3 active4 active4 box fend1 fixed

# Copy the file event_details.temp to $user_filename

file copy -force event_details.temp $user_filename

# Writing the details of the events to the file.
# Information related to point objects will be stored in
# point.dat and rest will be in integral.dat

set color [string trimleft "$color" "#"]
set color c$color

set fileid [open point.dat "a+"]

if {[array exists po]} {
    foreach index [array names po] {
        set val $po($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
            set input "point($val,$color,1)."
            puts $fileid $input
        }
    }
    set input ""
    puts $fileid $input
    puts "Data written to file point.dat"
    close $fileid
}

set fileid [open integral.dat "a+"]

if {[array exists ie]} {
    foreach index [array names ie] {
        set val $ie($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
            set input "integral(0,1200,$color,1200)."
            puts $fileid $input
        }
    }
}

if {[array exists fr]} {
    foreach index [array names fr] {
        set val $fr($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
        }
    }
}
```

APPENDIX B. SOURCE CODE

```
        set input "integral(0,$val,$color,$val)."  
        puts $fileid $input  
    }  
}  
  
if {[array exists fl]} {  
    foreach index [array names fl] {  
        set val $fl($index)  
        if {$val == -1} {  
            continue  
        } else {  
            set val [expr int($val)*100]  
            set diff [expr 1200 - $val]  
            set input "integral($val,1200,$color,$diff)."  
            puts $fileid $input  
        }  
    }  
}  
  
if {[array exists f]} {  
    for {set index1 0} {$index1<=$countFE} {incr index1} {  
        set vall $f($index1,1)  
        set vall [expr int($vall)*100]  
  
        set val2 $f($index1,2)  
        set val2 [expr int($val2)*100]  
        set diff [expr $val2-$vall]  
        if {$vall != -1} {  
            set input "integral($vall,$val2,$color,$diff)."  
            puts $input  
            puts $fileid $input  
        }  
    }  
}  
  
    set input ""  
puts $fileid $input  
    puts "Data written to file integral.dat"  
close $fileid  
set color #2230f0  
  
# Unsetting the arrays  
if {[array exists po]} { unset po }  
if {[array exists ie]} { unset ie }  
if {[array exists fl]} { unset fl }  
if {[array exists fr]} { unset fr }  
if {[array exists f]} { unset f }  
  
# Reinitializing the variables used in various arrays  
set count -1  
    set countIL -1  
set countFL -1  
set countFR -1  
set countFE -1
```

APPENDIX B. SOURCE CODE

```
update
focus $w.entry.event_entry
if {$nfname != ""} {
    file copy -force event_details.dat $nfname
}
}

#Procedure invoked when the user clicks on the "Done" button.
proc done {w filename color} {
    puts "Procedure done called "
    upvar #0 point_Obj po
    upvar #0 infinite_Obj ie
    upvar #0 finiter_Obj fr
    upvar #0 finitel_Obj fl
    upvar #0 finite_Obj f

    global event_details.temp
    global user_filename nfname
    global count countIL countFL
    global countFR countFE
    global numFE
    set numFE $countFE
    # Copy the file event_details.temp to $user_filename
    if {[file exists event_details.temp]} {
        file copy -force event_details.temp $user_filename
    }

    # Writing the details to the file
    # Information related to point objects will be stored in
    # point.dat and rest will be in integral.dat

    set color [string trimleft "$color" "#"]
    set color c$color

    set fileid [open point.dat "a+"]

    if {[array exists po]} {
        foreach index [array names po] {
            set val $po($index)
            if {$val == -1} {
                continue
            } else {
                set val [expr int($val)*100]
                set input "point($val,$color,1)."
                puts $fileid $input
            }
        }
        set input ""
        puts $fileid $input
        puts "Data written to file point.dat"
        close $fileid
    }

    set fileid [open integral.dat "a+"]

    if {[array exists ie]} {
```

APPENDIX B. SOURCE CODE

```
        foreach index [array names ie] {
            set val $ie($index)
            if {$val == -1} {
                continue
            } else {
                set val [expr int($val)*100]
                set input "integral(0,1200,$color,1200)."
```

APPENDIX B. SOURCE CODE

```
    puts "Data written to file integral.dat"
close $fileid

set color #2230f0

# Unsetting the arrays
if {[array exists po]} { unset po }
if {[array exists ie]} { unset ie }
if {[array exists fl]} { unset fl }
if {[array exists fr]} { unset fr }
if {[array exists f]} { unset f }

# Reinitializing the variables used in various arrays
set count -1
set countIL -1
set countFL -1
set countFR -1
set countFE -1

if {$nfname != ""} {
    file copy -force event_details.dat $nfname
}

destroy $w
}

canvas $c -width 17c -height 6c
pack $w.c -side top -fill x

# Create a font to show the text on the ruler.

font create txtft2 -family Courier -size 12 -slant italic \
    -weight bold
set newf txtft2

#Info about the box

set demo_boxInfo(a) 0
set demo_boxInfo(motionProc) MoveNull
if {[wininfo depth $c] > 1} {
    set demo_boxInfo(boxStyle) "-fill {} -outline \
black -width 1"
    set demo_boxInfo(active) "-fill red \
-outline black -width 1"
} else {
    set demo_boxInfo(boxStyle) "-fill {} -outline \
black -width 1"
    set demo_boxInfo(active) "-fill black \
-outline black -width 1"
}

#Info about the ruler

set demo_rulerInfo(grid) .25c
set demo_rulerInfo(left) [wininfo fpixels $c 1c]
```


APPENDIX B. SOURCE CODE

```
set demo_rulerInfo(right) [winfo fpixels $c 13c]
set demo_rulerInfo(top) [winfo fpixels $c 1c]
set demo_rulerInfo(bottom) [winfo fpixels $c 1.75c]
set demo_rulerInfo(size) [winfo fpixels $c .4c]
set demo_rulerInfo(normalStyle) "--fill $color"
puts "color is $color"
if {[winfo depth $c] > 1} {
    set demo_rulerInfo(activeStyle) "--fill red -stipple {}"
    set demo_rulerInfo(deleteStyle) [list -fill red \
        -stipple @[file join $tk_library demos images gray25.bmp]]
} else {
    set demo_rulerInfo(activeStyle) "--fill black -stipple {}"
    set demo_rulerInfo(deleteStyle) [list -fill black \
        -stipple @[file join $tk_library demos images gray25.bmp]]
}

$c create line 1c 0.5c 1c 1c 13c 1c 13c 0.5c -width 2
$c create text .5c .75c -text "AM" -font $newf -anchor s -fill maroon
for {set i 0} {$i < 12} {incr i} {
    set x [expr $i+1]
    if {$i==0} {
        $c create line ${x}c 1c ${x}c 0.6c -width 1
        $c create line $x.25c 1c $x.25c 0.8c -width 1
        $c create line $x.5c 1c $x.5c 0.7c -width 1
        $c create line $x.75c 1c $x.75c 0.8c -width 1
        $c create text $x.15c .75c -text "12" -font $newf -anchor s \
            -fill maroon
    } else {
        $c create line ${x}c 1c ${x}c 0.6c -width 1
        $c create line $x.25c 1c $x.25c 0.8c -width 1
        $c create line $x.5c 1c $x.5c 0.7c -width 1
        $c create line $x.75c 1c $x.75c 0.8c -width 1
        $c create text $x.15c .75c -text $i -font $newf -anchor s \
            -fill maroon
    }
}
# prints the last 12 on the scale
set x [expr $x+1]
$c create text $x.17c .75c -text $i -font $newf -anchor s -fill maroon
$c create text [expr $x+1]c .75c -text "PM" -anchor s -font $newf \
    -fill maroon

# Tags below are for the circular object
$c addtag well withtag [$c create rect 2c 4.5c 3c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well withtag [rulerMkTab $c [winfo pixels $c 2.5c] \
    [winfo pixels $c 4c]]
$c create text 2.25c 5.25c -text "Point" -font $newf -fill maroon
$c create text 2.25c 5.75c -text "Event" -font $newf -fill maroon

# Tags below are for the Infinite Line object
$c addtag well1 withtag [$c create rect 4.5c 4.5c 5.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well1 withtag [rulerMkTabIL $c [winfo pixels $c 4.5c] \
    [winfo pixels $c 4.15c]]
```

APPENDIX B. SOURCE CODE

```
$c create text 4.75c 5.25c -text "Limitless" -font $newf -fill maroon
$c create text 4.75c 5.75c -text "Event" -font $newf -fill maroon
```

```
# Tags below are for the Fixed End object
$c addtag well2 withtag [$c create rect 7c 4.5c 8c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well2 withtag [rulerMkTabFE $c [wininfo pixels $c 7.02c] \
    [wininfo pixels $c 4.15c]]
$c create text 7.5c 5.25c -text "Fixed" -font $newf -fill maroon
$c create text 7.5c 5.75c -text "Event" -font $newf -fill maroon
```

```
# Tags below are for the Fixed left object
$c addtag well3 withtag [$c create rect 9.5c 4.5c 10.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well3 withtag [rulerMkTabFL $c [wininfo pixels $c 9.52c] \
    [wininfo pixels $c 4.15c]]
$c create text 10c 5.25c -text "FixedLeft" -fill maroon -font $newf
$c create text 10c 5.75c -text "Event" -fill maroon -font $newf
```

```
# Tags below are for the Fixed right object
$c addtag well4 withtag [$c create rect 12c 4.5c 13c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well4 withtag [rulerMkTabFR $c [wininfo pixels $c 12.02c] \
    [wininfo pixels $c 4.15c]]
$c create text 13c 5.25c -text "FixedRight" -fill maroon -font $newf
$c create text 12.5c 5.75c -text "Event" -fill maroon -font $newf
```

```
#Bindings below are for the Infinite Line object
$c bind well1 <1> "rulerNewTabIL $c %x %y"
$c bind tab1 <1> "rulerSelectTabIL $c %x %y"
$c bind well1 <Bl-Motion> "rulerMoveTabIL $c %x %y"
$c bind well1 <Any-ButtonRelease-1> "rulerReleaseTabIL $c"
```

```
#Bindings below are for the Point object
$c bind well <1> "rulerNewTab $c %x %y"
$c bind tab <1> "rulerSelectTab $c %x %y"
$c bind well <Bl-Motion> "rulerMoveTab $c %x %y"
$c bind well <Any-ButtonRelease-1> "rulerReleaseTab $c"
```

```
#Bindings below are for the Fixed End object
$c bind well2 <1> "rulerNewTabFE $c %x %y"
$c bind tab2 <1> "rulerSelectTabFE $c %x %y"
$c bind well2 <Bl-Motion> "rulerMoveTabFE $c %x %y"
$c bind well2 <Any-ButtonRelease-1> "rulerReleaseTabFE $c"
```

