MPC REPORT NO. 94-36

AN EVALUATION GUIDEBOOK FOR RURAL AND SMALL URBAN TRANSPORTATION SYSTEMS IN THE MOUNTAIN-PLAINS REGION

John Bitzan
and
Jill Hough

September 1994

EXECUTIVE SUMMARY

The objective of this study was to enhance the value of data received by maintenance foremen for making decisions regarding winter highway maintenance. More specifically, to develop a microcomputer-based Decision Support System (DSS) to facilitate the acquisition and interpretation of information from weather advisory services and from remote sensors at Remote Weather Information Systems (RWIS).

The project was a joint venture with Utah State University (USU), the Utah Department of Transportation (UDOT), and SSI Corporation, St. Louis, MO. SSI provided a RWIS at reduced cost to UDOT; UDOT installed the equipment and provided technical assistance for the project; and USU developed new user interfaces based on expert system techniques.

Three approaches were evaluated for developing the new interfaces: 1) a commercial expert system shell, 2) the Prolog programming language, and 3) the C++ programming language. Work with the shell was pursued for about two months and then abandoned. The shell worked very well for quickly establishing rules and it provided an impressive user interface. However, we were unable to communicate effectively with the software for the remote sensors.

We found Prolog to be a powerful programming language especially when used in conjunction with a commercial library for the graphical user interface (GUI). However, like the shell, Prolog also lacked the flexibility we needed to communicate effectively with the remote sensors. An application was developed using Prolog to interface with the WSI Weather Information Service.

We found C++ to be the best tool for developing a knowledge-base decision support system to interface with the SSI remote weather information system. An application was developed in C++ that provides a new prototype menu-driven user interface and links it to a knowledge-base decision support system.

A panel of experts was formed with maintenance foremen from Utah and Colorado who had extensive experience with winter highway maintenance. The computer interfaces were designed according to the recommendations of the panel. The experts created the knowledge-base for the SSI decision-support system and evaluated the preliminary applications. The SSI program was also tested by giving a non-expert historical data from two storms, and asking him to make decisions regarding the dispatch and release of maintenance crews. The non-expert reached the same conclusions as the expert for a small sample of relatively simple situations. Although the overall approach and the design of the graphical user interface were judged to be effective, the current rule-base is suitable only for prototype demonstrations.

The WSI weather information system interface was evaluated by maintenance personnel and judged to be very satisfactory. It was obvious from the interchange with the experts that the use of a graphical user interface is necessary in order to effectively use existing weather services by eliminating the need for the user to memorize complex command syntax.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION ... 1

STUDY OBJECTIVES ... 2

STUDY ORGANIZATION ... 2

- Preliminary Study Design ... 3
  - Background ... 3
  - Structure for the Decision-Support System ... 5
- Final Study Design ... 7

ORGANIZATION OF THE REPORT ... 11

CHAPTER 2

WORK BY OTHERS ... 13

CHAPTER 3

DECISION-SUPPORT SYSTEM ... 21

RULE-BASE ... 21

- The Variables File ... 22
- The Actions File ... 22
- Rules and Tests ... 23
- The Preprocessor ... 23
- The Fact Data Base File ... 23

THE WRAPPER ... 24

THE TRIPOD OBJECT ... 24

PROGRAM PROCEDURE ... 24

- Overview ... 24
- The Rule Engine ... 25

TRIPOD DATA STRUCTURES ... 27

- The TRIPOD Class ... 27
- The RBVar and RBChoice Classes ... 30
- The RBAct and RBOpAry Classes ... 30
- The RBRue and RBRueTst Classes ... 31

CHAPTER 4

PROTOTYPE APPLICATION 33

RULE-BASE DEVELOPMENT 35

The Variables File 36

The Actions File 37

The Rules File 39

PRELIMINARY TESTING 41

CHAPTER 5

THE WEATHER INFO INTERFACE 47

Using Weather Service Information (WSI) to Meet a Specific Need 47

Weather Brief and WXVIEW+- 48

The WeatherInfo Interface 49

The Main Menu 50

Choose New Options 51

Show Existing Data 55

Download Data 56

The Status Line and Messages 57

Choosing the Weather Information Items to be Downloaded 57

Downloading Weather Information 58

The Tools Used to Create WeatherInfo 58

Evaluation and Use of WeatherInfo and WSI's Weather Information System 58

Advantages of WeatherInfo. 60

Additional Enhancements 61

CHAPTER 6

SUMMARY AND CONCLUSIONS 63

Summary 63

Conclusions 64

REFERENCES 67

APPENDIX 71

LIST OF TABLES

Table 1. Input Format for the Variables File 36
Table 2. A Partial Listing of the Variables File (.IDV) 38
Table 3. Input Format for the Actions File 39
Table 4. A Partial Listing of the Actions File (.IDA) 40
Table 5. Input Format for the Rules File 40
Table 6. A Partial Listing of the Rule File (.IDR) 44
Table 7. Satellite and Radar Information 53
Table 8. Present and Past Conditions 53
Table 9. Forecasts 54
Table 10. Maps and Forecasts 54
Table 11. Custom Graphics 55

LIST OF FIGURES

Figure 1. Conceptualized Decision Support System 6
Figure 2. The Decision-Support System Application Process 22
Figure 3. Program Procedure 26
Figure 4. Tripod Data Structure 29
Figure 5. The Main Menu for the Decision-Support System Interface 34
Figure 6. Standard Interface Screen 45
Figure 7. WeatherInfo Menu System Interface 50
Figure 8. A Radar Image and a Satellite Image of the Utah Region Downloaded with WeatherInfo 51
Figure 9. Present Conditions for the Entire United States 52
Figure 10. A Sequence in a Satellite Image Loop 61

CHAPTER 1
INTRODUCTION

Each winter, most highway departments in the northern United States spend a major portion of their total annual budget for equipment, manpower, and materials to keep roads free of snow and ice. Snow removal and de-icing operations usually consists of plowing the snow from the roads and applying salt and sand to prevent or eliminate slick conditions. The Utah Department of Transportation (UDOT) had a statewide budget of over $7,500,000 for snow and ice removal for the 1990 fiscal year.

The highway maintenance engineer is faced with the problem of how to use manpower and equipment in the best way possible. He must decide if and when to take action to prevent dangerous road conditions, which roads require maintenance, which equipment to use, and what combination of salt and sand will be most effective. The correct timing of the application of the salt and sand is critical. If the salt and sand are put on the pavement too soon, the motion of passing vehicles may blow it off the highway before it becomes effective. If the maintenance engineer waits too long before beginning salting operations, the highway may become icy, and more salt will be required to melt the ice that has formed than would have been required to prevent its formation in the first place. The timing of the dispatch and release of road crews affects costs, as well as safety. If crews were dispatched too early or released too late, excessive costs would be incurred.

In order to make good decisions, the maintenance engineer must have timely and accurate information on current pavement and weather conditions, and on weather forecasts. However, just having access to raw data is usually not sufficient. The information must be easily retrieved and automatically displayed in a form that is understandable and useful for decision making.

STUDY OBJECTIVES

The broad objective of the study was to enhance the value of data received by maintenance foremen for making decisions regarding winter highway maintenance. More specifically, to develop a microcomputer-based Decision Support System (DSS) to facilitate the acquisition and interpretation of information from weather advisory services and from remote sensors at Remote Weather Information Systems (RWIS). The following major tasks were identified:

1. Establish a formal agreement among Utah State University (USU), the Utah Department of Transportation (UDOT), and Surface Systems, Inc. (SSI) of Saint Louis, MO for the purpose of installing a SSI remote weather information system in Sardine Canyon, UT. The agreement provides for SSI to supply the equipment at reduced cost, UDOT to purchase and install the equipment, and USU to develop an enhanced microcomputer interface.
2. Develop a prototype microcomputer-based decision-support system to assist maintenance foremen in interpreting the information they receive from SSI remote weather information systems.
3. Develop a graphical user interface to facilitate the acquisition of weather data and forecasts from weather advisory services.
4. Demonstrate the prototype decision-support system to personnel from SSI and UDOT at the conclusion of the project.

STUDY ORGANIZATION

The two-year study commenced in 1989 with the expectation that Task 1 could be quickly accomplished, and that the equipment would be installed for the 1989-90 winter season. However, we greatly underestimated the time and effort necessary to negotiate a mutually acceptable formal contract. The proposal identified the responsibilities for each participant and had been approved by all parties. After the award of the Mountain Plains Consortium (MPC) funding, USU initiated the process for establishing a formal contract.

During the process we discovered that it was very difficult to incorporate all the legal clauses required by the various contracting offices. Repeatedly, we would expect the contract to receive all signatures only to learn that a new complication had arisen that would cause a delay of another two or three months. The contract was finally signed by all parties during the summer of 1990 and the equipment was ordered. The equipment arrived in late fall, and due to inclement weather, installation was not completed until the Fall of 1991, more than two years after the start of the project.

Commitments to fund graduate students had been made at the beginning of the project before we expected the prolonged delays in acquiring equipment. During the first year, three master's level students entered and then left the project because of the uncertainty of completing a thesis before the date they had planned to graduate. Consequently research was limited to the identification of needs, and to the selection and development of software tools for creating decision-support systems.

Eventually, two students who were supported by the contract completed research projects that provided prototype modules for the decision-support system (Marshall 1993 and Bjerregaard 1993). Their work is incorporated into this report based on preliminary thesis materials. Marshall's work contributed to the development of the C++ knowledge-base decision-support system, and Bjerregaard's work included the development of the weather information system management interface.

Preliminary Study Design

Background

For most Utah highways, maintenance engineers rely on local media weather and the National Weather Service for weather forecasts and weather information. The maintenance engineer must watch television or listen to the radio for periodic forecasts from the National Weather Service to obtain this information. Maintenance engineers rely mostly on weather forecasts from television to plan their future manpower and equipment needs.

Other sources for information about weather and pavement conditions are also used to plan and decide when, or if, to start snow and ice removal operations. These sources of information include Highway Patrol reports, personal observations, and personal contacts. A Highway Patrol report is most often a phone call warning UDOT personnel that a Highway Patrol Trooper has observed dangerous highway pavement conditions at some location. Personal observations may be as simple as looking out the window, or driving on the highway to observe the conditions. By intuitively combining these observations with information from weather forecast and personal knowledge of the effects of local topography on the weather, a maintenance engineer can gain added insight into what the pavement conditions will be in the near future.

For example, a maintenance engineer for Utah District 2 reported that he often has a UDOT employee drive west on Interstate I-80 about 80 miles west of Salt Lake City to observe conditions when a storm is approaching from the west. These observations provide direct information on current pavement conditions on I-80 and on the timing and intensity of the approaching storm. Some maintenance engineers have personal contacts that can provide weather information to which that they would not otherwise have access. These contacts include friends in distant areas of their maintenance district and meteorologists at local National Weather Service offices, television stations, airports, and military bases.

Faced with limited and uncertain weather information, UDOT's de-icing procedures and management are understandably conservative. This conservative approach leads to safer road conditions but at a considerable cost in manpower, materials, and equipment. For example, the maintenance shed located in northern Utah on State Highway 84 between Brigham City and Wellsville is manned nearly continuously by a maintenance crew from November to April. Rather than risk dangerous highway conditions, maintenance crews are on the job and ready to sand, salt, and plow any time there is even a slight chance of snow or ice on the highway. This approach is understandable given the uncertainty of weather forecasts for this mountainous area, but it also increases the costs of winter maintenance.

Considerable experience is required to make consistently good decisions about de-icing and snow removal operations. The maintenance engineer must decide when and where to begin de-icing and snow removal operations, the manpower and equipment needed, and what combination of salt and sand will be most effective. To make these decisions, the maintenance engineer must have knowledge of the resources available, knowledge of the effects of local topography on weather, and knowledge of local traffic patterns.

Structure for the Decision-Support System

The decision-support system would be based on a knowledge-base expert system (KBS) incorporating information gathering components. Figure 1 illustrates the conceptualized framework for the decision-support system. Starting at the upper left-hand corner of the diagram, interviews with maintenance engineers (domain experts) produce a knowledge base. The knowledge base encapsulates the knowledge of the experts by establishing some facts and defining a set of logical rules to operate on the facts. The computerized decision support system (DSS) is indicated by the heavy line in Figure 1. The knowledge base is implemented in the DSS by means of two files in the computer: the rule-base and the fact-base. In addition, a module is provided in the DSS to acquire real-time data from external sources. The inference engine is an algorithm that utilizes the logical rules, the facts, and the real-time data to reach a conclusion regarding appropriate action for the current situation.

Plans called for providing access to three data sources as shown in Figure 1. National Weather Service data are available through a national data base service (WSI 1988). Highway remote sensors data and the SCAN-CAST (WSI 1989) weather forecast service are available through Surface Systems Incorporated, St. Louis, MO. Data could be automatically extracted from these sources as needed to satisfy the logical sequence of operations directed by the inference engine.

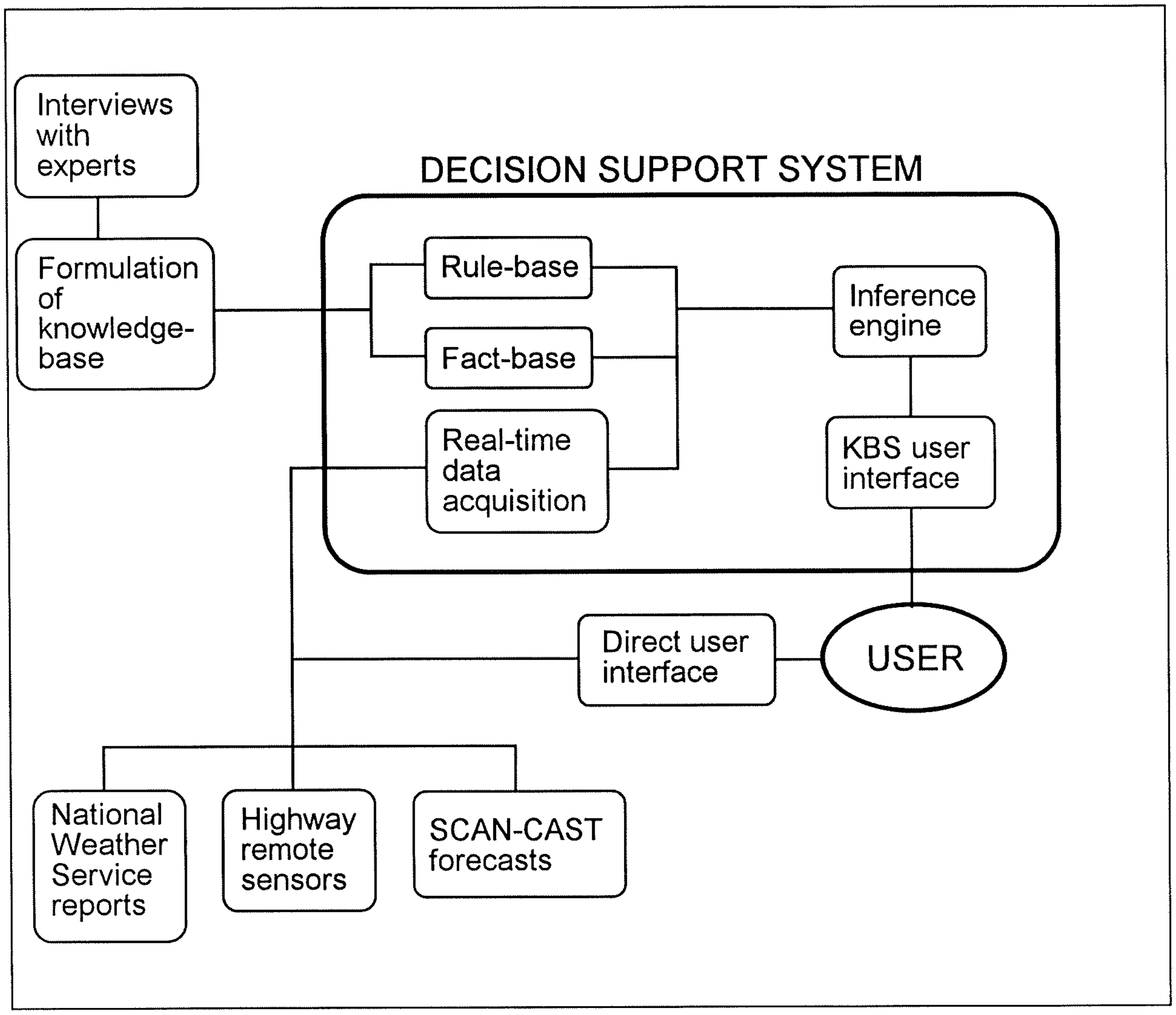


Figure 1. Conceptualized Decision-Support System.

Two computer interfaces are provided for the user. The KBS user interface is provided for the inexperienced user. The user may wish to use the decision-support system to assist him/her in reaching a reasonable conclusion for the current highway and weather conditions. He/she is willing to rely on the logic in the rule-base and the available data to lead him/her to the appropriate decision. An experienced user may wish to bypass the decision-support system and access the external data sources directly. The direct user interface is provided for that purpose. This interface simply provides a convenient mechanism for accessing the data, and for presenting meaningful displays of the data.

Final Study Design

Because of the delay in obtaining equipment, we were unable to completely implement the system illustrated by Figure 1. Most of the study effort went into the development of the knowledge-base decision-support system (DSS) tools and the graphical user interface (GUI) for accessing weather forecasts.

The first activity undertaken was the establishment of a panel of experts. Four foremen from highway maintenance departments in Colorado and Utah were invited to form the panel. All four had lengthy experience making decisions regarding snow plow and de-icing operations. During the first meeting, the experts listed the decisions that they make during a wide variety of winter storm events, and identified the data that would be nice to have available in order to decide on a particular course of action. They considered the following features essential for a useful DSS:

1. Lead a novice user through a step-by-step process to reach a reasonable conclusion.
2. Display tables and graphics quickly and concisely for experienced users.
3. Provide optional supplemental information to assist with the interpretation of displayed data.
4. Interface with commercial spreadsheets, data bases, and other third-party executable modules.
5. Operate on a computer platform readily available to maintenance foremen.

This last feature dictated the use of IBM 386-class microcomputers because they are affordable or already available at highway maintenance sites. Also, the software interface provided by the manufacturer of the remote sensors operates under DOS™[1]. However, we also considered portability an important feature. MS Windows™[2], OS/2™[3], and UNIX are

[1] DOS is a registered trademark of Microsoft®, Redmond, WA.
[2] MS Windows is a registered trademark of Microsoft®, Redmond WA.
[3] OS/2 is a registered trademark of Internation Business Machines (IBM).

encroaching on DOS™, and we would like to adapt to these operating systems as they become prevalent.