```
#Bindings below are for the Fixed Left object
$c bind well3 <1> "rulerNewTabFL $c %x %y"
$c bind tab3 <1> "rulerSelectTabFL $c %x %y"
$c bind well3 <Bl-Motion> "rulerMoveTabFL $c %x %y"
$c bind well3 <Any-ButtonRelease-1> "rulerReleaseTabFL $c"
```

```
#Bindings below are for the Fixed Right object
$c bind well4 <1> "rulerNewTabFR $c %x %y"
```

APPENDIX B. SOURCE CODE

```
$c bind tab4 <1> "rulerSelectTabFR $c %x %y"
$c bind well4 <B1-Motion> "rulerMoveTabFR $c %x %y"
$c bind well4 <Any-ButtonRelease-1> "rulerReleaseTabFR $c"

font delete txtft2
source Bruler.tcl
#!/usr/local/bin/wish

#Procedures controlling the behaviour of various objects

#----- OBJECT POINT -----

# Position gives the position of the point objects on the ruler;
# count helps in keeping a count of the number of point objects
# test is a boolean variable, to check if a new object is
# created or not, initially set to false (0)

set Position -1
set count -1
set test 0

# rulerNewTab --
# Does all the work of creating a tab stop, including creating the
# point object and adding tags to it to give it tab behavior.
#
# Arguments:
# c - The canvas window.
# x, y - The coordinates of the tab stop.

proc rulerNewTab {c x y} {
    global count
    global test
    incr count
    set test 1

    upvar #0 demo_rulerInfo v
    $c addtag active withtag [rulerMkTab $c $x $y]
    $c addtag tab withtag active
    set v(x) $x
    set v(y) $y
    rulerMoveTab $c $x $y
}

# rulerSelectTab --
# This procedure is invoked when mouse button 1 is pressed over
# a tab. It remembers information about the tab so that it can
# be dragged interactively.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse (identifies the point by
# which the tab was picked up for dragging).

proc rulerSelectTab {c x y} {
    global test
```

APPENDIX B. SOURCE CODE

```
global Position
set test 0

upvar #0 demo_rulerInfo v
set v(x) [$c canvasx $x $v(grid)]
set v(y) [expr $v(top)+2]
$c addtag selected closest $v(x) $v(y)
set unit 35.457
set tmp [expr int((( $v(x)/$unit)-1)*100)]
set Position [expr $tmp/100.00]

$c addtag active withtag current
eval "$c itemconf active $v(activeStyle)"
$c raise active
$c bind tab <Bl-Motion> "rulerMoveTab $c %x %y"
$c bind tab <Any-ButtonRelease-1> "rulerReleaseTab $c"
}

# rulerMoveTab --
# This procedure is invoked during mouse motion events to drag a tab.
# It adjusts the position of the tab, and changes its appearance if
# it is about to be dragged out of the ruler.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse.

proc rulerMoveTab {c x y} {
    upvar #0 demo_rulerInfo v
    if {[ $c find withtag active] == ""} {
        return
    }
    set cx [$c canvasx $x $v(grid)]
    set cy [$c canvasy $y]
    if {$cx < $v(left)} {
        set cx $v(left)
    }
    if {$cx > $v(right)} {
        set cx $v(right)
    }
    if {($cy >= $v(top)) && ($cy <= $v(bottom))} {
        set cy [expr $v(top)+2]
        eval "$c itemconf active $v(activeStyle)"
    } else {
        set cy [expr $cy-$v(size) -2]
        eval "$c itemconf active $v(deleteStyle)"
    }
    $c move active [expr $cx-$v(x)] [expr $cy-$v(y)]
    set v(x) $cx
    set v(y) $cy
}

# rulerReleaseTab --
# This procedure is invoked during button release events that end
# a tab drag operation. It deselects the tab and deletes the tab if
```

APPENDIX B. SOURCE CODE

```
# it was dragged out of the ruler.
#
# Arguments:
# c -      The canvas widget.
# x, y -   The coordinates of the mouse.

proc rulerReleaseTab c {
    upvar #0 demo_rulerInfo v
    upvar #0 point_Obj p

    global count
    global test
    global Position
    global color

    if {[${c} find withtag active] == {}} {
        return
    }
    if {$v(y) != [expr $v(top)+2]} {
        ${c} delete active
        if {$test != "1"} {
            foreach index [array names p] {
                if {$p($index) == $Position} {
                    set vall $index
                }
            }
            set p($vall) -1
            incr count -1
        }
    } else {
        eval "${c} itemconf active -fill $color"

        # xvalue will be shown only if the event is on
        # the ruler line and not otherwise.

        set unit 35.457
        set tmp [expr int(((v(x)/$unit)-1)*100)]
        set xvalue [expr $tmp/100.00]
        if {$test == "1"} {
            set p($count) $xvalue
        } elseif { $test == "0"} {
            foreach index [array names p] {
                if {$p($index) == $Position} {
                    set val $index
                }
            }
            set p($val) $xvalue
        }
        ${c} dtag active
    }
}

#----- OBJECT-IL-----

set countIL -1
```

APPENDIX B. SOURCE CODE

```
# rulerNewTabIL --
# Does all the work of creating a tab stop, including creating the
# Infinite line object and adding tags to it to give it a tab behavior.
#
# Arguments:
# c -           The canvas window.
# x, y -       The coordinates of the tab stop.

proc rulerNewTabIL {c x y} {
    global countIL
    incr countIL
    upvar #0 demo_rulerInfo v
    $c addtag activel withtag [rulerMkTabIL $c $x $y]
    $c addtag tabl withtag activel
    set v(x) $x
    set v(y) $y
    rulerMoveTabIL $c $x $y
}

# rulerSelectTabIL --
# This procedure is invoked when mouse button 1 is pressed over
# a tab. It remembers information about the tab so that it can
# be dragged interactively.
#
# Arguments:
# c -           The canvas widget.
# x, y -       The coordinates of the mouse (identifies the point by
#              which the tab was picked up for dragging).

proc rulerSelectTabIL {c x y} {
    upvar #0 demo_rulerInfo v
    set v(x) [$c canvasx $x $v(grid)]
    set v(y) [expr $v(top)+2]
    $c addtag activel withtag current
    eval "$c itemconf activel $v(activeStyle)"
    $c raise activel
    $c bind tabl <B1-Motion> "rulerMoveTabIL $c %x %y"
    $c bind tabl <Any-ButtonRelease-1> "rulerReleaseTabIL $c"
}

# rulerMoveTabIL --
# This procedure is invoked during mouse motion events to drag the IL.
# It adjusts the position of the IL, and changes its appearance if
# it is about to be dragged out of the ruler.
#
# Arguments:
# c -           The canvas widget.
# x, y -       The coordinates of the mouse.

proc rulerMoveTabIL {c x y} {
    upvar #0 demo_rulerInfo v
    if {[[$c find withtag activel] == ""]} {
        return
    }
}
```

APPENDIX B. SOURCE CODE

```
}
set cx [$c canvasx $x $v(grid)]
set cy [$c canvasy $y]
if {$cx < $v(left)} {
    set cx $v(left)
}
if {$cx > $v(right)} {
    set cx $v(right)
}
if (($cy >= $v(top)) && ($cy <= $v(bottom))) {
    set cy [expr $v(top)+2]
    eval "$c itemconf activel $v(activeStyle)"
} else {
    set cy [expr $cy-$v(size)-2]
    eval "$c itemconf activel $v(deleteStyle)"
}
$c move activel [expr $cx-$v(x)] [expr $cy-$v(y)]
set v(x) $cx
set v(y) $cy
}

# rulerReleaseTabIL --
# This procedure is invoked during button release events that end
# a IL drag operation. It deselects the IL and deletes the IL if
# it was dragged out of the ruler.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse.

proc rulerReleaseTabIL c {
    upvar #0 demo_rulerInfo v
    upvar #0 infinite_Obj ie
    global countIL
    if {[ $c find withtag activel ] == {}} {
        return
    }
    if {$v(y) != [expr $v(top)+2]} {
        $c delete activel
        incr countIL -1
    } else {
        $c delete activel
        $c addtag tabl withtag [rulerMkTabIL2 $c \
            [winfo pixels $c lc] [winfo pixels $c lc]]

        set unit 35.457
        set tmp [expr int((( $v(left) )/$unit)-1)*100)]
        set left [expr $tmp/100.00]

        set tmp [expr int((( $v(right) )/$unit)-1)*100)]
        set right [expr $tmp/100.00]

        set ie($countIL) 1
        $c bind tabl <1> "rulerSelectTabIL2 $c %x %y"
        $c dtag activel
    }
}
```

APPENDIX B. SOURCE CODE

```
}

proc rulerSelectTabIL2 {c x y} {
    upvar #0 demo_rulerInfo v
    $c addtag active1 withtag current
    eval "$c itemconf active1 $v(activeStyle)"
    $c raise active1
    $c bind tab1 <Bl-Motion> "rulerMoveTabIL $c %x %y"
    $c bind tab1 <Any-ButtonRelease-1> "rulerReleaseTabIL2 $c"
}

proc rulerReleaseTabIL2 c {
    upvar #0 demo_rulerInfo v
    upvar #0 infinite_Obj ie
    global countIL
    if {[ $c find withtag active1 ] == {}} {
        return
    }
    if { $v(y) != [expr $v(top)+2] } {
        $c delete active1
        set ie($countIL) -1
        incr countIL -1
    } else {
        eval "$c itemconf active1 $v(normalStyle)"
        $c dtag active1
    }
}

#----- OBJECT-FE -----

# Setting up a counter
set countFE -1

# rulerNewTabFE --
# Does all the work of creating a tab stop, including creating the
# Finite End object and adding tags to it to give it a tab behavior.
#
# Arguments:
# c -          The canvas window.
# x, y -      The coordinates of the tab stop.

proc rulerNewTabFE {c x y} {
    upvar #0 demo_rulerInfo v
    upvar #0 demo_boxInfo w
    global countFE
    incr countFE
    $c dtag active2
    $c dtag tab2
    $c dtag box1
    set w(a) 0
    $c addtag active2 withtag [rulerMkTabFE $c $x $y]
    $c addtag tab2 withtag active2
    set v(x) $x
    set v(y) $y
    rulerMoveTabFE $c $x $y
}

}
```


APPENDIX B. SOURCE CODE

```
# rulerSelectTabFE --
# This procedure is invoked when mouse button 1 is pressed over
# a tab. It remembers information about the tab so that it can
# be dragged interactively.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse (identifies the point by
# which the tab was picked up for dragging).

proc rulerSelectTabFE {c x y} {
    upvar #0 demo_rulerInfo v
    set v(x) [$c canvasx $x $v(grid)]
    set v(y) [expr $v(top)+2]

    $c addtag selectedFE closest $v(x) $v(y)
    set unit 35.457
    set tmp [expr int((($v(x)/$unit)-1)*100)]
    set PositionFE [expr $tmp/100.00]
    set tags [$c gettags current]

    $c addtag active2 withtag current
    $c addtag tab2 withtag active2
    eval "$c itemconf active2 $v(activeStyle)"
    $c raise active2
    $c bind tab2 <B1-Motion> "rulerMoveTabFE $c %x %y"
    $c bind tab2 <Any-ButtonRelease-1> "rulerReleaseTabFE $c"
}

# rulerMoveTabFE --
# This procedure is invoked during mouse motion events to drag the FE.
# It adjusts the position of the FE, and changes its appearance if
# it is about to be dragged out of the ruler.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse.

proc rulerMoveTabFE {c x y} {
    upvar #0 demo_rulerInfo v
    if {[ $c find withtag active2] == ""} {
        return
    }
    set cx [$c canvasx $x $v(grid)]
    set cy [$c canvasy $y]
    if {$cx < $v(left)} {
        set cx $v(left)
    }
    if {$cx > $v(right)} {
        set cx $v(right)
    }
    if (($cy >= $v(top)) && ($cy <= $v(bottom))) {
        set cy [expr $v(top)+2]
    }
}
```

APPENDIX B. SOURCE CODE

```
    eval "$c itemconf active2 $v(activeStyle)"
  } else {
    set cy [expr $cy-$v(size)-2]
    eval "$c itemconf active2 $v(deleteStyle)"
  }
  $c move active2 [expr $cx-$v(x)] [expr $cy-$v(y)]
  set v(x) $cx
  set v(y) $cy
}

# rulerReleaseTabFE --
# This procedure is invoked during button release events that end
# a FE drag operation. It deselects the FE and deletes the FE if
# it was dragged out of the ruler.
#
# Arguments:
# c - The canvas widget.
# x, y - The coordinates of the mouse.

proc rulerReleaseTabFE c {
  upvar #0 demo_rulerInfo v
  upvar #0 finite_Obj f
  global countFE
  if {[ $c find withtag active2] == {}} {
    return
  }
  if {$v(y) != [expr $v(top)+2]} {
    $c delete active2 tab2 box1
    set f($countFE,1) -1
    set f($countFE,2) -1
    incr countFE -1
  } else {
    eval "$c itemconf active2 $v(normalStyle)"
    source box.tcl
    $c bind tab2 <1> "rulerSelectTabFE2 $c %x %y"
  }
}

proc rulerSelectTabFE2 {c x y} {
  upvar #0 demo_rulerInfo v
  #Capturing the object closest to mouse cursor
  set vx [ $c canvasx $x $v(grid)]
  set vy [expr $v(top)+2]
  $c addtag fendl closest $vx $vy
  set unit 35.457
  set tmp [expr int((( $vx/$unit)-1)*100)]
  set positionFE [expr $tmp/100.00]
  set tags [ $c gettags current]

  $c addtag active2 withtag current
  $c addtag tab2 withtag active2
  eval "$c itemconf active2 $v(activeStyle)"
  $c raise active2
  $c bind tab2 <B1-Motion> "rulerMoveTabFE $c %x %y"
```

APPENDIX B. SOURCE CODE

```
        $c bind tab2 <Any-ButtonRelease-1> "rulerReleaseTabFE $c"
    }

source Cruler.tcl
#!/usr/local/bin/wish

#----- OBJECT-FL -----

# CountFL is used to keep a count of the # of objects created
# for Fixed Left event
set countFL -1

# rulerNewTabFL
# Does all the work of creating a tab stop, including creating the
# Fixed Left object and adding tags to it to give it a tab behavior.
#
# Arguments:
# c -          The canvas window.
# x, y -       The coordinates of the tab stop.

proc rulerNewTabFL {c x y} {
    upvar #0 demo_rulerInfo v
    global countFL
    incr countFL
    $c addtag active3 withtag [rulerMkTabFL $c $x $y]
    $c addtag tab3 withtag active3
    set v(x) $x
    set v(y) $y
    rulerMoveTabFL $c $x $y
}

# rulerSelectTabFL --
# This procedure is invoked when mouse button 1 is pressed over
# a tab. It remembers information about the tab so that it can
# be dragged interactively.
#
# Arguments:
# c -          The canvas widget.
# x, y -       The coordinates of the mouse (identifies the point by
#              which the tab was picked up for dragging).

proc rulerSelectTabFL {c x y} {
    upvar #0 demo_rulerInfo v
    set v(x) [$c canvasx $x $v(grid)]
    set v(y) [expr $v(top)+2]
    $c addtag active3 withtag current
    eval "$c itemconf active3 $v(activeStyle)"
    $c raise active3
    $c bind tab3 <B1-Motion> "rulerMoveTabFL $c %x %y"
    $c bind tab3 <Any-ButtonRelease-1> "rulerReleaseTabFL $c"
}
}
```

APPENDIX B. SOURCE CODE

```
# rulerMoveTabFL --
# This procedure is invoked during mouse motion events to drag the FL.
# It adjusts the position of the FL, and changes its appearance if
# it is about to be dragged out of the ruler.
#
# Arguments:
# c -      The canvas widget.
# x, y -   The coordinates of the mouse.

proc rulerMoveTabFL {c x y} {
    upvar #0 demo_rulerInfo v
    if {[${c} find withtag active3] == ""} {
        return
    }
    set cx [${c} canvasx $x $v(grid)]
    set cy [${c} canvasy $y]
    if {$cx < $v(left)} {
        set cx $v(left)
    }
    if {$cx > $v(right)} {
        set cx $v(right)
    }
    if {($cy >= $v(top)) && ($cy <= $v(bottom))} {
        set cy [expr $v(top)+2]
        eval "${c} itemconf active3 $v(activeStyle)"
    } else {
        set cy [expr $cy-$v(size)-2]
        eval "${c} itemconf active3 $v(deleteStyle)"
    }
    ${c} move active3 [expr $cx-$v(x)] [expr $cy-$v(y)]
    set v(x) $cx
    set v(y) $cy
}

# rulerReleaseTabFL --
# This procedure is invoked during button release events that end
# a FL drag operation. It deselects the FL and deletes the FL if
# it was dragged out of the ruler.
#
# Arguments:
# c -      The canvas widget.
# x, y -   The coordinates of the mouse.

proc rulerReleaseTabFL c {
    upvar #0 demo_rulerInfo v
    upvar #0 finitel_Obj fl
    global countFL
    if {[${c} find withtag active3] == {}} {
        return
    }
    if {$v(y) != [expr $v(top)+2]} {
        ${c} delete active3
        incr countFL -1
    } else {
        set unit 35.457
    }
}
```

APPENDIX B. SOURCE CODE

```
    set tmp [expr int((( $v(x)$ )/$unit)-1)*100]
    set xvalue [expr $tmp/100.00]
    set fl($countFL) $xvalue

    $c delete active3
    $c addtag tab3 withtag [rulerMkTabFL2 $c \
    $v(x) [wininfo pixels $c lc]]
    $c bind tab3 <1> "rulerSelectTabFL2 $c %x %y"
    $c dtag active3
  }
}

proc rulerSelectTabFL2 {c x y} {
  upvar #0 demo_rulerInfo v
  puts "Invoked from proc rulerSelectTabFL2"
  global position
  set v(x) [$c canvasx $x $v(grid)]
  set v(y) [expr $v(top)+2]

  $c addtag selected closest $v(x) $v(y)

  set unit 35.457
  set tmp [expr int((( $v(x)$ )/$unit)-1)*100]
  set position [expr $tmp/100.00]

  $c addtag active3 withtag current
  eval "$c itemconf active3 $v(activeStyle)"
  $c raise active3
  $c bind tab3 <B1-Motion> "rulerMoveTabFL $c %x %y"
  $c bind tab3 <Any-ButtonRelease-1> "rulerReleaseTabFL2 $c"
  set unit 35.457
  set tmp [expr int((( $v(x)$ )/$unit)-1)*100]
  set xvalue [expr $tmp/100.00]
}

proc rulerReleaseTabFL2 c {
  upvar #0 demo_rulerInfo v
  upvar #0 finitel_Obj fl
  global countFL
  global position
  if {[ $c$  find withtag active3] == {}} {
    return
  }
  if { $v(y)$  != [expr $v(top)+2]} {
    $c delete active3
    foreach index [array names fl] {
      if { $fl($index) == $position} {
        set vall $index
      }
    }
    set fl($vall) -1
    incr countFL -1
  } else {
    set unit 35.457
  }
}
```

APPENDIX B. SOURCE CODE

```
        set tmp [expr int(((v(x)/$unit)-1)*100)]
        set xvalue [expr $tmp/100.00]
        foreach index [array names fl] {
            if { $fl($index) == $position} {
                set val $index
            }
        }
        set fl($val) $xvalue

    $c delete active3
    $c addtag tab3 withtag [rulerMkTabFL2 $c \
        $v(x) [wininfo pixels $c lc]]
    $c bind tab3 <1> "rulerSelectTabFL2 $c %x %y"
    $c dtag active3
}
}

# ----- OBJECT-FR -----

# countFR is used to keep a count of the # of objects created
# for Fixed Right event
set countFR -1

# rulerNewTabFR --
# Does all the work of creating a tab stop, including creating the
# Fixed Right object and adding tags to it to give it a tab behavior.
#
# Arguments:
# c -           The canvas window.
# x, y -       The coordinates of the tab stop.

proc rulerNewTabFR {c x y} {
    upvar #0 demo_rulerInfo v
    global countFR
    incr countFR
    $c addtag active4 withtag [rulerMkTabFR $c $x $y]
    $c addtag tab4 withtag active4
    set v(x) $x
    set v(y) $y
    rulerMoveTabFR $c $x $y
}

# rulerSelectTabFR
# This procedure is invoked when mouse button 1 is pressed over
# a tab. It remembers information about the tab so that it can
# be dragged interactively.
#
# Arguments:
# c -           The canvas widget.
# x, y -       The coordinates of the mouse (identifies the point by
# which the tab was picked up for dragging).

proc rulerSelectTabFR {c x y} {
```

APPENDIX B. SOURCE CODE

```
    upvar #0 demo_rulerInfo v
    set v(x) [$c canvasx $x $v(grid)]
    set v(y) [expr $v(top)+2]
    $c addtag active4 withtag current
    eval "$c itemconf active4 $v(activeStyle)"
    $c raise active4
    $c bind tab4 <B1-Motion> "rulerMoveTabFR $c %x %y"
    $c bind tab4 <Any-ButtonRelease-1> "rulerReleaseTabFR $c"
}