The second activity undertaken to design the DSS involved the selection of a modeling technique (language or shell) for the tool. The rule-base structure was selected as the basis for the DSS for the following reasons. It is a natural way to represent the decision-making process, it is a better way than most to capture the thought processes of experts, and it can be expanded and updated with a minimum of reprogramming effort (Walters and Nielsen 1988, Turban 1990, Adeli 1990). We decided to evaluate three techniques: 1) a commercial shell, 2) Prolog, and 3) C++. Three graduate students worked on three separate prototype projects. One used the INSIGHT 2+ commercial shell (Information Builders 1986), another used Turbo Prolog (Borland International, 1988), and the third developed a prototype using Turbo C++ (Borland 1990). We compared progress weekly during the evaluation period.

Work with the shell was pursued for about two months and then abandoned. The shell worked very well for quickly establishing rules and it provided an impressive user interface. However, we had difficulty interfacing the software for the remote sensors. This software was an undocumented stand-alone package composed of some complex DOS batch files that executed Pascal programs. The software changed environmental variables, dominated memory, and frequently switched the screen between text and graphics modes. The software was proprietary and we did not have access to the source code. In the final analysis, we just did not have the expertise necessary to interface the remote sensors with the shell. The shell did give us, however, a good initial set of rules for the other two projects.

The second student had no previous experience with Prolog. After climbing the learning curve, he came to prefer Prolog over C and FORTRAN. He utilized Borland's Turbo Prolog Toolbox (Borland International 1988) to develop effective pull-down menus. He coded a module which responds to a menu item and automatically accesses a national weather data base (Robertson Software 1988). He developed routines to select data from the data base and display it in graphical formats. He made an initial attempt to interface the sensor manufacture's software package and was not successful. After the difficulties encountered during our first experience, we elected not to spend additional time trying to interface the sensors, but instead to complete his project as a prototype interface to the national weather database (Bjerregaard 1993).

During the third evaluation project, we found it difficult to adjust from traditional structured programming to the Object Oriented Programming (OOP) approach advocated by C++ developers. However, it was worth the effort. OOP is intuitively consistent with the event-driven paradigm which is becoming increasingly popular, and which we found to be an effective way to design the DSS. Although we still had trouble interfacing with the remote sensors, many alternatives for work-arounds were available. We decided to select C++ over a commercial shell and Prolog. The main reasons follow:

1. The commercial shell was effective for quickly constructing a prototype. However, learning to use it for a practical application involving remote sensors appeared to be almost as difficult as learning a lower level language, and the shell certainly lacked the procedural programming capabilities. A commercial shell would not be as portable to other platforms as C/C++. Also, commercial shells often have a royalty fee associated with each run-time license.
2. Turbo Prolog was found to be a very powerful language for this application, especially when used in conjunction with Borland's Toolbox. Although popular in other parts of the world, Prolog appears to be declining and losing support in the USA. For example, Borland discontinued support for Turbo Prolog. We did not think that Prolog would be as portable to other platforms as C/C++.
3. Momentum is growing in support and extensions for C++. Borland's C++ provides a framework for developing object oriented programs. OOP is intuitively consistent with the event-driven paradigm which is becoming increasingly popular.

It should be emphasized that this evaluation of the three products was far from scientific or comprehensive. It focused on one application, and, in particular, on portability and the interfacing of procedural modules. At the very least, it was biased by our backgrounds as C programmers. However, in summary, it is safe to say that neither the commercial shell nor Prolog offers sufficient advantages to switch from the C/C++ family for this application.

Another consideration played an important part in the decision. The experts had trouble formulating their ideas directly into typical antecedent-consequence type rules in conventional formats. However, they were quite effective developing rules by means of a three-step procedure. The first step was to make a list of the important actions (consequences) that they would perform during a winter storm event. The second step was to make a list of variables (types of data) that were useful in deciding which action to take. The third step involved the development of a decision matrix for associating actions with variables and their values. The decision matrix defines the rules for the application. An example of the rule format is presented later in this report.

The three data sets assembled by the above procedure completely define the rule-base. We wanted to develop a preprocessor that would automatically convert the three data sets into a single rule-base that could be efficiently input to a DSS of our design. It seemed like a good idea to translate directly from formats that the experts were able to construct and comprehend, to a rule-base file that the DSS could efficiently read. By doing so, the experts could quickly see the results of a change to one of their files without an intermediate interpretation by a knowledge engineer. All of the knowledge engineering goes into the formation of the three data sets which, for this case, were easily understood by the experts. We found that C/C++ was an effective language for developing both the preprocessor and the DSS.

ORGANIZATION OF THE REPORT

Chapter 1 of this report provides an overview of the objectives, plans, and difficulties encountered during the research project. Chapter 2 is a summary of literature on related work performed by others. Chapter 3 describes the decision-support tools that were developed for this study. Chapter 4 describes the use of the tools to implement a prototype application for winter highway maintenance. Chapter 5 describes the graphical user interface for accessing data from remote data bases containing weather forecasts that was developed during the evaluation of Prolog. Chapter 6 contains the summary and conclusions.

CHAPTER 2
WORK BY OTHERS

The literature available on the detection of ice on highways by remote sensors is limited. The technology used in developing the hardware is relatively new, and models using this information to predict the formation of ice have only recently been utilized. No literature was found linking ice-warning systems with knowledge-based systems or expert systems. Although there are numerous articles dealing with knowledge-based systems, there is no evidence that a knowledge-based system has been used as part of an ice warning system.

In the early 1970's, the Federal Highway Administration (FHWA) evaluated the best available remote ice sensors. Evaluation under laboratory and real highway conditions showed that the sensors at that time were not reliable nor accurate enough to operate warning signs for motorists on bridges. Numerous false alarms, missed detections of icing events, and hardware breakdowns were reported. The remote sensors tested used three different detection methods, conductivity/temperature, electrical capacitance/conductivity, and latent heat of fusion. The FHWA reported that the sensors using the conductivity/temperature method of ice detection showed the most potential for being improved to the point where they would be useful for remote ice detection on highways (Mawhinney, Lovell, and Ruden, 1975).

Brinkman (1977) pointed out that it is theoretically possible to detect ice on pavement by measuring spectral temperature, which is the product of actual temperature and surface emissivity. He concluded that such a detector would be superior to other available detectors because it would be capable of detecting ice over a large area rather than at a point, thus overcoming the detection problems associated with nonuniform distribution of the ice over the pavement surface.

Each day during the winter maintenance season, the maintenance engineer must decide whether or not to spread salt and sand on the roads in his maintenance area. Thornes, Wood, and Blackmore (1977) identified three distinct types of errors in the decision-making process:

1) meteorological errors in the forecast, 2) geographical errors due to topography and nonuniformity in the pavement surfaces, and 3) judgment errors by the maintenance engineer during the interpretation of the data. The authors suggest that better weather forecasts, thermal mapping, and increased awareness by the maintenance engineer would reduce these errors.

Ice-warning systems for highways are manufactured by several companies. These include Surface Systems Inc. (SSI) of St. Louis, MO, Findlay-Irvine Ltd of Great Britain, Vaisala of Finland, and Boschung of Switzerland. These systems are similar in many ways. All use sensors located in the pavement at select locations to monitor surface temperature, surface condition, i.e., wet or dry, and the presence of salt. In addition, several atmospheric sensors monitor air temperature, relative humidity or dew point, and wind speed and direction. The data from sensors are transmitted to a computer where they are processed and can be accessed at the request of the user.

According to studies of highway maintenance departments in several counties in Great Britain by Ponting (1984) and Chorlton (1986), the use of remote sensors and thermal mapping, as opposed to relying on weather forecasts alone, has resulted in a considerable cost savings. Chorlton (1986) reports additional benefits including increased safety and reduced application of salt.

Parmenter and Thornes (1986) explained that the usefulness of data from remote ice detectors is limited unless it can be used to predict surface conditions far enough in advance to allow maintenance crews time to take proper action. Computer models to predict the formation of ice on road surfaces have been developed. A numerical model was developed by the authors at the University of Birmingham. The computer model described as "essentially a heat balance model" used the data from the sensors and local weather forecasts to predict the temperature of the road surface. The model was tested for real-time forecasting during the winters of 1983-84 and 1984-85. The accuracy of the surface temperature forecasts were found to be limited only by the accuracy of the meteorological forecasts (Thornes 1985, Parmenter and Thornes 1986).

Thermal mapping has also been established as a useful tool in the prediction of surface temperature. A thermal mapping technique was developed in 1983 at the University of Birmingham. The technique uses a vehicle-mounted infra-red thermometer to show how road surface temperatures varied with road structure, local topography, and traffic flow. Results indicate that temperature variation along the road is systematic for a given weather condition and traffic flow. Using one or more sensor stations and a thermal map made under the appropriate conditions, road conditions for an entire area can be predicted. (Sugrue, Thornes, and Osborne, 1983). Thornes (1985) recommends that thermal mapping be used to determine locations for installation of remote ice detection sensors.

Parmenter and Thornes (1987) describe how a computer model can be used to combine data from road surface and atmospheric sensors with conventional weather forecasts to produce a prediction minimum pavement temperature. The forecasts for the selected sites can be interpreted over stretches of highway using a "thermal fingerprint" determined from thermal mapping. The authors conclude that thermal mapping and a computer prediction model linked to a few carefully located road sensors should provide the operational manager with a better predictive capability. Such a system was operated by Cheshire County Counsel. According to Faiz (1987), Cheshire Council reported winter maintenance cost savings of about 8 percent and, in addition, the maintenance engineer experienced less pressure to make the correct decision.

Weather radar images are used in Devon County, Great Britain, as part of their winter maintenance program. Chorlton (1986) credits elimination of unnecessary application of salt, better management of resources in emergencies, and improved advanced planning to the use of the weather radar information in conjunction with ice detection equipment. According to Chorlton:

> There is no doubt that the use of ice detection equipment and weather radar data as part of a comprehensive control and forecasting system can produce savings and provide an improved winter maintenance service. By themselves the data they produce is helpful but as part of a total management system weather

> monitoring equipment has a substantial positive contribution to make. (Chorlton 1986, p. 256)

Harverson (1985a, 1985b, 1986a, 1986b) reports that ICELERT ice warning systems by Findlay-Irvine Ltd are being used extensively by highway maintenance districts in Great Britain and have resulted in reduced salt application and other cost savings.

Ice detection and forecasting systems are currently being used many states and cities in the United States. In the United States, the system manufactured by SSI is the most commonly used. Several writers including Immordino (1984), Cosby (1986), Kelley, Trask, and Hunt (1987), Public Works (1987), Better Roads (1987), and Transportation Research News (1990) report that evaluations of these ice detection systems as have shown them to be reliable and cost effective. Maintenance engineers report increased efficiencies in the use of manpower and deicing chemicals. The most important benefit of a properly used system is increased safety.

Tackle (1990) developed an expert system for forecasting frost formation on bridges and roadways for the central part of the State of Iowa. The objective was to provide an 8-hour warning in advance of the formation of frost, which usually occurs shortly before sunrise. The system was based on rules derived from examination of meteorological conditions and Iowa department of Transportation frost reports for a period of 4 years including 2608 observations of bridge frost and 1615 observation of roadway frost. When predictions from the expert system were compared to those from a human forecaster, the forecaster and the expert system had comparable hit rates; but the expert system had a lower false alarm rate.

Pouliot and Wilson (1992) conducted a survey of motorists traveling on Interstate 80 between Laramie and Cheyenne, WY to evaluate the information needs of motorists during adverse winter travel conditions. The results indicated that motorists have generally consistent adverse winter travel information needs. Pavement conditions (dry/wet, slick in spots/slushy, or snow packed/icy), visibility (clear, limited, or very limited), and wind (calm/breezy or strong/gusty) were found to be the principal parameters. Pavement condition was perceived to be the most important by the motorists, and visibility the second most important. Local commuters most often sought road and travel information from the winter travel advisory phone. The primary source for interstate truckers was the CB radio. Changeable Message Signs (CMS) were indicated as an important source of information by almost 70% of the local commuters and 40% of the interstate truckers surveyed.

French and Wilson (1992) evaluated the effectiveness of remote weather information system (RWIS) systems for two main communication objectives: 1) to provide the road user information about the severity of the road and travel conditions so that the road user may determine whether or not to proceed with a potentially unnecessary trip, and 2) to provide the road user with adequate warning so that driving habits may be adjusted accordingly. The real time road and travel information can be communicated to road users using a variety of devices including changeable message signs. The authors found that the conditions described by the single RWIS on Interstate 80 between Laramie and Cheyenne did not relate to the overall conditions of the roadway as described by either the road users or the snow plow operators. The authors concluded the data from the RWIS should not be used as the sole source of information to determine poor road and travel conditions; however, more meaningful data might be provided if the RWIS was upgraded to include more weather sensor locations and visibility measurement devices.

Larson and Fleege (1989) conducted studies with Remote Weather Information Systems (RWIS) in Sweden, Finland, the United Kingdom, and the State of Wisconsin. They concluded that the existing systems were effective in reducing costs, increasing safety, and automating decisions, and they suggest that expansion in those areas would be beneficial. They further recommended that the State of Minnesota, which was considering the implementation of automated road information systems at that time, proceed immediately. They stressed, however, that management understanding and management commitment were the key to the success of the Road-Weather Programs, and without proper planning and long-term commitments the systems cost effectiveness, safety value, and ability to serve as a decision making tool would be hindered. In the four sites reviewed (Sweden, Finland, the United

Kingdom, and the State of Wisconsin) management training and planning was considered the key to the systems success. As a result, maintenance personnel felt more comfortable in their decisions, some reduced their travel time for call outs, and others eliminated their night patrol altogether.

An evaluation of the SCAN 16 EF Moisture, Frost and Ice Early Warning System (developed by Surface Systems Inc.) installed by the New Jersey Department of Transportation was conducted by Balgowan (1988). Four sites in Region IV containing remote processing units (RPU's) were used in the evaluation. A comparison of field collected data to the data supplied by the SCAN system showed that the data from the SCAN system was 91% accurate overall. The author also showed the system hardware, software and other associated equipment to be highly reliable as well. The author concluded that the system's greatest potential would be as a real time statewide roadway weather information system interactively shared with other agencies. In particular, he suggested aeronautics could use the information to assist pilots in developing flight plans, and data from the system could be used by the State Police to supplement their Emergency Management System. Also noted was a savings of 54,170 dollars for each spread saved statewide by reducing deicing chemical usage, improved weather forecasts for the State of New Jersey, reduced snow and ice related accidents, and a savings in manpower and equipment usage. A savings (statewide) of 11,500 dollars per hour could be realized for each hour saved by delaying crew call-outs and expediting crew releases.

Scholl and Rusnak (1988) evaluated the SCAN 16 EF system performance on the Medord Viaduct. The Medord Viaduct (located in Oregon) is a 3230-foot-long structure which carries Interstate 5 across Bear Creek and several city streets. As a result of two ice-related accidents occurring on the structure in December of 1984, the SCAN system was installed and evaluated over a period of two winters (1986-87 and 1987-88). Opinions of the installations success and reliability varied. Highway maintenance personnel, who use the system directly, found it to be somewhat useful in prioritizing their time while managing ice problems throughout the city, whereas the local Maintenance Supervisor did not believe that its benefits outweighed its costs. Overall, the state police were more satisfied with the system than the highway maintenance personnel.

As a result of the study, Scholl and Rusnak (1988) recommended that system reliability could be improved in several ways:

1) Adding more sensors to strategic locations. The most severe icing on the structure is caused by early morning shading of the deck by the median barrier - a location where there is no sensor.
2) Improved humidity sensors, now available, could make frost detection more reliable.
3) Additional training in using the system to open the way to benefits not previously understood or anticipated. Training could help users better predict how conditions are changing, and thus make better use of de-icing chemicals and crew dispatching.

Scholl and Rusnak are quick to point out, however, that the two winters the system was tested over did not include severe or frequent ice or snow conditions, and the evaluation of the systems accuracy, therefore, "is not conclusive." The concern that only 5 sensor sites may be inadequate to represent the worse condition on the entire 3200-foot structure was pointed out, and furthermore, Scholl and Rusnak stated the Medford installation of the SCAN system did not demonstrate the same level of reliability and accuracy experienced at other locations with similar equipment. The findings in this report suggest that there may be some unusual cases where special attention to installation and training is needed in order for the system to be fully effective at these extraordinary sites.

CHAPTER 3
DECISION-SUPPORT SYSTEM

Considering the list of essential features previously listed for the DSS, some are more suitable to a rule-base design and some to a procedural design. The decision-making and informative aspects of the application lend themselves to the rule-base approach. The sensor interfacing and data manipulation aspects are best suited to the procedural approach. We wanted a hybrid tool that provides the flexibility of both approaches.

We distinguished between two fundamental approaches:

1) formulating the application in a declarative architecture which utilizes and coordinates procedural modules; and
2) formulating the application in a procedural architecture which utilizes and coordinates rule-base objects.

The commercial shell and, to some extent, Prolog are declarative approaches that permit interfacing with procedural modules. We found significant problems with data transfer among modules. Because we were much more confident in our procedural than declarative programming skills, we decided to design a tool composed of a procedural "wrapper" which has the capability to instantiate multiple rule-base objects. Because our rule-base object obtains knowledge from three data sets, we named it "Tripod." Information exchange (data and commands) is accomplished by messages sent between the wrapper and the Tripod object (Grenney 1993).

RULE-BASE

Figure 2 illustrates the DSS application process. The terminology used in the following discussion evolved during interviews with the panel of experts. Terms were selected that provided the best communication with the experts. Three ASCII files, shown at the left of the figure are prepared by the experts and the developer.

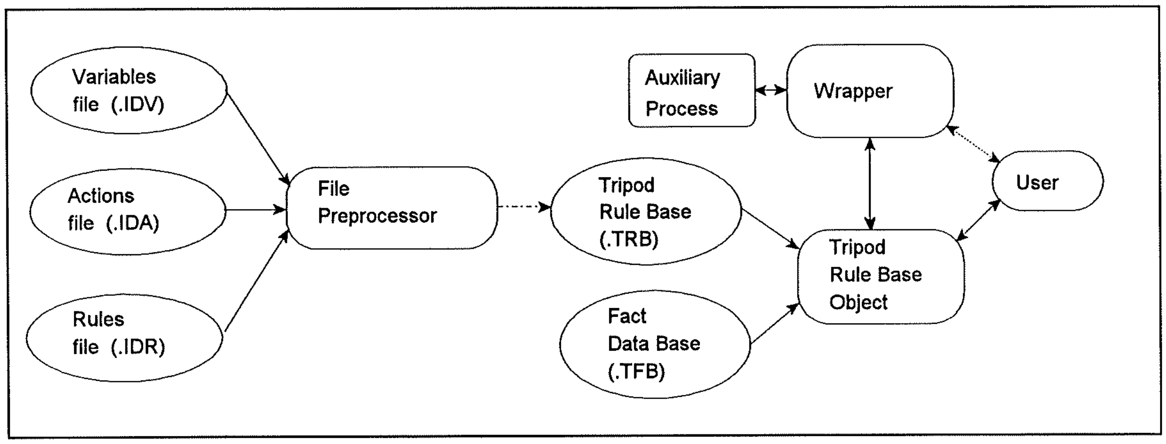


Figure 2. The Decision-Support System Application Process.