# rulerMoveTabFR
# This procedure is invoked during mouse motion events to drag the FL.
# It adjusts the position of the FR, and changes its appearance if
# it is about to be dragged out of the ruler.
#
# Arguments:
# c -      The canvas widget.
# x, y -   The coordinates of the mouse.

proc rulerMoveTabFR (c x y) {
    upvar #0 demo_rulerInfo v
    if {[{$c find withtag active4] == ""} {
        return
    }
    set cx [$c canvasx $x $v(grid)]
    set cy [$c canvasy $y]
    if {$cx < $v(left)} {
        set cx $v(left)
    }
    if {$cx > $v(right)} {
        set cx $v(right)
    }
    if {($cy >= $v(top)) && ($cy <= $v(bottom))} {
        set cy [expr $v(top)+2]
        eval "$c itemconf active4 $v(activeStyle)"
    } else {
        set cy [expr $cy-$v(size)-2]
        eval "$c itemconf active4 $v(deleteStyle)"
    }
    $c move active4 [expr $cx-$v(x)] [expr $cy-$v(y)]
    set v(x) $cx
    set v(y) $cy
}

# rulerReleaseTabFR
# This procedure is invoked during button release events that end
# a FR drag operation. It deselects the FR and deletes the FR if
# it was dragged out of the ruler.
#
# Arguments:
# c -      The canvas widget.
# x, y -   The coordinates of the mouse.

proc rulerReleaseTabFR c {
```

APPENDIX B. SOURCE CODE

```
    upvar #0 demo_rulerInfo v
#
#Value nv adds [winfo fpixels $c lc] to v(x)
#
    set nv [expr $v(x)+35.457]
    upvar #0 finiter_Obj fr
    global countFR
    if {[{$c find withtag active4} == {}] {
        return
    }
    if {$v(y) != [expr $v(top)+2]} {
        $c delete active4
        incr countFR -1
    } else {
        set unit 35.457
        set tmp [expr int(($v(x)/$unit)*100)]
        set xvalue [expr $tmp/100.00]
        set fr($countFR) $xvalue

        $c delete active4
        $c addtag tab4 withtag [rulerMkTabFR2 $c \
            $nv [winfo pixels $c lc]]
        $c bind tab4 <1> "rulerSelectTabFR2 $c %x %y"
        $c dtag active4
    }
}

proc rulerSelectTabFR2 {c x y} {
    upvar #0 demo_rulerInfo v
    global positionFR
    set v(x) [$c canvasx $x $v(grid)]
    set v(y) [expr $v(top)+2]

    $c addtag selected closest $v(x) $v(y)

    set unit 35.457
    set tmp [expr int(($v(x)/$unit)*100)]
    set positionFR [expr $tmp/100.00]
    set positionFR [expr $positionFR - 1]

    $c addtag active4 withtag current
    eval "$c itemconf active4 $v(activeStyle)"
    $c raise active4
    $c bind tab4 <Bl-Motion> "rulerMoveTabFR $c %x %y"
    $c bind tab4 <Any-ButtonRelease-1> "rulerReleaseTabFR2 $c"
}

proc rulerReleaseTabFR2 c {
    upvar #0 demo_rulerInfo v
    upvar #0 finiter_Obj fr
    global countFR
    global positionFR
    if {[{$c find withtag active4} == {}] {
        return
    }
    if {$v(y) != [expr $v(top)+2]} {
```


APPENDIX B. SOURCE CODE

```

    $c delete active4
    foreach index [array names fr] {
        if { $fr($index) == $positionFR} {
            set vall $index
        }
    }
    set fr($vall) -1
    incr countFR -1
} else {
    set unit 35.457
    set tmp [expr int(($v(x)/$unit)*100)]
    set xvalue [expr $tmp/100.00]
    set xvalue [expr $xvalue - 1]
    foreach index [array names fr] {
        if { $fr($index) == $positionFR} {
            set val $index
        }
    }
    set fr($val) $xvalue

    $c delete active4
    $c addtag tab4 withtag [rulerMkTabFR2 $c \
    $v(x) [winfo pixels $c 1c]]
    $c bind tab4 <1> "rulerSelectTabFR2 $c %x %y"
    $c dtag active4
}
}

#!/usr/local/bin/wish

# This file creates the query screen and its underlying functionality.
#-----
#                               Query.tcl
#-----

# proc positionWindow to sets the position of the Window on the screen.

proc positionWindow w {
    wm geometry $w +300+300
}

#Font variable
set font {Helvetica 14}

#Default color for the widget items

set color #2230f0

#begin
# rulerMkTab --
# This procedure creates a new circular polygon in a canvas to
# represent a point event.
#
```

APPENDIX B. SOURCE CODE

```
# Arguments:
# c -      The canvas window.
# x, y -   Coordinates at which to create the tab stop.
#
# Arguments:
# w -      The name of the window to position.

proc rulerMkTab {c x y} {
    global color
    puts "Color for point object is $color"
    set vl [wininfo fpixels $c .3c]
    $c create oval [expr $x-$vl/2] [expr $y-$vl/2] [expr $x+$vl/2] \
        [expr $y+$vl/2] -fill black
}

# rulerMkTabIL --
# This procedure creates a new Infinite line item in a canvas.
#
# Arguments:
# c -      The canvas window.
# x, y -   Coordinates at which to create the IL item.
#
# Arguments:
# w -      The name of the window to position.

proc rulerMkTabIL {c x y} {
    set vl [wininfo fpixels $c 1c]
    $c create line $x $y [expr $x+$vl] $y \
        -arrow both -fill black -width 8
}

# rulerMkTabIL2 --
# This procedure creates a new Infinite line item in a canvas.
# Behaves similar to the above but covers the entire length.

proc rulerMkTabIL2 {c x y} {
    global color
    set vl [wininfo fpixels $c 12c]
    $c create line $x $y [expr $x+$vl] $y -arrow both -fill $color \
        -width 8
}

# rulerMkTabFE --
# This procedure creates a new Fixed End line item in a canvas.
#
# Arguments:
# c -      The canvas window.
# x, y -   Coordinates at which to create the FE item.
#
# Arguments:
# w -      The name of the window to position.
# Here vl plays a role in positioning the item especially w.r.t y axis.

proc rulerMkTabFE {c x y} {
    set vl [wininfo fpixels $c 1c]
```

APPENDIX B. SOURCE CODE

```
    $c create line $x $y [expr $x+$v1] $y -fill black -width 8
}

# rulerMkTabFL
# This procedure creates a new Fixed Left item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#
# Arguments:
# w -           The name of the window to position.
# v1 plays a role in positioning the item w.r.t y axis.

proc rulerMkTabFL {c x y} {
    set v1 [wininfo fpixels $c lc]
    $c create line $x $y [expr $x+$v1] \
        $y -fill black -width 8 -arrow last
}

# rulerMkTabFL2
# This procedure creates a new Fixed left item in a canvas.
# Behaves similar to the above but covers the right length.

proc rulerMkTabFL2 {c x y} {
    upvar #0 demo_rulerInfo v
    global color
    set v1 [wininfo fpixels $c lc]
    $c create line $x $y $v(right) \
        $y -fill $color -width 8 -arrow last
}

# rulerMkTabFR
# This procedure creates a new Fixed Right item in a canvas.
#
# Arguments:
# c -           The canvas window.
# x, y -       Coordinates at which to create the FE item.
#
# Arguments:
# w -           The name of the window to position.
# v1 plays an important role in positioning the item.

proc rulerMkTabFR {c x y} {
    set v1 [wininfo fpixels $c lc]
    $c create line $x $y [expr $x+$v1] $y \
        -fill black -width 8 -arrow first
}

# rulerMkTabFR2
# This procedure creates a new Fixed Right item in a canvas.
# Behaves similar to the above but covers the left length.
```

APPENDIX B. SOURCE CODE

```
proc rulerMkTabFR2 { c x y } {
    upvar #0 demo_rulerInfo v
    global color
    set vl [wininfo fpixels $c lc]
    $c create line $v(left) $y $x $y -fill black -width 3 \
    -fill $color -width 3 -arrow first
}

# rulerMkTabQues
# This procedure creates a new Question item in the canvas.
# Clicking on this tells everything about the selected event

proc rulerMkTabQues { c x y } {
    $c create bitmap 12.00c 4.00c -bitmap questhead
}

# Sets up the window manager options.
set w .query_ruler
global w
global tk_library
catch {destroy $w}
toplevel $w
wm title $w "Querying the System"
wm iconname $w "QueryRuler"
positionWindow $w
set c $w.c

# Frame-1.

# Create a droplist for the events...
# drop_test_var is the global variable holding the value of
# the current selected event

global drop_test_var

proc DropListCreate {
    basename text width height variable initial_value } {

    upvar #0 $variable var
    set var "$initial_value"

    # Name of top-level widget to create.
    set top $basename.top

    #
    # Widgets to enter data.
    #
    frame $basename -bd 0
    label $basename.lbl -text $text -anchor e
    entry $basename.ent -width $width
    $basename.ent insert 0 "$initial_value"
    DropButton $basename.drop $basename.top $basename.ent

    bind $basename.ent <Return> \
        "DropListSetVal $basename.ent $variable"
```

APPENDIX B. SOURCE CODE

```
bind $basename.ent <Key-Escape> "wm withdraw $top"

pack $basename.lbl -side left -ipady 3 -pady 7
pack $basename.ent -side left -expand 1 -ipady 3 -pady 7
pack $basename.drop -side left -ipady 3 -pady 7

#
# Drop-list is a top-level temporary window.
#
toplevel $top -cursor top_left_arrow
wm overriddenirect $top 1
wm withdraw $top

# Create list
set frm $top.frame
frame $frm -bd 4 -relief sunken

listbox $frm.list -height $height -width $width \
    -selectmode single \
    -yscrollcommand "$frm.scrollbar set"

bind $frm.list <Key-Escape> "wm withdraw $top"

# Create scrollbar
scrollbar $frm.scrollbar \
    -command "$frm.list yview"

pack $frm.scrollbar -side right -fill y
pack $frm.list -side left
pack $frm -side top

bind $frm.list <ButtonRelease-1> \
    "DropListClick $top $basename.ent $variable"

pack $basename

#
# Return list widget so you can fill it.
#
return $frm.list
}

# Returns selected item for a single-select list.
proc list_selected { listname } {
    set indx [$listname curselection]

    if { $indx != "" } {
        set item [$listname get $indx]

        return $item
    } else {
        return ""
    }
}
```

APPENDIX B. SOURCE CODE

```
# Places value in global variable.
proc DropListSetVal { entry variable } {
    upvar #0 $variable var

    set value [$entry get]

    if { $value != "" } {
        set var $value
    }
}

# Handles click on drop list widget.
proc DropListClick { basename entry variable } {

    catch {
        set selected [list_selected $basename.frame.list]

        if { $selected != "" } {
            #
            # Put item into entry widget.
            #
            $entry delete 0 end
            $entry insert 0 "$selected"

            DropListSetVal $entry $variable
        }
    }

    wm withdraw $basename
}

# Makes drop list visible. Create with DropListCreate.
proc ShowDropList { basename associated_widget } {
    set x [winfo rootx $associated_widget]
    set y [winfo rooty $associated_widget]
    set y [expr $y + [winfo height $associated_widget]]

    wm geometry $basename "+$x+$y"

    wm deiconify $basename
    raise $basename

    focus $basename.frame.list
}

# Creates a button with a drop-down bitmap.
proc DropButton { name toplevel entry } {

    button $name -image darrow \
        -command "ShowDropList $toplevel $entry"
```

APPENDIX B. SOURCE CODE

```
    return $name
}

#
# Bitmap data for down arrow bitmap.
#
set darrow_data "
#define darrow2_width 18
#define darrow2_height 18
static unsigned char darrow2_bits[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x00, 0xf8, 0x7f,
0x00,
    0xf8, 0x7f, 0x00, 0xf0, 0x3f, 0x00, 0xf0, 0x3f, 0x00, 0xe0, 0x1f,
0x00,
    0xc0, 0x0f, 0x00, 0xc0, 0x0f, 0x00, 0x80, 0x07, 0x00, 0x80, 0x07,
0x00,
    0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff,
0x00,
    0xfc, 0xff, 0x00, 0x00, 0x00, 0x00};
"
image create bitmap darrow -data $darrow_data

set drop_test_var "Unset"
global drop_test_var

proc test_value { } {
    global drop_test_var
    global list_of_colors
    global list_of_events
    global color

    set index [lsearch $list_of_events $drop_test_var]
    set color_val [lindex $list_of_colors $index]
    puts "Color selected is $color_val"
    set color $color_val
    puts "Value of variable=$drop_test_var"
}

# Test procedure creating a drop-down list
proc test_drop_list { wl } {

    global drop_test_var
    global list_of_events
    global list_of_colors
    global list_of_descrip
    set filename event_details.dat

    set data ""
    set list_of_events ""
    set list_of_colors ""
    set list_of_descrip ""

    if {[file readable $filename]} {
        set fileid [open $filename "r"]
    }
}
```

APPENDIX B. SOURCE CODE

```
while {[eof $fileid] != 1} {
    gets $fileid data
    set list_of_events [linsert $list_of_events end $data]
    gets $fileid data
    set list_of_colors [linsert $list_of_colors end $data]
    gets $fileid data
    set list_of_descrip [linsert $list_of_descrip end $data]
}
close $fileid
}

#
# Determine initial value.
# Enclose in quotes because it may
# have spaces in the value.
#
set initial_value [lindex $list_of_events 0]

# Create drop list.
set list [DropListCreate $w1 "Select Event: " \
    40 8 drop_test_var "$initial_value"]

# Fill in drop list with events.
foreach event $list_of_events {
    $list insert end $event
}
button $w1.ok -text "OK" -command test_value
pack $w1.ok -side left -expand 1
}

# Comment out next line to remove test code.

set newframe $w.frameList
test_drop_list $newframe

# Frame-2.
# The following commands create the frame buttons.
frame $w.buttons
pack $w.buttons -side bottom -fill x -pady 2m
button $w.buttons.query -text " Query " -command { query }
button $w.buttons.new_query -text "New Query"\
    -command { new_query $w}
button $w.buttons.exit -text " Exit " -command { query_close $w }
button $w.buttons.help -text " Help " -command { source
queryHelp.tcl}
pack $w.buttons.query $w.buttons.new_query $w.buttons.exit \
    $w.buttons.help -side left -expand 1

#
# Procedure query_proc, which queries the eclipse depending
```


APPENDIX B. SOURCE CODE

```
# on the event selected by the user. It requires information
# about the event name, value on the time scale (obtained through
# upvar).
#

proc query { } {
    upvar #0 point_Obj po
    upvar #0 infinite_Obj ie
    upvar #0 finiter_Obj fr
        upvar #0 finitel_Obj fl
    upvar #0 finite_Obj f

    global color countFE
    global list_of_events
    global list_of_colors
    global drop_test_var
    global tk_library
    set numFE $countFE
    puts "No. of Fixed objects are $countFE"

    # Delete knowledge_base (older version) and create again
    # by joining integral.dat and point.dat

    file delete knowledge_base
    set command1 "cat integral.dat point.dat > knowledge_base"
    eval exec $command1

    #
    # Get the index of the event from list_of_events and pick
    # the corresponding color from list_of_colors
    #

    set index [lsearch $list_of_events $drop_test_var]
    set color_val [lindex $list_of_colors $index]
    set color $color_val
        puts "Color selected is $color_val"
        set color_val [string trimleft "$color_val" "#"]
    set color_val c$color_val

    #
    # The following lines of code open a channel and the
    # eclipse commands are executed from the tcl script
    #

    set pipe [open "|eclipse" w+]
    fconfigure $pipe -buffering line
    puts $pipe compile(inference_engine).
    puts $pipe compile(knowledge_base).

    flush $pipe
    if {[array exists po]} {
        foreach index [array names po] {
            set val $po($index)
            if {$val == -1} {
                continue
            } else {
                set val [expr int($val)*100]
            }
        }
    }
}
```

APPENDIX B. SOURCE CODE

```
        set command "pt($val,$color_val,1)."
        puts $pipe $command
    }
}

if {[array exists ie]} {
    foreach index [array names ie] {
        set val $ie($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
            set command "int(0,1200,$color_val,1200)."
            puts $pipe $command
        }
    }
}

if {[array exists fr]} {
    foreach index [array names fr] {
        set val $fr($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
            set command "int(0,$val,$color_val,$val)."
            puts $pipe $command
        }
    }
}

if {[array exists fl]} {
    foreach index [array names fl] {
        set val $fl($index)
        if {$val == -1} {
            continue
        } else {
            set val [expr int($val)*100]
            set diff [expr 1200 - $val]
            set command "int($val,1200,$color_val,$diff)."
            puts $pipe $command
        }
    }
}

if {[array exists f]} {
    for {set index1 0} {$index1<=$numFE} {incr index1} {

        set val1 $f($index1,1)
        set val1 [expr int($val1)*100]
        set val2 $f($index1,2)
        set val2 [expr int($val2)*100]
        set diff [expr $val2-$val1]
        if {$val1 != -1} {
            set command "int($val1,$val2,$color_val,$diff)."
        }
    }
}
```

APPENDIX B. SOURCE CODE

```
        puts $pipe $command
    }

}

puts $pipe "halt."

#
# The output of the execution is written to the file
# outfile.dat
#

set fileid [open outfile.dat "w+"]

fileevent $pipe readable [list Reader $pipe]
while {[eof $pipe] != 1} {
    gets $pipe response
    puts $response
    puts $fileid $response
}

close $fileid

# The following lines of code close a channel once
# it becomes readable

proc Reader { pipe } {
    if [eof $pipe] {
        catch {close $pipe}
        return
    }
    gets $pipe response
    puts $response
}

#
# The following lines of code read the data which is
# written onto the file outfile.dat and show a Message/
# dialog box giving the results.

proc showResult { d } {
    if { [regexp {(yes|.|bye)$} $d] } {
        return 1
    } elseif { [regexp {[eclipse.*$} $d] } {
        return 1
    } elseif { [regexp "^[ \t]*$" $d] } {
        return 1
    } else {
        return 0
    }
}

if {[file readable outfile.dat]} {
```

APPENDIX B. SOURCE CODE

```
        set counter 1
        set flag 1
        set fileid [open outfile.dat "r"]
        #
        #Are we at the end of the file?
        #
        while {[gets $fileid data] >=0 } {
            incr counter
            if {$counter >= 19} {
                set val [showResult $data]
                if { $val == 0} {
                    set flag 0
                }
            } else {
                continue
            }
        }
        close $fileid
    }

    #
    # show Message/Dialog box
    #
    if { $flag == 1} {
        set result [tk_messageBox -parent .query_ruler \
            -title Result -type ok -icon info \
            -message "Query results: True !" ]
    } else {
        set result [tk_messageBox -parent .query_ruler \
            -title Result -type ok -icon info \
            -message "Query results: No !" ]
    }
}

# Procedure invoked when New Query button is pressed.
proc new_query { w } {
    upvar #0 point_Obj po
    upvar #0 infinite_Obj ie
    upvar #0 finitel_Obj fl
    upvar #0 finiter_Obj fr
    upvar #0 finite_Obj f

    global count countIL countFL
    global countFR countFE
    set color #2230f0
    $w.c delete box1 tab tab1 tab2 tab3 tab4 active active1 \
        active2 active3 active4 box fend1 fixed

    # Unsetting the various arrays
    if {[array exists po]} { unset po }
    if {[array exists ie]} { unset ie }
    if {[array exists fl]} { unset fl }
    if {[array exists fr]} { unset fr }
    if {[array exists f]} { unset f }
```

APPENDIX B. SOURCE CODE

```
# Reinitializing various variables used in arrays
set count -1
set countIL -1
set countFL -1
set countFR -1
set countFE -1

# Giving Visual cues to the user.
$w.c delete well5
$w.c addtag well5 withtag [$w.c create rect 11.5c 4.5c 12.5c 3.5c
\
    -outline black -fill [lindex [$w.c config -bg] 4]]
$w.c create bitmap 12.00c 4.00c -bitmap questhead \
    -background gray -tags well5

update
focus $w.frameList
}
#Procedure invoked when Exit button is pressed
proc query_close { w } {
    destroy $w
}

#Creating the Query Screen.

canvas $c -width 17c -height 7c
pack $w.c -side top -fill x

# Create a font to show the text in the query window.

font create txtft2 -family Courier -size 12 -slant italic \
    -weight bold
set newf txtft2