The Variables File

A variable is a quantifiable characteristic of the system. The state of the system at any time is defined by the values of the variables. The variable file (identified by the extension .IDV) contains a list of unique identification symbols and associated information for each variable, such as the type of the variable (i.e., integer, float, logical, menu, or string). Each variable also has a question associated with it that, when needed, will be displayed on the screen requesting a value from the user.

The Actions File

An action is a consequence which is invoked when an associated rule fires (e.g., evaluates true). The actions file (identified by extension .IDA) contains a list of unique identification symbols and associated information for each action, such as the type of action to be performed. An action may display text, display graphics, or perform an operation. A variety of operations may be performed by an action, for example, manipulating the values of variables, sending a message to the wrapper, and branching to a specified rule in the rule-base.

Rules and Tests

A rule is an IF-THEN statement specifying the appropriate action for a particular state of the system, as defined by the values of the variables. It has the following form:

IF (variable_1 == ruleValue_1) **AND** (variable_2 <= ruleValue_2) **AND** .... **THEN** (action_1)

The antecedent is made up of "tests," shown in parentheses. For a test, if the value of the variable (i.e. variable_1) satisfies the comparison (i.e., equals ruleValue_1), then the test is true. If all of the tests for a rule are true, then the rule is true and the action is invoked. If any one of the tests is false, then the rule is false. The rule file (identified by the extension .IDR) contains a list of unique identification symbols and associated information for each rule. Among other things, each rule has a list of tests which can include variables, actions, and other rules.

The Preprocessor

A preprocessor performs error checking on the three files and produces a rule-base file which can be efficiently read by the Tripod object. Changes to the rule-base file may be made only through the variables, actions, and rules files.

The Fact Data Base File

Figure 2 also shows a "Fact Data Base" file. This file contains initial values for variables in the rule-base. It is an ASCII file with a list of variable identification symbols, each followed by its initial value. The Fact Data Base may be updated any time prior to the instantiation of the Tripod object. Variables not listed in the Fact Data Base are automatically initialized to an "unknown" state by the object.

THE WRAPPER

Figure 2 shows the wrapper interacting with a Tripod object and an auxiliary process (such as a remote sensor). The wrapper is a procedural main program which instantiates and coordinates Tripod objects as well as performs auxiliary processes. The wrapper is the master of the application; it corresponds with Tripod objects by means of messages and gathers information to respond to the messages by means of procedural functions (auxiliary processes). The wrapper is tailored for the rule-base, or set of rule-bases, in a specific application. The wrapper must know how to respond to messages it receives. Messages are integer values. For example, action_message "2" from the Tripod object requests data from a remote sensor buffer. The wrapper often responds by sending a message containing the data back to the Tripod object.

THE TRIPOD OBJECT

When a Tripod object is instantiated by the wrapper, it fills its data structures from the Rule-Base File. It then initializes appropriate data member values from the Fact Data Base file, and returns control to the wrapper. The object is dormant until it receives a message from the wrapper to begin testing its rule-base at a specified rule. Rule testing is conducted by a member function in the Tripod object which is described later in this paper. Rules are tested in sequence until one fires. When a rule fires the associated action is performed by another Tripod member function, a "rule_fired" message is sent, and program control is returned to the wrapper.

PROGRAM PROCEDURE

Overview

Figure 3 illustrates the program procedure as a series of communication events between the wrapper and the Tripod object. The circled numbers in the figure correspond to numbers in parentheses in the following discussion.

The following sequence starts at the top left of Figure 3. The user executes the wrapper and the wrapper instantiates a Tripod Rule-base object (1). The object's constructor loads the rule-base file, initializes variables from the Fact Data Base file, and returns control to the wrapper (2). The wrapper sends a message to the Tripod object to begin testing rules in sequence, starting with a designated rule (3). The rule-testing algorithm in the object evaluates the tests associated with current rule. Most tests will request a response from the user as indicated in the figure.

Whenever the Tripod object request a response, the user has the option to send a message to the wrapper via the object. When this happens the object stores its current status, sends the message and returns control to the wrapper (4). After the wrapper responds to the message, which might be the execution of an auxiliary process for example, it sends a message to the object to resume from where it left off (5).

When a rule is true (i.e., all of its tests evaluate true) the associated action is invoked, and a "rule_fired" message is sent and control is returned to the wrapper (6). The wrapper typically requests the user to decide whether to continue or to terminate the run.

The Rule Engine

The Rule Engine is the member function of the Tripod object that evaluates the rules. It takes as a parameter the order number of a rule in the rule-base. It starts by evaluating the first test in the list of tests associated with that rule. If the first test evaluates true, it goes to the second, and so on. If all of the tests are true, the Rule Engine calls the function **execAction()**. This function takes as a parameter the pointer to an action, and performs the operation specified by the action. For example, it might replace the value of one variable with the value from another, or it might just display text on the screen. After **execAction()** returns, the Rule Engine stores the current status of the Tripod object, sends a "rule_fired" message and returns control to the wrapper.

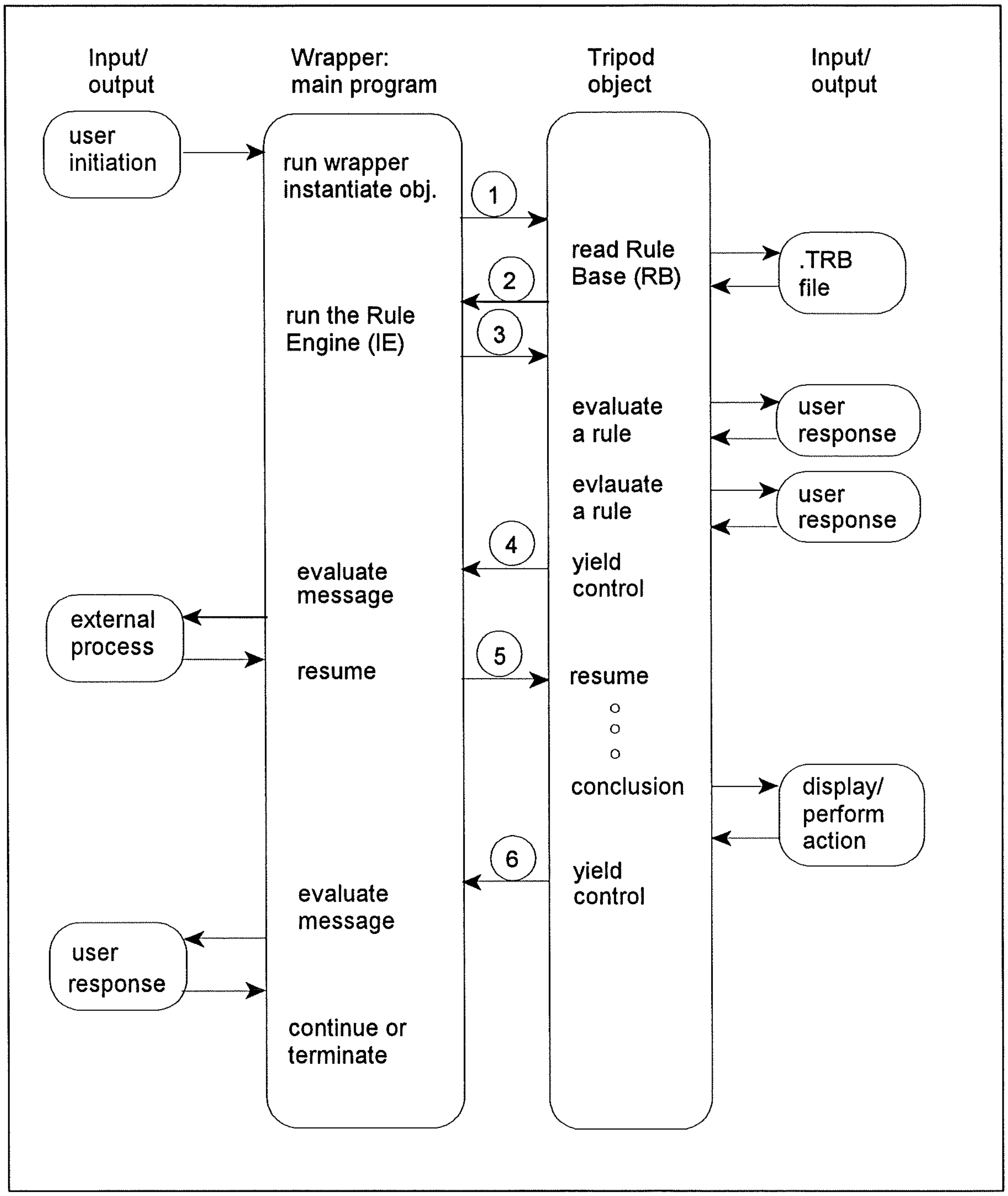


Figure 3. Program Procedure.

If a test evaluates false, the Rule Engine discontinues evaluating tests for the rule, marks the rule as false, and goes on to the next rule in the rule-base. This process continues until a rule is encountered in which all tests evaluate true.

Three types of tests are permitted: variable, rule, and action. The variable-type test provides the symbol for the variable and a value (or range of values). The Rule Engine first checks to see if the variable has been previously evaluated and if not calls a member function **query()**. This function displays the question associated with the variable, and compares the user's response with the test value (s). It returns a true or false flag to the Rule Engine depending on the results of the comparison.

A rule type test provides a pointer to another rule. The Rule Engine calls itself to evaluate this rule. If the rule is true then the test is true and vise versa. The Rule Engine operates recursively on nested rules to any reasonable depth.

An action type test provides a pointer to an action. The rule engine checks to see if the action has previously been invoked (i.e., associated with a rule that has already fired), and if not begins testing all of the rules associated with the action. If one of these rules fires, then the action is invoked and the test is true. If none of these rules fire, then the test is false.

Branching is handled by a special operation in an action. When an action is invoked by **execAction()** it may perform several types of operations. One type is to branch to a specified rule. This is accomplished by returning a message to the Rule Engine directing it to leave off what it has been doing and begin evaluating the specified rule. For example, if a nested rule fires and its action specifies a branch to another section of the rule-base, the Rule Engine does not finish evaluating the higher rules in the nest.

TRIPOD DATA STRUCTURES

The TRIPOD Class

Continuity is maintained by continually updating the Tripod data structures. The Tripod object retains the results of all previous operations. For example, it records which rules have fired, which actions have been invoked, which variables have been evaluated and what values have been assigned. This is facilitated by encapsulating the data for the actions, variables, and rules separately.

Figure 4 shows the most important data members in the class structure. The Tripod class contains **varAnchor, actAnchor,** and **rueAnchor** which are pointers to lists of variables, actions, and rules respectively. The data members **nbrVar**, **nbrAct**, and **nbrRue** are the number of entries in the respective lists. The data members **kbFile** and **kbTrace** are file handles for the rule-base input file and an output file to record a trace of the steps performed by the user. The other data members are for control purposes:

- **rueEngIdx[2]** records the indices of the principal rule (the highest in the hierarchy of nested rules) and the current nested rule being evaluated.
- **nestDth** is the depth of the current rule being evaluated within a set of nested rules.
- **nbrRueTsted** is the number of rules which have been tested during the current run.
- **curCode** is the latest message sent to the wrapper from the Rule Engine.
- **curRueKey** is the key (integer code) associated with current rule being evaluated in by the Rule Engine. The key is assigned by the developer in the rules file. It is often used to indicate a help or supplemental information file (i.e., 54.txt).
- **curVarKey** is the key (integer code) associated with the current variable being evaluated by the Rule Engine. The key is assigned by the developer in the variables file.
- **curActKey** is the key (integer code) associated with the current rule being evaluated by the Rule Engine. It is assigned by the developer in the actions file.

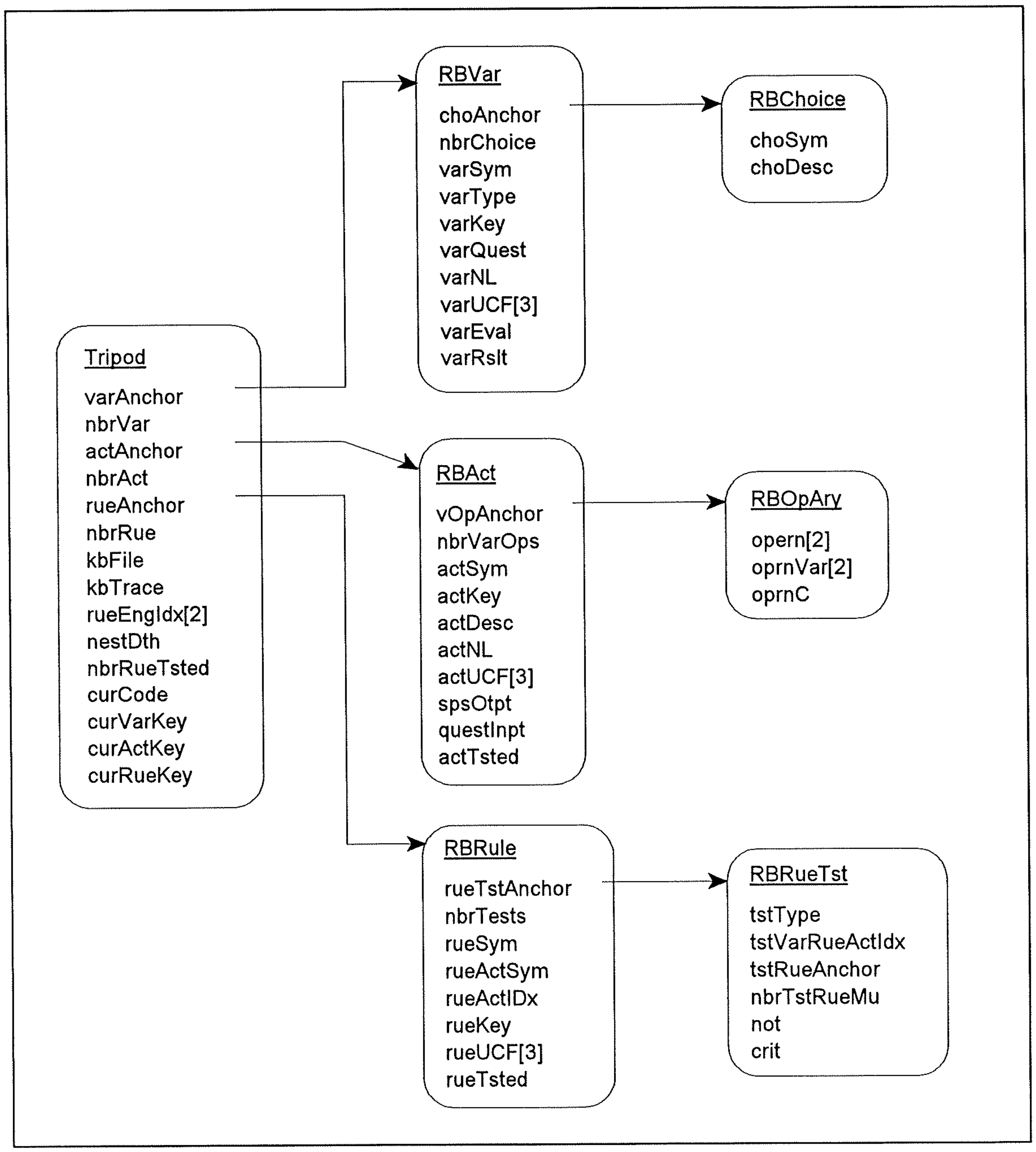


Figure 4. Tripod Data Structure.

The RBVar and RBChoice Classes

The **RBVar** (Rule-Base Variables) class contains data for each decision variable. The data member **choAnchor** points to a list of menu choices for a menu type of variable, and **nbrChoice** is the number of choices in the list. The **RBChoice** class contains the information for each menu item: **choSym** is a unique character string identification symbol and **choDesc** is a character string description for the item. **varSym** is a character string for a unique identification symbol for this variable. **varType** is the type of the variable which may be one of the following: menu, logical, string, integer, or floating point. **varKey** is the integer code assigned by the developer in the variables file to specify an auxiliary process.

An auxiliary process is performed by the wrapper. It is normally a function programmed by the developer to accomplish a specific tasks; for example, reading a remote sensor, instantiating another Tripod object with a different rule-base, or displaying a text file on the screen. A developer may wish to use key "54" to display text file 54.txt on the screen when the user requests help with the query for a particular variable. The developer assigns the value 54 to the key for the variable in the variable file. He/she then programs a function for the wrapper to respond to a "variable key" message having the value 54 by displaying 54.txt.

varQuest is the question for the variable and **varNL** is the number of lines in the question. **varUCF[3]** stores three arbitrary coefficients for the variable. Originally these coefficients were going to be used to represent uncertainty. However, this feature has not yet been implemented and the coefficients are unused. **varEval** is a flag indicating whether or not this variable has been previously evaluated. If it has been previously evaluated, **varRslt** stores the value. **varRslt** is a data union that stores a Boolean, string, float, or integer depending on the type of variable.

The RBAct and RBOpAry Classes

The **RBAct** (Rule-Base Actions) class contains data for each action. The data member **vOpAnchor** points to a list of operations that the action is to perform on decision variables, and **nbrVarOps** is the number of operations in the list. The **RBOpAry** class contains information for the operation. **opern[2]** stores two character symbols indicating the type of operation to be performed. **oprnVar[2]** contains the two operands. Three types of operations are currently available, and they are described in the next section.

Other data members in **RBAct** include **actSym**, **actKey**, and **actUFC[3]** which are analogous to **varSym**, **varKey**, and **varUCF[3]** previously defined. **actDesc** is a character string with the description of the action, and **actNL** is the number of lines in the description. **spsOtpt** is a flag to suppress display of the description. **actTsted** is a flag to indicate whether or not the action has been previously invoked. Setting this flag suppresses subsequent executions of this action. **questInpt** is a flag modifying the action. Three flags are permitted at this time, and they are described in the next section.

The RBRue and RBRueTst Classes

The **RBRue** (Rule-Base Rules) class contains data for each rule. The data member **rueTstAnchor** points to the list of tests for the rule, and **nbrTests** is the number of tests in the list. **rueSym** is a character string containing the unique identifier for the rule. **rueActSym** is the symbol of the action associated with this rule, and **rueActPtr** is a pointer to it. **rueKey**, **rueUCF[3]**, and **rueTsted** are analogous to **actKey**, **actUCF[3]**, and **actTsted**.

The **RBRueTst** class contains information for each test. **tstType** is a character indicating the type of test: "V" for variable, "R" for rule, and "A" for action. **tstVarRueActIdx** is the identifier of the variable, rule, or action for this test. **tstRueAnchor** is a pointer to a list of pointers to rules nested in this test. **nbrTstRueMu** is the number of nested rules in this test. **not** is a flag indicating whether NOT should be applied to the results of this test (i.e., true becomes false). **crit** is a data union for a Boolean, string, float, or integer. It contains the values which will cause the test to evaluate true.