#Info about the box

set demo_boxInfo(a) 0
set demo_boxInfo(motionProc) MoveNull
if {[wininfo depth $c] > 1} {
    set demo_boxInfo(boxStyle) "--fill {} -outline \
black -width 1"
    set demo_boxInfo(active) "--fill red \
-outline black -width 1"
} else {
    set demo_boxInfo(boxStyle) "--fill {} -outline \
black -width 1"
    set demo_boxInfo(active) "--fill black \
-outline black -width 1"
}

set demo_rulerInfo(grid) .25c
set demo_rulerInfo(left) [wininfo fpixels $c 1c]
set demo_rulerInfo(right) [wininfo fpixels $c 13c]
set demo_rulerInfo(top) [wininfo fpixels $c 1c]
```

APPENDIX B. SOURCE CODE

```
set demo_rulerInfo(bottom) [wininfo fpixels $c 1.75c]
set demo_rulerInfo(size) [wininfo fpixels $c .4c]
set demo_rulerInfo(normalStyle) "-fill $color"
if {[wininfo depth $c] > 1} {
    set demo_rulerInfo(activeStyle) "-fill red -stipple {}"
    set demo_rulerInfo(deleteStyle) [list -fill red \
        -stipple @[file join $tk_library demos images gray25.bmp]]
} else {
    set demo_rulerInfo(activeStyle) "-fill black -stipple {}"
    set demo_rulerInfo(deleteStyle) [list -fill black \
        -stipple @[file join $tk_library demos images gray25.bmp]]
}

$c create line 1c 0.5c 1c 1c 13c 1c 13c 0.5c -width 2
$c create text .5c .75c -text "AM" -font $newf -anchor s -fill maroon
for {set i 0} {$i < 12} {incr i} {
    set x [expr $i+1]
    if {$i==0} {
        $c create line ${x}c 1c ${x}c 0.6c -width 1
        $c create line $x.25c 1c $x.25c 0.8c -width 1
        $c create line $x.5c 1c $x.5c 0.7c -width 1
        $c create line $x.75c 1c $x.75c 0.8c -width 1
        $c create text $x.15c .75c -text "12" -font $newf \
            -anchor s -fill maroon
    } else {
        $c create line ${x}c 1c ${x}c 0.6c -width 1
        $c create line $x.25c 1c $x.25c 0.8c -width 1
        $c create line $x.5c 1c $x.5c 0.7c -width 1
        $c create line $x.75c 1c $x.75c 0.8c -width 1
        $c create text $x.15c .75c -text $i -font $newf \
            -anchor s -fill maroon
    }
}
# prints the last 12 on the scale
set x [expr $x+1]
$c create text $x.17c .75c -text $i -font $newf -anchor s -fill maroon
$c create text [expr $x+1]c .75c -text "PM" -font $newf \
    -anchor s -fill maroon

# Tags below are for the Point object
$c addtag well withtag [$c create rect 1.5c 4.5c 2.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well withtag [rulerMkTab $c [wininfo pixels $c 2c] \
    [wininfo pixels $c 4c]]
$c create text 2c 5.25c -text "Point" -fill maroon -font $newf
$c create text 2c 5.75c -text "Event" -fill maroon -font $newf

# Tags below are for the Infinite Line object
$c addtag well1 withtag [$c create rect 3.5c 4.5c 4.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well1 withtag [rulerMkTabIL $c [wininfo pixels $c 3.5c] \
    [wininfo pixels $c 4.15c]]
$c create text 4c 5.25c -text "Limitless" -fill maroon -font $newf
$c create text 4c 5.75c -text "Event" -fill maroon -font $newf
```

APPENDIX B. SOURCE CODE

```
# Tags below are for the Fixed End object
$c addtag well2 withtag [$c create rect 5.5c 4.5c 6.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well2 withtag [rulerMkTabFE $c [winfo pixels $c 5.52c] \
    [winfo pixels $c 4.15c]]
$c create text 6c 5.25c -text "Fixed" -fill maroon -font $newf
$c create text 6c 5.75c -text "Event" -fill maroon -font $newf

# Tags below are for the Fixed left object
$c addtag well3 withtag [$c create rect 7.5c 4.5c 8.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well3 withtag [rulerMkTabFL $c [winfo pixels $c 7.52c] \
    [winfo pixels $c 4.15c]]
$c create text 8c 5.25c -text "Fixed" -fill maroon -font $newf
$c create text 8c 5.75c -text "Left" -fill maroon -font $newf

# Tags below are for the Fixed right object
$c addtag well4 withtag [$c create rect 9.5c 4.5c 10.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well4 withtag [rulerMkTabFR $c [winfo pixels $c 9.52c] \
    [winfo pixels $c 4.15c]]
$c create text 10c 5.25c -text "Fixed" -fill maroon -font $newf
$c create text 10c 5.75c -text "Right" -fill maroon -font $newf

# Tags below are for the question mark
$c addtag well5 withtag [$c create rect 11.5c 4.5c 12.5c 3.5c \
    -outline black -fill [lindex [$c config -bg] 4]]
$c addtag well5 withtag [rulerMkTabQues $c [winfo pixels $c 11.52c] \
    [winfo pixels $c 4.15c]]
$c create text 12c 5.25c -text "What's" -fill maroon -font $newf
$c create text 12c 5.75c -text "True" -fill maroon -font $newf

#Bindings below are for the Infinite Line object
$c bind well1 <1> "rulerNewTabIL $c %x %y"
$c bind tab1 <1> "rulerSelectTabIL $c %x %y"
$c bind well1 <B1-Motion> "rulerMoveTabIL $c %x %y"
$c bind well1 <Any-ButtonRelease-1> "rulerReleaseTabIL $c"

#Bindings below are for the circular object
$c bind well <1> "rulerNewTab $c %x %y"
$c bind tab <1> "rulerSelectTab $c %x %y"
$c bind well <B1-Motion> "rulerMoveTab $c %x %y"
$c bind well <Any-ButtonRelease-1> "rulerReleaseTab $c"

#Bindings below are for the Fixed End object
$c bind well2 <1> "rulerNewTabFE $c %x %y"
$c bind tab2 <1> "rulerSelectTabFE $c %x %y"
$c bind well2 <B1-Motion> "rulerMoveTabFE $c %x %y"
$c bind well2 <Any-ButtonRelease-1> "rulerReleaseTabFE $c"

#Bindings below are for the Fixed Left object
$c bind well3 <1> "rulerNewTabFL $c %x %y"
```

APPENDIX B. SOURCE CODE

```
$c bind tab3 <1> "rulerSelectTabFL $c %x %y"
$c bind well3 <B1-Motion> "rulerMoveTabFL $c %x %y"
$c bind well3 <Any-ButtonRelease-1> "rulerReleaseTabFL $c"

#Bindings below are for the Fixed Right object
$c bind well4 <1> "rulerNewTabFR $c %x %y"
$c bind tab4 <1> "rulerSelectTabFR $c %x %y"
$c bind well4 <B1-Motion> "rulerMoveTabFR $c %x %y"
$c bind well4 <Any-ButtonRelease-1> "rulerReleaseTabFR $c"

#Bindings below are for the QuestionHead object
$c bind well5 <1> "rulerInfoAll $c"
# Controls the working of the What's true icon
proc rulerInfoAll {w} {

    global drop_test_var
    puts "$drop_test_var"
    $w delete well5
    $w addtag well5 withtag [$w create rect 11.5c 4.5c 12.5c 3.5c \
        -outline black -fill [lindex {$w config -bg} 4]]
    $w create bitmap 12.00c 4.00c -bitmap questhead \
        -background blue -tags well5
    }

}

font delete txtft2
source Bruler.tcl
#!/usr/local/bin/wish

# Procedure boxSetup help in manipulating the small box
# attached to the Fixed Event icon.

proc boxSetup c {
    upvar #0 demo_rulerInfo v
    upvar #0 demo_boxInfo w
    upvar #0 finite_Obj f
    global color countFE
    set vl [wininfo fpixels $c .25c]
    set tags [$c gettags current]
    if { $tags != "" } {
        set cur [lindex $tags [lsearch -glob $tags box?]]
    } else {
        set cur ""
    }
    $c delete active2 box1 tab2
    eval "$c create line $v(x) $v(y) [expr $v(x)+$vl+$w(a)] $v(y) \
        -fill $color -width 8 -tags {tab2 fixed}"

    $c addtag active2 withtag tab2

    # Creating box for reshaping the item....

    eval "$c create rect [expr $v(x)+$w(a)+$vl-6] [expr $v(y)-6] \
        [expr $v(x)+$w(a)+$vl] [expr $v(y)] $w(boxStyle) \
        -tags {box1 box}"
}
```


APPENDIX B. SOURCE CODE

```
        if {$cur != ""} {
            eval $c itemconfigure box1 $w(active)
        }
        set unit 35.457
        set temp [expr int(((v(x)/$unit)-1)*100)]
        set xvalue [expr $temp/100.00]
        set temp2 [expr int(((w(a)+v(x))/$unit)-1)*100)]

#       Value of 0.25 is added to the length because the starting
offset
#       of the length is like that and also to come to the right figure
#       it is important for us to add 0.25 to length.

        set ends [expr $temp2/100.00 + 0.25]
        set f($countFE,1) $xvalue
        set f($countFE,2) $ends
    }

upvar #0 demo_boxInfo w

#Bindings for the box.
boxSetup $c
$c bind box <Enter> "$c itemconf current $w(active)"
$c bind box <Leave> "$c itemconf current $w(boxStyle)"
$c bind box <B1-Enter> ""
$c bind box <B1-Leave> ""
$c bind box1 <1> {set demo_boxInfo(motionProc) boxMove1 }
$c bind box1 <B1-Motion> "boxMove1 $c %x %y"
$c bind box1 <Any-ButtonRelease-1> "boxSetup $c"

# procedure boxMove1 helps in moving the box on the time scale.

proc boxMove1 { c x y} {
    upvar #0 demo_rulerInfo v
    upvar #0 demo_boxInfo w
    set tmp [expr ([c canvasx $x v(grid)]]

    if {$tmp <= v(x)} {
        set tmp v(x)
    }
    if {$tmp >= v(right)} {
        set tmp v(right)
    }
    set newA [expr ($tmp-v(x))]
    if {$newA != w(a)} {
        $c move box1 [expr ($newA-w(a))] 0
        set w(a) $newA
    }
}

#!/usr/local/bin/wish
#Tk color dialog window.
#The script below creates a window for the user to
#pick a color for an event.

proc color_get { parent initialcolor } {
    upvar #0 demo_rulerInfo v
```

APPENDIX B. SOURCE CODE

```
set filename event_details.dat

global list_of_events
global list_of_descrip
global list_of_colors

set data ""
set list_of_events ""
set list_of_descrip ""
set list_of_colors ""
if {[file readable $filename]} {
    set fileid [open $filename "r"]
    while {[eof $fileid] != 1} {
        gets $fileid data
        set list_of_events [linsert $list_of_events end $data]
        gets $fileid data
        set list_of_colors [linsert $list_of_colors end $data]
        gets $fileid data
        set list_of_descrip [linsert $list_of_descrip end
$data]
    }
    close $fileid
}

set color [tk_chooseColor \
    -parent $parent \
    -initialcolor $initialcolor \
    -title "Color" ]

set v(normalStyle) "-fill $color"

#
# To search if the color currently selected is already in
# the database
#
set ans [lsearch $list_of_colors $color]
if { $ans != -1 } {
    set result [tk_messageBox -parent . \
        -title Error -type ok -icon error \
        -message \
            "Color already exists! Pick another"]
} else {
    return $color
}
}

set color [color_get . maroon]

puts "Color=<$color>"

#!/usr/local/bin/wish

# Use of the html_library for online help.

#
# For this example, you must have
```

APPENDIX B. SOURCE CODE

```
# html_library.tcl in the current directory.
#
puts "Loading html_library.tcl"
source html_library.tcl

#
# Control look of links.
# The default is raised bevel.
#
global HMevents
array set HMevents {
    Enter    {-underline 1}
    Leave    {-underline 0}
    1        {-underline 0}
    ButtonRelease-1 {-underline 0}
}

# Global variables
set help_initialized 0
global help_initialized

# Creates a help window and calls up HTML helpfile.
proc help { helpfile } {
    global help_initialized
    global html

    set text [help_create $helpfile]

    # Load help file
    set html [help_load_html $helpfile]

    # Initialize HTML
    if { $help_initialized == 0 } {
        set help_initialized 1

        HMinit_win $text

        HMset_state $text -size 2
        HMset_indent $text 1.2
    } else {
        HMreset_win $text
    }

    HMparse_html $html "HMrender $text"
}

# Handle the copy action for a text widget.
proc edit_copy { textwidget } {

    # Check if any text is selected in textwidget.
    set owner [selection own]

    if { $owner == $textwidget } {
        # Clear clipboard.
    }
}
```

APPENDIX B. SOURCE CODE

```
        clipboard clear

        catch {
            clipboard append [selection get]
        }
    }
}

# Creates help window, called by help.
proc help_create { filename } {

    set top .helpwindow
    set frm $top.frm

    if { [wininfo exists $top ] } {
        wm deiconify $top
        return $frm.text
    }

    toplevel $top
    # Set up wm options for .help

    # Menubar
    frame $top.menubar -bd 1 -relief raised

    menubutton $top.menubar.file -text "File" -underline 0 \
        -menu $top.menubar.file.menu
    menubutton $top.menubar.edit -text "Edit" -underline 0 \
        -menu $top.menubar.edit.menu

    menu $top.menubar.file.menu

        # Reset to original message.
    $top.menubar.file.menu add command -label "Original Topic" \
        -command "help_disp_file $frm.text $filename"

    $top.menubar.file.menu add command -label "Close" \
        -command "destroy $top"

    menu $top.menubar.edit.menu
    $top.menubar.edit.menu add command -label "Copy" \
        -command "edit_copy $frm.text"

    pack $top.menubar.file $top.menubar.edit -side left
    pack $top.menubar -side top -fill x

    # Main help area.
    frame $frm -bd 0
    text $frm.text -width 60 -height 20 \
        -yscrollcommand "$frm.v_scroll set" \
        -xscrollcommand "$frm.h_scroll set"

    scrollbar $frm.v_scroll \
        -command "$frm.text yview"
```

APPENDIX B. SOURCE CODE

```
scrollbar $frm.h_scroll -orient horizontal \  
    -command "$frm.text xview"  
  
pack $frm.v_scroll -side right -fill y  
pack $frm.h_scroll -side bottom -fill x  
pack $frm.text -expand 1 -fill both  
  
pack $frm -side top -expand 1 -fill both  
  
# Return name of text widget.  
return $frm.text  
}  
  
# Private procedure to handle a link.  
proc Hmlink_callback {win href} {  
    global html  
  
    #  
    # If this is not a file link,  
    # or if it has file://, you may  
    # need to parse out the type  
    # of URL.  
    #  
  
    # Load up HTML file.  
    set html [help_load_html $href]  
  
    # Display in text widget.  
    help_disp_html $win $html  
}  
  
# Private procedure to display HTML.  
proc help_disp_html {win html} {  
  
    # Display in text widget.  
    HMreset_win $win  
    HMparse_html $html "HMrender $win"  
}  
  
# Private procedure to load HTML file and display.  
proc help_disp_file {win filename} {  
  
    set html [help_load_html $filename]  
  
    help_disp_html $win $html  
}  
  
# Private procedure to load HTML file.  
proc help_load_html { filename } {
```

APPENDIX B. SOURCE CODE

```
# Default data in case of errors.
set data "<title>Bad file $filename</title>
        <h1>Error reading $filename</h1><p>"

catch {
    set fileid [open $filename]

    set data [read $fileid]
    close $fileid
}

return $data
}

help mainhelp.htm

:- use_module(library(fd)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Note: only works for info that is T/F

% user types in pt(t,f,x) to find out if f(t)=x, and
% int(a,b,f,x) to find out if the integral of f between a and b is x.

% solve directly
pt(T,F,X) :- point(T,F,X).

% F is true throughout some interval containing T.
pt(T,F,1) :-
    A #< T,
    B #> T,
    C #= B-A,
    integral(A,B,F,C).

% solve directly
int(A,B,F,X) :- integral(A,B,F,X), !.

% F is true over a super-interval of (A,B)
int(A,B,F,C) :-
    X #<= A,
    Y #>= B,
    Z #= Y-X,
    integral(X,Y,F,Z),
    C #= B-A.

% Sub-divide the interval
int(A,B,F,C) :-
    X #<= A,
    Y #> A,
    Y #< B,
```

APPENDIX B. SOURCE CODE

```
Z #= Y-X,
integral(X,Y,F,Z),
int(Y,B,F,Temp),
C #= Y - A + Temp.

:- use_module(library(fd)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% all interval based information is represented using the integral.
% integral(a,b,f,x) is true iff the integral of f from a to b is x.
% e.g.: integral(0,10,running,10) -- running is true throughout (0,10).

% information true at an isolated point is represented with point.
% point(t,f,x) is true iff f(t)=x.
% e.g.: point(20,running,1) -- running is true at time 20
% 1 == true; 0 == false.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example %%%%%%%%%%%%%%%

% running is true over (0,10), false over (10,15), true over (15,20)
% and (20,30)
integral(0,10,running,10).
integral(10,15,running,0).
integral(15,20,running,5).
integral(20,30,running,10).

% ran for an hour between 30 and 40.
integral(30,40,running,1).

%-----write after this line-----

integral(600,700,cb08a60,100).

integral(0,1200,cb0c860,1200).

integral(0,1000,cb030f6,1000).

integral(900,1200,c6cbebc,300).

integral(600,700,cb0742c,100).

integral(600,700,cd84c60,100).
integral(700,800,cd84c60,100).

integral(100,1000,c546e60,900).

%point based info
point(10,running,0).
point(15,running,1).
```

APPENDIX B. SOURCE CODE

```
point(20,running,1).
point(25,running,1).

% -----write after this line-----

point(1000,cfc3060,1).

point(900,c5c7e60,1).
point(1000,c5c7e60,1).

point(1000,cb0742c,1).

point(1000,c2230f0,1).

point(700,cd84c60,1).

#!/usr/local/bin/wish

#
# This program creates a window to accept the new file
# name from the user which will be used as to save
# event details.
#

set nf .newf
toplevel $nf -class Dialog
wm title $nf "New File"
wm transient $nf .
wm geometry $nf +300+300

set filename event_details.dat

proc proc_ok {main} {
    global nfname
    if {$nfname == ""} {
        set pmess [tk_messageBox -parent $main\
            -title {Error} -type ok -icon error\
            -message\
            "Missing file name."]
    } else {
        destroy $main
    }
}

#
#Create the heading
#

global nfname
frame $nf.fr
label $nf.fr.filename -text "Enter new file name:"
entry $nf.fr.fileentry -width 25 -textvariable nfname

grid config $nf.fr.filename -column 0 -row 1 -sticky w
grid config $nf.fr.fileentry -column 1 -row 1 -sticky snw
```


APPENDIX B. SOURCE CODE

```
pack $nf.fr

frame $nf.frl
button $nf.frl.b -text " OK " -command {proc_ok $nf}
button $nf.frl.b1 -text "Cancel" -command {destroy $nf}
pack $nf.frl.b $nf.frl.b1 -side left -padx 2 -pady 2
pack $nf.frl

focus $nf.fr.fileentry
<html>
<head><TITLE>System Database</TITLE></head>
<BODY BGCOLOR=#FFFFFF TEXT=#000>

<center><h1>How to Use Temporal Expert System Shell<br>
<u>Finding the contents of the Database</u></h1>
<h3><i>
<DT><A HREF="overviewhelp.htm">Overview of the System</A>
<DT><A HREF="enterhelp.htm">Entering the information</A>
<DT><A HREF="queryhelp.htm">Querying the system</A>
<DT><A HREF="symbolhelp.htm">Symbols and Icons</A>
<DT><A HREF="mainhelp.htm">Return to Main Menu</A>

</h3></center>
<h2><u>Finding the contents of the Database</u></h2>

User can find the contents of the Database by clicking on the
View Database choice available under the "Options" menu.
<br><br>
It displays the "Event Name", "Associated Color" and the
"Event Description" in a tabular fashion.
<br><br>
It is a dialog window and clicking on the "OK" button closes
the window.

</body></html>

<html>
<head><TITLE>Entering Information</TITLE></head>
<BODY BGCOLOR=#FFFFFF TEXT=#000>

<center><h1>How to Use Temporal Expert System Shell<br>
<u>Entering the Information</u></h1>
<h3><i>
<DT><A HREF="overviewhelp.htm">Overview of the System</A>
<DT><A HREF="queryhelp.htm">Querying the System</A>
<DT><A HREF="symbolhelp.htm">Symbols and Icons</A>
<DT><A HREF="databasehelp.htm">Finding the contents of
Database </A>
<DT><A HREF="mainhelp.htm">Return to Main Menu</A>

</h3></center>
<h2><u>Entering the Information</u></h2>

User can enter an event details through the window called Enter
Information. This window is invoked by clicking on the Enter
information option under the Edit menu.
```

APPENDIX B. SOURCE CODE

Every event has a name, description and a temporal component.
For example:

John had a meeting at 4 p.m.

The event name for this event is "Meeting", the event description is "John had a meeting at 4 p.m. The text entry box labeled "Event Name" is used for entering the name of the event. The "Event description" text box provides a location for adding event details. The user can move from one data field to another either by tabbing to, or by clicking in the desired field. The system prompts the user with an error message, if any of the boxes are left empty.

The next step is to pick a color. The "Pick Color" button in the Enter Information window invokes the color window. The user can select a color for an event by filling in the numerical values for red, green and blue. It is also possible to select a color by sliding the triangular tab on the color scale for each color. The user confirms his/her selection by clicking on the "OK" button.

There are different categories of events that can be represented. Depending upon the category of an event, the user chooses an icon from one of the boxes labeled "Point Event", "Limitless Event" etc. The user can modify or change the position of the icon/icons on the time scale.

In order to select an icon, the user points and clicks on the desired icon and it can be dragged along with the mouse pointer to any position on the time scale. The icon may be positioned at any of the grid intervals. The user may modify the position of the icon on the time scale by pointing and clicking on the icon and dragging it along with the mouse pointer to a new position.

After entering the current event, the user clicks on the "Continue" button to add another event. Finally, clicking on the "Done" button closes the current window, saving the details to a file. Cancel button aborts the current operation, clears the screen and takes the control back to the "Event Name" text box.