CHAPTER 4
PROTOTYPE APPLICATION

After the TRIPOD library was created and tested, a prototype decision-support system was developed for winter highway maintenance. The user interface was tailored to accommodate the features available with the Surface Systems, Inc. (SSI) remote weather information systems (RWIS). These RWISs includes sensors for the following parameters:

1) Temperature at the pavement surface
2) Temperature at about 50 centimeters below the pavement
3) Wind speed and direction
4) Air temperature
5) Humidity
6) Precipitation (photocell to detect precipitation, but not type or rate).

Figure 5 shows the main window menu and the menu tree. The first menu command, "File," contains two items: "User Programs," and "Exit." The first item provides a list of independent programs and utilities that a user may wish to execute from this window. The second item exits the application.

The second menu command also contains two items. The "Operational Rule Base" is the decision-support system described later in this chapter. The "Tutorial" item is provided to allow for future incorporation of a self-learning system to teach new users how to use the application. The TRIPOD rule-base approach has been found to be an effective way to implement a computer-based learning module (Grenney and Senti 1992).

The third menu command is for the scan color graphics terminal (S.C.G.T.). It provides enhanced view, storage, and communications for the SSI RWIS data. The package can be set up to periodically call several Scan systems automatically, saving their data for later display. The "Scan Color Terminal" item automatically accesses the most recent data from the remote

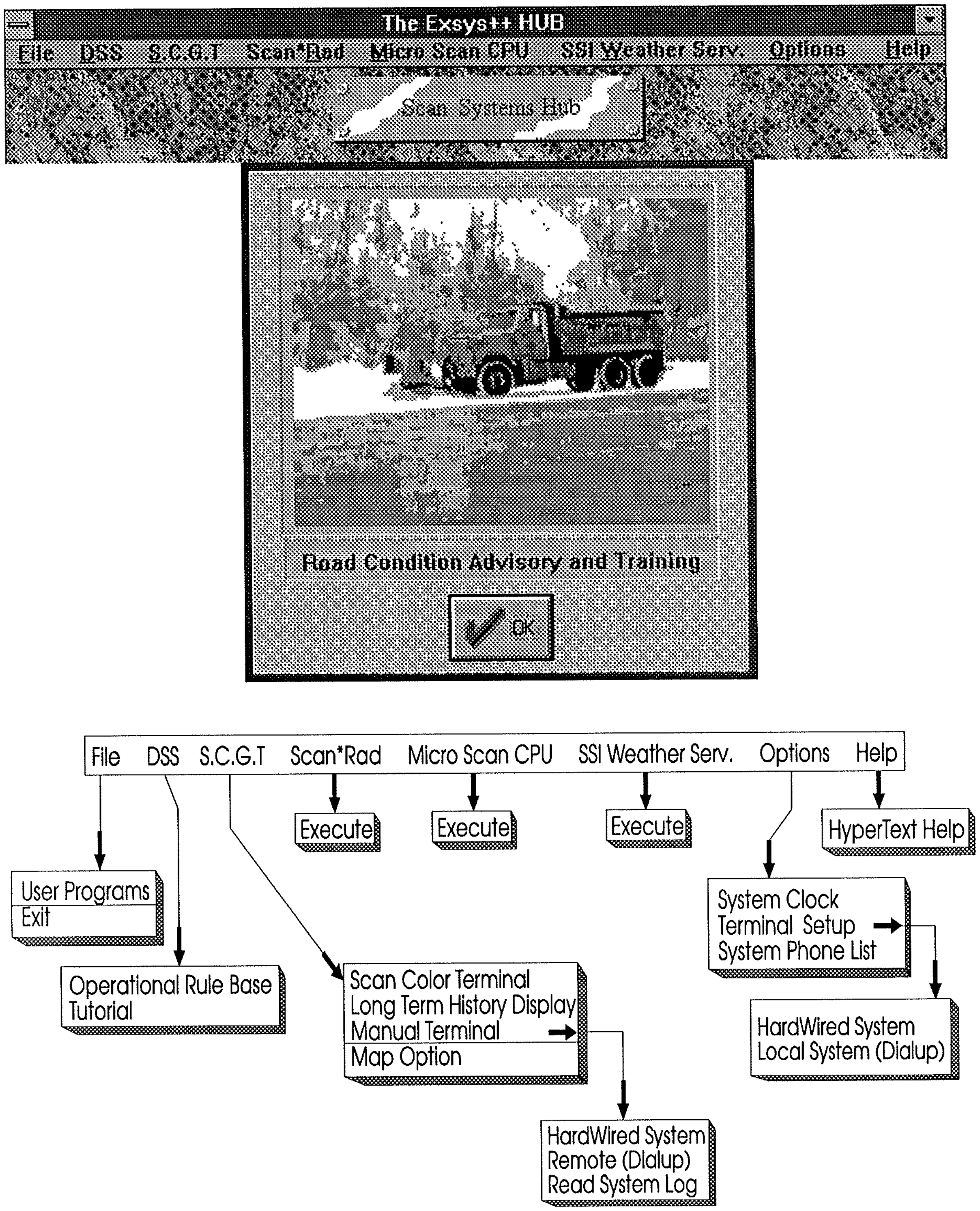


Figure 5. The Main Menu for the Decision-Support System Interface.

weather information system. The system log contains a list of the CPUs that have been accessed for data.

Scan*Rad is SSI's X-band color weather radar system. It consists of the actual radar unit and a software package that lets the user view the radar images from a remote location. The Micro Scan CPU menu command is provided to access SSI's software package that lets the user directly communicate with the remote sensors at the RWIS.

The SSI Weather Service is a weather-forecast service that users may subscribe to. It operates 24 hours per day during the winter from SSI's offices in St. Louis, and utilizes data many sources (including National Weather Service and RWIS data) to forecast pavement conditions at the subscriber's local site. The service is called SCAN-CAST. Historical data from SCAN-CAST was used to test the decision support system as described later in this chapter.

The last two commands in the main menu allow the user to set system parameters and to obtain online help regarding the use of the system.

RULE-BASE DEVELOPMENT

Decisions regarding the deployment of personnel and heavy equipment for snow and ice prevention and removal are made by the maintenance foremen in geographical districts. Districts range in size from a few hundred to several thousand square kilometers. Data available to the foremen include local and national weather forecasts, real time data from remote sensors, and personal observations. During a winter event, foremen rely upon these data in varying degrees and upon personal rules of thumb for dispatching crews and equipment, and for releasing crews near the end of the event.

During the first three meetings with the panel of experts, the scope of the problem was discussed and terminology was established that was meaningful to the members of the panel. The objective of the study was focused to prepare a simple prototype of a DSS that would help foremen make decisions about the dispatch and release of road crews during a winter event.

The prototype should provide table and graphic displays of data from remote sensors, and provide a step-by-step advisor for novice users (Grenney and Marshall 1991).

The importance of terminology used for communicating with the panel of experts should not be underestimated. Terms were established by the panel defining three categories of information: "variables," "actions," and "rules." Forms were developed to assemble information in these categories. These forms were the basis for the input files for the preprocessor.

The Variables File

The variables file contains the input data for the variables. Table 1 shows the input format for each variable in the file.

```
( variableSymbol )       M3    2      1      0.1     0.2     0.3
This line contains the question for the user.
choiceSymbol_1          First menu choice
choiceSymbol_2          Second menu choice
choiceSymbol_3          Third menu choice
```

Table 1. Input Format for the Variables File.

A unique symbol for the variable is enclosed in parentheses. The symbol may be up to 20 characters long. This is followed by the declaration of the type of variable: menu (M), string (S), logical (L), integer (I), or floating point (F). A menu-type declaration must be followed by the number of items in the menu list, as shown in Table 1. The next field contains the integer "key" code (2). This code is passed to the wrapper with a message from the Rule Engine when an external process is requested for the variable. The next item in the first row is an integer specifying the number of lines in the question. The last three fields (0.1, 0.2, 0.3) were originally intended for coefficients quantifying uncertainty, but this feature is not yet implemented.

The second row contains the question that will be displayed for the user when this variable is encountered in a rule test. If the variable is a menu type, as show in Table 1, the remaining rows contain the choices for the menu. If the variable is an integer of float type, only one row follows the question, and it contains the minimum and maximum permissible values for the variable.

Table 2 contains a partial listing of the variables identified by the experts. A key of "1" signals the wrapper to obtain and display information from the ScanCast weather forecasting system (SSI 1990). This procedure requires accessing a remote database over telephone lines. A key of "2" signals the wrapper to obtain current meteorological data from remote sensors at a highway station. This information includes air temperature, pavement temperature, wind speed, precipitation, relative humidity, and other pertinent data.

The Actions File

The actions file contains specific actions to be performed as a consequence of the state of the system. An action may be a screen display giving recommendations to the user, an operation to be performed on a set of variables, or a message to the wrapper to execute an external process. Table 3 shows the format for each action in the actions file.

A unique symbol for the action is enclosed in parentheses, followed by the key code. Three flags are permitted following the key:

1) /S suppresses the screen display of the action description.
2) /Q signifies "questionnaire" and will force all tests of a rule having this action to be evaluated. Normally, the evaluation of tests within a rule will be discontinued at the first false.
3) /O indicates that this action is to perform operations as specified by codes following the description line (s).

The next field in the first row specifies the number of lines in the description, and the last three fields are place holders for future development. The second row contains the description of the action. This text will be displayed to the user when this action is invoked unless the /S flag is

```
( expectPrecip )                 M3     2       1
Is there a possibility of precipitation in the next 3 hours?
rain            There is a possibility of rain in the next 3 hours.
snow            There is a possibility of snow in the next 3 hours.
no              No possibility of precipitation in the next 3 hours.

( expectDrift )         L       2       1
Is there a possibility of drifting in the next 3 hours?

( snoRateNow )          M3      1       1
What is the rate of snow fall now?
light           Light (less than 1 in/hr).
medium          Medium (between 1 and 2 in/hr).
heavy           Heavy (greater than 2 in/hr).

( hwyPriority )         I       0       1
What highway priority are you considering (1 to 4)?
1  4

( crewOnDuty )          M4      0       1
Is there a crew on duty?
noCrew          No crew is on duty.
nightCrew       A normal night crew is on duty.
skelCrew        A skeleton crew is on duty.
maintCrew       A maintenance crew is on duty.

( stormDirection )      M4      2       1
The storm approaches from which direction?
southEast       Storm from the southeast (moisture).
northWest       Storm from the northwest (winds).
west            Storm from the west (moisture).
southWest       Storm from the southwest (severe).

( timeOfDay )           F       0       1
What is the time to the nearest half hour (24 hour clock)?
0  24.5

( specialEvent )        M4      0       1
What type of special event is scheduled?
sport           Major sports event.
conv            Major convention.
holiday         Holiday celebration.
polit           Major political event.
```

Table 2. A Partial Listing of the Variables File (.IDV).

```
( actionSymbol )          57/S/Q/O      1       0.1      0.2      0.3
This line contains the description of the action.
=V   variableSymbol_1              variableSymbol_2
=C   variableSymbol_3              constantValue
B    ruleSymbol
```

Table 3. Input Format for the Actions File.

set. If an operation flag (/O) has been set, then additional rows are needed to specify the operations to be performed. Three types of operations are currently available:

1) "=V" means set the value of the first variable equal to the value of the second.
2) "C" means set the value of the specified variable equal to the value of the constant.
3) "B" means branch to the specified rule.

Table 4 contains a partial listing of the actions identified by the experts. Each action is made of a unique symbol and an expanded description. No keys were needed for this application. Blank fields in the input file are interpreted as null by the file preprocessor.

The Rules File

Rules associate the state of the system, as characterized by the values of the decision variables, with the actions to be performed. Each rule contains a list of tests. A test may be based on a variable, an action, or another rule. In its simplest form, the rule-base may be represent by a decision matrix where each row is a rule and each column is a test. The elements in a row contain the values which will cause the tests to evaluate true. An action is associated with each rule, and when a specific rule fires, it is this action which is invoked.

The experts used a decision matrix to establish the rules for this application. Each row (rule) in the decision matrix was represented in the input file as shown in Figure 5. A unique symbol for the rule is enclosed in parentheses, followed by the symbol for the action associated with this rule, the key code, and the three unused coefficients. The second row contains the description of the rule.

```
( Wait )                0       1
No decision necessary at this time.

( AlertCrew )                   0       1
Alert crews for possible dispatch in 2 to 3 hr.

( EmergencyAlert )      0       2
Emergency condition expected, alert contractors, and/or highway patrol.

( DispatchSklCrew )     0       1
Dispatch a skeleton crew.

( DispatchCrewSnd )     0       1
Dispatch a standard crew with sand trucks.

( DispatchCrewP/S )     0       1
Dispatch a standard crew with plow and sand.

( EmergencyAct )                0       1
Implement emergency actions: contractors, state patrol, etc.

( ReduceToSkel )                0       1
Reduce to skeleton crew.

( ReleaseCrew )                 0       1
Release Crew.

( AlertRelCrew )                0       1
Alert relief crew for possible dispatch in 2 to 3 hr.

( DispatchRelCrew )     0       1
Dispatch a relief crew to replace a standard crew.
```

Table 4. A Partial Listing of the Actions File (.IDA).

```
( ruleSymbol )             actionSymbol  111    0.1     0.2     0.3
This line contains the description of the rule.
V     variableSymbol_1               criteriaForTRUE
R     ruleSymbol_1                   ruleSymbol_2   ruleSymbol_3 ...
!A    actionSymbol                   ruleSymbol_4   ruleSymbol_5 ...
```

Table 5. Input Format for the Rules File.

Each row following the description describes a test for the rule. The first two characters in each row indicate the kind of test. A "!" as the first character indicates that the result of the test is to be reversed (i.e., true becomes false) after evaluation. Three kinds of tests are permitted:

1) "V" means to test a variable against the specified criteria.
2) "R" means to test the following list of rules in the specified order.
3) "A" means to test an action. A list of rule symbols (associated with the action) may follow to dictate the way in which the action is to be evaluated.

Table 6 contains a partial listing of the input file for the rules developed by the experts. The experts would first select an action, then decide which variables were important for decisions regarding the action, and, finally, what values of the variables would cause the action to be taken. In Table 6, integer and floating point variables are followed by a range of values. The variable will test true for any value within this range, and false otherwise. Menu variables are followed by a list of choices. Selection of any one of these items will cause the variable to test true.

PRELIMINARY TESTING

Most of the testing effort went into exercising the procedures in the preprocessor, the wrapper, and the Tripod object. During this process, many error traps and warnings were added in order to ensure consistency among the input files. The preprocessor prepares a list of all variable and action symbols not used in the rule file so that the developer can see if important elements have been inadvertently omitted from the rule-base. In the case where the preprocessor operates on two variables, the preprocessor checks to ensure that the variables are of the same type. If they are menu-type variables, the first variable must have at least as many menu items as the second and a warning is displayed if corresponding choice symbols are not identical.

The preprocessor also checks to see if the developer has used a rule in a test for that rule. For example, in a test containing several levels of nested rules, a developer could easily include the rule within a test for itself. Obviously such a mistake could result in runtime endless recursion, a situation that occurred on several occasions before this trap was added.

Next to displaying text for the user, the branch is the most useful action operation. We used it as a means to partition a rule-base. For example by evaluating a few rules first, we could branch to major sections of a rule-base to acquiring detail information about a particular subject. Also, we could design the rule-base to skip over major sections of rules which are no longer relevant after a particular action has been invoked. Branching can be used to construct loops in the rule-base; however, we did not have an immediate practical need for this construct. Like nesting rules, using branches must be done carefully in order to avoid run-time thrashing. We have not devised a technique for the preprocessor to detect inappropriate branching.

Figure 6 shows the standard interface screen for this application. A menu-type question is displayed in the figure. The user answers the question by clicking on the appropriate menu item. The appearance of the screen was designed over a period of several months by direct interaction with potential users. The buttons down the left side of the screen provide convenient access to the functions that were considered to be the most useful. By clicking on the "I Don't Know" button, the user can gain access to supplemental information that will help him to answer the question. For example, in the case of the question in Figure 6, the user will be supplied with the wind speeds and directions at the remote sensors of his choice. The next button, labeled with a "C," is the comment button. When clicked, this button will activate the user's word processor and will automatically insert the current question at the end of the existing text. The user can conveniently make notations and maintain a record of the steps that he went through to make a decision.

The next button is labeled "New." This button will terminate the current session and start a new session. The "Backup" button allows the user to backup to any previous step in the decision-support system. This is a convenient way for a user to changes his answers without completely starting over.

The main menu bar is shown at the top of Figure 6. The main menu commands are "File," "Preferences," "Action History," and "Help." The "File" command allows the user to save partially completed sessions, and come back later to pick up where he left off. The "Preferences" command provides the user with options for setting interface parameters. The "Action History" command provides the user with the option for automatically recording every step that is taken in the decision-making process. The "Help" command provides on-line help for the user.

During prototype testing, the wrapper and rule-base performed quickly and efficiency. The rule-engine utilizes pointers extensively, and can evaluate more that 100 tests without an observable delay between display screens when operating on a 25-MHz 386-class microcomputer.

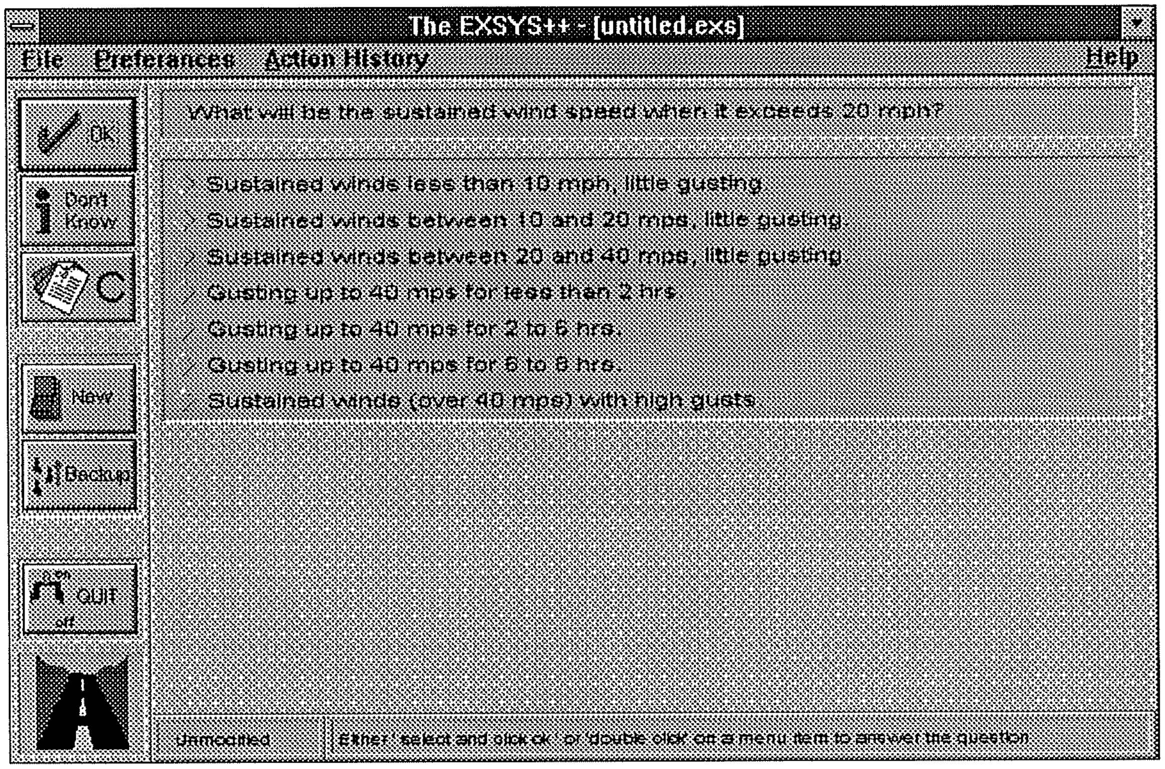
Preliminary testing of the prototype decision-support system by the experts led to some adjustments to the existing rules and formulation of additional rules. The experts found the three data sets (variables, actions, and rules) to be a practical approach for constructing the rule-base.

The prototype was used in a case study for the Denver, Colorado area. The Colorado Department of Transportation maintains records of major storm events, including weather forecasts prior to and during the storm, remote sensor data, and personnel and equipment allocations. Archived printouts of SCAN-CAST data for Denver Metro - South during the Spring of 1991 were uses in the study.

A nonexpert was given the historical data from two storms, and asked to make decisions regarding the dispatch and release of maintenance crews.

```
(102)                     Alert Crew      0
Put a crew on stand-by for snow
v     expectPrecip              snow
v     snowStart                 2  6
v     crewNotified              false
v     crewOnDuty                noCrew

( 103 )                   AlertCrew       0
Put a crew on stand-by for rain.
v     expectPrecip              rain
v     rainStart                 2  6
v     rainRateThen              mist     drizzle  rain
v     crewNotified              false
v     crewOnDuty                noCrew
v     pvmtFrzStart              2  6

( 109 )                   DispatchCrewP/S      0
Dispatch crew with plow and sand.
v     expectPrecip              snow
v     snowStart                 0  2
v     snoRateThen               light    medium heavy

( 110 )                   DispatchCrewSnd      0
Dispatch a standard crew with sand.
v     expectPrecip              rain
v     pvmtFrzStart              0  2
v     rainStart                 0  1

( 111 )                   DispatchCrewP/S      0
Dispatch crew with plow and sand.
v     expectPrecip              snow
v     snoRateNow                light    medium heavy

( 112 )                   DispatchCrewP/S      0
Dispatch crew with plow and sand.
v     expectDrift               true
v     windStart                 lt_1            1_to_3
v     windSusThen               10_to_20
v     SnoForDrift               10  100
```

Table 6. A Partial Listing of the Rule File (.IDR).



What will be the sustained wind speed when it exceeds 20 mph?

- Sustained winds less than 10 mph, little gusting.
- Sustained winds between 10 and 20 mps, little gusting.
- Sustained winds between 20 and 40 mps, little gusting.
- Gusting up to 40 mps for less than 2 hrs.
- Gusting up to 40 mps for 2 to 6 hrs.
- Gusting up to 40 mps for 6 to 8 hrs.
- Sustained winds (over 40 mps) with high gusts.

Unmodified | Either ' select and click ok ' or 'double click' on a menu item to answer the question

Figure 6. Standard Interface Screen.

Using the decision-support model, the non-expert reached the same conclusions as the expert. The prototype was used for a small sample of relatively simple situations. Expansion of the rule-base and additional testing of much larger sample sizes are needed in order to establish a useful decision-support system. Although the overall approach and the design of the graphical user interface were judged to be effective, the current rule-base is suitable only for prototype demonstrations.

Additional testing of the Tripod Object was conducted for four more applications. One was a screening tool for transportation agencies to quickly evaluate the appropriateness of a wide variety of hazardous waste remediation technologies for a specific waste at a specific site (Grenney and Penmetsa 1993, Penmetsa and Grenney 1993). This application demonstrated the coordination of three rule-bases, a database, and a simulation model by means of a wrapper and three Tripod objects. We found it to be more convenient to assemble three rule-bases which could be instantiated as needed, than to assemble one large rule-base.

The second use of Tripod was a conceptual application to reconcile non-graphic data with digital maps for use in Geographic Information Systems (O'Neill and Grenney 1991). In this case, there was very little need to solicit information from the user during a session. The actions operate on the values of the variables according to a set of predefined conditions which are nearly the same for all non-graphic data files and digital map files. We found this approach to be a viable alternative to programming a complex system of IF-THEN-ELSE statements directly in the code. The difference in computational time between in-line code and a rule-base object was significant but not excessive for this application.

The Tripod system can be effectively used to create self-study tutorials and self-administered exams. A prototype certification exam for the use of radar guns has been developed for the Utah Police Academy (Grenney and Senti 1992a). All police officers must be certified annually for the use of radar guns. This self-study computer program is intended to reduce the cost of providing field training for hundreds of police officers each year.

The forth application of Tripod was for the National Ecology Research Center in Fort Collins, CO (Grenney and Senti, 1992b and 1992c). The center has developed a manual for evaluating study plans for projects that alter river flows. The manual focuses on impacts to instream fish habitat. The manual is being implemented as a computer-based decision-support system for the U.S. Fish and Wildlife Service field offices.

CHAPTER 5

THE WEATHER INFO INTERFACE

Using Weather Service Information (WSI) to Meet a Specific Need

During meetings with the panel of experts, it was suggested that decision makers have greater and more direct access to weather information services. Direct access to weather information would allow them to obtain weather information at their convenience. Although few have formal meteorological training, it would also be an advantage for maintenance personnel to view radar and satellite images.

Weather forecasts for Utah are very general. A typical forecast issued by the National Weather Service or the news media will divide the state into northern and southern regions, with the entire area south of the Wasatch Front described as southern Utah. The forecasts for the mountains are usually limited to the range of snowfall expected within the region. Although weather forecasts are very general, actual conditions can be extremely variable because of differences in elevation and localized effects of topography. The direction from which a storm is approaching can greatly affect which areas receive precipitation, especially in mountainous areas. Such information often is not included in available weather forecasts. It may be useful to get the weather information directly to highway maintenance personnel, so they can use their knowledge of local terrain to determine what effect the forecasted weather conditions will have on the roads in their maintenance district. Two weather information services were considered for this study: 1) Surface Systems, Inc. SCAN-CAST (SSI 1990) and 2) WSI (WSI 1988).

Weather Brief and WXVIEW+ -

Both WXVIEW+ and WeatherBrief use special software which allows the user to access current weather information from a database on a remote computer. Both software packages provide modem communications for a personal computer, allowing the user to download desired weather information to his own computer. Both systems also charge for the time spent connected to the database computer.

WXVIEW+ requires that the user enter a series of the commands while the computer is on-line with the WSI's weather data base. These commands are quite complex and must be entered exactly making the system difficult to use for the average user. WXVIEW+ provides access to a wide variety of weather information including short- and long-term forecasts, weather data from National Weather Service weather stations, forecast maps, radar detected precipitation images, and satellite images. Information, including radar and satellite images in the database is updated every two hours,

WeatherBrief is menu-driven and easier to use, but the information available on the database using the menus is limited. Additional information can be obtained from the WeatherBrief database using interactive on-line commands similar to those used by WXVIEW+. WeatherBrief provides access to weather information including forecasts, weather data from recording stations, radar detected precipitation images, forecast maps, and cloud cover maps. Satellite images were not available using WeatherBrief. Weatherbrief graphics were not as detailed as those provided by WXVIEW+, but they did download in much less time.

It was determined that the WXVIEW+ weather information system was more suitable for providing weather information to UDOT maintenance personnel because the graphics are more detailed, satellite images are available in the database, and the system is compatible with other programs by WSI.

The WeatherInfo Interface

A new interface called "Weather Info" was developed for WSI's weather information service (WSI 1988, 1989). It is a menu-driven user interface for maintenance personnel to easily access weather information using WSI's WXVIEW+. This prototype was called WeatherInfo. WeatherInfo eliminates that need for users to type in detailed commands specifying the information that they want, and makes it easier to display data once it is downloaded.

Although the prototype was developed for use under a DOS environment, its components can easily be ported to other operating systems and environments. For instance, much of the successful components of the WeatherInfo program, particularly the menuing system, can easily be integrated into an object-oriented menuing system under Microsoft Windows or other graphical user interfaces. WeatherInfo was developed so it could be operated independently with WXVIEW+ (Robertson Software, 1988) or easily incorporated in a future knowledge-based system for overall management of de-icing and snow removal operations.

The program WeatherInfo is a menu-driven front end for WSI's WXVIEW+ weather information and communications software. Using menus, WeatherInfo allows the user to choose which weather information will be downloaded, download the data, and display downloaded data. By providing help for the user through a status line and on screen messages, the program is user- friendly and requires little computer experience to use. Figure 7 is a breakdown of the menu system used by the WeatherInfo Prototype.

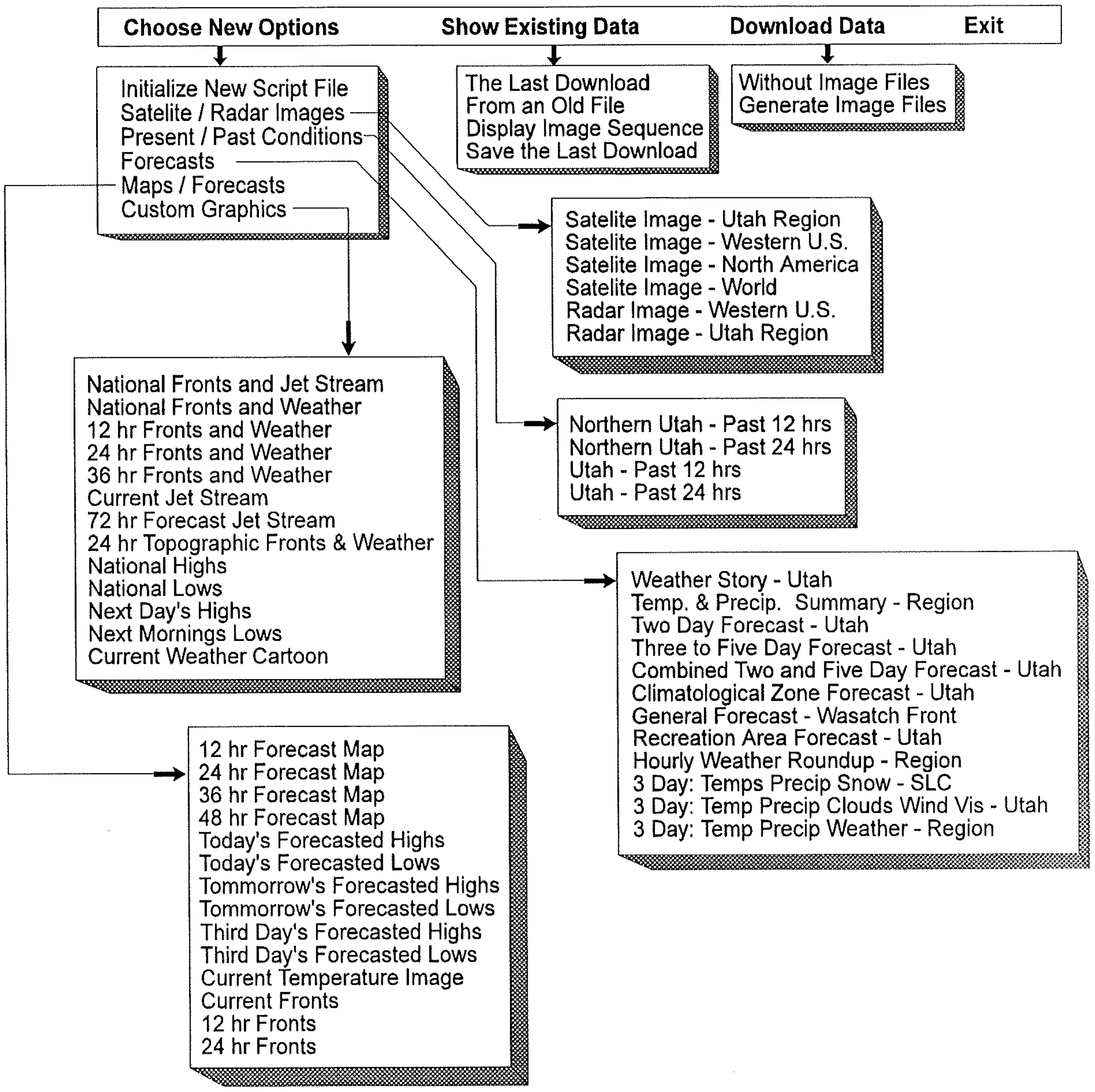

Choose New Options
Show Existing Data
Download Data
Exit
Initialize New Script File
Satelite / Radar Images
Present / Past Conditions
Forecasts
Maps / Forecasts
Custom Graphics
The Last Download
From an Old File
Display Image Sequence
Save the Last Download
Without Image Files
Generate Image Files
Satelite Image - Utah Region
Satelite Image - Western U.S.
Satelite Image - North America
Satelite Image - World
Radar Image - Western U.S.
Radar Image - Utah Region
Northern Utah - Past 12 hrs
Northern Utah - Past 24 hrs
Utah - Past 12 hrs
Utah - Past 24 hrs
Weather Story - Utah
Temp. & Precip. Summary - Region
Two Day Forecast - Utah
Three to Five Day Forecast - Utah
Combined Two and Five Day Forecast - Utah
Climatological Zone Forecast - Utah
General Forecast - Wasatch Front
Recreation Area Forecast - Utah
Hourly Weather Roundup - Region
3 Day: Temps Precip Snow - SLC
3 Day: Temp Precip Clouds Wind Vis - Utah
3 Day: Temp Precip Weather - Region
National Fronts and Jet Stream
National Fronts and Weather
12 hr Fronts and Weather
24 hr Fronts and Weather
36 hr Fronts and Weather
Current Jet Stream
72 hr Forecast Jet Stream
24 hr Topographic Fronts & Weather
National Highs
National Lows
Next Day's Highs
Next Mornings Lows
Current Weather Cartoon
12 hr Forecast Map
24 hr Forecast Map
36 hr Forecast Map
48 hr Forecast Map
Today's Forecasted Highs
Today's Forecasted Lows
Tommorrow's Forecasted Highs
Tommorrow's Forecasted Lows
Third Day's Forecasted Highs
Third Day's Forecasted Lows
Current Temperature Image
Current Fronts
12 hr Fronts
24 hr Fronts

Figure 7. WeatherInfo Menu System Interface.

<u>The Main Menu</u>

WeatherInfo's main menu is a line menu with four choices. Three of the main menu choices, Choose New Options, Show Existing Data, and Download Data, have corresponding pull-down menus. The item "Exit" on the main menu does not have a pull-down menu and is simply a command that exits the program.

Choose New Options

The six items on this menu allow you to choose the weather data that will be downloaded. The first item on the menu is a command that initialize a new script file. The remaining five items are categories of weather information. Choosing one of the five categories of weather information invokes a third level of menus containing weather items. The menus containing the weather items are multiple choice menus. This means that one or more of the weather items listed on the menu can be chosen.

Listed below are the selections in the "Choose New Options" pull-down menu and a brief description of their purpose:

Initialize New Script - This item is a command that will initialize a new script file. Initializing a new script file means that all previously selected weather items in the script file are erased and new weather items must be selected in order to download data. The script file is used to tell WXVIEW+ which items should be downloaded.

Satellite/Radar Images - This menu contains items which allows you to view radar and infrared satellite images for the Utah Region, western United States, and North America. See Figure 5 for a complete list of these menu items and their command line equivalents. Table 8 shows an image downloaded by selecting items from this menu.

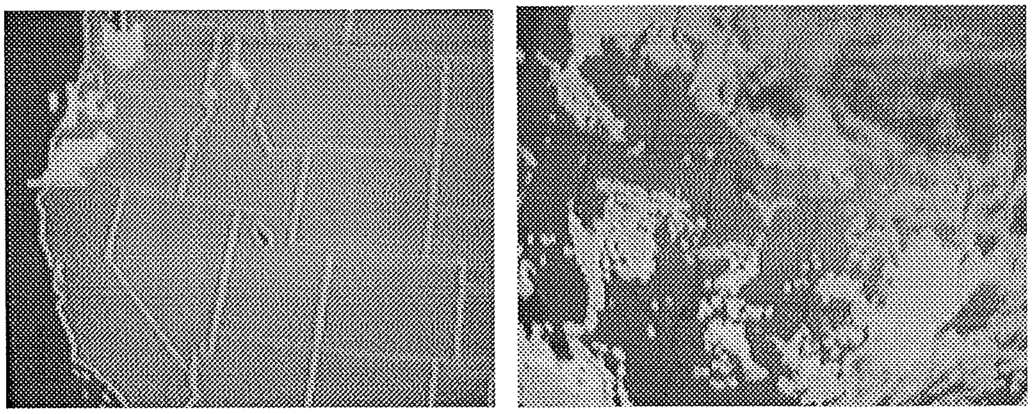


Figure 8. A Radar Image and a Satellite Image of the Utah Region Downloaded with WeatherInfo.

Present/Past Conditions - This menu contains items which give you access to weather data from Utah stations for the past 12 to 24 hours. See Figure 6 for a complete list of these menu items. Figure 3 is an example of present conditions for the entire United States.

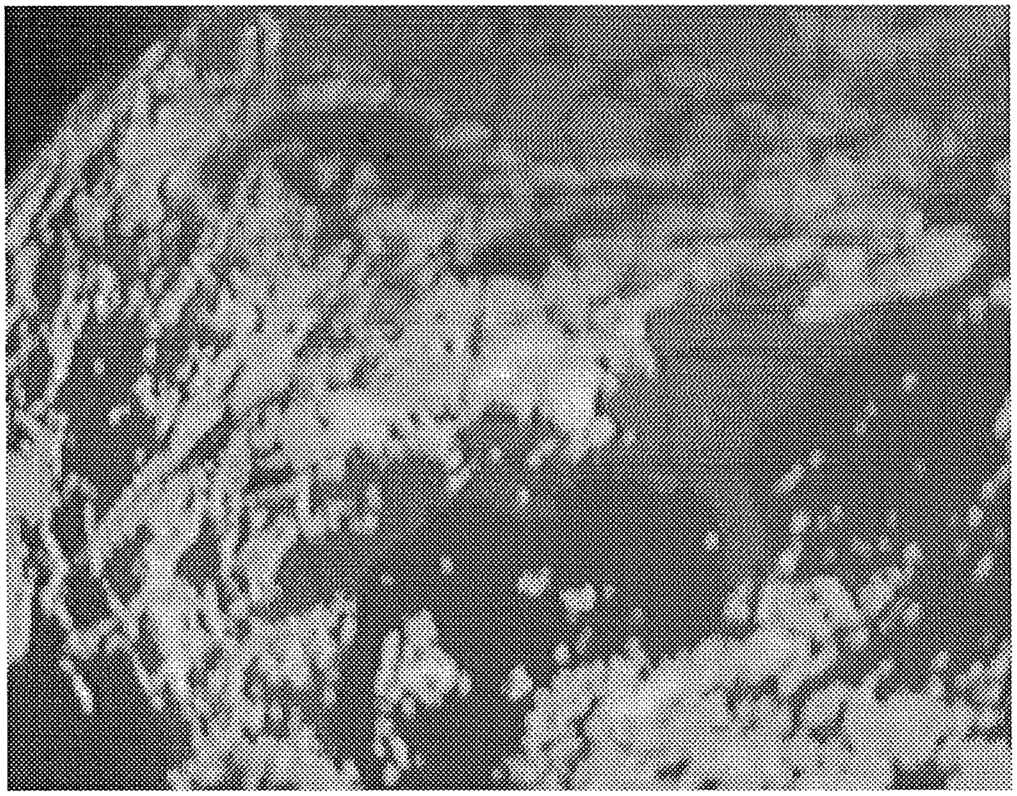


Figure 9. Present Conditions for the Entire United States.

Forecasts - The items on this menu are the National Weather Service Forecasts for Utah and the output from the FOUS forecast models. See Table 9 for a complete list of these menu items.

Maps/Forecasts - The items on this menu allow you to view graphics of forecast maps, forecasted and current frontal positions, forecasted high temperatures, and forecasted low temperatures. See Table 10 for a list of these menu items.

Custom Graphics - The items on this menu are special graphics of national weather forecasts and current conditions. WXVIEW+ charges extra to access items on this menu (See the WXVIEW+ Users Manual for details). See Table 11 for a complete list of these menu items.

Tables 7 through 11 list the specific weather information items available on the menus and gives the associated WSI command. By using the WeatherInfo interface, the need to memorize or know how to use these commands is removed. All functions can be performed by selecting a menu item instead.

Table 7. Satellite and Radar Information.

Menu Item	WSI Command
Satellite Image - Utah Region	SUPSAT SLC
Satellite Image - Western U.S.	SUPSAT WEST
Satellite Image - North America	SUPSAT GOES
Satellite Image - World	SUPSAT WORLD
Radar Image - Western U.S.	SUPSAT RADAR SLC
Radar Image - Utah	REGRAD SLC

Table 8. Present and Past Conditions.

Menu Item	WSI Command
Northern Utah - Past 12 hrs	USWX SLC,OGD,HIF,ENV 12-
Northern Utah - Past 24 hrs	USWX SLC,OGD,HIF,ENV 24-
Utah - Past 12 hrs	USINFO @UT TXAZGBZRZS 12-
Utah - Past 24 hrs	USINGO @UT TXAZGNZRZS 24-

61

Table 9. Forecasts.

Menu Item	WSI Command
Weather Story - Utah	STORY UT
Temp.& Precip. Summary - Region	STPDAY UT
Two Day Forecast - Utah	TWODAY UT
Three to Five Day Forecast - Utah	EXTEND UT
Combined Two and Five Day Forecast - Utah	STATE UT
Climatological Zone Forecast - Utah	ZONES UT
General Forecast - Wasatch Front	METRO SLC
Recreation Area Forecast - Utah	RECREA @UT
Hourly Weather Roundup - Region	WXRNDP @UT
3 Day: Temps,Precip,Snow -SLC	POPTYP SLC
3 Day: Temp,Precip,Clouds,Wind,Vis - Utah	MOSTYP SLC,ENV
3 Day: Temp,Precip,Weather - Region	USINFO SLC,CDC, BOI,RNO YAYBYCYE

Table 10. Maps and Forecasts.

Menu Item	WSI Command
12 hr Forecast Map	FAX 12PROG
24 hr Forecast Map	FAX 24PROG
36 hr Forecast Map	FAX 36PROG
48 hr Forecast Map	FAX 48PROG
Today's Forecasted Highs	SUPSAT HITEMP
Today's Forecasted Lows	SUPSAT LOTEMP
Tomorrow's Forecasted Highs	SUPSAT HITEMP T
Tomorrow's Forecasted Lows	SUPSAT LOTEMP T
Third Day's Forecasted Highs	SUPSAT HITEMP TT
Third Day's Forecasted Lows	SUPSAT LOTEMP TT
Current Temperature Image	SUPSAT TEMP
Current Fronts	SUPSAT FRNT
12 hr Fronts	SUPSAT FRNT12
24 hr Fronts	SUPSAT FRNT24

Table 11. Custom Graphics.

Menu Item	WSI Command
National Fronts and Jet Stream	SUPSAT CGMAP1
National Fronts and Weather	SUPSAT CGMAP2
12 hr Fronts and Weather	SUPSAT CGMAP3
24 hr Fronts and Weather	SUPSAT CGMAP4
36 hr Fronts and Weather	SUPSAT CGMAP5
Current Jet Stream	SUPSAT CGJET
72 hr Forecast Jet Stream	SUPSAT CG72HR
24 hr Topographic Fronts & Weather	SUPSAT CGTOP3
National Highs	SUPSAT CGHIGH
National Lows	SUPSAT CGLOW
Next Day's Highs	SUPSAT CGFHI
Next Morning Lows	SUPSAT CGFLO
Current Weather Cartoon	SUPSAT CGTDAY

Show Existing Data

The items on this menu allow you to view the weather data that you have previously downloaded. The first three items on this menu are commands which load WSI programs. The last choice, "Save Last Download," allows you to save the file you have most recently downloaded. If you do not choose "Save Latest Download," the file will be overwritten and the data lost the next time you download.

You have the choice of viewing the most recently downloaded data file or any other previously downloaded files which you have saved. A sequence of satellite images displayed in a loop can also be shown.

Listed below are the selections in the "Show Existing Data" pull-down menu and a brief description of their purpose:

The Last Download - Displays the weather information obtained during the most recent download session by loading the WSI program OLDWX.BAT and the file LATEST.WX.

From an Old File - Displays weather information from a file of your choice by first prompting for the filename and then loading the WSI program OLDWX.BAT and the file you specified.

Display Image Sequence - Displays satellite images of the Western United States (GOES WEST) in a loop by first prompting for the image filenames and then loading the WSI program WXACTION.EXE and the files which you specified.

Save Last Download - This command copies the information in LATEST.WX, the file created during the last download, to a new file, MM-DD-HH.WX; where MM, DD, and HH, are integers representing the month, day, and hour, according to your computer's internal clock.

Download Data

The two items on this menu are commands allowing you to download weather information by loading a WSI program. You can choose to download the data as a single file or to download the data and create a separate image file of the GOES WEST satellite image. Image files can later be displayed in a loop.

Listed below are the selections in the "Download Data" pull-down menu and a brief description of their function:

Without Generating Images - Downloads the requested weather data to a single file, LATEST.WX, by loading the WSI program NEWWX.BAT along with the script file COMMANDS.

Generate Images - Downloads the requested weather data to the file, LATEST.WX, and creates an image file of the GOES WEST satellite image by first issuing a DOS level command and then loading the WSI program NEWWX.BAT along with the script file COMMANDS.

The Status Line and Messages

When WeatherInfo is running, the bottom line on the screen is a status line. The status line instructs you which keys to push in order to perform specific tasks. For example, at the main menu, the status line instructs you how to select an item by displaying, "Select with arrows or use first upper case letter."

Messages are written to the screen to give instruction, prompt for input of filenames, and prevent users from inadvertently downloading data or exiting the program. When messages are written to the screen giving instruction, the message will remain until the you push a key. This information is shown by the status line reading, "Press any key." When messages are written to the screen prompting for filename input, the message will remain until you type in the requested filename(s) ending with a carriage return. Hitting the escape key when a filename is requested will cause the program to "crash." After selecting "Exit" or a command to download data, a message is written to the screen asking if this is really what you want to do. You must press the "Y" or "y" key to proceed with the selected command. Pressing any other key will cause the program to return to the menu. The status line will display, "Press 'Y' for YES or any other key for NO."

Choosing the Weather Information Items to be Downloaded

You do not need to choose new weather information items every time you want to download weather data. If you want to download the same weather items that you downloaded in a previous session, then simply download the data as you did before, without selecting the weather items again. If you wish to add items to your list, then simply choose the new items and they will be added.

Downloading Weather Information

After you have chosen the desired weather items, you can call the WSI computer and download the weather information to a file on your computer. Once you have downloaded the data, you can display it or get a printout at your convenience and as many times as you would like. You can also rename your downloads so that newer downloads will not overwrite existing ones. This enables you to create histories of what the weather has done. All downloads, regardless of time and name, can be viewed the WeatherInfo Utility.

The Tools Used to Create WeatherInfo

WeatherInfo was written in Borland's Turbo Prolog and incorporates several user interface tools from Turbo Borland's Prolog Toolbox. Prolog is a declarative language. This means that when given the necessary facts and rules, Turbo Prolog will use deductive reasoning to solve the programming problem. Prolog differs from traditional computer languages, such as Basic, FORTRAN, and Pascal, which are procedural languages. With procedural languages, the programmer must provide step-by-step instructions that tell the computer exactly how to solve the given problem. Prolog's declarative approach to problem solving makes the language efficient and powerful, especially for programming knowledge-based systems. A Prolog program for a given application will typically require fewer program lines as the corresponding program written in Pascal or C.

Evaluation and Use of WeatherInfo and WSI's Weather Information System

UDOT began using WSI's weather information system in January, 1990, for winter maintenance of the highways up Big Cottonwood Canyon and Little Cottonwood Canyon. Bjerregaard 1993), part of the responsibility of winter maintenance in the canyons is avalanche forecasting. The WSI weather information system was used to obtain weather information for purposes of general snow removal operations and dangerous avalanche conditions prediction.

UDOT personnel found that, in addition to convenient forecasts, the satellite images graphically showing relative cloud top temperatures were the most useful additional weather information supplied by WSI's weather information system. The satellite images indicate which direction forecasted storms are coming from and, therefore, help maintenance personnel anticipate the effect of local topography. For instance, storms approaching from the south generally produce very little precipitation in the Salt Lake Valley. Storms approaching from the northwest tend to produce the most snow along the benches on the east side of the valley.

One of the best uses for satellite images is to determine whether or not a storm is over. For example, the National Weather Service may have forecasted 2 to 4 inches of snow for the northern Utah valleys. The storm front may pass through and leave only about an inch of snow and partially clearing can be seen on the western horizon. Maintenance personnel realize that the snow-producing portion of a storm sometimes arrives with the front and sometimes arrives several hours after the front. After a front passes, "wait ???" must decide when to send the crews home. The "???" question they face is: has the storm exhausted itself or when a significant amount of snow will follow several "???" hours behind the front? Maintenance personnel must now decide if the storm has fizzled and snowfall was less forecasted to determine if the crews can be sent home or if they should remain on duty. Access to a current satellite image can help maintenance personnel determine if the storm is over or if the rest of the forecasted snow is just over the horizon.

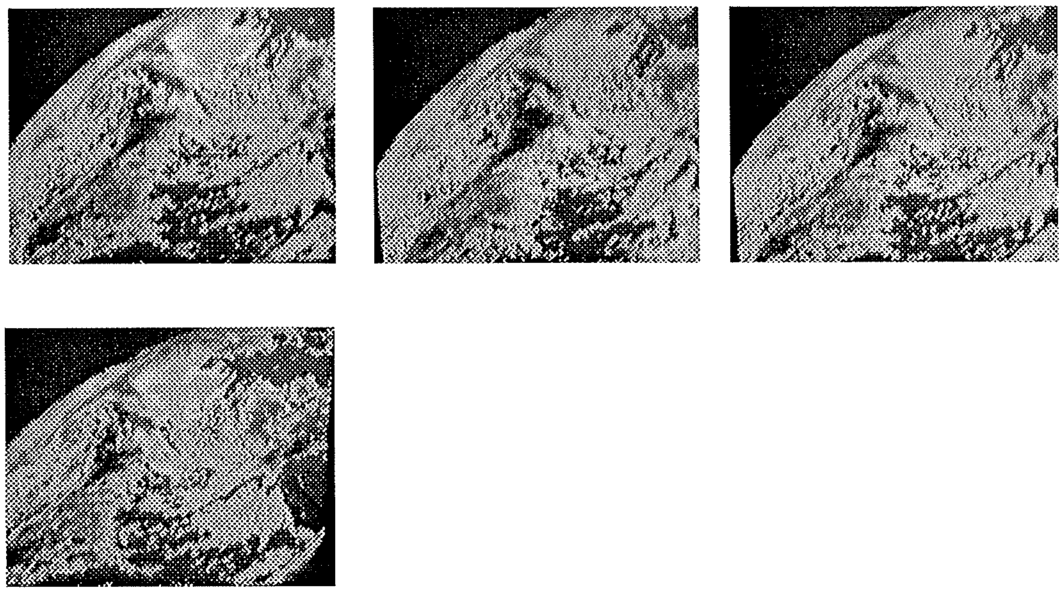
UDOT personnel were using WXVIEW, rather than WXVIEW+, to access WSI weather information. Although WXVIEW and WXVIEW+ are both WSI programs and operate essentially the same, WeatherInfo will only work with WXVIEW+. Therefore, a demonstration version of WeatherInfo was prepared and supplied to the maintenance personnel at shed (Bjerregaard 1993). This program operated independently of WSI's program and demonstrated how weather information items are specified, downloaded, and displayed with WeatherInfo. The demonstration program for WeatherInfo was evaluated by UDOT maintenance personnel in terms of its advantages over obtaining weather information using only WXVIEW. Those who evaluated WeatherInfo had varying computer use experience ranging from beginner to experienced.

Advantages of WeatherInfo

WeatherInfo was found to be easy for computer novices to use. Some of the personnel using WSI's weather information service have limited typing skills and do not want to remember all commands. Eliminating the need to enter commands saved on-line time and made the weather information service were economical to use.

It was also anticipated that WeatherInfo would reduce "on-line" time by reducing the tendency of some users to view the data as it is being downloaded. Because downloading is automatic and continuous, users are less likely to stop the process and view the data. The data can easily be recalled after it has been downloaded.

WXVIEW+ has the capability of creating special files of satellite images so they can viewed in a designated sequence (looped). Using this feature WXVIEW+ requires a detailed understanding of DOS file names. UDOT personnel indicated that the WeatherInfo's user interface makes looping satellite images much easier for the average user and that the looping feature was potentially very useful. Image files can be automatically created and displayed with menu choices in the user interface. Figure 10 shows an example of a sequence in a satellite loop that can be viewed using WeatherInfo.



Figures 10. A Sequence in a Satellite Image Loop.

Additional Enhancements

UDOT personnel who evaluated WeatherInfo had the following suggestions for improving the user interface:

1) Put the menu choices and WSI commands in an ASCII text file so the menus can be customized at a user level.
2) Modify the program so it will read and display the file names available for image files and past downloads. This allows the user to specify a series of satellite images for looping.
3) Add more options to the looping features. It was agreed that options to download the last 6, 12, and 24 hours of satellite images for the purpose of looping. Currently the user can only download the most recent satellite image. Images must be downloaded regularly and saved to use the looping feature.

CHAPTER 6

SUMMARY AND CONCLUSIONS

Summary

The objective of this study was to enhance the value of data received by maintenance foremen for making decisions regarding winter highway maintenance. More specifically, to develop a microcomputer-based Decision Support System (DSS) to facilitate the acquisition and interpretation of information from weather advisory services and from remote sensors at Remote Weather Information Systems (RWIS).

The project was a joint venture with Utah State University (USU), the Utah Department of Transportation (UDOT), and SSI Corporation, St. Louis MO. SSI provided a RWIS at reduced cost to UDOT, UDOT installed the equipment and provided technical assistance for the project, and USU developed new user interfaces based on expert system technology.

Three approaches were evaluated for developing the new interfaces: 1) a commercial expert system shell, 2) the Prolog programming language, and 3) the C++ programming language. Work with the shell was pursued for about two months and then abandoned. The shell worked very well for quickly establishing rules and it provided an impressive user interface. However, we were unable to communicate effectively with the software for the remote sensors.

We found Prolog to be a powerful programming language especially when used in conjunction with a commercial library for the graphical user interface (GUI). However, like the commercial shell, Prolog lacked the flexibility we needed to communicate effectively with the remote sensors. An application was developed using Prolog to interface with the WSI weather information service.

We found C++ to be the best tool for developing a knowledge-base decision support system to interface with the SSI remote weather information system. An application was developed in C++ that provides a new prototype menu-driven user interface and links it to a knowledge-base decision support system.

A panel of experts was formed with maintenance foremen from Utah and Colorado who had experience with winter highway maintenance. The computer interfaces were designed according to their recommendations. The experts created the knowledge-base for the SSI decision-support system and evaluated the preliminary applications. The SSI program was also tested by giving a non-expert historical data from two storms, and asking him to make decisions regarding the dispatch and release of maintenance crews. The non-expert reached the same conclusion as the expert for a small sample of relatively simple situations. Although the overall approach and the design of the graphical user interface were judged to be effective, the current rule-base is suitable only for prototype demonstrations.

The WSI weather information system interface was evaluated by maintenance personnel and judged to be very satisfactory. It was obvious from the interchange with the experts that the use of a graphical user interface is necessary in order to effectively use existing weather services by eliminating the need for the user to memorize complex command syntax.

Conclusions

A C++ class called Tripod and a file preprocessor were developed to provide a tool for developing knowledge-base decision support systems. The approach outlined in this report was found to provide several advantages over traditional procedural programming for this application:

1) The benefits of a rule-base decision making structure (a Tripod object) and the benefits of procedural functions (the wrapper) can be effectively combined.

2) Multiple small rule-bases can be implemented by the wrapper as an alternative to one large rule-base.
3) The application logic can be modified and expanded without changing the program code, and numerical algorithms can be modified or added without changing the rule-base.
4) The approach utilizes memory efficiently and permits interfacing with third-party programs which often reset environmental variables, dominate memory, and return unique status codes.

Based on the Tripod class, a prototype knowledge-base model was developed with the capability to interface with remote sensors and external meteorological databases. The model was applied to produce a prototype decision-support system for winter highway maintenance in the Intermountain West.

The rule-base for the model was developed by interaction with a panel of domain experts. The model was tested on a small sample of historical data and responded favorably. It has the potential to help an inexperienced person make reasonable decisions consistent with those of an experienced person in similar situations.

The Weather Info program in its present state, was considered satisfactory by the maintenance personnel who tested the prototype. It was obvious from this study that a highly intuitive application maximizes existing programs by removing the need for the user to memorize and to use complex command syntax. The solution is to place the commands in a more sophisticated user interface, most often with the use of menus and graphics, where each command is replaced by a meaningful menu description rather than a cryptic command. Furthermore, by implementing the suggestions offered by the maintenance personnel, the program will fit to what end users consider an applicable tool.

Although the original program was written using Borland Prolog, recent advancement in object-oriented technology suggest that rewriting the program under C++ would provide greater flexibility and portability to other operating environments. The rise of GUIs (Graphical User Interfaces) has brought in a new standard of user interfaceability and, with this new standard, comes the need to update DOS and text-based programs. WeatherInfo fits well into this category because it already contains much of the menuing system that would be used in a graphics environment such as Microsoft Windows. Porting the program to C++ would allow it to be integrated into various program schemes with little manipulation or reimplementation. Furthermore, with the added useability of a GUI, particularly the use of global clipboards, use of the mouse as an input device, and multimedia functions such as digitized sounds and video, the application becomes a much more substantial source for information. Nearly all GUIs are inherently multi-tasking environments as well. This allows for several programs to run asynchronously, therefore allowing programs like WeatherInfo to interact with word processors, digitizers, and spreadsheets. The added flexibility of a modularized program (using C++) in conjunction with a highly expandable GUI, allows WeatherInfo to adapt to new functions as needs change. It is essential that the proper languages and delivery tools be used so as not to render a product obsolete within months of its implementation.

REFERENCES

Adeli, H. (1990). *Knowledge Engineering*, McGraw-Hill, New York, NY.

Balgowan, R. M. (1988). Evaluation of the Accuracy, Reliability, Effectiveness, Expandability, and Additional Potential Benefits of the SCAN 16 EF Moisture, Frost and Ice Early Warning System. Final Report, New Jersey Department of Transportation, Division of Construction and Maintenance, Trenton, NJ. 22p.

Better Roads (1987). Bridge Alert. *Better Roads,* May, Pg. 26-29.

Bjerregaard, J. D. (1993) Use of a Weather Information System for Management of Winter Highway Maintenance, Master of Science thesis, Utah State University (in publication).

Borland International (1988). *Turbo Prolog User's Guide*, Scotts Valley, CA.

Borland International (1990). *Borland C++ Programmer's Guide*, Scotts Valley, CA.

Brinkman, C.P. (1977). Highway ice detection. *Transportation Engineering,* v47, n 11, Nov., p 34-36.

Chorlton, E. (1986). Use of weather monitoring equipment in the management of highway maintenance. *Municipal Engineering,* v3, n5, Oct., p. 245-256.

Cosby, D. R. (1986). Virginia installs SCAN ice detector. *Better Roads,* v56, n6, Jan., p. 60-61.

Faiz, O. (1987). Ice prediction package cuts cost on Cheshires Gritting Operation. *Highways* (Corydon, England), v55, n1927, Apr., p. 21-22.

French, K. A., and E. M. Wilson (1992). Remote Sensing Rural Rad and Travel Conditions. Presented at the 32nd Annual .ITE Intermountain Section Meeting. Jackson, WY. 17p.

Grenney, W. J. (1993). A C++ Class for Rule-Base Objects. *Scientific Programming.* (in publication for April 1993).

Grenney, W. J. and H. N. Marshall (1991). "Prototype Rule-Based Decision support System for Winter Highway Maintenance," In *Artificial Intelligence and Civil Engineering,* International Conference on the Application of Artificial Intelligence Techniques to Civil and Structural Engineering, Oxford, England, pg. 71-75.

Grenney, W. J, and R. K. Penmetsa (1993). Computer Methodology for Transportation Agencies to Screen Technologies for Hazardous Waste Remediation. Mountain-Plains Consortium. North Dakota State University, Fargo, ND. 112p.

Grenney, W. J., and T. E. Senti (1992a). POSAT: Police Officer Standards and Training. Manual for the Prototype Radar Certification Tutorial and Exam. Center for Applied and Advanced Transportation Studies, Utah State University, Logan, UT. 27 p.

Grenney, W. J., and T. E. Senti (1992b). Users Manual for the SPECS Study Plan Evaluation & Checking System. National Ecology Research Center, Fort Collins, CO. 17p. (preliminary).

Grenney, W. J., and T. E. Senti (1992c). Description of the Knowledge-Base for the SPECS Study Plan Evaluation & Checking System. National Ecology Research Center, Fort Collins, C). 25p.

Harverson, D. (1985a) Ice warning systems on British roads. *Highways* (Corydon, England), v53, n1900, Apr., p. 26-27.

Harverson, D. (1985b) Ice warning systems: Communication or control? *Highways* (Corydon England), v53, n1907, Nov., p. 8-9.

Harverson, D. (1986a) Ice warning system cut the cost of winter maintenance. *Surveyor* (Sutton England), v166, n4877, Jan., p. 8-9.

Harverson, D. (1986b) Ice warning systems - the met men join in. *Highways* (Corydon, England), v54, n1917, Apr., p. 12-14.

Immordino, K. M. (1984) New Jersey evaluates ice detection system. *Public Works* 115, p. 66-68, Dec.

Information Builders, Inc. (1986). Level 5 Expert System Software User's Manual. Information Builders, New York, NY.

Kelley, J. R., D. C. Trask, and O. M. Hunt. 1987. Pavement sensors: a crystal ball for road authorities. *Traffic Safety* 87, p. 6-11, Nov./Dec.

Larson, D., and E. Fleege (1989). Road-weather instrumentation recommendations for snow and ice control at Minnesota Department of Transportation. Fact Finding Study Report and Recommendations. Minneapolis, MN.

Marshall, H. N. (1993). Microcomputer Interface for a Remote Weather Information System. Master's thesis, Utah State University, Logan, UT. (in preparation).

Mawhinney R. C., C. C. Lovell, and R. J. Ruden. (1975). Snow and ice detection and warning systems. Final Report No. FHWA-RD-76-25. Federal Highway Administration.

O'Neill, W. A., and W. J. Grenney (1991). "Prototype Knowledge-Base Model for Matching Non-Graphic Road Inventory Files with Existing Digital Cartographic Databases," Proceedings of the 29th Annual Conference of the Urban and Regional Information Systems Association, San Francisco, CA.

Parmenter, B. S., and J. E. Thornes (1986). The use of a computer model to predict the formation of ice on road surfaces. Research Report 71, Transport and Road Research Laboratory (Crowthorne, Berkshire, England), 21 p.

Parmenter, B. S., and J. E. Thornes (1987). New technology for the winter maintenance of roads. *Municipal Engineering* v4, n1, Feb., p. 7-14.

Penmetsa, R. K. and W. J. Grenney (1993?). A Computer Methodology for Screening Technologies for Hazardous Waste Remediation, *American Society of Civil Engineers, Environmental Division.* (in publication).

Ponting, M. (1984). Weather prediction systems. *Journal of the Institution of Highwaysand Transportation*, Nov., p. 24-32.

Pouliot, S. G., and E. M. Wilson (1992). Motorist Information Needs and Changeable Sigh Messages for Adverse Winter Travel. Presented at the 32nd Annual ITE Intermountain Section Meeting, Jackson, WY. 23p.

Public Works (1987). Forewarned is forearmed: Winter weather data from the field. *Public Works* 118:52-3, Jul., p. 52-53.

Robertson Software (1988). *WX-View: A Weather Graphics Terminal*, Geneva, IL.

Scholl, L. G., and J. S. Rusnak (1988). Medford Viaduct Ice Detection System. Experimental Feature Final Report OR-85-03. Oregon State Highway Division, Salem, OR. 14p.

SSI (1990). Equipment manufactured by Surface Systems, Inc. Saint Louis, MO. 1989.

Sugrue, J.E., J.E. Thornes, and R. D. Osborne (1983). Thermal mapping of road surface temperatures. *Physical Technology*, v14, p. 212-213.

Tackle, E. S. (1990). Bridge and Roadway Frost: Occurrence and prediction by use of an expert system. *Journal of Applied Meteorology.* v29, n8, p. 727-734.

Thornes, J. E. (1985). Prediction of ice formation on roads. *Highways and Transportation* v32, n8, Aug-Sep, p. 3-12.

Thornes, J., L. Wood, and R. Blackmore (1977). To salt or not to salt? *New Scientist, February* 10, p. 327-328.

Transportation Research News (1990). Wisconsin's winter weather system. *Transportation Research News*, March-April, p. 22-23.

Turban, E. (1990). *Decision Support and Expert Systems: Management Support Systems*, Second Edition, MacMillan Series in Information Systems.

Walters, J., and N. R. Nielsen (1988). *Crafting Knowledge-Based Systems*. John Wiley & Sons, New York, NY.

WSI (1988). The WSI *Corporation Manual for Weather Service Information*, Bedford, MA. 40p.

WSI (1989). *User Manual for SSI Weather Service Information.* WSI Corporation, Bedford, MA. 21p.

APPENDIX

```
/**********************************************************************

                    WEATHERINFO

            *** A MENU DRIVEN FRONT END ***
          *** FOR WSI WEATHER INFORMATION ***

                  BY JOHN BJERREGAARD


   This program is written in Borland's Turbo Prolog.  It uses
   tools from Borland's Turbo Prolog Toolbox to help create the
   menus and the status line.  In order to compile this program,
   the files listed in the include section of this program must
   be present.  In order to run this program the following steps
   must be completed:

       - Install WXVIEW+ on drive C
       - Copy NEWWX.BAT to the directory C:\WSIWX
       - Copy OLDWX.BAT to the directory C:\WSIWX

   In order to create and view image files, the following steps
   must be completed:

       - Copy WXACTION.BAT to the directory C:\WSIWX
       - Copy IMAGE.BAT to the directory C:\WSIWX
       - Make a new directory C:\WSIWX\WEST

**********************************************************************/
/*********************************************************************
   Declare the Domains, Database, and Predicates, and include
   the required Toolbox Files
**********************************************************************/

include "tdoms.pro"

DATABASE

  pdwstate(ROW,COL,SYMBOL,ROW,COL)

include "tpreds.pro"       \*****  From Turbo Prolog Toolbox ****/
include "status.pro"       \*****  From Turbo Prolog Toolbox ****/
include "pulldown.pro"     \*****  From Turbo Prolog Toolbox ****/
include "menu.pro"         \*****  From Turbo Prolog Toolbox ****/
```

81

```
PREDICATES

 msg(STRING)
 write_list(INTEGER,INTEGERLIST)
 write_choices(INTEGER,INTEGERLIST)
 getcommand(INTEGER,INTEGER,STRING)
 get_date_hr(STRING,STRING,STRING)
 puttogether(STRING,STRING,STRING,STRING)
 getfilename(STRING,STRING)
 question(STRING,CHAR)
 download(CHAR)
 quit(CHAR)
 initialize

/*********************************************************************
   The Goal Section creates a line-menu with four initial
   choices.  It also creates subchoices on drop down menus
   corresponding to each initial choice.  The only way the
   goal can fail is if the <escape> key is pushed when the
   program prompts for input from the keyboard.
*********************************************************************/

GOAL

/*
          1         2         3         4         5         6         7
01234567890123456789012345678901234567890123456789012345678901234567890123456789012345
  Choose New Options    Show Existing Data    Download Data    Files
*/
 makewindow(1,80,15,"",0,0,24,80),
 makestatus(112," Select with arrows or use first upper case letter"),
 pulldown(113,
    [curtain(3,"Choose New Options",
               ["Initialize New Command File",
                "Satelite/Radar Images",
                "Present/Past Conditions",
                "Forecasts",
                "Maps/Forecasts"]),
     curtain(26,"Show Existing Data",
               ["The Last Download",
                "From an Old File",
                "Display Image Sequence"]),
     curtain(49,"Download Data",
               ["Without Generating Images",
                "Generate Images"]),
     curtain(67,"Files",
               ["Rename Latest Data File",
              "Quit" ])],CH,SUBCH),
    write("\n    CH = ",CH),
    write("\n SUBCH = ",SUBCH),nl.

CLAUSES

\*************************************************************************
  The clause "pdwaction(CH,SUBCH)" is bound when the user makes
  a choice from the line menu.  The variable "CH" is bound to an
  integer from 1 to 4 depending upon the choice on the line menu.
  The variable "SUBCH" is also bound to an integer depending upon
  the users choice in the drop down menu.
*************************************************************************/


 pdwaction(1,1):-
        initialize,
        msg("      New options must now be chosen \n
         in order to Download Data.").

 pdwaction(1,2):-
        changestatus(" F10:End   Cr:Select or Delete  Esc:Abort"),

\*********************************************************************
  "Menu_mult" creates a box menu from which the user can choose
  a one or more of the items that are listed.  The choices are
  saved and handled by the program as a list.
*********************************************************************/

        menu_mult(7,31,49,113,["SAT Image - West Hemishere",
                              "SAT Image - North America",
                    "SAT Image - Western U.S.",
                              "RADAR Image - Western U.S.",
                              "RADAR Image - Utah Region"],
                    "Make Your Choices",[],Choices),

        openappend(df,"C:\\WSIWX\\REQUEST\\COMMANDS"),
        write_choices(1,CHOICES),
        closefile(df),
      changestatus(" Select with arrows or use first upper case letter").

  pdwaction(1,3):-
        changestatus(" F10:End   Cr:Select or delete  Esc: abort"),

        menu_mult(7,31,49,113,["12 Hours",
                                 "24 Hours",
                                 "48 hrs",
                                 "72 Hours"],
```

83

```
                "Make Your Choices",[],Choices),
        openappend(df,"C:\\WSIWX\\REQUEST\\COMMANDS"),
        write_choices(2,CHOICES),
        closefile(df),
      changestatus(" Select with arrows or use first upper case letter").

   pdwaction(1,4):-
        changestatus(" F10:End   Cr:Select or delete  Esc: abort"),

        menu_mult(5,31,49,113,["General Forecast - Utah",
                               "Temp.& Precip. Summary - Region",
                               "Two Day Forecast - Utah",
                               "Three to Five Day Forecast - Utah",
                               "Combined Two and Five Day Forecast - Utah",
                               "Climatological Zone Forecast - Utah",
                               "General Forecast - Wasatch Front",
                               "Recreation Area Forecast - Utah",
                               "Hourly Weather Roundup - Region",
                               "3 Day: Temps,Precip,Snow -SLC",
                               "3 Day: Temp,Precip,Clouds,Wind,Vis - Utah",
                               "3 Day: Temp,Precip,Weather - Region"],
                                 "Make Your Choices",[],Choices),
        openappend(df,"C:\\WSIWX\\REQUEST\\COMMANDS"),
        write_choices(3,CHOICES),
        closefile(df),
      changestatus(" Select with arrows or use first upper case letter").


   pdwaction(1,5):-
        changestatus(" F10:End   Cr:Select or delete  Esc: abort"),

        menu_mult(5,31,49,113,["12 hr Forecast Map - Western U.S.",
                               "24 hr Forecast Map - Western U.S.",
                               "36 hr Forecast Map - Western U.S.",
                               "48 hr Forecast Map - Western U.S.",
                               "Today's Forecasted High Temps - U.S.",
                               "Today's Forecasted Low Temps - U.S.",
                               "Tommorrow's Forecasted High Temps - U.S.",
                               "Tommorrow's Forecasted Low Temps - U.S.",
                               "Third Day's Forecasted High Temps - U.S.",
                               "Third Day's Forecasted Low Temps - U.S.",
                               "Current Temperature Image - U.S.",
                               "Analyzed Fronts - U.S.",
                               "12 hr Forecasted Fronts - U.S.",
                               "24 hr Forecasted Fronts - U.S."],
                                 "Make Your Choices",[],Choices),
        openappend(df,"C:\\WSIWX\\REQUEST\\COMMANDS"),
        write_choices(4,CHOICES),
        closefile(df),
    changestatus(" Select with arrows or use first upper case letter").

pdwaction(2,1):-
        system("OLDWX"),
    changestatus(" Select with arrows or use first upper case letter").

pdwaction(2,2):-
        changestatus("Push 'Enter' to see the latest downloaded data"),
        getfilename("Type in the filename (MM-DD-HH.WX):", OLDFILE),
        concat("OLDWX ",OLDFILE,COM),
        system(COM),
    changestatus(" Select with arrows or use first upper case letter").

pdwaction(2,3):-
        changestatus(" Type the filenames or use wild cards
                (examp.- 10FEB?.00Z ,*.00Z)"),
        getfilename("List the Images: ",OLDFILE),
        concat("WXACTION ",OLDFILE,COM),
        system(COM),
    changestatus(" Select with arrows or use first upper case letter").

pdwaction(3,1):-
        changestatus("Press 'Y' for YES or any other key for NO"),
        question("    'Download Data' was selected ! \n
                Are you sure ? (Y/N)" ,Answer),
        download(ANSWER),
        changestatus(" Select with arrows or use first upper case letter").

pdwaction(3,2):-
        system("C:\\WSIWX\\IMAGE"),
        changestatus("Press 'Y' for YES or any other key for NO"),
        question("    'Download Data' was selected ! \n
                Are you sure ? (Y/N)  ",Answer),
        download(ANSWER),
        changestatus(" Select with arrows or use first upper case letter").

pdwaction(4,1):-
      msg("The current LATEST.WX file will become: \n\n
          MONTH-DAY-YEAR.WX (MM-DD-YY.WX)"),
        get_date_hr(MM,DD,HH),
        puttogether(MM,DD,HH, FILENAME),
        concat("copy c:\\wsiwx\\latest.wx c:\\wsiwx\\",FILENAME,RENAME),
```

85

```
        system(RENAME),
     changestatus(" Select with arrows or use first upper case letter").
 pdwaction(4,2):-
        changestatus("Press 'Y' for YES or any other key for NO"),
        question("\n Do you really want to Quit ? (Y/N)  ",Answer),
        quit(ANSWER).


\**********************************************************************
   "Msg" writes a message to the screen to give the user an
   instruction or a warning.
**********************************************************************/


 msg(S):-
     makestatus(112,"Press any key"),
        makewindow(2,113,112," Message ",16,18,5,44),
        window_str(S),
        readkey(_),
        removewindow,
        removestatus.


\*******************************************************************
   "Question" prompts the user for input.
*******************************************************************/


 question(S,CH1):-
        makewindow(2,113,112," Question ",16,20,4,40),
        window_str(S),
        readchar(CH1),
        removewindow.


\******************************************************************
   "Download" activates NEWWX which dials WSI and downloads the
    data if the argument is bound to 'y' or 'Y'.
******************************************************************/


 download('y'):-
        system("NEWWX COMMANDS",1,_),
        msg("    A new data file has been downloaded. \n
           Select 'Show Existing Data' to view"),!.

 download('Y'):-
        system("NEWWX COMMANDS",1,_),
        msg("      A new data file has been downloaded. \n
            Select 'Show Existing Data' to view."),!.
```

86

```
 download(_).  /* Do nothing if the user has not entered a 'y' or 'Y *\


\*************************************************************************
   "Quit" exits the program if the argument is bound to 'y' or 'Y'.
*************************************************************************/

 quit('y'):- removewindow,removewindow,removewindow,exit.
 quit('Y'):- removewindow,removewindow,removewindow,exit.
 quit(_):- changestatus("Select with arrows or use first uppercase letter").


\*************************************************************************
   "Write_choices" and "Write_list" together write the commands
    necessary to download the data chosen by the user.  These
    commands are written to a script file called "Commands".
*************************************************************************/

 write_choices(_,[]) :- msg("\n  None of the options were selected !!!").
 write_choices(CH1,CHOICES) :- write_list(Ch1,Choices).

 write_list(_,[]).          /* If the list is empty, do no more*/
 write_list(CH1,[H|T]) :-
        getcommand(CH1,H,COMMAND),
        writedevice(df),
        write(COMMAND),nl,
        write_list(CH1,T).


\*************************************************************************
   "Get_date_hr" returns the integers representing the month, day
    and hour, according to the computer's internal clock.
*************************************************************************/

 get_date_hr(MM,DD,HH):-
        date(_,MONTH,DAY),
        time(HOUR,_,_,_),
        str_int(MM,MONTH),str_int(DD,DAY),str_int(HH,HOUR).

\*************************************************************************
   "Puttogether" creates a filename from the integers - MM, DD,
    and HH, returned by the predicate  "get_date_hr".
*************************************************************************/
```

```
 puttogether(MM,DD,HH,FILENAME):-
        concat("-",DD,TEMP1),
        concat(TEMP1,"-",TEMP2),
        concat(MM,TEMP2,TEMP3),
        concat(TEMP3,HH,TEMP4),
        concat(TEMP4,".WX",FILENAME).

\*********************************************************************
   "Getfilename" prompts the user to input a filename.
*********************************************************************/

 getfilename(INSTRUCTION,OLDFILE):-
        makewindow(3,113,112,"User Input",12,15,3,50),
        window_str(INSTRUCTION),
        readln(OLDFILE),
        removewindow.


\*********************************************************************
   "Initialize" creates a new script file called "Commands".
    The file is empty until the user chooses the data to be
    downloaded.
*********************************************************************/
  initialize:-
        openwrite(df,"C:\\WSIWX\\REQUEST\\COMMANDS"),
        closefile(df).
\*********************************************************************
   "Getcommand" returns the WSI command corresponding to the
    menu choice.  The command is written in the script file
    "Commands".
*********************************************************************/
 getcommand(1,1,"SUPSAT WORLD").
 getcommand(1,2,"SUPSAT GOES").
 getcommand(1,3,"SUPSAT WEST").
 getcommand(1,4,"SUPSAT RADAR SLC").
 getcommand(1,5,"REGRAD SLC").
 getcommand(2,1,"USWX SLC,OGD,HIF,ENV,DPG,PVU,U16,VEL 12-").
 getcommand(2,2,"USWX SLC,OGD,HIF,ENV,DPG,PVU,U16,VEL 24-").
 getcommand(2,3,"USWX SLC,OGD,HIF,ENV,DPG,PVU,U16,VEL 48-").
 getcommand(2,4,"USWX SLC,OGD,HIF,ENV,DPG,PVU,U16,VEL 72-").
 getcommand(3,1,"STORY UT").
 getcommand(3,2,"STPDAY UT").
 getcommand(3,3,"TWODAY UT").
 getcommand(3,4,"EXTEND UT").
 getcommand(3,5,"STATE UT").
 getcommand(3,6,"ZONES UT").
 getcommand(3,7,"METRO SLC").
 getcommand(3,8,"RECREA @UT").
 getcommand(3,9,"WXRNDP @UT").
 getcommand(3,10,"POPTYP SLC").
 getcommand(3,11,"MOSTYP HIF,SLC,ENV").
 getcommand(3,12,"USINFO SLC,CDC,BOI,PIH,RNO YAYBYCYDYEYFYGYHYI").
 getcommand(4,1,"FAX 12PROG").
 getcommand(4,2,"FAX 24PROG").
 getcommand(4,3,"FAX 36PROG").
 getcommand(4,4,"FAX 48PROG").
 getcommand(4,5,"SUPSAT HITEMP").
 getcommand(4,6,"SUPSAT LOTEMP").
 getcommand(4,7,"SUPSAT HITEMP T").
 getcommand(4,8,"SUPSAT LOTEMP T").
 getcommand(4,9,"SUPSAT HITEMP TT").
 getcommand(4,10,"SUPSAT LOTEMP TT").
 getcommand(4,11,"SUPSAT TEMP").
 getcommand(4,12,"SUPSAT FRNT").
 getcommand(4,13,"SUPSAT FRNT12").
 getcommand(4,14,"SUPSAT FRNT24").


/*******************************************************************
   Turbo Prolog Toolbox
   TPREDS.PRO
*******************************************************************/

PREDICATES
 nondeterm repeat

CLAUSES
 repeat.
 repeat:-repeat.

/*******************************************************************/
/*                  miscellaneous                                  */
/*******************************************************************/

PREDICATES
 maxlen(STRINGLIST,COL,COL)            /* The length of the longest string */
 listlen(STRINGLIST,ROW)          /* The length of a list              */
 writelist(ROW,COL,STRINGLIST)            /* used in the menu predicates          */
 reverseattr(ATTR,ATTR)           /* Returns the reversed attribute   */
 min(ROW,ROW,ROW)            min(COL,COL,COL)              min(LEN,LEN,LEN)
min(INTEGER,INTEGER,INTEGER)
 max(ROW,ROW,ROW)            max(COL,COL,COL)              max(LEN,LEN,LEN)
max(INTEGER,INTEGER,INTEGER)
```

89

```
CLAUSES
 maxlen([H|T],MAX,MAX1) :-
         str_len(H,LENGTH),
         LENGTH>MAX,!,
         maxlen(T,LENGTH,MAX1).
 maxlen([_|T],MAX,MAX1) :- maxlen(T,MAX,MAX1).
 maxlen([],LENGTH,LENGTH).

 listlen([],0).
 listlen([_|T],N):-
         listlen(T,X),
         N=X+1.

 writelist(_,_,[]).
 writelist(LI,ANTKOL,[H|T]):-
         field_str(LI,0,ANTKOL,H),
         LI1=LI+1,
         writelist(LI1,ANTKOL,T).

 min(X,Y,X):-X<=Y,!.
 min(_,X,X).

 max(X,Y,X):-X>=Y,!.
 max(_,X,X).

 reverseattr(A1,A2):-
         bitand(A1,$07,H11),
         bitleft(H11,4,H12),
         bitand(A1,$70,H21),
         bitright(H21,4,H22),
         bitand(A1,$08,H31),
         A2=H12+H22+H31.


/******************************************************************/
/*      Find letter selection in a list of strings              */
/*    Look initially for first uppercase letter.                */
/*    Then try with first letter of each string.                */
/******************************************************************/

PREDICATES
 upc(CHAR,CHAR)  lowc(CHAR,CHAR)
 try_upper(CHAR,STRING)
 tryfirstupper(CHAR,STRINGLIST,ROW,ROW)
 tryfirstletter(CHAR,STRINGLIST,ROW,ROW)
 tryletter(CHAR,STRINGLIST,ROW)
```

90

```
CLAUSES
 upc(CHAR,CH):-
        CHAR>='a',CHAR<='z',!,
        char_int(CHAR,CI), CI1=CI-32, char_int(CH,CI1).
 upc(CH,CH).

 lowc(CHAR,CH):-
        CHAR>='A',CHAR<='Z',!,
        char_int(CHAR,CI), CI1=CI+32, char_int(CH,CI1).
 lowc(CH,CH).

 try_upper(CHAR,STRING):-
        frontchar(STRING,CH,_),
        CH>='A',CH<='Z',!,
        CH=CHAR.
 try_upper(CHAR,STRING):-
        frontchar(STRING,_,REST),
        try_upper(CHAR,REST).

 tryfirstupper(CHAR,[W | _],N,N) :-
        try_upper(CHAR,W),!.
 tryfirstupper(CHAR,[_ | T],N1,N2) :-
        N3 = N1+1,
        tryfirstupper(CHAR,T,N3,N2).

 tryfirstletter(CHAR,[W | _],N,N) :-
        frontchar(W,CHAR,_),!.
 tryfirstletter(CHAR,[_ | T],N1,N2) :-
        N3 = N1+1,
        tryfirstletter(CHAR,T,N3,N2).

 tryletter(CHAR,LIST,SELECTION):-
        upc(CHAR,CH),tryfirstupper(CH,LIST,0,SELECTION),!.
 tryletter(CHAR,LIST,SELECTION):-
        lowc(CHAR,CH),tryfirstletter(CH,LIST,0,SELECTION).


/*****************************************************************/
/* adjustwindow takes a windowstart and a windowsize and adjusts */
/* the windowstart so the window can be placed on the screen.    */
/* adjframe looks at the frameattribute: if it is different from */
/* zero, two is added to the size of the window                  */
/*****************************************************************/

PREDICATES
 adjustwindow(ROW,COL,ROW,COL,ROW,COL)
 adjframe(ATTR,ROW,COL,ROW,COL)
```

```
CLAUSES
 adjustwindow(LI,KOL,DLI,DKOL,ALI,AKOL):-
            LI<25-DLI,KOL<80-DKOL,!,ALI=LI,AKOL=KOL.
 adjustwindow(LI,_,DLI,DKOL,ALI,AKOL):-
            LI<25-DLI,!,ALI=LI,AKOL=80-DKOL.
 adjustwindow(_,KOL,DLI,DKOL,ALI,AKOL):-
            KOL<80-DKOL,!,ALI=25-DLI, AKOL=KOL.
 adjustwindow(_,_,DLI,DKOL,ALI,AKOL):-
            ALI=25-DLI, AKOL=80-DKOL.

 adjframe(0,R,C,R,C):-!.
 adjframe(_,R1,C1,R2,C2):-R2=R1+2, C2=C1+2.


/******************************************************************/
/*                    Readkey                                     */
/* Returns a symbolic key from the KEY domain                     */
/******************************************************************/

PREDICATES
 readkey(KEY)
 readkey1(KEY,CHAR,INTEGER)
 readkey2(KEY,INTEGER)

CLAUSES
 readkey(KEY):-readchar(T),char_int(T,VAL),readkey1(KEY,T,VAL).

 readkey1(KEY,_,0):-!,readchar(T),char_int(T,VAL),readkey2(KEY,VAL).
 readkey1(cr,_,13):-!.
 readkey1(esc,_,27):-!.
 readkey1(break,_,3):-!.
 readkey1(tab,_,9):-!.
 readkey1(bdel,_,8):-!.
 readkey1(ctrlbdel,_,127):-!.
 readkey1(char(T),T,_) .

 readkey2(btab,15):-!.
 readkey2(del,83):-!.
 readkey2(ins,82):-!.
 readkey2(up,72):-!.
 readkey2(down,80):-!.
 readkey2(left,75):-!.
 readkey2(right,77):-!.
 readkey2(pgup,73):-!.
 readkey2(pgdn,81):-!.
 readkey2(end,79):-!.
 readkey2(home,71):-!.
```

92

```
 readkey2(ctrlleft,115):-!.
 readkey2(ctrlright,116):-!.
 readkey2(ctrlend,117):-!.
 readkey2(ctrlpgdn,118):-!.
 readkey2(ctrlhome,119):-!.
 readkey2(ctrlpgup,132):-!.
 readkey2(fkey(N),VAL):- VAL>58, VAL<70, N=VAL-58, !.
 readkey2(fkey(N),VAL):- VAL>=84, VAL<104, N=11+VAL-84, !.
 readkey2(otherspec,_).


/****************************************************************
 TDOMS.PRO

   Turbo Prolog Toolbox
   (C) Copyright 1987 Borland International.

   (Modified slightly for inclusion in WeatherInfo)
 In order to use the tools, the following domain declarations
 should be included in the start of your program
****************************************************************/

GLOBAL DOMAINS
 ROW, COL, LEN, ATTR  = INTEGER
 STRINGLIST = STRING*
 INTEGERLIST = INTEGER*
 KEY   = cr; esc; break; tab; btab; del; bdel; ctrlbdel; ins;
         end ; home ; fkey(INTEGER) ; up ; down ; left ; right ;
         ctrlleft; ctrlright; ctrlend; ctrlhome; pgup; pgdn;
         ctrlpgup; ctrlpgdn; char(CHAR) ; otherspec
 FILE = sf; df; sp; dp


/****************************************************************
                 MENU.PRO

Copyright (c) 1986, 88 by Borland International, Inc.
  (Modified and included in WeatherInfo)
****************************************************************/

PREDICATES
 menu(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGER,INTEGER)
 menuinit(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,ROW,COL)
 menu1(SYMBOL,ROW,ATTR,STRINGLIST,ROW,COl,INTEGER)
 menu2(KEY,STRINGLIST,ROW,ROW,ROW,SYMBOL)
```

```
CLAUSES
 menu(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHOICE,CHOICE) :-
        menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,LEN),
        ST1=STARTCHOICE-1,max(0,ST1,ST2),MAX=NOOFROW-1,min(ST2,MAX,STA
RTROW),
        menu1(cont,STARTROW,WATTR,LIST,NOOFROW,LEN,CHOICE),
        removewindow.

 menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL):-
        maxlen(LIST,0,MAXNOOFCOL),
        str_len(HEADER,HEADLEN),
        HEADL1=HEADLEN+4,
        max(HEADL1,MAXNOOFCOL,NOOFCOL),
        listlen(LIST,LEN), LEN>0,
        NOOFROW=LEN,
        adjframe(FATTR,NOOFROW,NOOFCOL,HH1,HH2),
        adjustwindow(ROW,COL,HH1,HH2,AROW,ACOL),
        makewindow(81,WATTR,FATTR,HEADER,AROW,ACOL,HH1,HH2),
        writelist(0,NOOFCOL,LIST).

 menu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,CHOICE):-!,
        reverseattr(ATTR,REV),
        field_attr(ROW,0,NOOFCOL,REV),
        cursor(ROW,0),
        readkey(KEY),
        field_attr(ROW,0,NOOFCOL,ATTR),
        menu2(KEY,LIST,MAXROW,ROW,NEXTROW,CONT),
        menu1(CONT,NEXTROW,ATTR,LIST,MAXROW,NOOFCOL,CHOICE).
 menu1(esc,ROW,_,_,_,_,CHOICE):-!,CHOICE=ROW+1.
 menu1(_,ROW,ATTR,_,_,NOOFCOL,CHOICE):-
        CHOICE=ROW+1,
        reverseattr(ATTR,REV),
        field_attr(ROW,0,NOOFCOL,REV).

 menu2(esc,_,_,_,-1,esc):-!.
 menu2(fkey(10),_,_,ROW,ROW,stop):-!.
 menu2(char(C),LIST,_,_,CH,selection):-tryletter(C,LIST,CH),!.
 menu2(cr,_,_,ROW,CH,selection):-!,CH=ROW.
 menu2(up,_,_,ROW,NEXTROW,cont):-ROW>0,!,NEXTROW=ROW-1.

menu2(down,_,MAXROW,ROW,NEXTROW,cont):-NEXTROW=ROW+1,NEXTROW<
MAXROW,!.
 menu2(end,_,MAXROW,_,NEXT,cont):-!,NEXT=MAXROW-1.
 menu2(pgdn,_,MAXROW,_,NEXT,cont):-!,NEXT=MAXROW-1.
 menu2(home,_,_,_,0,cont):-!.
 menu2(pgup,_,_,_,0,cont):-!.
 menu2(_,_,_,ROW,ROW,cont).
```

```
/*****************************************************************/
/*                        menu_leave                          */
/* As menu but the window is not removed on return.                */
/*****************************************************************/

PREDICATES
 menu_leave(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGER,INTEGER)

CLAUSES
 menu_leave(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHOICE,CHOICE) :-
	menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL),
	ST1=STARTCHOICE-1,max(0,ST1,ST2),MAX=NOOFROW-1,min(ST2,MAX,STA
RTROW),
	menu1(cont,STARTROW,WATTR,LIST,NOOFROW,NOOFCOL,CHOICE).


/*****************************************************************
                        menu_mult
 Implements a popup-menu which allows a multiple number of
 selections.

 Each selection is made by pressing RETURN. All selections are
 then activated by pressing F10.

  The arguments to menu_mult are:


menu(ROW,COL,WINDOWATTR,FRAMEATTR,STRINGLIST,HEADER,STARTLIST,
NEWLIST)

	ROW and COL determine the position of the window
	WATTR and FATTR determine the attributes for the window
		and its frame - if FATTR is zero there
		will be no frame around the window.
	STRINGLIST is the list of menu items
	HEADER is the text to appear at the top of the menu window
	STARTLIST determines the items to be highlighted when
		 the menu is first displayed
	NEWLIST   contains the list of selections

  Ex:	menu_mult(5,5,7,7,[this,is,a,test],"select words",[1],NEWLIST)

*****************************************************************/
```

95

```
PREDICATES

menu_mult(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGERLIST,INTEGERLI
ST)

multmenu1(SYMBOL,ROW,ATTR,STRINGLIST,ROW,COL,INTEGERLIST,INTEGERLI
ST)
 highlight_selected(INTEGERLIST,COL,ATTR)
 handle_selection(INTEGER,INTEGERLIST,INTEGERLIST,COL,ATTR)
 try_del(INTEGER,INTEGERLIST,INTEGERLIST,COL,ATTR)

CLAUSES
 menu_mult(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHLIST,CHLIST) :-
        menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL),
        multmenu1(cont,0,WATTR,LIST,NOOFROW,NOOFCOL,STARTCHLIST,CHLIS
T),
        removewindow.

 multmenu1(stop,_,_,_,_,_,CHL,CHL):-!.
 multmenu1(esc,_,_,_,_,_,_,[]):-!.

multmenu1(selection,ROW,ATTR,LIST,MAXROW,NOOFCOL,OLDCHLIN,CHLOUT):-
        CHOICE=1+ROW,
        handle_selection(CHOICE,OLDCHLIN,NEWCHLIN,NOOFCOL,ATTR),
        multmenu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,NEWCHLIN,CHLOU
T).
 multmenu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,CHLIN,CHLOUT):-
        reverseattr(ATTR,REV),
        highlight_selected(CHLIN,NOOFCOL,REV),
        cursor(ROW,0),
        readkey(KEY),
        menu2(KEY,LIST,MAXROW,ROW,NEXTROW,CONT),
        multmenu1(CONT,NEXTROW,ATTR,LIST,MAXROW,NOOFCOL,CHLIN,CHL
OUT).

 highlight_selected([],_,_).
 highlight_selected([H | T],L,ATTR):-
        ROW=H-1,
        field_attr(ROW,0,L,ATTR),
        highlight_selected(T,L,ATTR).

 try_del(SELECTION,[SELECTION | REST],REST,LEN,ATTR):-
        ROW=SELECTION-1,
        field_attr(ROW,0,LEN,ATTR),!.
 try_del(SELECTION,[H | REST],[H | REST1],LEN,ATTR):-
        try_del(SELECTION,REST,REST1,LEN,ATTR).
```

```
handle_selection(SELECTION,OLDCHIN,NEWCHIN,LEN,ATTR):-
        try_del(SELECTION,OLDCHIN,NEWCHIN,LEN,ATTR),!.
handle_selection(SELECTION,OLDCHIN,[SELECTION | OLDCHIN],_,_).
```