```
</body></html>
```

```
<html>
<head><TITLE>Temporal Expert System Help</TITLE></head>
<BODY BGCOLOR=#FFFFFF TEXT=#000>
<center>
<h1>How to Use Temporal Expert System Shell<br>
<u>Main Menu</u></h1>
<h3><i>
<DT><A HREF="overviewhelp.htm">Overview of the System</A>
<DT><A HREF="enterhelp.htm">Entering the Information</A>
<DT><A HREF="queryhelp.htm">Querying the System</A>
<DT><A HREF="symbolhelp.htm">Symbols and Icons</A>
<DT><A HREF="databasehelp.htm">Finding the contents of
Database </A>
</i></h3><center>
</body></html>
```

```
<html>
<head><TITLE>System Overview</TITLE></head>
```

APPENDIX B. SOURCE CODE

```
<BODY BGCOLOR=#FFFFFF TEXT=#000>

<center><h1>How to Use Temporal Expert System Shell<br>
<u>Overview of the System</u></h1>
<h3><i>
<DT><A HREF="enterhelp.htm">Entering the Information</A>
<DT><A HREF="queryhelp.htm">Querying the System</A>
<DT><A HREF="symbolhelp.htm">Symbols and Icons</A>
<DT><A HREF="databasehelp.htm">Finding the contents of
      Database </A>
<DT><A HREF="mainhelp.htm">Return to Main Menu</A>

</h3></center>
<h2><u>Overview of the System</u></h2>

The Temporal Expert System Shell has two windows for Entering
the information and querying the information.
Clicking on the Edit menu and then on Enter Information will
invoke the window for Entering Information.
Similarly, Clicking on the Edit menu and then on the Query
Information will invoke the window for Querying Information.
<br><br>
Every event is associated with a unique color and the colors
currently in use can be found by clicking on the Colors Used
under the Options menu.
We can find out a list of events and their associated description
by clicking on the View Database under the Options menu.<br>
A list of all the symbols used is available under the Show symbols
category of the Options menu.
<br><br>
All the windows used are self explanatory and and easy to use.
The help options under the Enter Information and Query Information
provide help for those screens.
<br><br>
</body></html>

<html>
<head><TITLE>Querying the System</TITLE></head>
<BODY BGCOLOR=#FFFFFF TEXT=#000>

<center><h1>How to Use Temporal Expert System Shell<br>

<u>Quering the System</u></h1>
<h3><i>
<DT><A HREF="overview.htm">Overview of the System</A>
<DT><A HREF="enterhelp.htm">Entering the information</A>
<DT><A HREF="symbolhelp.htm">Symbols and Icons</A>
<DT><A HREF="databasehelp.htm">Finding the contents of
      Database </A>
<DT><A HREF="mainhelp.htm">Return to Main Menu</A>

</h3></center>
<h2><u>Querying the System</u></h2>

User can query an event through the window called Query Information.
This window is invoked by clicking on the Query information option
under the Edit menu. <br>
```

APPENDIX B. SOURCE CODE

The user can perform the following queries:

1. Is an event true at the given time ?

2. What is true about an event ?

The layout of the query window is similar to the enter information window. Select Event is the label for the dropdown event list box shown by a button with a downward arrow sign. The user can view list for the events by clicking on this button. An event is selected from the list by pointing and clicking on it. The user confirms his/her selection by clicking on the "OK" button.

To query the system about an event, the user selects and positions the appropriate icon on the time scale and clicks on the "Query" button. The system displays the result of the query in a separate dialog window.

As the name suggests the "New Query" button allows the user to perform a new query by clearing the screen and returning control to the "Select Event" dropdown list box. The "Exit" button closes the current window and transfers control back to the main application window.

</body></html>

<html>

<head><TITLE>Symbols and Icons</TITLE></head>

<BODY BGCOLOR=#FFFFFF TEXT=#000>

<center><h1>How to Use Temporal Expert System Shell

<u>Symbols and Icons</u></h1>

<h3><i>

<DT>Overview of the System

<DT>Entering the information

<DT>Querying the system

<DT>Finding the contents of
Database

<DT>Return to Main Menu

</h3></center>

<h2><u>Symbols and Icons </u></h2>

There are various Symbols and Icons which are used in Entering and Querying information. These Symbols are explained below:

POINT EVENT: Used for events which occur at precise points over the given time scale. E.g., Phone rang at 2:00 pm.

LIMITLESS EVENT: Event that occurred at some unknown time in past and continues in future.E.g., It has been raining today.

FIXED EVENT: Event that occurred between two points on the given time scale. Activity has precise start and end points.

E.g., We had lunch between 2 & 3 pm.

FIXEDLEFT EVENT: Event that starts at a known fixed point and continues in future. E.g., Basketball game started at 6 pm and went on till late evening.

FIXEDRIGHT EVENT: Event which started at some unknown point in past and has fixed end point. Complement of FixedLeft Event.

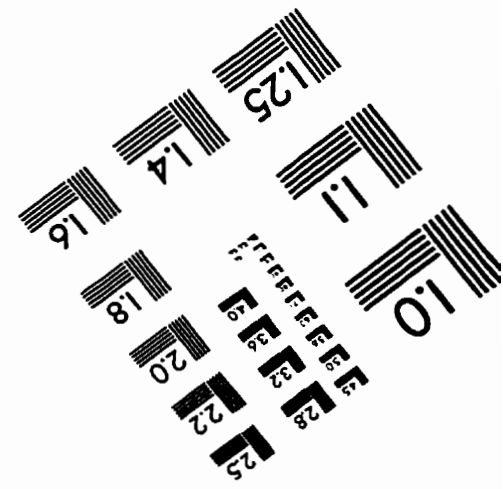
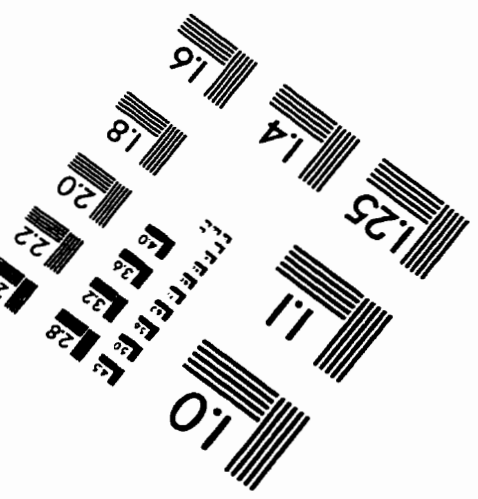
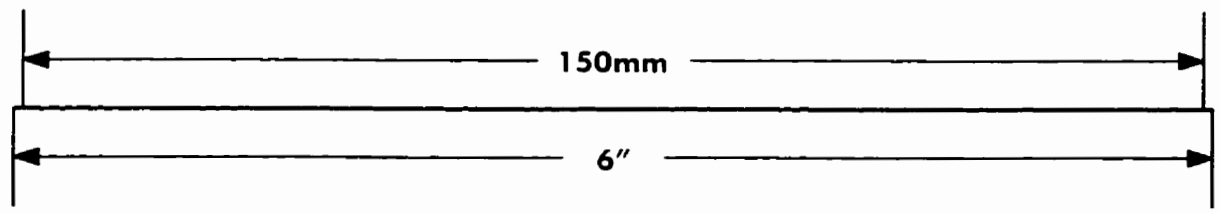
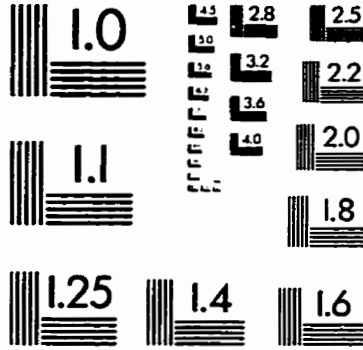
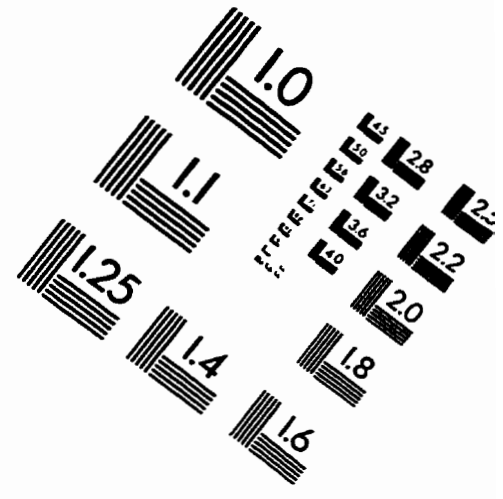
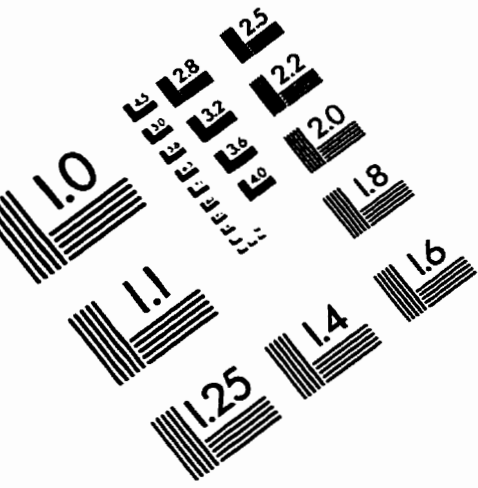
APPENDIX B. SOURCE CODE

E.g., After a long sleep, I woke up at 12:00 pm.

WHAT'S TRUE: This symbol is used for querying the system.
It tells, all what is true for a selected event.

</body></html>

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved