01 Dec 1989

Development of an Expert System to Convert Knowledge-based Geological Engineering Systems into Fortran

Jill J. Cress

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

DEVELOPMENT OF AN EXPERT SYSTEM TO CONVERT KNOWLEDGE-BASED GEOLOGICAL ENGINEERING SYSTEMS INTO FORTRAN

J. J. Cress and R. W. Wilkerson

CSc-89-6

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the M.S. thesis of the first author, completed December 1989.

ABSTRACT

A knowledge-based geographic information system (KBGIS) for geological engineering map (GEM) production was developed in GoldWorks, an expert system development shell. GoldWorks allows the geological engineer to develop a rule base for a GEM application. Implementation of the resultant rule base produced a valid GEM, but took too much time. This proved that knowledge-based GEM production was possible but in GoldWorks implementation failed as a practical production system. To solve this problem, a Conversion Expert System was developed which accepted, as input, a KBGIS and produced, as output, the equivalent Fortran code. This allowed the engineer to utilize GoldWorks for development of the rule base while implementing the rule base in a more practical manner (as a Fortran program). Testing of the Fortran program generated by this Conversion System confirmed that the GEMs produced were identical to those from the KBGIS, and execution time was significantly reduced. There was an additional benefit; since use of the Fortran program did not require access to the GoldWorks System, a single GoldWorks package could be used with the Conversion System to develop several Fortran production systems. These systems could then be used at remote production sites. However, each Fortran production system still required access to the Earth Resources Data Analysis System (ERDAS) that supplied the GIS input and output files. Thus, this Conversion System achieved two major objectives; it dramatically reduced GEM production time, and it added versatility.

ACKNOWLEDGEMENT

The author would like to express her sincere thanks to: Dr. Ralph Wilkerson, her advisor, for his support and guidance during the development and completion of this project; Dr. John Prater, Dr. George Zobrist, and Dr. David Barr for their participation as committee members; Andy Hinely and all of the other members of the USGS Software Engineering Section for their constant support; and Robin Deister for her willingness to share her time and knowledge.

The author would also like to express her deepest thanks to her parents, for their enduring support, love, and encouragement.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGEMENT iv

LIST OF ILLUSTRATIONS vii

LIST OF TABLES viii

SECTION

I. INTRODUCTION 1

A. ARTIFICIAL INTELLIGENCE 1

B. EXPERT SYSTEMS 4

C. LISP 9

II. REVIEW DEVELOPMENT OF A KBGIS 11

A. GEOLOGICAL ENGINEERING MAPS (GEMS) 11

1. Production Method 12

2. Production Weaknesses 14

B. DEVELOPMENT OF THE KBGIS 16

1. Identification of Minimal Factors 17

2. Rule Base Creation 18

3. First KBGIS Implementation 19

4. Final KBGIS Implementation 22

5. Results and Conclusions 27

III. DEVELOPMENT OF CONVERSION EXPERT SYSTEM 29

A. ANALYSIS OF EXISTING KBGIS EXPERT SYSTEM 29

1. Weaknesses of the KBGIS Expert System 29

Page

2. Characteristics of the KBGIS Expert System 30

3. Results of the Analysis 31

B. ESTABLISH VALIDITY OF CONVERSION SYSTEM 32

C. DEVELOPMENT OF THE CONVERSION EXPERT SYSTEM 33

1. Conversion of KBGIS data structures 33

2. Conversion of the KBGIS Rules 34

3. Conversion of KBGIS User Interface 35

4. Implementation of the Conversion Expert System 36

D. RESULTS 37

IV. CONCLUSION 38

BIBLIOGRAPHY 39

VITA 42

APPENDICES

A. Listing of the Conversion Expert System Rules 43

B. LISP functions used by the Conversion Expert System 51

C. Fortran code generated by the Conversion Expert System 68

LIST OF ILLUSTRATIONS

Figure Page

1 Diagram of the Turing Test 2

2 Major Components of an Expert System 7

3 Conversion of Decision Tree into if-then Structured Rules 20

4 An Example of the Translation of a KBGIS rule into GoldWorks 23

5 Example of a Frame and an associated Instance 24

6 Example of KBGIS Implemented Recode Operation 25

7 Example of KBGIS Implemented Matrix Operation 26

8 Example of the Conversion of a GoldWorks Frame into a Fortran Array 34

9 Example of a Conversion of a GoldWorks rule into a Fortran rule . . . 35

10 An Example of the Conversion Expert System User Interface 36

LIST OF TABLES

Table Page

I GEMS USED FOR DETAILED INVESTIGATION 17

II FACTORS AND RESULTANT CLASSIFICATIONS 18

I. INTRODUCTION

A. ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is one of the newest, as well as one of the most controversial, areas in computer science. Its original purpose was perceived as simply creating a machine which could exhibit human intelligence. However, as often happens, this goal became much more difficult than was originally planned.

Most agree with the definition that AI means building a machine which can simulate intelligence. However, few can agree on exactly what intelligence is and, more to the point, how to know when a machine is simulating it. Alan Turing was one of the first scientists to address AI and to offer a possible method of recognizing its presence. In his article "Computing Machinery and Intelligence", which was published in 1950, he stated that it was possible to program a computer to exhibit intelligence (Charniak, 1986). He then went on to describe a test which would verify that a computer was exhibiting intelligence. This test, better known as the Turing Test, has since become one of the cornerstones in the study of AI.

The Turing Test simply states that a computer can be considered intelligent if it can fool someone into thinking it's human. To perform this test, a computer is placed in one room, a person in another, and a communication medium in a third (Figure 1). The communication medium is then used to ask questions of the other two rooms in an effort to determine which room holds the computer. If the questioner is unable to determine which room holds the computer, then the computer is accepted as exhibiting intelligence.

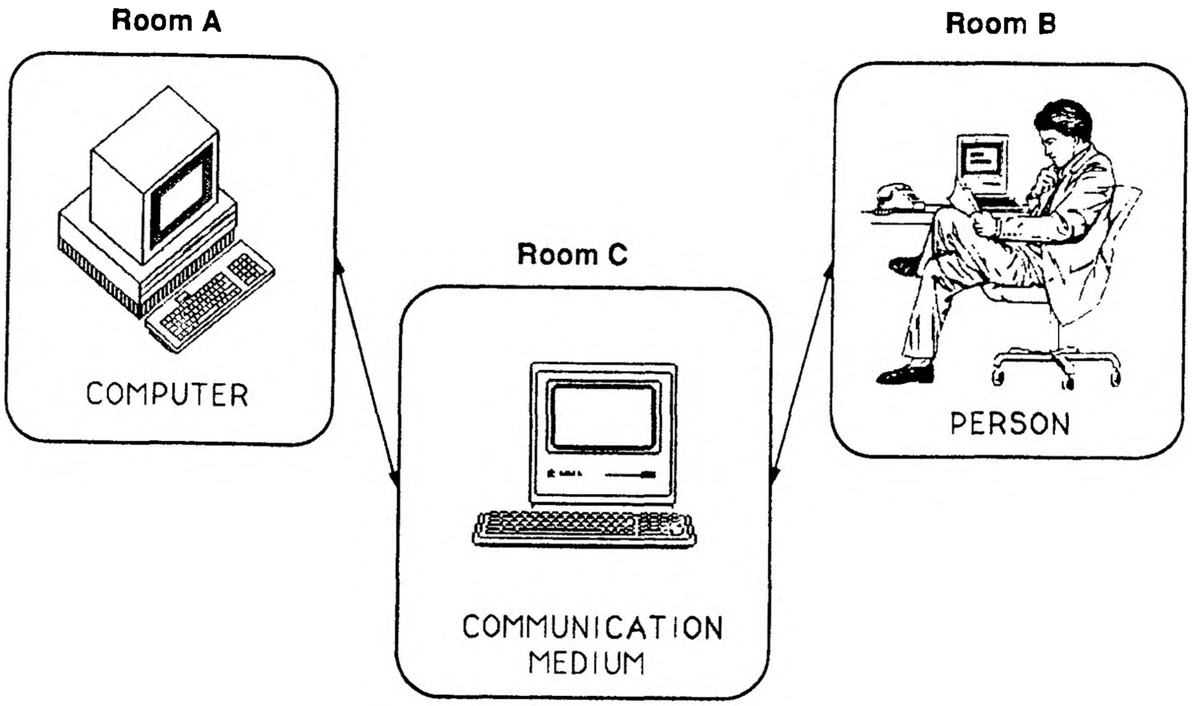


Figure 1. Diagram of the Turing Test

During the mid-50's, Allen Newell and Herbert Simon created a "thinking machine". It was a computer program called the Logic Theorist which proved theorems in symbolic logic by applying rules of reason (Lipkin, 1988). This program was considered unique since is was able "to deviate from the hard-cut paths typical of most programs" in an effort to discover new and better proofs for theorems (Lipkin, 1988).

The Logic Theorist represented a major advancement in the area of AI. But possibly even more important, were the discoveries made by Newell and Simon during its development. It was while studying the way decisions are made they discovered that, of major importance was our ability to limit the number of possible options towards a solution without having to explore each one in detail. For example, "in chess, there are more possible moves than there are stars in the universe. A master, clearly, has tricks to keep him from weighing out every one." (Lipkin, 1988). It was now obvious that a similar method for narrowing the search field would have to be created before an intelligent machine could be developed. In response to this need Simon and Newell developed heuristics -- "search-limiting rules [which] zero in on good solutions." (Lipkin, 1988).

Newell and Simon also discovered that, in order to fulfill their goals for an intelligent machine, a more advanced language would have to be developed. Not only did their program require that any language be much more flexible, it also depended on a language having the ability to manipulate symbolic types of data. They decided that the best way to perform symbolic manipulation was to treat symbols as lists and then process the lists. They then created the first list-processing language, IPL.

Six months after Newell and Simon's "thinking machine" was created, AI became a field of study as it is known today. A conference to study the field of artificial intelligence was organized by John McCarthy, an assistant mathematics professor at Dartmouth, and Marvin Minsky, from M.I.T.. The term "artificial intelligence" first appeared in print in McCarthy's written purpose for this conference (Charniak, 1986). The purpose stated that:

> "a two-month, ten-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." (Charniak, 1986).

McCarthy's statement is still used as the definition of AI. His suggested timeframe, however, grossly underestimated the complexity of the study of intelligence. The original conference group was unable to determine the nature of intelligence, let alone its features. This study marks one of the more controversial, as well as complex, areas of interest today, over thirty years later.

Today, the primary goal of AI is twofold; first, to discover the nature of intelligence and, second, to create an intelligent machine (Schank, 1987).

Unfortunately, intelligence continues to be one of the great mysteries of our time. Not only are we unable to determine what intelligence means, we are also unable to even agree on its characteristics. Roger Schank, in his article "What Is AI, Anyway?", attempts to define what he considers to be the critical characteristics of intelligence; communication, internal knowledge (self-awareness), world knowledge, intentionality (goal-driven), and creativity (Schank, 1987). However, he also points out that, while these characteristics as a unit attempt to define intelligence, the absence of one of these characteristics does not constitute a lack of intelligence. In other words, an entity could still be intelligent even if it did not have all of the above characteristics.

Because of the complexity of defining intelligence, many researchers have turned from this concern to that of applying what has already been learned in an endeavor to build intelligent machines. In other words, they have avoided the theoretical complexities and gone on to the practical applications of AI.

B. EXPERT SYSTEMS

The number and variety of applications within the field of AI have continued to grow since the introduction of the Logic Theorist; an automated theorem proving application. Automated theorem proving applications, simply put, are systems which infer a conclusion given a set of truths (axioms). In the case of the Logic Theorist the conclusion inferred is a proof of a specific theorem. Another type of theorem proving application is an expert system, currently one of the more commercially viable types of AI applications.

Early theorem proving systems concentrated on attempting to duplicate the human reasoning process. In 1957, Newell and Simon developed an improved version of the Logic Theorist, called General Problem Solver. This newer system incorporated abilities like "backward reasoning" (reasoning from the desired conclusion backwards), and "hill climbing" (determining when a solution is being neared) (Lipkin, 1988). Unfortunately, certain aspects of the human reasoning process still elude computer implementation.

As a result of this, some researchers narrowed in on specific areas of reasoning. One of these areas concentrated on the knowledge used to reach a conclusion since "much of human thinking seems to involve a small amount of reasoning using a large amount of knowledge" (Winston, 1984).

Edward Feigenbaum, a computer scientist at Stanford University, was one of the major contributors to this new area. He concentrated on empirical induction; the process of inferring information about objects by using observation. And "the result of his efforts to get computers to think empirically was the expert system" (Lipkin, 1988). In fact, he is acknowledged as the father of expert systems.

Feigenbaum, with the help of Joshua Lederberg, an organic chemist, created the very first expert system, Dendral. Dendral analyses the structure of complex molecules after being given the molecules' atomic formula and mass spectrogram.

The next major advance in expert systems came with the development of a system called Mycin. Mycin was created by Edward Shortliffe, from Stanford Medical School, and Bruce Buchanan, from the Stanford Computer Science Department, to aid doctors in the diagnosis and treatment of infectious blood diseases. However, it was not the expert system itself that proved to be so important but, rather, a discovery Shortliffe made during its development. He discovered that it was possible to separate the knowledge base from the inference engine, or logic mechanism. This allowed the creation of an empty expert system, consisting of an inference engine into which various knowledge bases could be inserted. He called this new machine Emycin (Empty Mycin), a factless inference engine. This type of system is now called an expert system shell, since it contains the logic reasoning shell into which specific facts, or knowledge, can be placed.

Today, an expert system is defined as an "artificial expert", a system which replaces or assists an expert in solving a problem. And since a large number of businesses are dependent on the knowledge of their experts, as well as the availability of that knowledge, expert systems have become one of the most popular forms of AI in the corporate world. Expert systems not only offer businesses the opportunity to preserve the knowledge of their experts, they also allow companies, who lack access to experts, knowledge via an expert system.

An expert system consists of three main elements: a user interface, inference engine, and knowledge base (Figure 2). The user interface handles the interaction between the user and the system, it allows the user to respond to system questions and input necessary data. The inference engine is the part of the system that actually performs the reasoning; it controls the user interface and generates the conclusions from the knowledge base. Finally, the knowledge base contains the knowledge of the system; this knowledge can be in various forms but the most common is that of production rules. The knowledge base is the part of the system that is developed by a knowledge engineer working with the expert.

An expert system is designed and developed by a knowledge engineer. The knowledge engineer is responsible for converting an expert's knowledge into a knowledge base. This process consists of four steps.

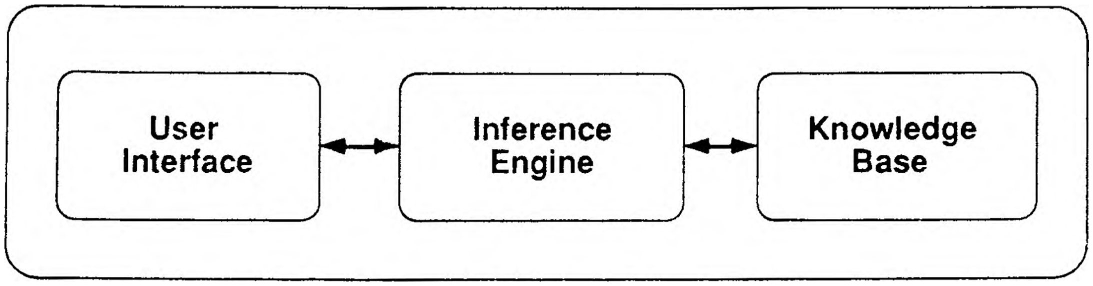


Figure 2. Major Components of an Expert System

First, the purpose of the system must be defined. This involves not only defining the problem/need of the expert system, but also determining if the system proposed is a valid one. Not all expert processes can be replicated by an expert system. Expert systems operate best when restricted to processes which serve a single purpose. It is also necessary to confirm that the knowledge needed to support the system exists, and that the necessary experts can be identified and consulted. It is also important to decide on the method of expert system development (various options for development will be discussed later).

Once the system has been clearly defined, it is time for step two, acquiring the knowledge of how the expert reaches decisions. This step involves extensive interaction between the knowledge engineer and the expert in order to identify and structure all of the knowledge which is used during the expert's decision making process. This includes identifying all items such as rules of thumb and instincts, as well as any relationships within the knowledge.

Converting the knowledge into code is the third step in expert system development. This is done after all of the knowledge has been identified and structured. The expert system development method chosen will strongly affect this step. The knowledge base is converted into rules and facts, or whatever structure has been chosen, and then entered into the system.

The fourth and final step in expert system development involves verification and validation of the system. This means confirming that the system gives creditable results when compared to the expert's results. If any changes in the system are required, the above steps will be repeated as necessary.

There are currently a variety of software tools available to aid in expert system development. These tools can be grouped into three categories, each representing a different method of expert system development.

The first of these categories is called custom development. It involves starting from scratch using an AI development language to create a specific system. This means that in addition to developing a specific knowledge base, the knowledge engineer also creates an inference engine. While this method of development results in a highly specialized system, it also involves enormous development time and money.

Using an expert system shell is the second method of development. This method involves purchasing (or using) a shell system which consists of a generic inference engine, user interface, and knowledge base structure. The knowledge engineer then builds the specific knowledge base and adds it to the generic shell structure. The time savings of this method are obvious since the engineer is spared writing an inference engine. An additional bonus is the ability to use the same shell to create a variety of expert systems.

Expert system application packages are the final development method or category. These packages consist of a complete expert system which only needs a few parameters of input before it is ready to run. This method of development was first marketed in 1985 and, although these systems require the least amount of development, they are also the most restricting since the knowledge base is already defined. A company would only find this type of system useful if it was needed for a relatively common expert area, such as finance. But even though this is the more limiting type of expert system development, it continues to grow in popularity by offering pre-designed and developed systems to companies that lack either the expertise or the resources necessary to develop a system any other way. The cost of these, however, is still somewhat an inhibiting factor since such systems can cost as much as $46,000 (Newquist, 1986).

Although these development tools differ in a variety of ways, they all eventually produce an expert system, and they are all based on a list-processing language.

C. LISP

Although McCarthy accepted that the best way to handle symbolic manipulation was list processing, he felt that a better language than IPL could be developed for this purpose. He began working on his own list-processing language shortly after the Dartmouth Conference. Before long LISP (List Processing) was born.

McCarthy started by designing four commands which would enable him to access his machines memory locations. Of these, "car", which returns the first element of a list, and "cdr", which returns all but the first element of a list, are still in use today. Next he wrote "read" and "write" routines, and designed what he referred to as a "universal function", called "eval". He stated that "eval" was so powerful that it could be used to interpret any of the other LISP functions. In fact, he wrote a paper stating that LISP could be used to express computations in automata theory, replacing the use of Turing machines, since the "eval" function was so powerful (Charniak, 1986). Although

McCarthy was responsible for designing the "eval" function, it was one of his students, Steve Russell, who actually wrote the code which allowed the function to run.

LISP is today considered the native language of AI. In fact "people who want to know about computer intelligence at some point need to understand LISP, if their understanding is to be complete" (Winston, 1984). A variety of different LISP versions currently exist, but Common LISP has become the accepted standard. And, although the specifics of the various versions may vary, the basic concepts of the language remain the same.

LISP is an interpreted language instead of a compiled one (although it is also possible to compile LISP). The difference between a compiled and an interpreted language is that a compiled one translates everything (all lines of code) into machine language before it can be executed. An interpreted language reads and executes each line. LISP treats symbols as atoms and then creates lists consisting of atoms and/or other lists. Symbolic expressions are then created by combining atoms and lists and symbolic manipulation is performed on these expressions. LISP also supports both procedure and data abstraction, which allows it to treat procedures as data. This ability makes LISP an ideal language for the creation of compilers and interpreters.

II. REVIEW DEVELOPMENT OF A KBGIS

Expert system development is rapidly becoming an integral part of almost every field. This trend can be traced to the growing need to add "knowledge" to processes and the proven success of using expert systems to fulfill this need. In fact, the need for a more knowledgeable production process recently lead to the introduction of expert systems into yet another area, that of map production in geological engineering. This introduction resulted in the development of a knowledge-based geographic information system (KBGIS) for geological engineering map (GEM) production.

A. GEOLOGICAL ENGINEERING MAPS (GEMS)

GEMs are maps that represent the "geological engineering conditions" of an area; they "show surficial and bedrock geologic patterns classified according to engineering suitability for urban development..." (Usery, Deister and Barr, 1988; Usery et. al., 1988). The specific information which is represented on any single GEM is determined by the chosen classification scheme, which, in turn, is based on the intended use of the map.

Classification schemes refer to the methods used to classify the data of a GEM. A classification scheme is developed by determining exactly what features are to be represented on the map and then defining each class in terms of the chosen features. A key is also included on the map to show the definition of each class. Features can refer to properties such as soil, geology, and topography, as well as specific attributes of these properties, i.e. flooding, permeability, karst, plasticity index, and slope.

One of the primary uses of GEMs is as a tool in determining the best site for a particular type of development. Site selection GEMs would show the geological characteristics of an area classified according to their suitability for the specified type of development. For example, if a possible landfill site was being investigated, characteristics such as flooding frequency and soil permeability would obviously be important. Therefore, in response to this need, a GEM would be produced to classify the area under investigation in terms of these factors.

1. Production Method.

The GEM production cycle begins by identifying an area which needs to be mapped. Once this area has been chosen, a series of six processing steps are performed. These steps, which are performed each time a GEM is produced, are usually done by a geological engineer. In fact, the first three processing steps rely heavily on input and interpretation from an engineer skilled in both geological principles and mapping concepts. Most, if not all, of these production steps are performed manually.

Obtaining the area maps necessary to generate the GEM is the first production step. This involves determining what features are going to be used and then gathering the maps of the area which contain information about these features. If this information can not be either found on, or derived from, existing published maps then a field investigation is required in order to develop a map of the missing feature or features.

Field investigation forms the basis of all geological maps and involves a significant investment in terms of both time and money. When a field investigation is performed in order to acquire the field data necessary to produce a map, the amount and quality of data gathered is a direct result of the amount of time spent in the field (Usery et. al., 1988). In other words, the better the quality of the data the more expensive the investigation. This means that each time a field investigation is performed a determination is made between the quality of the produced map verses the cost of the investigation. Because of this dilemma, there is a growing effort to find alternatives to field investigation for generating the basic data needed to produce a map.

Once all of the necessary maps have been obtained, the next production step involves establishment of the specific criteria. The engineer determines which characteristics are the most important for the specific map being produced. Referring back to the earlier example of the landfill site selection map, flooding and soil features were mentioned. The engineer would convert these features into criteria by stating that the area chosen for the landfill site must not be prone to flooding and must have access to impermeable soil.

After the desired criteria has been established, it is now time for the third production step. All of the maps that were obtained in the first step, both published and on site investigations, are analyzed. Each map is studied by the engineer in an effort to identify the sites which meet the criteria. One of the methods used for this analysis is to place a clear mylar sheet over each of the maps and then mark any of the areas that meet one or more of the criteria. Once all of the maps have been studied the mylar sheet can then be used to create the final GEM since it has marked on it all the areas which meet the specified criteria.

Before the GEM is actually made, however, the engineer must also decide how much information he wants the finished map to contain. In other words, how many classes should be specified on the map. If he wants an "all or nothing" product, then only those areas which identify sites meeting all of the criteria would be classified. But if a more flexible end product were desired, then in addition to classifying the areas which meet all criteria, the areas that meet only some of the criteria would also be classified.

The fourth step of the production process is printing the map. And since little or no additional analysis is required at this point, the engineer, for the first time, is not required. In fact, the engineer can be totally isolated from the actual map generation, if necessary.

Determining acceptance of the map is the fifth step in the GEM production system. This involves not only confirming that the map meets all quality guidelines, but also gaining the approval of its users. If any problems are identified at this point, or if the users request any changes, then the affected production steps must be repeated. This iteration is continued until the map is finally both approved and accepted by its users.

Even though the map has been accepted, it is not considered complete until it has been field checked; the last step in the production process. Field checking means confirming that the new map is correct by actually going into the field to check it. This check is performed by making sure that the classifications for a specific area on the map match the actual geological conditions found for that area. If any discrepancies are discovered, then the reason for the differences must be found and any needed corrections made. As with step five, any of the necessary production steps must be repeated, including again gaining the users approval, until the map is finally determined to be correct.

Once the GEM has been successfully field tested, the production process is complete and a published map is produced.

2. Production Weaknesses.

Several weaknesses can be observed in this GEM production system. One of the more serious of these is the non-standardization of the system. Since there are no set policies or guidelines for the development of classifications, it is entirely possible to discover two maps which define the same specific classification in different ways. In fact, studies of existing maps show that a definite variance exists among classification definitions. These definitions are obvious victims of the lack of standardization.

Even though each engineer applies the same basic geological principles when developing these classification definitions, subjectivity is also introduced. This subjectivity occurs when the engineer relies on his own personal base of knowledge and interpretation in order to define the classification. Obviously, different engineers have different knowledge bases and interpretations and could, therefore, end up with slightly different definitions. The result of this subjectivity is a severe weakening of the power of the different classification schemes. This weakening becomes visible when a user of a specific GEM discovers different definitions for the same specific class. For example one map might define "class-IA" as having a plasticity index of 20 whereas another map would state a plasticity index of 10 in its definition.

Another weakness inherent in the GEM production process is the demands it places, in terms of both time and effort, on a skilled geological engineer. Few of the steps can be performed without extensive interaction with the engineer, which makes the system extremely inefficient. The system also uses a "back to the drawing board" approach for all of its resource gathering and data manipulation. This means that not only does the engineer have to spend a large amount of time developing each GEM, he will also be performing the same basic processing steps over and over again.

In summary, although the GEM production system fulfills its purpose by producing a GEM, it does so in neither an efficient nor reliable manner. In fact, the weaknesses inherent in this system -- its lack of standardization and extreme reliance on interaction with a skilled engineer -- make it a prime target for modernization.

B. DEVELOPMENT OF THE KBGIS

Robin Deister, a PH.D. candidate in Geological Engineering at the University of Missouri-Rolla (UMR), felt that the GEM production system could be significantly improved by automation. Preliminary investigations determined that the best way to automate would be to "...utilize expert knowledge in GIS [geographic information system] processing" (Usery et. al., 1988). In other words, create a rule based expert system with the ability to interact with existing GISs, i.e., a knowledge-based GIS (KBGIS) (Usery et. al.,1988; Smith and Pazner, 1984).

GIS refers to an information system which allows for the input, manipulation, and output of geographic databases. Geographic databases refer to files which contain digitized geographic data. There are a variety of different types of geographic databases; including digital line graphs (DLGs), which are created by digitizing an existing map.

A joint research project to develop a KBGIS for GEM production was started in 1987 by the UMR Geological Engineering Department and the United States Geological Survey (USGS). The first objective of this project was to discover the "minimum number of Earth resource data sets needed to make engineering judgments about areas..." (Usery et. al., 1988). And then apply this knowledge to the development of a rule based expert system for GEM production.

During the initial investigation of existing maps it was discovered that the variety of GEM classifications varied dramatically from region to region. It was clear that the development of a single set of valid classifications would be virtually impossible. This discovery resulted in the decision to narrow the scope of the investigation so that it only covered the Midwest region.

1. Identification of Minimal Factors.

In an effort to determine the minimal factors, i.e., properties, necessary for the creation of a GEM, a sample of five GEMs were chosen for detailed investigation (Table I, Usery et. al., 1988). This involved not only studying the maps but also contacting their creators in an effort to establish exactly what thought processes were used in their development. The result of this investigation was a list of each output classification and exactly what factors made up the classification. Also included in the list were any implied outputs; factors which were not explicitly stated but that were implied.

Table I. GEMS USED FOR DETAILED INVESTIGATION

AREA	TYPE OF MAP
Creve Coeur, Missouri	Engineering Geology
San Mateo County, California	Engineering Character of Hillside Materials
Northeast Corridor Washington D.C., to Boston, Mass	Engineering Geology
Jefferson and St. Louis Counties, Missouri	Engineering Geology
St. Louis County, Missouri	Engineering Geology

Once a complete list had been developed, it was then converted into a table format listing the factors, classification type, integer representation of classification type, and the confidence factor (Table II). The confidence factor reflected how much weight should be given to the specified data. It was based on whether or not any discrepancies were found among the classification definitions.

The table was then used as input into 1st Class, an expert system shell developed by Programs-In-Motion (Programs-In-Motion, 1987). 1st Class is an example-based system which was used to develop a decision tree from the table of classifications and factors. An advantage of the 1st Class decision tree evaluation method is its ability to list only the factors which were necessary for the creation of unique classifications (Usery et. al., 1988). For instance, figure 3 shows that only the flooding and plasticity index factors are necessary in determining classification 2, or "Ib".

Experimentation with the 1st Class system resulted in the discovery that by reordering the factors on the table, different decision trees could be created. The table was then repeatedly reordered until a decision tree was developed which contained the minimum number of factors needed to create a GEM. These factors were identified as slope, karst, plasticity index, and flooding.

Table II. FACTORS AND RESULTANT CLASSIFICATIONS

GEOLOGY	FLOODING	KARST	PI	SLOPE	CLASS	RESULT	WEIGHT
ALLUVIAL	YES	NO	20.0	2.	Ia	1	2.00
ALLUVIAL	YES	NO	10.0	2.	Ib	2	2.00
ALLUVIAL	OCCASIONAL	NO	10.0	2.	Ic	4	2.00
ALLUVIAL	NO	NO	20.0	20.	Id	10	2.00
ALLUVIAL	NO	NO	27.0	9.	Ie	5	2.00
LIMESTONE	NO	YES	15.0	25.	IIa	6	2.00
LIMESTONE	NO	YES	32.0	20.	IIb	7	2.00
LIMESTONE	NO	YES	15.0	20.	IIc	6	2.00
LIMESTONE	NO	YES	20.0	40.	IId	9	2.00
LIMESTIONE	NO	YES	40.0	30.	IVa	7	2.00
LIMESTONE	NO	YES	25.0	30.	IVb	19	2.00
CYCLIC	NO	NO	15.0	20.	V	14	2.00
LIMESTONE	NO	YES	25.0	20.	VIa	6	2.00
LIMESTONE	NO	YES	20.0	30.	V!b	19	2.00
ALLUVIAL	YES	NO	10.0	2.	Ia	2	1.00
ALLUVIAL	NO	NO	10.0	20.	Ib	3	1.00
ALLUVIAL	NO	NO	27.0	9.	Ic	5	1.00
LIMESTONE	NO	YES	15.0	25.	IIa	6	1.00

WEIGHT KEY	
2.00	Confident
1.00	Less Confident

2. <u>Rule Base Creation.</u>

An appropriate classification scheme had to be chosen before the rules could be developed. In other words, a standardized classification scheme had to be established.

This meant, that in addition to resolving any variations between classification definitions, a determination had to be made on what kind of end product was desired.

It was decided that a general GEM, instead of a specific site selection GEM, was a desirable end product. This general end product increased the efficiency of the production system by removing the need to repeat the entire production process each time a specific site selection GEM was needed. Instead of repeating the process, the general GEM could be used to derive any of the specific end products. In other words, a one step translation from a general GEM to a specific GEM could easily be performed.

After a reinvestigation of the maps in Table I, it was decided that the scheme used for the St. Louis County GEM would be used as the standardized scheme since it "forms a basic structure for a general classification in the Midwest" (Usery et. al., 1988; Lutzen and Rockaway, 1971). It was also chosen because it defines specific classifications in terms of their general engineering characteristics and then supplies a table which states how each of the classifications would "perform under different site-utilization situations." (Usery et. al., 1988).

Development of the rule base was essentially completed once the minimum factors and a classification scheme were developed and chosen. All that remained was simply converting the correct decision tree into if-then structured rules. Figure 3 shows an example of this translation process.

3. <u>First KBGIS Implementation</u>.

The first KBGIS design used the LISP environment to implement the rule-base. A LISP program was created that interacted with both the rule base and the input files (overlays) in order to generate a file of the GIS operations needed to produce a GEM.

```
---- start of rule ----
FLOODING ??
 YES :  PI ??
        < 15.00 : ............................................................ 2   ( Ib )
        > 15.00 : ............................................................ 1   ( Ia )
 NO :  KARST ??
       YES :  PI ??
              < 28.50 :  SLOPE ??
                         < 27.50 : ........................................... 6   ( IIa )
                         > 27.50 :  PI ??
                                    < 22.50 :  SLOPE ??
                                               < 35.00 : ....... 19  ( VIb )
                                               > 35.00 : ....... 9   ( IId )
                                    > 22.50 : .......................... 19  ( VIb )
              > 28.50 :  SLOPE ??
                         < 14.50 :  PI ??
                                    < 44.00 : .......................... 7   ( IIb )
                                    > 44.00 : .......................... 18  ( IIb )
                         > 14.50 : ........................................... 7   ( IIb )
       NO :  PI ??
             < 19.00 :  SLOPE ??
                        < 30.00 :  PI ??
                                   < 16.50 :  PI ??
                                              < 12.50 : ........ 3   ( Ib )
                                              > 12.50 : ........ 14  ( V )
                                   > 16.50 : ........................... 15  ( Xa )
                        > 30.00 : ............................................ 13  ( VI )
             > 19.00 :  SLOPE ??
                        < 9.50 :   PI ??
                                   < 29.50 : ........................... 5   ( Ic )
                                   > 29.50 :  PI ??
                                              < 43.00 : ........ 16  ( Xb )
                                              > 43.00 : ........ 17  ( Xc )
                        > 9.50 :   PI ??
                                   < 22.50 : ........................... 10  ( IIIa )
                                   > 22.50 :  PI ??
                                              < 30.00 : ....... 12  ( V )
                                              > 30.00 : ....... 11  ( IIIb)
OCCASIONAL : ......................................................................... 4   ( Ic )
---- end of rule ----
```

IF Flooding is None or Rare and Slope is 5-30% and Plasticity Index is 20-30 and Karst is No THEN Class is Ig

IF Flooding is None or Rare and Slope is 5-30% and Plasticity Index is 30-40 and Karst is No THEN Class is IIb

IF Karst is Yes THEN Class is IIc

IF Slope is >30% THEN Class is IId

●

●

●

●

Figure 3. Conversion of Decision Tree into if-then Structured Rules

Production of the GEM was then simply a matter of using a GIS to perform these operations. The Earth Resources Data Analysis System (ERDAS) software package, which is "a raster-based GIS and image processing system", was chosen as the GIS for this implementation (Usery et. al., 1988).

ERDAS files, which represent geographic overlays, are stored in a 512 bytes per record format consisting of 128 bytes of header information followed by the map data. Each byte of this data, which represents a pixel, contains a numerical value between 0 and 255. Although a variety of GIS commands are available for manipulating these data files, knowledge based implementation requires only the recode and matrix operations in order to produce a GEM.

In the ERDAS environment, a recode "essentially reassigns classification values to new zones...For example, slope may be represented in 5 percent increments from 0 to 100 percent in a GIS overlay by numbers from 1 to 20. A recoding of the slopes to a total of three classes, 1 for numbers 0 to 5, 2 for numbers 6 to 12, and 3 for numbers 13 to 20, can be performed if the analysis requires values of only low, medium, and high." (Usery et. al., 1988). In other words, the recode operation recodes (converts) a range of pixel values into a new single pixel value.

A matrix operation, in the ERDAS environment, "analyzes two overlays and produces a new overlay containing class values that are coded to indicate how the class values from the original files coincide or overlap." (Usery et. al., 1988). This is done by using the class values of one of the files as the matrix columns and the class values of the other file as the matrix rows. This matrix can then be used to create "logical combinations of classes such as union, intersection, complement, or any combination" (Usery et. al., 1988). For instance, the logical intersection of two overlays could be found by sequentially numbering the matrix positions and then using this number as the resultant classification (Usery et. al., 1988). Unfortunately the GIS matrix operation can only analyze two overlays at a time. Therefore in order to logically combine more then two overlay files, a complex series of both recode and matrix operations is required. This is one of the data manipulation weaknesses that is inherent in any GIS environment.

The first KBGIS implementation, the LISP program, was tested by using a combination of actual and simulated data for the area of Aspen, Colorado (Usery et. al., 1988). This information, which was formatted as ERDAS files, was then input into the LISP program and an output file of the needed ERDAS commands was generated. ERDAS was then used to execute the commands from this file and actually produce the GEM. After this was completed, the resultant GEM values were compared to manually calculated results in order to determine the validity of the system. This comparison confirmed that the knowledge-base was creating valid results by its generation of recode and matrix operations. It was also observed, however, that the use of the LISP environment to implement the knowledge base caused the results of the system to vary, depending on the order of the rules in the knowledge base. It was, therefore, concluded that another type of system would have to be designed in order to implement the knowledge base.

4. Final KBGIS Implementation.

"The final system design implements GIS processing within the expert system" (Usery et. al., 1988). In other words, the final system design required that an expert system be developed that would actually produce the GEM, instead of just generating a file of ERDAS commands. The GoldWorks expert system shell package from Gold Hill was used to develop this expert system.

GoldWorks provides "a knowledge-based expert system development environment integrated with Gold Hill's Golden Common Lisp (GCLISP) Developer software" (Gold Hill, 1987). It also provides two different interfaces which allow both the programmer and non-programmer to easily develop expert systems. The Menu interface can be used by anyone, regardless of their programming ability, to quickly and easily develop and modify expert systems since it supplies a completely menu driven environment. Of particular importance is its ability to allow quick and easy modification of a knowledge base. More experienced programmers are offered the opportunity to use the Developers interface which uses the GMACS editor and GCLISP to allow for the creation of more powerful systems.

A knowledge base consists of both active (rules) and passive (facts) knowledge. GoldWorks provides an easily understood and used structure for both rules and facts. Rules are structured in if-then format, and can be easily input and modified by using the Menu interface (Figure 4). GoldWorks uses a frame-based structure to represent the facts of the knowledge base. A frame is used to represent the structure of a class of objects. Frames consist of a set of slots, each representing an attribute of the frame. The actual occurrence of a specific object is represented by creating an instance of the frame. Facts can then be placed in the slots of the instance. In other words, slots can now be given slot values. Figure 5 gives an example of one of the frames and instances in the KBGIS. In this example, the frame KARST has a slot for each of the possible karst input values.

```
IF Flooding is None or Rare
   and Slope is 5-30 %
   and Plasticity Index is 20-30
   and Karst is No
THEN Class is Ig
```

$\longrightarrow$

```
IF
 (( Instance ?X is Pixel with Flooding None)
   or
  ( Instance ?X is Pixel with Flooding Rare))
 and
 ( Instance ?X is Pixel
              with Slope 5-30%
              with Plasticity 20-30
              with Karst No)
THEN
  ( Instance ?X is Pixel
              with Gem Class-Ig)
```

Figure 4. An Example of the Translation of a KBGIS rule into GoldWorks

Rules are used to deduce new facts from the existing fact base. This process is controlled by the inference engine and in GoldWorks the inference engine supports three inference techniques; forward-chaining, backward-chaining, and goal-directed forward-chaining. The rule base can use any one of these techniques in order to deduce new facts. It can even use a combination of all three. In the KBGIS rule base, the forward-chaining technique was used.

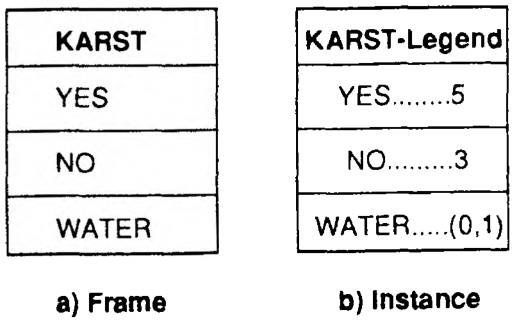


Figure 5. Example of a Frame and an associated Instance

The KBGIS uses conceptual values, that represent the symbolic values of the various overlays, for processing. For instance, when the karst input is being processed by the expert system, it will have one of the following conceptual values; "no", "yes", or "water". These conceptual values are then used by the rule base to determine a conceptual GEM classification value, like "class-Ia". In other words, since both the input and output files consist of numerical pixels, the expert system must have the ability to perform the following mappings; "pixel to conceptual" and "conceptual to pixel".

These mappings are performed by implementing the GIS recode operation in the KBGIS. The recode operation is more complex in the KBGIS environment then it was in the GIS, since it requires more then a "pixel to pixel" mapping. Referring back to the recoding example for the slopes, in the KBGIS the final values would be the conceptual ones, "low", "medium", and "high".

Implementation of the KBGIS recode operation was done by making use of the frame-based data structures. In order to perform the KBGIS recode operation, first a frame structure had to be created for each of the GIS data files. The set of slots for each frame represented all of its possible conceptual values. For example a frame called karst would have "yes", "no", and "water" as its slots. In order to perform the recode operation, each time the KBGIS is executed an instance for each of the frames is created and the slot values are set equal to the appropriate pixel values (Figure 6). The KBGIS recode can also map multiple pixel values into a single conceptual value since GoldWorks allows multi-valued slots. The pixel values that are placed in the specific slots are user-supplied for the input GIS files and system-supplied for the output GEM GIS file.

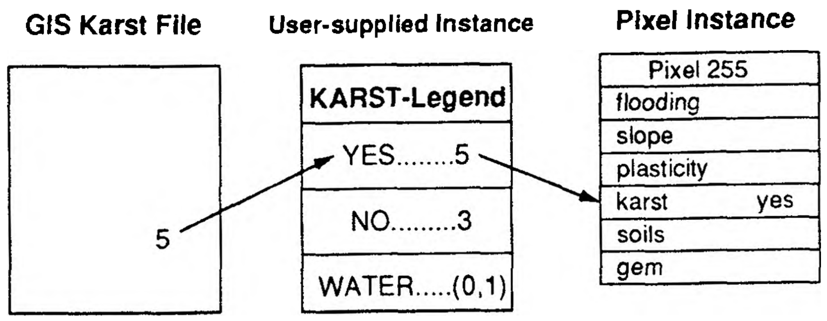
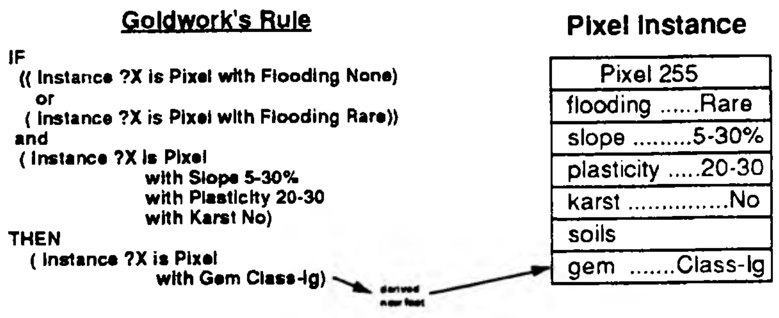


Figure 6. Example of KBGIS Implemented Recode Operation

After the instances have been created, they are then used as keys in order to perform the "pixel to conceptual" and "conceptual to pixel" recodes. A GIS input pixel to KBGIS conceptual recode is done by accessing the correct instance and then finding the slot value which matches the input pixel. It is then simply a matter of setting the conceptual value equal to the slot name which held the correct slot value. A conceptual to pixel recode is performed in a similar manner, the main difference being that the conceptual value is used as the key. The correct instance is referenced and the slot name which matches the conceptual value is accessed, then the pixel value is set equal to the slot value of the accessed slot. The KBGIS process accepts pixel input and produces pixel output; therefore, two intermediate KBGIS recodes must be performed.

In addition to the recode operation, the KBGIS also implements the ERDAS matrix operation. Implementation of the matrix operation in the KBGIS is much more powerful then its equivalent in the GIS environment because it has the ability to matrix any number of files simultaneously. Implementation of the matrix operation also relies on the use of frames. The matrix operation, however, only required the creation of a single frame, one that represented a pixel. Its slots were named after each of the GIS files that were either input or output. The slot values themselves are conceptual values which are recoded from the appropriate pixel values, either input or output. Each time a set of pixels are read from the input files, a single instance of the pixel frame is created and the appropriate slot values are filled (Figure 7). This creation of a new fact in the knowledge base causes the rule base to be invoked. The rules, which use conceptual values, then inspect the slots of the instance pixel in an effort to set the GEM slot. Figure 7 shows an example of this process. It can be seen that the number of files which can be utilized by matrix is only limited by the number of slots contained in the frame pixel.



```
IF
 (( Instance ?X is Pixel with Flooding None)
    or
  ( Instance ?X is Pixel with Flooding Rare))
 and
 ( Instance ?X is Pixel
                   with Slope 5-30%
                   with Plasticity 20-30
                   with Karst No)
THEN
   ( Instance ?X is Pixel
                    with Gem Class-lg)
```

Pixel 255
flooding ......Rare
slope 5-30%
plasticity 20-30
karst ....No
soils
gem ....Class-lg

Figure 7. Example of KBGIS Implemented Matrix Operation

Currently, the KBGIS can produce a GEM given one of two different types of input. The first type of input requires four separate GIS overlay files; flooding, slope, plasticity, and karst. In addition to supplying these input files, the user must also supply the information necessary to map the input pixel value to a conceptual value for each of the files. This information is then used by the system to convert the input pixel from each of these files into a conceptual value, which is then placed in a pixel instance. The rule base then performs the matrix operation and sets the GEM slot of the pixel instance. This conceptual value is then recoded into a pixel value and output to the GIS formatted GEM file. Once all of the pixels have been processed, a GIS trailer file is created which contains a legend to aid in interpreting the GEM pixels.

The second type of input that the KBGIS can accept is that of a GIS soils file. A soils file is simply a composite of the flooding, slope, plasticity, and karst overlays. Therefore, it can also be used to produce a GEM file. In order to use a soils file, however, the user must first of all describe to the system each of the possible soil pixels in terms of the flooding, slope, plasticity, and karst overlays. In other words, a specific pixel instance is created for each of the possible soils pixels and the appropriate conceptual values are filled in. The system then performs a matrix operation to set the GEM slot for each of these pixels. To produce a GEM file now simply requires a KBGIS recode of the soils file and the creation of a trailer file.

5. Results and Conclusions.

The final KBGIS was tested by first selecting an area that already had a manually produced GEM and then generating a KBGIS GEM for it. These two maps were then compared and the results of the comparison were analyzed.

The Creve Coeur, Missouri, area was chosen for this test because a manually produced GEM already existed for it (Usery et. al., 1988; Lutzen and Rockaway, 1970). Once the area had been selected, it was then necessary to generate the KBGIS GEM for that area for the eventual comparison testing.

The first generation step required that the needed input files be in ERDAS GIS format. It was decided that the soils file would be used for the GEM generation. Therefore, the published map of the soils overlay was digitized and then converted into ERDAS format. Once this input was in the correct format, the KBGIS was used to produce a GEM. This GEM was then displayed on a graphics system, as was the digitized manually produced GEM, and side by side comparisons were performed.

It was concluded that a "good" correlation existed between the two maps, taking into account land use changes that may have occurred in the 10 years since the manual GEM was produced (Usery et. al., 1988). The major differences that were observed between the two maps were caused by differences in the classification methods, rather then problems with the system (Usery et. al., 1988). In fact, the KBGIS produced a more detailed GEM then the one produced manually. It was, therefore, concluded that a KBGIS could be used to produce a GEM.

Unfortunately, the system did not prove to be very practical. The amount of execution time required by the KBGIS severely hampered its practicality. Generation of the Creve Coeur GEM took in excess of 76 hours. In fact, the system was not allowed to run to completion. Instead, processing was stopped and the missing classifications where tabulated by hand. This meant that, although the KBGIS automated the GEM production process, it was not at all efficient in terms of time.

III. DEVELOPMENT OF CONVERSION EXPERT SYSTEM

Results of the testing of the final KBGIS supported the theory that a automated approach could be used to produce GEM maps. In fact, automation strengthened the production process by standardizing the classification definitions, reducing the time commitment required of an engineer, and removing the need to start from scratch each time a specific site selection GEM was required. Unfortunately, the automated process still required a significant amount of production time.

In response to this problem, an investigation was performed to determine how the developed knowledge base could be more efficiently implemented as a production system. This resulted in development of an expert system which converts any existing KBGIS into Fortran code, which can then be compiled and used efficiently in production.

A. ANALYSIS OF EXISTING KBGIS EXPERT SYSTEM

1. Weaknesses of the KBGIS Expert System.

One of the primary causes of the slow KBGIS execution time is the large size of the input and output files, since a typical GIS overlay file could easily contain as much as 250,000 bytes of data. The overhead created by the processing of such large files is significant. Expert system inference algorithms, while supporting extremely powerful reasoning abilities, also, by their very nature, require a larger execution time for the processing of larger data files.

Another weakness of the KBGIS as a production system is its reliance on the GoldWorks package. Purchasing GoldWorks requires a major initial financial investment, since both the software package and the appropriate hardware would have to be purchased in order to use the KBGIS GEM production system. This could possibly be an inhibiting factor for a smaller GEM production environment, such as a field office. This could result in only the larger production companies being able to afford the system.

2. Characteristics of the KBGIS Expert System.

A detailed study of the KBGIS expert system has shown that the knowledge base can essentially be split into two separate parts; an application-specific knowledge base, and a GIS knowledge base. The GIS part consists of the knowledge needed to perform the recode and matrix operations that enable the system to convert the input data into a GEM. This represents the stable part of the knowledge base; regardless of the specific application, this part of the system remains unchanged. The application knowledge base, however, is not stable. This is the part of the knowledge base consisting of the specific rules that were developed to generate the GEM from the minimal factors, both of which were established for the Midwest region only. This means that in order to create a production system for any other region, the development process would have to be repeated to establish a valid application knowledge base for the new region.

GoldWorks is vital to the application knowledge base development process. Its ease of use for non-programmers makes it possible for geological engineers to develop the rule base themselves, without requiring that they have access to programmers. GoldWorks also offers the engineer the ability to develop prototype KBGISs quickly and easily. This is done by plugging experimental application rules into the established GIS knowledge base and generating GEMs. Needed rule base modifications can then be made quickly and easily through GoldWorks, and the entire process repeated until an acceptable rule base has been established. During this development process, in order to overcome the problem of excessive execution time, limited test data sets could be used for the preliminary prototyping and testing.

The application knowledge base consists of two parts; data structures for the minimal factors and the rules which determine the classifications. The rules themselves follow a fairly standard if-then format. The if part consists of a combination of the minimal factors and associated conceptual values; the then part sets the correct classification value. These rules are mutually exclusive, and will always be mutually exclusive since classification schemes demand that unique factor combinations be established for each classification definition. The number of rules will, therefore, be equal to the number of possible classifications, which will vary region to region. The data structure of the minimal factors will also remain basically stable region to region. This structure associates with each of the minimal factors all of its possible input values.

3. Results of the Analysis.

Although the GoldWorks expert system is obviously vital to the development of the application knowledge base, it is inefficient when used to implement the resultant knowledge base in a production environment. It was also observed that once the rule base is established, its characteristics did not require implementation in an expert system environment. This discovery lead to the conclusion that, once the final rule base was established, it could be converted into another, more efficient, programming language for production purposes. However, it was also observed that this conversion from a GoldWorks based expert system into another language would require access to an experienced computer programmer. This conversion process would also have to be performed multiple times and possibly over a prolonged time period, since any changes to a region's established rule base must also be reflected in the production system.

It was, therefore, decided that an intermediate expert system should be developed that could accept as input any KBGIS knowledge base and convert it into another computer language. This new program could then be compiled and used for the production system. Development of an expert system to perform the conversion process would solve both the need to have access to a programmer, and the need to perform the conversion process repetitively.

B. ESTABLISH VALIDITY OF CONVERSION SYSTEM

The first step in developing a conversion expert system was to choose the language to convert the KBGIS into, and then confirm that significant time savings would be obtained. Fortran was chosen as the language because of its inherently fast execution time for data intense files, which can be seen in Fortran based ERDAS processing. It was also chosen because of its availability, since the majority of the computer systems currently available in the field support Fortran.

After Fortran was chosen, the existing KBGIS for the Midwestern region was manually converted into equivalent Fortran code. This hand-coded program was compiled and an executable program developed. A sample data set was then used to develop a GEM, using both the Fortran program and the KBGIS expert system. Comparisons were made between the two GEMs to confirm that the Fortran code was producing identical results. A byte by byte comparison was made between the two GEM files and no discrepancies were found. Therefore, it was concluded that the Fortran program was producing valid results.

A comparison was also done between the execution times of the two GEM production systems, which were both executed on the same hardware in order to eliminate any hardware related execution factors. The Fortran program was able to produce the GEM in 5 minutes and 49 seconds whereas the KBGIS had still not completed production after 3 hours. It was, therefore, concluded that the Fortran production system was able to produce a valid GEM with a significant time savings as compared to the KBGIS expert system.

C. DEVELOPMENT OF THE CONVERSION EXPERT SYSTEM

The GoldWorks expert system development shell was chosen as the development tool for the Conversion Expert System. GoldWorks was chosen, not only because the KBGIS was implemented in it, but because it also allowed for the creation and use of powerful LISP functions in its Developers interface.

1. Conversion of KBGIS data structures.

The first task of this expert system was to convert the GoldWorks framed-based data structures into appropriate Fortran data structures. Fortran arrays where chosen, since they allowed a specific number of elements to be associated with a single data structure. This was a vital requirement, since each of the frames in the KBGIS had associated slots. Conversion of a frame into an equivalent Fortran array was done by creating an instance of the specific frame, and then giving the name of the equivalent Fortran array element as a value for each of the slots (Figure 8). The name of the array is used as the instance name, and a key list containing the frame name and associated array name was constructed.

An equivalent Fortran array data structure was created for each of the frames representing one of the minimal factors. Determination of these frames was performed by searching the rule base and developing a list of all frames used in the antecedent part of the rules which set the GEM classification. This list was then used as a key for

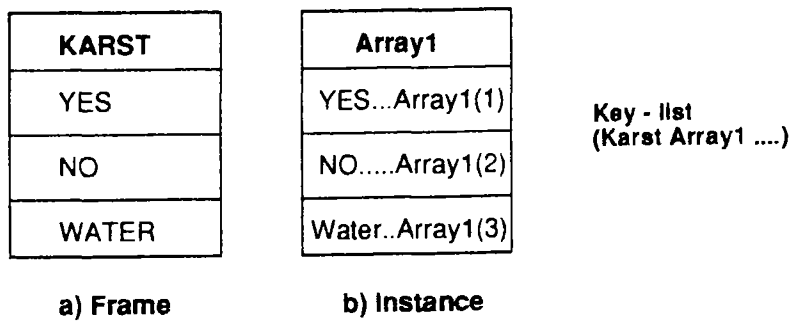


Figure 8. Example of the Conversion of a GoldWorks Frame into a Fortran Array

which frames needed to be converted. Once all of the appropriate frames were converted, it was time to convert the rules.

2. Conversion of the KBGIS Rules.

Conversion of the GoldWorks rules into Fortran if-then rules required extensive use of recursion. Only the rules which deduce a GEM classification needed to be converted. Therefore, the first step was to identify those rules. This was done by retrieving each rule's consequence and then searching it for the name which represented the classification slot of the pixel frame. GEM is the slot name in the existing KBGIS. If a match was found, then the specific classification value which the rule sets was checked for validity. Confirmation of validity was done by making sure that the classification value specified in the rule matched one of the slots for the GEM frame in the KBGIS.

Once it was established that a rule needs to be converted, a list of the rule's antecedent was retrieved. This list can consist of several layers, depending on the complexity of the antecedent. Each member of the list was searched until the minimal sublist was found. This sublist is a list which actually contains the pattern used to search the fact base. This list is then converted into Fortran code and, if necessary, the appropriate logical connective is also converted into Fortran (Figure 9). This process was recursively repeated until each of the minimal sublists were converted. After the rules antecedent was processed, the consequence was retrieved and converted. This conversion was much simpler, since the list of the consequence contains only atoms, that can be easily converted (Figure 9).

<u>Goldwork's Rule</u>

```
IF
 (( Instance ?X is Pixel with Flooding None)
    or
  ( Instance ?X is Pixel with Flooding Rare))
 and
  ( Instance ?X is Pixel
                 with Slope 5-30%
                 with Plasticity 20-30
                 with Karst No)
THEN
   ( Instance ?X is Pixel
                 with Gem Class-lg)
```

<u>FORTRAN</u>

```
IF ( ( (  Flooding .EQ. Array3(1) )
     .OR. ( Flooding .EQ. Array3(3) )
    .AND. ( Slope .EQ. Array2(2) )
    .AND. ( Plasticity .EQ. Array4(3) )
    .AND. ( Karst .EQ. Array1(2) ) )

THEN
   GEM = 6
```

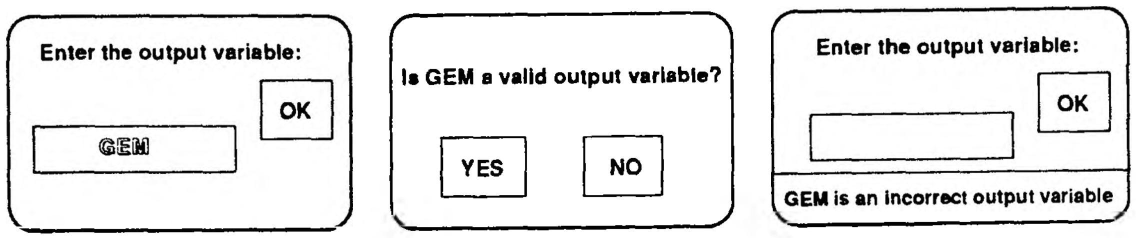
Figure 9. Example of a Conversion of a GoldWorks rule into a Fortran rule

3. <u>Conversion of KBGIS User Interface.</u>

The only part of the KBGIS system left to convert was its user interface. This conversion was relatively simple, since the only anticipated changes would be the names and numbers of the input files since these are based on the minimal factors used by the system. The format of both the input and output files was constant since it was based on the use of GIS files; and the creation of a trailer file was also constant. Essentially, all that was required for this part of the conversion was to create the user input interface. This required using the list of the minimal factors (developed earlier in the conversion) to generate the Fortran code needed for input/output processing by using formatted stream output statements.

4. Implementation of the Conversion Expert System.

In order to use the Conversion Expert System, the KBGIS to be converted must first of all be loaded into the system. Once this is done, the system can be executed. The Conversion System is extremely easy to use and requires a minimum amount of input from the user. User interface functions in a very user-friendly environment. Each time the system requests input from the user, not only is the user given the opportunity to change the answer, but the system also confirms that a valid answer is given. If either the user or the system decides that an incorrect entry is given, the user is asked to enter the input again. An example of a series of user queries are given in figure 10.


Enter the output variable:

GEM

Is GEM a valid output variable?

Enter the output variable:

GEM is an incorrect output variable

Figure 10. An Example of the Conversion Expert System User Interface

After entering the system from the opening screen, the user is required to tell the system the directory path name for the generated Fortran code. This consists of six files called ftr0.dat to ftr5.dat that must be concatenated together in ascending numerical order to produce the complete Fortran source code. This is easily done by using the DOS copy command. Then the user is asked for input and output variable names, as well as a default output classification. Finally, the user is asked whether the specified factor, which the system has retrieved from the KBGIS rule base, can be derived from the soils file, represents an independent overlay, or both.

D. RESULTS

Once the generated Fortran code is converted into an executable program, it can then be used to produce a GEM. This program is now equivalent to the KBGIS system in terms of input and output requirements, i.e., it requires one of two types of input and produces two outputs (GEM and trailer). The only differences in the systems are their methods of processing; the Fortran program performs pixel to pixel recodes instead of pixel to conceptual to pixel recodes like the KBGIS. In addition, the Fortran code is limited to one-to-one mappings, unlike the KBGIS which supports multiple values to a single value mapping. This is, however, the only additional constraint added by the Fortran production system. And, if a multiple to single mapping is required of the system, it can easily be performed in ERDAS as a pre-processing step.

The Conversion Expert System was tested by converting the Midwestern KBGIS expert system. After the KBGIS was converted into Fortran code by the expert system, an executable Fortran program was created. In other words, the Fortran code which was output by the expert system was compiled and linked. Next, the executable Fortran program was used to produce a GEM and then compared to a GEM produced by the KBGIS from the same input. Comparisons resulted in no differences being found between the GEMs.

The Fortran production system was able to process 179 bytes per second (.0056 sec/byte) as compared to the KBGIS which processed only .77 bytes per second (1.30 sec/byte). And since this comparison was based on a data set of only 62500 bytes and it had already been determined that the larger the data set the worse the KBGIS system performs (in fact the time required to process a byte grows exponentially) it was obvious that the Fortran production system performs in a significantly more efficient manner.

IV. CONCLUSION

It can easily be seen that while the KBGIS approach to development of a GEM production system has definite benefits, it fails to fulfill its obligations as a production system. The introduction of an intermediate step, however, was able to bridge the gap between ease of development and efficiency of use. This bridge was the development of a Conversion Expert System.

This system significantly improved the practicality of the automated production of GEMs by drastically reducing the production time. The Fortran approach was also able to solve the possibly inhibiting effect of the cost of the system. Although the GoldWorks package is still a requirement in order to develop such a KBGIS, once the system has been developed and converted into Fortran, any system that can execute Fortran can be used as a production system. This means that a hub approach could be taken by a production company which could result in significant financial savings. A hub approach means that only the central regional office would need to purchase the GoldWorks package and associated hardware. The KBGIS could then be developed and converted at the central office, and only the resulting executable program would need to be sent out to the field offices.

In conclusion, the development of a KBGIS for the production of GEMs, resulted in major improvements over existing production methods. Then the development of a Conversion System to be used in association with the KBGIS made the resulting production process both efficient and affordable.

BIBLIOGRAPHY

Carter, James R. "A Typology of Geographic Information Systems," Proceedings of the American Congress on Surveying and Mapping/American Society for Photogrammetry and Remote Sensing , Vol. 5 (1988), 207-215.

Charniak, Eugene and Drew McDermott. Introduction to Artificial Intelligence. Reading, Massachusetts: Addison-Wesley Publishing Company, 1986.

Dangermond, Jack. "A Review of Digital Data Commonly Available and some of the Practical Problems of Entering them into a GIS," Proceedings of the American Congress on Surveying and Mapping/American Society for Photogrammetry and Remote Sensing , Vol. 5 (1988), 1-10.

Deister, Robin. Personal interview. October 25, 1989.

Denning, Peter J. "The Science of Computing: Blindness in Designing Intelligent Systems," American Scientist , Vol. 76 (March-April, 1988), 118-120.

ERDAS, Inc. ERDAS Software Toolkit Image Processing System User's Guide. Atlanta, Georgia: ERDAS, Inc., 1987.

Gold Hill. GoldWorks Expert System Development and Delivery. Cambridge, Massachusetts: Gold Hill Computers, Inc., 1987.

Lipkin, Richard. "Cover Story," Insight , (February 15, 1988), 8-17.

Lutzen, E. E. and J. D. Rockaway, Jr. "Engineering Geology of St. Louis County, Missouri," Engineering Geology Series No. 4 , 1971, 23.

Lutzen, E. E. and J. D. Rockaway, Jr. "Engineering Geology of the Creve Coeur Quadrangle, St. Louis County, Missouri," Engineering Geology Series No. 2 , 1970, 19.

Mishkoff, Henry C. Understanding Artificial Intelligence. Indianapolis, Indiana: Howard W. Sams and Company, 1985.

Newquist III, Harvey P. "Expert Systems: The Promise of a Smart Machine," Information Systems for Management: A Book of Readings. 1986, 170-181.

Programs-In-Motion. 1st Class User's Guide , Wayland, Massachusetts: Programs-In-Motion, Inc., 1987.

Schank, Roger C. "What Is AI, Anyway?," AI Magazine , Winter 1987, 59-65.

Steele, Guy L., Jr. Common LISP The Language. Digital Equipment Corporation, 1984.

Smith, T. R., and M. Pazner. "Knowledge-Based Control of Search and Learning in a Large-Scale GIS," Proceedings International Symposium on Spatial Data Handling , 1984, 498-519.

Tanimoto, Steven L. The Elements of Artificial Intelligence. Rockville, Maryland: Computer Science, Inc., 1987.

Tatar, Deborah G. A Programmer's Guide to Common LISP. Digital Equipment Corporation, 1987.

Usery, E. L., Phyllis Altheide, Robin Deister, and David Barr. "Knowledge-Based GIS Techniques Applied to Geological Engineering," Photogrammetric Engineering and Remote Sensing , Vol. 54 (November 1988), 1623-1628.

Usery, E. L., Robin Deister, and David Barr. "A Geological Engineering Application of a Knowledge-Based Geographic Information System," Proceedings of the American Congress on Surveying and Mapping/American Society for Photogrammetry and Remote Sensing , Vol. 2 (1988), 176-185.

Waldrop, M. Mitchell. "Artificial Intelligence," The Washington Post , (February 22, 1988), C3.

Waterman, Donald A. A Guide to Expert Systems. Reading, Massachusetts: Addison-Wesley Publishing Company, 1986.

Williamson, Mickey. Artificial Intelligence for Micro-computers: The Guide for Business Decisionmakers. New York: Brady Communications Company, Inc., 1986.

Winston, Patrick Henry, and Berthold Klaus Paul Horn. LISP. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984.

VITA

Jill Janene Cress was born January 3, 1964 in Ft. Collins, Colorado. She received her primary and secondary education in Rolla, Missouri. She has received her college education from the University of Missouri-Rolla, in Rolla, Missouri. She received a Bachelor of Science degree in Computer Science from the University of Missouri-Rolla, in Rolla, Missouri in May 1986.

She has been enrolled in the Graduate School of the University Missouri-Rolla since June 1987. She is currently working as a programmer analyst for the United States Geological Survey in Rolla, Missouri.

APPENDIX A

LISTING OF THE CONVERSION EXPERT SYSTEM RULES

```
(DEFINE-INSTANCE MAIN-SCREEN
 ( print-name "MAIN-SCREEN"
   doc-string "Main screen control"
   is  SCREEN-CONTROL)
 (SCREEN-LAYOUTS (INTRO-SCREEN))
 (TEMPLATE-AREA-HEIGHT 23)
 (STATUS  INITIAL-STATUS)
 )

(DEFINE-INSTANCE INTRO-SCREEN
 ( print-name "INTRO-SCREEN"
   doc-string ""
   is  SCREEN-LAYOUT)
 (SYSTEM-MENU  YES)
 (SCREENS-MENU  YES)
 (MENU-BAR-BORDER-COLOR  LIGHT-GRAY)
 (MENU-BAR-TEXT-COLOR  LIGHT-GRAY)    ;; Slot MENU could not be saved of type NULL
 (SCREEN-TEMPLATES
      ( (OPENING-SCREEN  LEFT 0  TOP 2  WIDTH 79  HEIGHT
          20)))
 (PARENT-SCREEN-CONTROL MAIN-SCREEN)
 )

(DEFINE-INSTANCE OPENING-SCREEN
 ( print-name "OPENING-SCREEN"
   doc-string "Opening screen layout"
   is  SCREEN-TEMPLATE)
 (OBJECTS
      ( (CONTINUE  MENU START-CODE "   CONTINUE   ")
        (LEAVE  MENU LEAVE-SCREEN "     EXIT    ")))
 (CONTENTS
      ( (" ") (" ")
        ("                   WELCOME TO THE            ")
        ("              FORTRAN CONVERSION EXPERT SYSTEM      ")
        (" ") (" ") (" ")
        ("        THIS SYSTEM CONVERTS A KBGIS EXPERT SYSTEM INTO")
        ("       FORTRAN CODE WHICH CAN THEN BE COMPILED AND EXECUTED")
        (" ") (" ")
        (" ") (" ")
        ("             " CONTINUE "         " LEAVE)))
 (TEXT-COLOR  GREEN)
 (BORDER-COLOR  BLUE)
 (BORDER  DOUBLE)
 (ORIENTATION  ROW)
 (LAYOUT  LINEAR)
 (SPACES-BETWEEN-COLUMNS 0)
 (SPACES-BETWEEN-ROWS 0)
 )

(DEFINE-INSTANCE LEAVE-SCREEN
 ( print-name "LEAVE"
   doc-string "Asks the user if they are sure they want to exit the system."
   is  POPUP-CONFIRM)
 (TARGET-SLOT CLOSE-REQUEST)
 (TARGET-INSTANCE MAIN-SCREEN)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (DEFAULT-ANSWER  YES)
 (CONTENTS (" Do you want to exit the system? "))
 )

(DEFINE-INSTANCE MESSAGE-SCREEN
 ( print-name "MESSAGE-SCREEN"
   doc-string "Used to output messages to the screen."
   is  OUTPUT-WINDOW)
 (LEFT 0)
 (TOP 21)
 (WIDTH 80)
 (HEIGHT 4)
 (FOREGROUND-COLOR  RED)
 (BACKGROUND-COLOR  WHITE)
 (AUTO-NEWLINE  YES)
 (SCROLLING  SCROLL)
 (CLEAR-BEFORE-NEW-DISPLAY  YES)
 )

(DEFINE-INSTANCE START-CODE
 ( print-name "START-CODE"
   doc-string "Asks the user for the output file path."
   is  POPUP-ASK-USER)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (ANSWER-WIDTH 10)
 (INSTRUCTIONS ("Example  \\gw\\kbgis\\ "))
 (CONTENTS ("Enter the complete directory path for the output files "))
 )

(DEFINE-INSTANCE ENTER-INPUT
 ( print-name "ENTER-INPUT"
   doc-string "Asks the user to enter an input variable."
   is  POPUP-ASK-USER)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (ANSWER-WIDTH 10)
 (CONTENTS ("Enter an input variable  "))
 )

(DEFINE-INSTANCE CONFIRM-INPUT
 ( print-name "CONFIRM-INPUT"
   doc-string "Asks the user if the entered input variable is correct."
   is  POPUP-CONFIRM)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (DEFAULT-ANSWER  YES)
 )

(DEFINE-INSTANCE ENTER-OUTPUT
 ( print-name "ENTER-OUTPUT"
   doc-string "Asks the user to enter the output variable."
   is  POPUP-ASK-USER)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (ANSWER-WIDTH 10)
 (CONTENTS ("Enter the output variable  "))
 )

(DEFINE-INSTANCE CONFIRM-OUTPUT
 ( print-name "CONFIRM-OUTPUT"
   doc-string "Asks the user if the entered output variable is correct."
   is  POPUP-CONFIRM)
 (BORDER-COLOR  BLUE)    ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (DEFAULT-ANSWER  YES)
 )

(DEFINE-INSTANCE ENTER-DEFAULT
 ( print-name "ENTER-DEFAULT"
   doc-string "Asks the user to enter a default output value."
   is  POPUP-ASK-USER)
   (BORDER-COLOR  BLUE)     ;; Slot MENU could not be saved of type NULL
   (CENTER  X-AND-Y)
   (ANSWER-WIDTH 10)
   (CONTENTS ("Enter the default output value  "))
   )

 (DEFINE-INSTANCE CONFIRM-DEFAULT
   ( print-name "CONFIRM-DEFAULT"
     doc-string "Asks the user if the entered default is correct."
     is  POPUP-CONFIRM)
   (BORDER-COLOR  BLUE)     ;; Slot MENU could not be saved of type NULL
   (CENTER  X-AND-Y)
   (DEFAULT-ANSWER  YES)
   )

 (DEFINE-INSTANCE CHECK-FACTOR
   ( print-name "CHECK-FACTOR"
     doc-string "Asks the user what the basic factor should be used for."
     is  POPUP-CONFIRM)
 (BORDER-COLOR  BLUE)     ;; Slot MENU could not be saved of type NULL
 (CENTER  X-AND-Y)
 (DEFAULT-ANSWER  YES)
   )

 (DEFINE-RULE START-SYSTEM
   ( print-name "START-SYSTEM"
     doc-string "Starts the system."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE MAIN-SCREEN IS SCREEN-CONTROL
         WITH STATUS  STARTED)
   THEN
   (INSTANCE MAIN-SCREEN IS SCREEN-CONTROL
         WITH STATUS  RUNNING
         WITH NEW-SCREEN INTRO-SCREEN))

 (DEFINE-RULE GENERATE-CODE
   ( print-name "GENERATE-CODE"
     doc-string "Starts the FORTRAN code generation."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE START-CODE IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES
         WITH DISPLAY ("   STARTED CREATING FORTRAN CODE"))
   (EVALUATE (START ?A))
   (INSTANCE ENTER-INPUT IS POPUP-ASK-USER
         WITH GO  YES))

 (DEFINE-RULE QUIT-SYSTEM
   ( print-name "QUIT-SYSTEM"
     doc-string "Asks the user if they wish to leave the system."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE START-CODE IS POPUP-CONFIRM
         WITH ANSWER  NO)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES)
   (INSTANCE LEAVE-SCREEN IS POPUP-CONFIRM
         WITH GO  YES))

 (DEFINE-RULE CHECK-INPUT
   ( print-name "CHECK-INPUT"
     doc-string "Asks the user to confirm the input variable."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE ENTER-INPUT IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES)
   (INSTANCE CONFIRM-INPUT IS POPUP-CONFIRM
         WITH CONTENTS
     ("Is " ?A " a valid input variable?")
         WITH GO  YES))

 (DEFINE-RULE INCORRECT-INPUT
   ( print-name "INCORRECT-INPUT"
     doc-string "Input variable is invalid, user is asked to re-enter."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-INPUT IS POPUP-CONFIRM
         WITH ANSWER  NO)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES
         WITH DISPLAY ("  INCORRECT INPUT VARIABLE"))
   (INSTANCE ENTER-INPUT IS POPUP-ASK-USER
         WITH GO  YES))

 (DEFINE-RULE CORRECT-INPUT
   ( print-name "CORRECT-INPUT"
     doc-string "Input variable is correct, processing is continued."
     dependency NIL
     direction   FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-INPUT IS POPUP-CONFIRM
         WITH ANSWER  YES)
   (INSTANCE ENTER-INPUT IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES
         WITH DISPLAY ("     WORKING"))
   (EVALUATE (SET-INPUT-VARIABLE ?A))
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES)
   (INSTANCE ENTER-OUTPUT IS POPUP-ASK-USER
        WITH GO  YES))

 (DEFINE-RULE CHECK-OUTPUT
   ( print-name "CHECK-OUTPUT"
     doc-string "Asks the user to confirm the output variable."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 50
     sponsor TOP-SPONSOR)
   (INSTANCE ENTER-OUTPUT IS POPUP-ASK-USER
        WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES)
   (INSTANCE CONFIRM-OUTPUT IS POPUP-CONFIRM
        WITH CONTENTS
     ("Is " ?A " a valid output variable?")
        WITH GO  YES))

 (DEFINE-RULE INCORRECT-OUTPUT
   ( print-name "INCORRECT-OUTPUT"
     doc-string "Invalid output variable, user is asked to re-enter."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-OUTPUT IS POPUP-CONFIRM
        WITH ANSWER  NO)
   (INSTANCE ENTER-OUTPUT IS POPUP-ASK-USER
        WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY (?A " IS AN INCORRECT OUTPUT VARIABLE"))
   (INSTANCE ENTER-OUTPUT IS POPUP-ASK-USER
        WITH GO  YES))

 (DEFINE-RULE CORRECT-OUTPUT
   ( print-name "CORRECT-OUTPUT"
     doc-string "Correct output variable, processing is continued."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-OUTPUT IS POPUP-CONFIRM
        WITH ANSWER  YES)
   (INSTANCE ENTER-OUTPUT IS POPUP-ASK-USER
        WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY ("    WORKING"))
   (EVALUATE (OUTPUT ?A)))

 (DEFINE-RULE CHECK-DEFAULT
   ( print-name "CHECK-DEFAULT"
     doc-string "Asks the user to confirm the default output value."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 50
     sponsor TOP-SPONSOR)
   (INSTANCE ENTER-DEFAULT IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES)
   (INSTANCE CONFIRM-DEFAULT IS POPUP-CONFIRM
         WITH CONTENTS
     ("Is " ?A " a valid default output value?")
         WITH GO  YES))

 (DEFINE-RULE INCORRECT-DEFAULT
   ( print-name "INCORRECT-DEFAULT"
     doc-string "Invalid default, user is asked to re-enter."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-DEFAULT IS POPUP-CONFIRM
         WITH ANSWER  NO)
   (INSTANCE ENTER-DEFAULT IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES
         WITH DISPLAY (?A " IS AN INCORRECT DEFAULT VALUE"))
   (INSTANCE ENTER-DEFAULT IS POPUP-ASK-USER
         WITH GO  YES))

 (DEFINE-RULE CORRECT-DEFAULT
   ( print-name "CORRECT-DEFAULT"
     doc-string "Correct default, processing is continued."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 50
     sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-DEFAULT IS POPUP-CONFIRM
         WITH ANSWER  YES)
   (INSTANCE ENTER-DEFAULT IS POPUP-ASK-USER
         WITH ANSWER ?A)
   THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
         WITH CLEAR  YES
         WITH DISPLAY ("      WORKING"))
   (EVALUATE (SET-DEFAULT ?A)))

 (DEFINE-RULE FINISH-CODE
   ( print-name "FINISH-CODE"
     doc-string "Processing is continued."
     dependency NIL
     direction  FORWARD
     certainty  1.0
     explanation-string ""
     priority 0
   sponsor TOP-SPONSOR)
   (INSTANCE CONFIRM-DEFAULT IS POPUP-CONFIRM
        WITH ANSWER  YES)
 THEN
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY ('     FINDING OVERLAYS'))
   (EVALUATE (SET-COMP-VARIABLES))
   (EVALUATE (FIND-BASICS))
   (EVALUATE (BUILD-BASICS-CODE))
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY ('     BUILDING FORTRAN CODE'))
   (EVALUATE (BUILD-DEF-CODE))
   (EVALUATE (BUILD-INPUT-CODE))
   (EVALUATE (BUILD-FILE-MISSING-CODE))
   (EVALUATE (BUILD-READ-INPUT-CODE))
   (EVALUATE (CONVERT-RULES))
   (EVALUATE (FINISH-OUTPUT-CODE))
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY ('     CREATING FINAL LEGEND CODE'))
   (EVALUATE (BUILD-LEGEND-CODE))
   (EVALUATE (BUILD-END-CODE))
   (INSTANCE MESSAGE-SCREEN IS OUTPUT-WINDOW
        WITH CLEAR  YES
        WITH DISPLAY ('     FINISHED CREATING FORTRAN CODE'))
   (INSTANCE MAIN-SCREEN IS SCREEN-CONTROL
        WITH CLOSE-REQUEST  YES))
```

APPENDIX B

LISP FUNCTIONS USED BY THE CONVERSION EXPERT SYSTEM

```
; Program Ftr.lsp
; Written by Jill Cress
; Purpose  Lisp functions which are used to convert a GoldWorks
;          KBGIS Expert System into Fortran code.

;
; Sets the maximum dimension for the arrays.
(defconstant limit 12)

;
; Initializes the lists and variables which are used by the
;  other functions.
(defun start (path)
   (setf array-key '(array1 array2 array3 array4 array5 array6 array7 array8))
   (setf array-list '("ARRAY1(12)" "ARRAY1(1)" "ARRAY1(2)" "ARRAY1(3)"
      "ARRAY1(4)" "ARRAY1(5)" "ARRAY1(6)" "ARRAY1(7)" "ARRAY1(8)"
      "ARRAY1(9)" "ARRAY1(10)" "ARRAY1(11)" "ARRAY1(12)" "ARRAY2(12)"
      "ARRAY2(1)" "ARRAY2(2)" "ARRAY2(3)" "ARRAY2(4)" "ARRAY2(5)"
      "ARRAY2(6)" "ARRAY2(7)" "ARRAY2(8)" "ARRAY2(9)" "ARRAY2(10)"
      "ARRAY2(11)" "ARRAY2(12)" "ARRAY3(12)" "ARRAY3(1)" "ARRAY3(2)"
      "ARRAY3(3)" "ARRAY3(4)" "ARRAY3(5)" "ARRAY3(6)" "ARRAY3(7)"
      "ARRAY3(8)" "ARRAY3(9)" "ARRAY3(10)" "ARRAY3(11)" "ARRAY3(12)"
      "ARRAY4(12)" "ARRAY4(1)" "ARRAY4(2)" "ARRAY4(3)" "ARRAY4(4)"
      "ARRAY4(5)" "ARRAY4(6)" "ARRAY4(7)" "ARRAY4(8)" "ARRAY4(9)"
      "ARRAY4(10)" "ARRAY4(11)" "ARRAY4(12)" "ARRAY5(12)" "ARRAY5(1)"
      "ARRAY5(2)" "ARRAY5(3)" "ARRAY5(4)" "ARRAY5(5)" "ARRAY5(6)"
      "ARRAY5(7)" "ARRAY5(8)" "ARRAY5(9)" "ARRAY5(10)" "ARRAY5(11)"
      "ARRAY5(12)" "ARRAY6(12)" "ARRAY6(1)" "ARRAY6(2)" "ARRAY6(3)"
      "ARRAY6(4)" "ARRAY6(5)" "ARRAY6(6)" "ARRAY6(7)" "ARRAY6(8)"
      "ARRAY6(9)" "ARRAY6(10)" "ARRAY6(11)" "ARRAY6(12)" "ARRAY7(12)"
      "ARRAY7(1)" "ARRAY7(2)" "ARRAY7(3)" "ARRAY7(4)" "ARRAY7(5)"
      "ARRAY7(6)" "ARRAY7(7)" "ARRAY7(8)" "ARRAY7(9)" "ARRAY7(10)"
      "ARRAY7(11)" "ARRAY7(12)" "ARRAY8(12)" "ARRAY8(1)" "ARRAY8(2)"
      "ARRAY8(3)" "ARRAY8(4)" "ARRAY8(5)" "ARRAY8(6)" "ARRAY8(7)"
      "ARRAY8(8)" "ARRAY8(9)" "ARRAY8(10)" "ARRAY8(11)" "ARRAY8(12)"))
   (setf key-list '())
   (setf factor-list '())
   (setf basics-list '())
   (setf logic-unit 7)
   (setf def-unit 4)
   (setf label 100)
   (setf err-label 855)
;
; Sets up the output files.
   (setf filename0 (make-pathname  directory path
         name "ftr0"  type "dat"))
   (setf datafile (open filename0  direction  output
         element-type 'string-char))
   (setf filename1 (make-pathname  directory path
         name "ftr1"  type "dat"))
   (setf ftr1file (open filename1  direction  output
         element-type 'string-char))
   (setf filename2 (make-pathname  directory path
         name "ftr2"  type "dat"))
   (setf ftr2file (open filename2  direction  output
         element-type 'string-char))
   (setf filename3 (make-pathname  directory path
         name "ftr3"  type "dat"))
   (setf ftr3file (open filename3  direction  output
         element-type 'string-char))
   (setf filename4 (make-pathname  directory path
         name "ftr4"  type "dat"))
   (setf ftr4file (open filename4  direction  output
         element-type 'string-char))
   (setf filename5 (make-pathname  directory path
         name "ftr5"  type "dat"))
   (setf ftr5file (open filename5  direction  output
         element-type 'string-char))

;
; Start the fortran code generation.
   (start-data)
   (start-code))

;
; Builds the initial data declarations for the fortran program.
(defun start-data ()
   (format datafile "°%      PROGRAM FTRGIS")
   (format datafile "°%C")
   (format datafile "°%      CHARACTER*1       ANS")
   (format datafile "°%      REAL*4            COLS")
   (format datafile "°%      LOGICAL*4         COMP")
   (format datafile "°%      CHARACTER*30      FNAME")
   (format datafile "°%      CHARACTER*30      FNAME2")
   (format datafile "°%      CHARACTER*128     HEADER")
   (format datafile "°%      INTEGER*4         I")
   (format datafile "°%      INTEGER*4         IOE")
   (format datafile "°%      INTEGER*4         J")
   (format datafile "°%      INTEGER*4         NUMBER")
   (format datafile "°%      REAL*4            ROWS")
   (format datafile "°%      CHARACTER*8       TYPE"))

;
; Starts the fortran code.
(defun start-code ()
   (format ftr1file "°%C")
   (format ftr1file "°%C  Asks the user which type of map they wish to produce.")
   (format ftr1file "°%C")
   (format ftr1file "°%      WRITE(*,°A)" label)
   (format ftr1file "°%  °A FORMAT(' ENTER THE TYPE OF MAP TO BE PRODUCED',/, " label)
   (format ftr1file "°%     +       ' (GENERAL|LANDFILL) = = > ')")
   (setf label (+ label 10))
   (format ftr1file "°%      READ(*,°A) TYPE " label)
   (format ftr1file "°%  °A FORMAT(A8)" label)
   (format ftr5file "°%C")
   (format ftr5file "°%C File Error Messages."))

;
; Sets up the input variable.
(defun set-input-variable (frame)
   (setf input-var frame)
   (format datafile "°%      CHARACTER*1       °A" input-var))

;
; Sets up the array for the final-legend values, sets the result variable,
;  and establishs the valid result values.
(defun output (frame)
   (cond ((frame-p frame)
          (setf result frame)
          (format datafile "°%      INTEGER*2         °A" frame)
          (setf final-legend 'fortran-legend)
          (make-instance final-legend  is frame)
          (setf output-values (instance-all-slots final-legend))
          (setf (slot-value 'message-screen 'clear) ' yes)
          (setf (slot-value 'enter-default 'go) ' yes))
         (T
          (setf (slot-value 'confirm-output 'answer) ' no))))
```

```
;
; Sets up the default output value and puts the integer output values
;  into the output array instance.
(defun set-default (frame)
   (cond ((member frame output-values)
          (setf default frame)
          (setf slot-list output-values)
          (setf count 0)
          (dotimes (I (length output-values))
              (cond ((neq (car slot-list) default)
                     (setf count (+ count 1))
                     (setf (slot-value final-legend (car slot-list)) count))
                    (T
                     (setf (slot-value final-legend (car slot-list)) 0)))
              (setf slot-list (cdr slot-list))))
         (T
          (setf (slot-value 'confirm-default 'answer) ' no))))

;
; A composite factor will be used therefore the necessary variables are
;  initialized.
(defun set-comp-variables ()
   (setf comp-var 'SOILS)
   (format datafile "°%      INTEGER*4          °A" comp-var)
   (setf comp-array 'SLS)
   (setf comp-list '())
   (setf comp-unit 3))

;
; The following series of functions are used to develop a list
; of all of the factors which exist in the rule-base.
(defun find-basics ()
   (setf rules (all-rules))
   (find-basics-rules rules))

(defun find-basics-rules (rule-list)
   (cond ((neq rule-list '())
          (setf rule (car rule-list))
          (setf cons (rule-consequent rule))
          (cond ((and(member 'with cons) (member result cons))
                 (setf result-slot (nth 1 (member result cons)))
                 (cond ((member result-slot output-values)
                        (setf ante (rule-antecedent rule))
                        (basics-variable ante))
                       (T nil)))
                (T nil))
          (find-basics-rules (cdr rule-list)))
         (T nil)))

(defun basics-variable (ante-list)
   (cond ((OR(eq 'AND (car ante-list))(eq 'OR (car ante-list)))
          (dotimes (I (- (length ante-list) 1))
              (setf temp (nth (+ I 1) ante-list))
              (basics-variable temp)))
         ((eq 'INSTANCE (car ante-list))
          (setf a-list (member 'with ante-list))
          (build-basics-list a-list))
         (T nil)))

(defun build-basics-list (b-list)
   (cond ((neq b-list '())
          (setf basic (nth 1 b-list))
          (cond ((member basic basics-list) nil)
                (T
                 (setf basics-list (cons basic basics-list))))
          (setf b-list (member 'with (cdr b-list)))
          (build-basics-list b-list))
         (T nil)))

;
; Determines which of the factors should be composite overlays and/or
; non-composite factors.
(defun build-basics-code ()
    (setf temp-b-list basics-list)
    (dotimes (I (length basics-list))
        (setf basic-factor (car temp-b-list))
        (setf temp-b-list (cdr temp-b-list))
        (setf (slot-value 'check-factor 'contents)
            '("Is " basic-factor " a " comp-var " overlay "))
        (setf (slot-value 'check-factor 'go) ' yes)
        (if (eq (slot-value 'check-factor 'answer) ' yes)
            (comp-factors basic-factor))
        (setf (slot-value 'check-factor 'contents)
            '("Is " basic-factor " an independent overlay "))
        (setf (slot-value 'check-factor 'go) ' yes)
        (if (eq (slot-value 'check-factor 'answer) ' yes)
            (factor basic-factor))))

;
; Builds the fortran code to split the composite factor into the
;  seperate overlays.
(defun comp-factors (frame)
    (setf comp-list (cons frame comp-list))
    (setf array-name (car array-key))
    (setf array-key (cdr array-key))
    (setf array (car array-list))
    (setf array-list (cdr array-list))
    (format datafile "°%      INTEGER*2          °A" frame)
    (format datafile "°%      INTEGER*2          °A" array)
    (setf key-list (cons array-name key-list))
    (setf key-list (cons frame key-list))
    (make-instance array-name  is frame)
    (setf slot-list (instance-all-slots array-name))
    (setf slots (instance-all-slots array-name))
    (setf dimen (length slot-list))
    (dotimes (I limit)
        (cond ((neq slot-list '())
               (setf (slot-value array-name (car slot-list)) (car array-list))
               (setf slot-list (cdr slot-list)))
              (T nil))
        (setf array-list (cdr array-list)))
    (setf label (+ label 10))
    (setf return-label (- label 1))
    (format ftr2file "°%C")
    (format ftr2file "°%C Asks the user to enter the °A overlay value " frame)
    (format ftr2file "°%C which is associated with the input °A value." comp-var)
    (format ftr2file "°%  °A      WRITE(*,°A)" return-label label)
    (format ftr2file "°%  °A      FORMAT(' ENTER THE ASSOCIATED °A VALUE ',/, " label frame)
    (format ftr2file "°%     +            ' CHOICES  °S ',/, " slots)
    (format ftr2file "°%     +            ' = = > ')")
    (setf label (+ label 10))
    (setf slot-list (instance-all-slots array-name))
    (format ftr2file "°%          READ(*,°A) CHOICE" label)
    (format ftr2file "°%  °A      FORMAT(A5)" label)
    (format ftr2file "°%          IF (CHOICE .EQ. '°A') THEN" (car slot-list))
    (format ftr2file "°%           °A(°A,°A) = 1" comp-array comp-var (length comp-list))
    (setf slot-list (cdr slot-list))
    (dotimes (I (- (length slots) 1))
        (format ftr2file "°%          ELSEIF (CHOICE .EQ. '°A') THEN" (car slot-list))
        (format ftr2file "°%           °A(°A,°A) = °A"
        comp-array comp-var (length comp-list) (+ 1 2))
       (setf slot-list (cdr slot-list)))
    (format ftr2file "~%          ELSEIF (CHOICE .EQ. ' ') THEN" )
    (format ftr2file "~%           ~A(~A,~A) = 0" comp-array comp-var (length comp-list))
    (format ftr2file "~%          ELSE")
    (format ftr2file "~%            GOTO ~A" return-label)
    (format ftr2file "~%          ENDIF"))

;
; Builds the fortran code which creates the definition file entries
;  for each factor.
(defun factor (frame)
    (setf factor-list (cons frame factor-list))
    (cond ((member frame key-list)
          (setf temp-list (member frame key-list))
          (setf array-name (nth 1 temp-list))
          (setf slots (instance-all-slots array-name))
          (setf dimen (length slots)))
         (T
          (setf array-name (car array-key))
          (setf array-key (cdr array-key))
          (setf array (car array-list))
          (setf array-list (cdr array-list))
          (format datafile "~%      INTEGER*2          ~A" frame)
          (format datafile "~%      INTEGER*2          ~A" array)
          (setf key-list (cons array-name key-list))
          (setf key-list (cons frame key-list))
          (make-instance array-name  is frame)
          (setf slot-list (instance-all-slots array-name))
          (setf slots (instance-all-slots array-name))
          (setf dimen (length slot-list))
          (dotimes (I limit)
             (cond ((neq slot-list '())
                   (setf (slot-value array-name (car slot-list)) (car array-list))
                   (setf slot-list (cdr slot-list)))
                   (T nil))
             (setf array-list (cdr array-list)))))
    (setf label (+ label 10))
    (format ftr3file "~%C")
    (format ftr3file "~%C Asks the user to set the integer values associated")
    (format ftr3file "~%C with the given ~A choices." frame)
    (format ftr3file "~%          WRITE(*,~A)" label)
    (format ftr3file "~%  ~A      FORMAT(' ENTER THE INTEGER VALUES WHICH REPRESENT',/, "
     label)
    (format ftr3file "~%     +            ' THE ~A CHOICES  ',/," frame)
    (format ftr3file "~%     +            ' ~S ',/, " slots)
    (format ftr3file "~%     +            ' (USE FORMAT ===>XXX XXX XXX)',/,")
    (format ftr3file "~%     +            ' ==> ')")
    (setf label (+ label 10))
    (format ftr3file "~%          READ(*,~A) (~A(I),I=1,~A)" label array-name dimen)
    (format ftr3file "~%  ~A      FORMAT(" label)
    (dotimes (I (- dimen 1))
       (format ftr3file "I3,1X,"))
    (format ftr3file "I3)"))

; Builds the fortran code which asks the user if they want to use
;  a composite factor.
(defun build-def-code ()
    (setf label (+ label 10))
    (format ftr1file "~%C")
    (format ftr1file "~%C Asks the user if they have a ~A file." comp-var)
    (format ftr1file "~%       WRITE(*,~A)" label)
    (format ftr1file "~%  ~A FORMAT(' DO YOU HAVE A ~A FILE (Y|N) ==> ') " label comp-var)
    (setf label (+ label 10))
    (format ftr1file "~%       READ(*,~A) ANS" label)
```

```
   (format ftr1file "°%  °A FORMAT(A1)" label)
   (format ftr1file "°%      COMP = .FALSE.")
   (format ftr1file "°%C")
   (format ftr1file "°%C If they do have a °A file then its definition file " comp-var)
   (format ftr1file "°%C is accessed.")
   (format ftr1file "°%      IF(ANS .EQ. 'Y') THEN")
   (build-comp-def-code)
   (build-noncomp-def-code))

; Builds the fortran code which reads the composite definition file
;  if it already exists and builds it if it doesn't.
(defun build-comp-def-code ()
   (make-instance 'dummy  is comp-var)
   (setf dimen (length (instance-all-slots 'dummy)))
   (format datafile "°%      INTEGER*2          °A(°A,°A)"
    comp-array dimen (length comp-list))
   (format datafile "°%      CHARACTER*5      CHOICE")
   (format ftr1file "°%      COMP = .TRUE.")
   (format ftr1file "°%C")
   (format ftr1file "°%C Initializes the °A overlay arrays." comp-var)
   (setf temp-list comp-list)
   (dotimes (I (length comp-list))
      (setf c-factor (car temp-list))
      (setf temp-list (cdr temp-list))
      (setf array (nth 1 (member c-factor key-list)))
      (setf elements (length (instance-all-slots array)))
      (setf label (+ label 10))
      (format ftr1file "°%C")
      (format ftr1file "°%      DO °A I = 1,°A" label elements)
      (format ftr1file "°%        °A(I) = I" array)
      (format ftr1file "°%  °A   CONTINUE" label))
   (setf label (+ label 10))
   (format ftr1file "°%C")
   (format ftr1file "°%C Asks the user to enter the °A definition file name" comp-var)
   (format ftr1file "°%C  and whether or not the file already exists.")
   (format ftr1file "°%      WRITE(*,°A) " label)
   (format ftr1file "°%  °A   FORMAT(' ENTER THE COMPLETE PATH NAME OF'," label)
   (format ftr1file "°%     +         ' THE °A DEFINITION FILE ',/," comp-var)
   (format ftr1file "°%     +         ' ==> ')")
   (setf label (+ label 10))
   (format ftr1file "°%      READ(*,°A) FNAME" label)
   (format ftr1file "°%  °A   FORMAT(A30)" label)
   (setf label (+ label 10))
   (format ftr1file "°%      WRITE(*,°A) " label)
   (format ftr1file "°%  °A   FORMAT(' DOES THE DEFINITION FILE ALREADY EXIST (Y|N) ==> ') "
    label)
   (setf label (+ label 10))
   (format ftr1file "°%      READ(*,°A) ANS " label)
   (format ftr1file "°%  °A   FORMAT(A1)" label)
   (format ftr1file "°%C")
   (format ftr1file "°%C If the definition file already exists then it is")
   (format ftr1file "°%C  opened and read.")
   (format ftr1file "°%      IF(ANS .EQ. 'Y') THEN ")
   (setf err-label (+ err-label 5))
   (setf open-err err-label)
   (format ftr1file "°%        OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
    def-unit open-err)
   (format ftr1file "°%     +         ACCESS='SEQUENTIAL',FORM='FORMATTED')")
   (format ftr5file "°%  °A WRITE(*,°A) IOE" open-err (+ open-err 2))
   (format ftr5file "°%  °A FORMAT(' ERROR OPENING °A ',/, " (+ open-err 2) comp-var)
   (format ftr5file "°%     +      ' DEFINITION FILE, ERROR = ',I4) " )
   (format ftr5file "°%      GOTO 1000 ")
   (setf label (+ label 10))
   (setf err-label (+ err-label 5))
   (format ftr1file "°%        READ(°A,°A,ERR=°A,IOSTAT=IOE) ((°A(I,J),I=1,°A),J=1,°A)"
 def-unit label err-label comp-array dimen (length comp-list))
(format ftr1file "°%  °A     FORMAT(°A(I1))" label (* dimen (length comp-list)))
(format ftr5file "°%  °A WRITE(*,°A) IOE" err-label (+ err-label 2))
(format ftr5file "°%  °A FORMAT(' ERROR READING °A ',/, " (+ err-label 2) comp-var)
(format ftr5file "°%     +       ' DEFINITION FILE, ERROR = ',I4) " )
(format ftr5file "°%     GOTO 1000 ")
(format ftr1file "°%C")
(format ftr1file "°%C Since the definition file does not already exist ")
(format ftr1file "°%C  it is created and saved.")
(format ftr1file "°%        ELSE")
(format ftr1file "°%          OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='NEW',"
 def-unit open-err)
(format ftr1file "°%     +         ACCESS='SEQUENTIAL',FORM='FORMATTED')")
(setf label (+ label 10))
(format ftr1file "°%C")
(format ftr1file "°%C Initializes the composite array.")
(format ftr1file "°%          DO °A J = 1,°A" label (length comp-list))
(format ftr1file "°%            DO °A I = 1,°A" (- label 1) dimen)
(format ftr1file "°%             °A(I,J) = 0" comp-array)
(format ftr1file "°%  °A        CONTINUE" (- label 1))
(format ftr1file "°%  °A      CONTINUE" label)
(setf label (+ label 10))
(format ftr1file "°%C")
(format ftr1file "°%C Builds the composite definition file.")
(format ftr1file "°%          DO °A I = 1,°A" label dimen)
(format ftr2file "°%  °A      CONTINUE" label)
(setf label (+ label 10))
(format ftr1file "°%C")
(format ftr1file "°%C  Asks the user to input a valid °A value." comp-var)
(format ftr1file "°%            WRITE(*,°A)" label)
(format ftr1file "°%  °A        FORMAT(' ENTER THE INTEGER °A VALUE -- '," label comp-var)
(format ftr1file "°%     +               ' ENTER 999 TO END -- (FORMAT = = >XXX )',/,")
(format ftr1file "°%     +               ' = = > ')")
(setf label (+ label 10))
(format ftr1file "°%            READ(*,°A) °A" label comp-var)
(format ftr1file "°%  °A        FORMAT(I3)" label)
(setf label (+ label 10))
(format ftr1file "°%           IF (°A .EQ. 999) GOTO °A" comp-var label)
(format ftr2file "°%  °A      CONTINUE" label)
(setf label (+ label 10))
(setf err-label (+ err-label 5))
(format ftr2file "°%C")
(format ftr2file "°%C Saves the definition file.")
(format ftr2file "°%          WRITE(°A,°A,ERR=°A,IOSTAT=IOE) ((°A(I,J),I=1,°A),J=1,°A)"
 def-unit label err-label comp-array dimen (length comp-list))
(format ftr2file "°%  °A     FORMAT(°A(I1))" label (* dimen (length comp-list)))
(format ftr5file "°%  °A WRITE(*,°A) IOE" err-label (+ err-label 2))
(format ftr5file "°%  °A FORMAT(' ERROR WRITING °A ',/, " (+ err-label 2) comp-var)
(format ftr5file "°%     +       ' DEFINITION FILE, ERROR = ',I4) " )
(format ftr5file "°%     GOTO 1000 ")
(format ftr2file "°%          ENDFILE(°A) " def-unit)
(format ftr2file "°%        ENDIF")
(format ftr2file "°%        CLOSE(°A)" def-unit))

; Builds the fortran code which reads the non-composite definition file
;  if it already exists, and builds it if it doesn't.
(defun build-noncomp-def-code ()
   (format ftr2file "°%C")
   (format ftr2file "°%C Since a composite file is not being used the user is asked")
   (format ftr2file "°%C if they have the necessary non-composite files.")
   (format ftr2file "°%      ELSE ")
   (setf label (+ label 10))
   (format ftr2file "°%        WRITE(*,°A) " label)
   (format ftr2file "°%  °A   FORMAT(' DO YOU HAVE °S FILES '," label factor-list)
   (format ftr2file "°%     +         ' (Y|N) = = > ')")
```

```
(setf label (+ label 10))
(format ftr2file "~%      READ(*,~A) ANS" label)
(format ftr2file "~% ~A  FORMAT(A1)" label)
(format ftr2file "~%C")
(format ftr2file "~%C Since there are non-composite files the definition file")
(format ftr2file "~%C  name is asked for and whether or not the file already exists.")
(format ftr2file "~%      IF(ANS .EQ. 'Y') THEN ")
(setf label (+ label 10))
(format ftr2file "~%        WRITE(*,~A) " label)
(format ftr2file "~% ~A    FORMAT(' ENTER COMPLETE PATH NAME OF THE DEFINITION FILE ',/,"
 label)
(format ftr2file "~%    +         ' = = > ')")
(setf label (+ label 10))
(format ftr2file "~%        READ(*,~A) FNAME" label)
(format ftr2file "~% ~A    FORMAT(A30)" label)
(setf label (+ label 10))
(format ftr2file "~%        WRITE(*,~A) " label)
(format ftr2file "~% ~A    FORMAT(' DOES THE DEFINITION FILE ALREADY EXIST (Y|N) = = > ') "
 label)
(setf label (+ label 10))
(format ftr2file "~%        READ(*,~A) ANS " label)
(format ftr2file "~% ~A    FORMAT(A1)" label)
(format ftr2file "~%C")
(format ftr2file "~%C The definition file is opened and read.")
(format ftr2file "~%        IF(ANS .EQ. 'Y') THEN ")
(setf err-label (+ err-label 5))
(setf open-err err-label)
(format ftr2file "~%          OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
 def-unit open-err)
(format ftr2file "~%    +          ACCESS='SEQUENTIAL',FORM='FORMATTED')")
(format ftr5file "~% ~A WRITE(*,~A) IOE" open-err (+ open-err 2))
(format ftr5file "~% ~A FORMAT(' ERROR OPENING ~A ',/, " (+ open-err 2) factor-list)
(format ftr5file "~%    +     ' DEFINITION FILE, ERROR = ',I4) " )
(format ftr5file "~%     GOTO 1000 ")
(setf temp-list factor-list)
(setf err-label (+ err-label 5))
(dotimes (I (length temp-list))
  (setf factor (car temp-list))
  (setf temp-list (cdr temp-list))
  (setf array (nth 1 (member factor key-list)))
  (setf elements (length (instance-all-slots array)))
  (setf label (+ label 10))
  (format ftr2file "~%          READ(~A,~A,ERR=~A,IOSTAT=IOE) " def-unit label err-label)
  (format ftr2file "(~A(I),I=1,~A)" array elements)
  (format ftr3file "~%          WRITE(~A,~A,ERR=~A,IOSTAT=IOE) " def-unit label (+ err-label 5))
  (format ftr3file "(~A(I),I=1,~A)" array elements)
  (format ftr2file "~% ~A      FORMAT(" label)
  (dotimes (J (- elements 1))
    (format ftr2file "I3,"))
  (format ftr2file "I3)"))
(format ftr5file "~% ~A WRITE(*,~A) IOE" err-label (+ err-label 2))
(format ftr5file "~% ~A FORMAT(' ERROR READING ~A ',, " (+ err-label 2) factor-list)
(format ftr5file "~%    +     ' DEFINITION FILE, ERROR = ',I4) " )
(format ftr5file "~%     GOTO 1000 ")
(setf err-label (+ err-label 5))
(format ftr5file "~% ~A WRITE(*,~A) IOE" err-label (+ err-label 2))
(format ftr5file "~% ~A FORMAT(' ERROR WRITING ~A ',/," (+ err-label 2) factor-list)
(format ftr5file "~%    +     ' DEFINITION FILE, ERROR = ',I4) " )
(format ftr5file "~%     GOTO 1000 ")
(format ftr2file "~%C")
(format ftr2file "~%C The definition file is created and saved.")
(format ftr2file "~%        ELSE ")
(format ftr2file "~%          OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='NEW',"
 def-unit open-err)
(format ftr2file "~%    +          ACCESS='SEQUENTIAL',FORM='FORMATTED') ")
```

```
    (format ftr3file "°%          ENDFILE(°A) " def-unit)
    (format ftr3file "°%        ENDIF")
    (format ftr3file "°%        CLOSE(°A)" def-unit))

;
; Starts the fortran code which reads in the input.
(defun build-input-code ()
    (format ftr4file "°%C")
    (format ftr4file "°%C Reads the input from the °A file and converts" comp-var)
    (format ftr4file "°%C  it into seperate overlay values.")
    (format ftr4file "°%        IF (COMP) THEN")
    (build-comp-input-code)
    (format ftr4file "°%C")
    (format ftr4file "°%C Reads the input from the non-composite files.")
    (format ftr4file "°%        ELSE")
    (build-noncomp-input-code)
    (setf result-unit logic-unit)
    (setf logic-unit 7))

;
; Builds the fortran code which reads in the composite input and
;  converts it into seperate overlay values.
(defun build-comp-input-code ()
    (setf comp-unit 3)
    (setf label (+ label 10))
    (setf err-label (+ err-label 5))
    (format ftr4file "°%          READ(°A,°A,REC=1,ERR=°A,IOSTAT=IOE) °A"
     comp-unit label err-label input-var)
    (format ftr4file "°%  °A     FORMAT(A1) " label)
    (format ftr5file "°%  °A WRITE(*,°A) IOE" err-label (+ err-label 2))
    (format ftr5file "°%  °A FORMAT(' ERROR READING °A FILE INPUT, ERROR = ',I4) "
     (+ err-label 2) comp-var)
    (format ftr5file "°%      GOTO 1000")
    (format ftr4file "°%          °A = ICHAR(°A)" comp-var input-var)
    (setf subscript (- (length comp-list) 1))
    (dotimes (I (length comp-list))
       (setf factor (nth subscript comp-list))
       (format ftr4file "°%          °A = °A(°A,°A)" factor comp-array comp-var (+ I 1))
       (setf subscript (- subscript 1))))

;
; Builds the fortran which reads in the input from the non-composite files.
(defun build-noncomp-input-code ()
    (setf label (+ label 10))
    (setf err-label (+ err-label 5))
    (setf temp-list factor-list)
    (dotimes (I (length factor-list))
       (setf factor (car temp-list))
       (setf temp-list (cdr temp-list))
       (format ftr4file "°%          READ(°A,°A,REC=1,ERR=°A,IOSTAT=IOE) °A"
        logic-unit label err-label input-var)
       (setf logic-unit (+ logic-unit 1))
       (format ftr4file "°%          °A = ICHAR(°A) " factor input-var))
    (format ftr4file "°%  °A     FORMAT(A1) " label)
    (format ftr4file "°%        ENDIF")
    (format ftr5file "°%  °A WRITE(*,°A) IOE" err-label (+ err-label 2))
    (format ftr5file "°%  °A FORMAT(' ERROR READING A NON-COMPOSITE FILE, ERROR = ',I4) "
     (+ err-label 2))
    (format ftr5file "°%      GOTO 1000"))

;
; Builds the fortran code to handle the case of missing files.
(defun build-file-missing-code ()
    (format ftr3file "°%C")
    (format ftr3file "°%C Since neither the composite file or all of the")
    (format ftr3file "~%C  non-composite files are available an error message")
    (format ftr3file "~%C  is output and processing is stopped.")
    (format ftr3file "~%        ELSE")
    (setf label (+ label 10))
    (format ftr3file "~%          WRITE(*,~A)" label)
    (format ftr3file "~%  ~A     FORMAT(' CANNOT PRODUCE A ~A FILE ',/," label result)
    (format ftr3file "~%     +          ' -- CRITICAL DATA FILES ARE NOT AVAILABLE.')")
    (format ftr3file "~%          GOTO 1000")
    (format ftr3file "~%        ENDIF")
    (format ftr3file "~%      ENDIF"))
;
; Starts the fortran code which opens the GIS file(s) for input
;  and creates the output file.
(defun build-read-input-code ()
    (format ftr3file "~%C")
    (format ftr3file "~%C Opens the input and output files.")
    (format ftr3file "~%      IF (COMP) THEN")
    (build-read-comp-code)
    (format ftr3file "~%C")
    (format ftr3file "~%C Opens the input and output files.")
    (format ftr3file "~%      ELSE ")
    (build-read-noncomp-code))


;
; Builds the fortran code which opens the composite input file and
;  creates the result output file.
(defun build-read-comp-code ()
    (setf label (+ label 10))
    (format ftr3file "~%        WRITE(*,~A) " label)
    (format ftr3file "~%  ~A   FORMAT(' ENTER THE ~A FILE NAME  = = >  ')" label comp-var)
    (setf label (+ label 10))
    (format ftr3file "~%        READ(*,~A) FNAME " label)
    (format ftr3file "~%  ~A   FORMAT(A30)" label)
    (setf err-label (+ err-label 5))
    (setf open-err err-label)
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     comp-unit open-err)
    (format ftr3file "~%     +        ACCESS='DIRECT',FORM='UNFORMATTED',RECL=4)")
    (format ftr5file "~%  ~A WRITE(*,~A) IOE " open-err (+ open-err 2))
    (format ftr5file "~%  ~A FORMAT(' ERROR OPENING ~A FILE, ERROR  =  ',I4) "
     (+ open-err 2) comp-var)
    (format ftr5file "~%      GOTO 1000")
    (setf err-label (+ err-label 5))
    (setf head-err err-label)
    (format ftr3file "~%C")
    (format ftr3file "~%C Reads the number of cols and rows of input that")
    (format ftr3file "~%C  need to be processed.")
    (format ftr3file "~%        READ(~A,REC=5,ERR=~A,IOSTAT=IOE) COLS" comp-unit err-label)
    (format ftr3file "~%        READ(~A,REC=6,ERR=~A,IOSTAT=IOE) ROWS" comp-unit err-label)
    (format ftr5file "~%  ~A WRITE(*,~A) IOE " err-label (+ err-label 2))
    (format ftr5file "~%  ~A FORMAT(' ERROR READING ~A FILE HEADER, ERROR  =  ',I4) "
     (+ err-label 2) comp-var)
    (format ftr5file "~%      GOTO 1000")
    (format ftr3file "~%        CLOSE(~A) " comp-unit)
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     comp-unit open-err)
    (format ftr3file "~%     +        ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
    (setf label (+ label 10))
    (format ftr3file "~%        WRITE(*,~A) " label)
    (format ftr3file "~%  ~A   FORMAT(' ENTER THE ~A FILE NAME  = = >  ')" label result)
    (setf label (+ label 10))
    (format ftr3file "~%        READ(*,~A) FNAME2 " label)
    (format ftr3file "~%  ~A   FORMAT(A30)" label)
    (setf label (+ label 10))
    (format ftr3file "~%      WRITE(*,~A) " label)
    (format ftr3file "~%  ~A FORMAT(' DOES THE ~A FILE ALREADY EXIST (Y|N) = = > ')"
     label result)
    (setf label (+ label 10))
    (format ftr3file "~%      READ(*,~A) ANS " label)
    (format ftr3file "~%  ~A FORMAT(A1)" label)
    (format ftr3file "~%C")
    (format ftr3file "~%C The ~A file is opened." result)
    (format ftr3file "~%      IF(ANS .EQ. 'Y') THEN ")
    (setf err-label (+ err-label 5))
    (setf open-err2 err-label)
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME2,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     result-unit open-err2)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
    (format ftr5file "~%  ~A WRITE(*,~A) IOE" open-err2 (+ open-err2 2))
    (format ftr5file "~%  ~A FORMAT(' ERROR OPENING ~A FILE, ERROR = ',I4) "
     (+ open-err2 2) result)
    (format ftr5file "~%      GOTO 1000")
    (format ftr3file "~%      ELSE")
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME2,ERR=~A,IOSTAT=IOE,STATUS='NEW',"
     result-unit open-err2)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
    (format ftr3file "~%      ENDIF")
    (format ftr3file "~%C")
    (format ftr3file "~%C Reads the header information from the ~A file" comp-var)
    (format ftr3file "~%C  and writes it to the output file.")
    (format ftr3file "~%        READ(~A,REC=1,ERR=~A,IOSTAT=IOE) HEADER" comp-unit head-err)
    (setf err-label (+ err-label 5))
    (format ftr3file "~%        WRITE(~A,REC=1,ERR=~A,IOSTAT=IOE) HEADER" result-unit err-label)
    (format ftr5file "~%  ~A WRITE(*,~A) IOE" err-label (+ err-label 2))
    (format ftr5file "~%  ~A FORMAT(' ERROR WRITING ~A FILE HEADER, ERROR = ',I4) "
     (+ err-label 2) result)
    (format ftr5file "~%      GOTO 1000")
    (format ftr3file "~%        CLOSE(~A) " comp-unit)
    (format ftr3file "~%        CLOSE(~A) " result-unit)
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     comp-unit open-err)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME2,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     result-unit open-err2)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr3file "~%C")
    (format ftr3file "~%C Calulates the number of bytes of input data.")
    (format ftr3file "~%        NUMBER = INT(COLS * ROWS) + 128"))


;
; Builds the fortran code which opens the non-composite input files and
;  creates the result output file.
(defun build-read-noncomp-code ()
   (setf temp-list factor-list)
   (dotimes (I (- (length factor-list) 1))
      (setf factor (car temp-list))
      (setf temp-list (cdr temp-list))
      (setf label (+ label 10))
      (format ftr3file "~%        WRITE(*,~A) " label)
      (format ftr3file "~%  ~A   FORMAT(' ENTER THE ~A FILE NAME = = > ')" label factor)
      (setf label (+ label 10))
      (format ftr3file "~%        READ(*,~A) FNAME " label)
      (format ftr3file "~%  ~A   FORMAT(A30)" label)
      (setf err-label (+ err-label 5))
      (setf open-err err-label)
      (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
       logic-unit open-err)
      (format ftr3file "~%     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
      (setf logic-unit (+ logic-unit 1))
      (format ftr5file "~%  ~A WRITE(*,~A) IOE " open-err (+ open-err 2))
   (format ftr5file "°%  °A FORMAT(' ERROR OPENING °A FILE, ERROR = ',I4) "
    (+ open-err 2) factor)
   (format ftr5file "°%      GOTO 1000"))
 (setf factor (car temp-list))
 (setf temp-list (cdr temp-list))
 (setf label (+ label 10))
 (format ftr3file "°%       WRITE(*,°A) " label)
 (format ftr3file "°%  °A   FORMAT(' ENTER THE °A FILE NAME = = > ')" label factor)
 (setf label (+ label 10))
 (format ftr3file "°%       READ(*,°A) FNAME " label)
 (format ftr3file "°%  °A   FORMAT(A30)" label)
 (setf err-label (+ err-label 5))
 (setf open-err err-label)
 (format ftr3file "°%       OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
  logic-unit open-err)
 (format ftr3file "°%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=4)")
 (format ftr5file "°%  °A WRITE(*,°A) IOE " open-err (+ open-err 2))
 (format ftr5file "°%  °A FORMAT(' ERROR OPENING °A FILE, ERROR = ',I4) "
  (+ open-err 2) factor)
 (format ftr5file "°%      GOTO 1000")
 (setf err-label (+ err-label 5))
 (setf head-err err-label)
 (format ftr3file "°%C")
 (format ftr3file "°%C Reads the number of cols and rows of input that")
 (format ftr3file "°%C  need to be processed.")
 (format ftr3file "°%       READ(°A,REC=5,ERR=°A,IOSTAT=IOE) COLS" logic-unit err-label)
 (format ftr3file "°%       READ(°A,REC=6,ERR=°A,IOSTAT=IOE) ROWS" logic-unit err-label)
 (format ftr5file "°%  °A WRITE(*,°A) IOE " err-label (+ err-label 2))
 (format ftr5file "°%  °A FORMAT(' ERROR READING °A FILE HEADER, ERROR = ',I4) "
  (+ err-label 2) factor)
 (format ftr5file "°%     GOTO 1000")
 (format ftr3file "°%       CLOSE(°A) " logic-unit)
 (format ftr3file "°%       OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
  logic-unit open-err)
 (format ftr3file "°%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
 (setf label (+ label 10))
 (format ftr3file "°%       WRITE(*,°A) " label)
 (format ftr3file "°%  °A   FORMAT(' ENTER THE °A FILE NAME = = > ')" label result)
 (setf label (+ label 10))
 (format ftr3file "°%       READ(*,°A) FNAME2 " label)
 (format ftr3file "°%  °A   FORMAT(A30)" label)
 (setf label (+ label 10))
 (format ftr3file "°%     WRITE(*,°A) " label)
 (format ftr3file "°%  °A FORMAT(' DOES THE °A FILE ALREADY EXIST (Y|N) = = > ')"
  label result)
 (setf label (+ label 10))
 (format ftr3file "°%     READ(*,°A) ANS " label)
 (format ftr3file "°%  °A FORMAT(A1)" label)
 (format ftr3file "°%C")
 (format ftr3file "°%C The °A file is opened." result)
 (format ftr3file "°%     IF(ANS .EQ. 'Y') THEN ")
 (setf err-label (+ err-label 5))
 (setf open-err2 err-label)
 (format ftr3file "°%       OPEN(UNIT=°A,FILE=FNAME2,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
  result-unit open-err2)
 (format ftr3file "°%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
 (format ftr5file "°%  °A WRITE(*,°A) IOE " open-err2 (+ open-err2 2))
 (format ftr5file "°%  °A FORMAT(' ERROR OPENING °A FILE, ERROR = ',I4) "
  (+ open-err2 2) result)
 (format ftr5file "°%     GOTO 1000")
 (format ftr3file "°%     ELSE")
 (format ftr3file "°%       OPEN(UNIT=°A,FILE=FNAME2,ERR=°A,IOSTAT=IOE,STATUS='NEW',"
  result-unit open-err2)
 (format ftr3file "°%     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)")
 (format ftr3file "°%     ENDIF ")
```

```
    (format ftr3file "~%C")
    (format ftr3file "~%C Reads the header information from the ~A file" factor)
    (format ftr3file "~%C  and writes it to the output file.")
    (format ftr3file "~%        READ(~A,REC=1,ERR=~A,IOSTAT=IOE) HEADER" logic-unit head-err)
    (setf err-label (+ err-label 5))
    (format ftr3file "~%        WRITE(~A,REC=1,ERR=~A,IOSTAT=IOE) HEADER" result-unit err-label)
    (format ftr5file "~%  ~A WRITE(*,~A) IOE" err-label (+ err-label 2))
    (format ftr5file "~%  ~A FORMAT(' ERROR WRITING ~A FILE HEADER, ERROR = ',I4) "
     (+ err-label 2) result)
    (format ftr5file "~%      GOTO 1000")
    (format ftr3file "~%        CLOSE(~A) " logic-unit)
    (format ftr3file "~%        CLOSE(~A) " result-unit)
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     logic-unit open-err)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr3file "~%        OPEN(UNIT=~A,FILE=FNAME2,ERR=~A,IOSTAT=IOE,STATUS='OLD',"
     result-unit open-err2)
    (format ftr3file "~%     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr3file "~%C")
    (format ftr3file "~%C Calulates the number of bytes of input data.")
    (format ftr3file "~%        NUMBER = INT(COLS * ROWS) + 128")
    (format ftr3file "~%      ENDIF"))

;
; The following series of functions are used to convert the rule base
;  into valid fortran if,elseif logic.
(defun convert-rules ()
    (setf control 0)
    (setf rules (all-rules))
    (find-rules rules))

(defun find-rules (rule-list)
    (cond ((neq rule-list '())
           (setf rule (car rule-list))
           (setf cons (rule-consequent rule))
           (cond ((and(member 'with cons) (member result cons))
                  (setf result-slot (nth 1 (member result cons)))
                  (cond ((member result-slot output-values)
                         (cond ((neq default result-slot)
                                (if (eq control 0)
                                    (format ftr4file "~%        IF")
                                    (format ftr4file "~%        ELSEIF"))
                                (setf control 1)
                                (setf ante (rule-antecedent rule))
                                (convert-ante ante 'AND)
                                (convert-cons ))
                               (T (setf default-rule rule))))
                        (T nil)))
                 (T nil))
           (find-rules (cdr rule-list)))
          (T (format ftr4file "~%        ELSE")
             (setf control 0)
             (setf result-slot default)
             (convert-cons)
             (format ftr4file "~%        ENDIF"))))

(defun convert-ante (ante-list logic-var)
    (cond ((OR(eq 'AND (car ante-list))(eq 'OR (car ante-list)))
           (setf logic-var (car ante-list))
           (format ftr4file "(")
           (dotimes (I (- (length ante-list) 1))
              (setf temp (nth (+ I 1) ante-list))
              (convert-ante temp logic-var)
              (if (neq (+ I 1) (- (length ante-list) 1))
                  (format ftr4file "~%     +         .~A. " logic-var)
```

```
            (format ftr4file "")" ))))
        ((eq 'INSTANCE (car ante-list))
         (setf a-list (member 'with ante-list))
         (convert-variables a-list logic-var))
        (T nil)))

(defun convert-variables (v-list logic-var)
   (cond ((neq v-list '())
          (setf frame-key (nth 1 v-list))
          (setf slot-key (nth 2 v-list))
          (setf frame (nth 1 (member frame-key key-list)))
          (setf slot (slot-value frame slot-key))
          (format ftr4file "( ~A .EQ. ~A )" frame-key slot)
          (setf v-list (member 'with (cdr v-list)))
          (if (neq v-list '())
              (format ftr4file "~%     +          .~A. " logic-var))
          (convert-variables v-list logic-var))
        (T nil)))

(defun convert-cons ()
   (if (neq control 0)
       (format ftr4file " THEN"))
   (setf slot (slot-value final-legend result-slot))
   (format ftr4file "~%          ~A = ~A " result slot))
;
; Builds the fortran code which writes the output value.
(defun finish-output-code ()
   (format ftr4file "~%          ~A = CHAR(~A)" input-var result)
   (setf label (+ label 10))
   (setf err-label (+ err-label 5))
   (format ftr4file "~%       WRITE(~A,~A,REC=I,ERR=~A,IOSTAT=IOE) ~A"
    result-unit label err-label input-var)
   (format ftr4file "~% ~A   FORMAT(A1)" label)
   (format ftr5file "~% ~A WRITE(*,~A) IOE " err-label (+ err-label 2))
   (format ftr5file "~% ~A FORMAT(' ERROR WRITING ~A FILE, ERROR = ',I4)"
    (+ err-label 2) result)
   (format ftr5file "~%     GOTO 1000")
   (setf label (+ label 10))
   (format ftr3file "~%     DO ~A I = 129,NUMBER" label)
   (format ftr4file "~% ~A CONTINUE" label)
   (format ftr4file "~%     CLOSE(~A)" result-unit)
   (setf label (+ label 10))
   (format ftr4file "~%     WRITE(*,~A) " label)
   (format ftr4file "~% ~A FORMAT(' FINISHED CREATING RESULT.')" label))
;
; Builds the fortran code for the development of a final-legend and
;  trailer file.
(defun build-legend-code ()
   (format datafile "~%      CHARACTER*72      LINE1")
   (format datafile "~%      CHARACTER*32      LINE2")
   (setf count 0)
   (setf slot-list output-values)
   (setf final-values (mapcar #'(lambda (x) (slot-value final-legend x)) slot-list))
   (format ftr4file "~%C")
   (format ftr4file "~%C Opens and starts the Trailer file.")
   (build-trailer)
   (format ftr4file "~%C")
   (format ftr4file "~%C Puts the values in the trailer file.")
   (format ftr4file "~%      IF (TYPE .EQ. 'GENERAL ') THEN")
   (setf leg-cons (rule-consequent 'GENERAL-LEGEND))
   (build-legend-list (cdr leg-cons))
   (format ftr4file "~%      ELSEIF (TYPE .EQ. 'LANDFILL') THEN")
   (setf leg-cons (rule-consequent 'LANDFILL-LEGEND))
   (setf count 0)
   (build-legend-list (cdr leg-cons))
```

```
    (format ftr4file "°%       ELSE")
    (format ftr4file "°%         CLOSE(°A)" logic-unit)
    (setf label (+ label 10))
    (format ftr4file "°%         WRITE(*,°A) " label)
    (format ftr4file "°%  °A   FORMAT(' ERROR -- MAP TYPE INCORRECT,
     MUST BE GENERAL|LANDFILL',/," label)
    (format ftr4file "°%     +        ' TRAILER FILE COULD NOT BE CREATED.')")
    (format ftr4file "°%         GOTO 1000")
    (format ftr4file "°%       ENDIF")
    (setf label (+ label 10))
    (format ftr4file "°%       WRITE(*,°A) " label)
    (format ftr4file "°%  °A FORMAT(' FINISHED CREATING THE LEGEND FILE.')" label)
    (format ftr4file "°%       GOTO 1000"))
;
; Creates the code which opens the trailer file and writes the header
; information in it.
(defun build-trailer ()
    (setf logic-unit (+ result-unit 1))
    (setf label (+ label 10))
    (format ftr4file "°%       WRITE(*,°A) " label)
    (format ftr4file "°%  °A FORMAT(' ENTER COMPLETE PATH NAME OF THE GIS TRAILER FILE ',/,"
     label)
    (format ftr4file "°%     +        ' = = >  ')")
    (setf label (+ label 10))
    (format ftr4file "°%       READ(*,°A) FNAME" label)
    (format ftr4file "°%  °A FORMAT(A30)" label)
    (setf label (+ label 10))
    (format ftr4file "°%       WRITE(*,°A) " label)
    (format ftr4file "°%  °A FORMAT(' DOES THE TRAILER FILE ALREADY EXIST (Y|N) = = >  ')"
     label)
    (setf label (+ label 10))
    (format ftr4file "°%       READ(*,°A) ANS " label)
    (format ftr4file "°%  °A FORMAT(A1)" label)
    (format ftr4file "°%C")
    (format ftr4file "°%C The trailer file is opened.")
    (format ftr4file "°%       IF(ANS .EQ. 'Y') THEN ")
    (setf err-label (+ err-label 5))
    (setf open-err err-label)
    (format ftr4file "°%         OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
     logic-unit open-err)
    (format ftr4file "°%     +        ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr5file "°%  °A WRITE(*,°A) IOE" open-err (+ open-err 2))
    (format ftr5file "°%  °A FORMAT(' ERROR OPENING GIS TRAILER FILE',/, " (+ open-err 2))
    (format ftr5file "°%     +        ' ERROR  =  ',I4) " )
    (format ftr5file "°%       GOTO 1000 ")
    (format ftr4file "°%       ELSE")
    (format ftr4file "°%         OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='NEW',"
     logic-unit open-err)
    (format ftr4file "°%     +        ACCESS='DIRECT',FORM='FORMATTED',RECL=1)")
    (format ftr4file "°%       ENDIF")
    (setf label (+ label 10))
    (setf err-label (+ err-label 5))
    (format ftr4file "°%C")
    (format ftr4file "°%C  Blank out the file first.")
    (format ftr4file "°%       DO °A I = 1,2048" label)
    (format ftr4file "°%         WRITE(°A,°A,REC=I,ERR=°A,IOSTAT=IOE)"
     logic-unit (+ label 5) err-label)
    (format ftr4file "°%  °A   FORMAT('0')" (+ label 5))
    (format ftr5file "°%  °A WRITE(*,°A) IOE" err-label (+ err-label 2))
    (format ftr5file "°%  °A FORMAT(' ERROR WRITING GIS TRAILER FILE',/, " (+ err-label 2))
    (format ftr5file "°%     +        ' ERROR  =  ',I4) " )
    (format ftr5file "°%       GOTO 1000 ")
    (format ftr4file "°%  °A CONTINUE" label)
    (format ftr4file "°%       CLOSE(°A)" logic-unit)
    (format ftr4file "°%       OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
     logic-unit open-err)
    (format ftr4file "°%      +     ACCESS='DIRECT',FORM='UNFORMATTED',RECL=72)")
    (format ftr4file "°%      LINE1  =  'TRAILER'")
    (format ftr4file "°%      WRITE(°A,REC=1,ERR=°A,IOSTAT=IOE) LINE1"
     logic-unit err-label)
    (format ftr4file "°%      LINE1  =  'GEOLOGICAL ENGINEERING MAP°°'")
    (format ftr4file "°%      WRITE(°A,REC=2,ERR=°A,IOSTAT=IOE) LINE1"
     logic-unit err-label)
    (format ftr4file "°%      CLOSE(°A)" logic-unit)
    (format ftr4file "°%      OPEN(UNIT=°A,FILE=FNAME,ERR=°A,IOSTAT=IOE,STATUS='OLD',"
     logic-unit open-err)
    (format ftr4file "°%      +     ACCESS='DIRECT',FORM='UNFORMATTED',RECL=32)"))

;
; Associates the numerical result values with the correct text.
(defun build-legend-list (cons)
   (setf rec-num 64)
   (cond ((eq (nth 1 (car cons)) 'FINAL-LEGEND)
          (dotimes (I 256)
             (cond ((member I final-values)
                    (setf indx (- (length final-values) (length (member I final-values))))
                    (setf final-text (nth 1 (member (nth indx slot-list) (car cons))))
                    (setf rec-num (+ rec-num 1))
                    (format ftr4file "°%        LINE2  =  '°A°°'" final-text)
                    (format ftr4file "°%        WRITE(°A,REC=°A,ERR=°A,IOSTAT=IOE) LINE2"
                     logic-unit rec-num err-label))
                   (T nil)))
          (dotimes (I (- 4 (mod (length output-values) 4)))
             (setf rec-num (+ rec-num 1))
             (format ftr4file "°%        WRITE(°A,REC=°A,ERR=°A,IOSTAT=IOE)"
              logic-unit rec-num err-label)
             (format ftr4file "°%      +
              '°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°'")))
         (T (build-legend-list (cdr cons)))))

;
; Ends the fortran code.
(defun build-end-code ()
   (format ftr5file "°% 1000 CONTINUE")
   (format ftr5file "°%      END")
   (format ftr5file "°%")
   (close datafile)
   (close ftr1file)
   (close ftr2file)
   (close ftr3file)
   (close ftr4file)
   (close ftr5file))
```

APPENDIX C

FORTRAN CODE GENERATED BY THE CONVERSION EXPERT SYSTEM

```
      PROGRAM FTRGIS
C
      CHARACTER*1       ANS
      REAL*4            COLS
      LOGICAL*4         COMP
      CHARACTER*30      FNAME
      CHARACTER*30      FNAME2
      CHARACTER*128     HEADER
      INTEGER*4         I
      INTEGER*4         IOE
      INTEGER*4         J
      INTEGER*4         NUMBER
      REAL*4            ROWS
      CHARACTER*8       TYPE
      CHARACTER*1       PIXEL
      INTEGER*2         GEM
      INTEGER*4         SOILS
      INTEGER*2         KARST
      INTEGER*2         ARRAY1(12)
      INTEGER*2         SLOPE
      INTEGER*2         ARRAY2(12)
      INTEGER*2         FLOODING
      INTEGER*2         ARRAY3(12)
      INTEGER*2         PLASTICITY
      INTEGER*2         ARRAY4(12)
      INTEGER*2         SLS(101,4)
      CHARACTER*5       CHOICE
      CHARACTER*72      LINE1
      CHARACTER*32      LINE2
C
C  Asks the user which type of map they wish to produce.
C
      WRITE(*,100)
  100 FORMAT(' ENTER THE TYPE OF MAP TO BE PRODUCED',/,
     +        ' (GENERAL|LANDFILL) = = > ')
      READ(*,110) TYPE
  110 FORMAT(A8)
C
C Asks the user if they have a SOILS file.
      WRITE(*,280)
  280 FORMAT(' DO YOU HAVE A SOILS FILE (Y|N) = = > ')
      READ(*,290) ANS
  290 FORMAT(A1)
      COMP = .FALSE.
C
C If they do have a SOILS file then its definition file
C  is accessed.
      IF(ANS .EQ. 'Y') THEN
        COMP = .TRUE.
C
C Initializes the SOILS overlay arrays.
C
        DO 300 I = 1,6
          ARRAY4(I) = I
  300   CONTINUE
C
        DO 310 I = 1,5
          ARRAY3(I) = I
  310   CONTINUE
C
        DO 320 I = 1,4
          ARRAY2(I) = I
```

```
 320   CONTINUE
C
      DO 330 I = 1,3
        ARRAY1(I) = I
 330   CONTINUE
C
C Asks the user to enter the SOILS definition file name
C  and whether or not the file already exists.
      WRITE(*,340)
 340   FORMAT(' ENTER THE COMPLETE PATH NAME OF',
    +        ' THE SOILS DEFINITION FILE ',/,
    +        ' = = > ')
      READ(*,350) FNAME
 350   FORMAT(A30)
      WRITE(*,360)
 360   FORMAT(' DOES THE DEFINITION FILE ALREADY EXIST (Y|N) = = > ')
      READ(*,370) ANS
 370   FORMAT(A1)
C
C If the definition file already exists then it is
C  opened and read.
      IF(ANS .EQ. 'Y') THEN
        OPEN(UNIT=4,FILE=FNAME,ERR=860,IOSTAT=IOE,STATUS='OLD',
    +        ACCESS='SEQUENTIAL',FORM='FORMATTED')
        READ(4,380,ERR=865,IOSTAT=IOE) ((SLS(I,J),I=1,101),J=1,4)
 380     FORMAT(404(I1))
C
C Since the definition file does not already exist
C  it is created and saved.
      ELSE
        OPEN(UNIT=4,FILE=FNAME,ERR=860,IOSTAT=IOE,STATUS='NEW',
    +        ACCESS='SEQUENTIAL',FORM='FORMATTED')
C
C Initializes the composite array.
        DO 390 J = 1,4
          DO 389 I = 1,101
            SLS(I,J) = 0
 389       CONTINUE
 390     CONTINUE
C
C Builds the composite definition file.
        DO 400 I = 1,101
C
C  Asks the user to input a valid SOILS value.
          WRITE(*,410)
 410       FORMAT(' ENTER THE INTEGER SOILS VALUE -- ',
    +            ' ENTER 999 TO END -- (FORMAT = = >XXX )',/,
    +            ' = = >')
          READ(*,420) SOILS
 420       FORMAT(I3)
          IF (SOILS .EQ. 999) GOTO 430
C
C Asks the user to enter the KARST overlay value
C which is associated with the input SOILS value.
 119       WRITE(*,120)
 120       FORMAT(' ENTER THE ASSOCIATED KARST VALUE ',/,
    +            ' CHOICES: (YES NO WATER) ',/,
    +            ' = = >')
          READ(*,130) CHOICE
 130       FORMAT(A5)
          IF (CHOICE .EQ. 'YES') THEN
            SLS(SOILS,1) = 1
          ELSEIF (CHOICE .EQ. 'NO') THEN
            SLS(SOILS,1) = 2
          ELSEIF (CHOICE .EQ. 'WATER') THEN
          SLS(SOILS,1) = 3
         ELSEIF (CHOICE .EQ. ' ') THEN
          SLS(SOILS,1) = 0
         ELSE
          GOTO 119
         ENDIF
C
C Asks the user to enter the SLOPE overlay value
C which is associated with the input SOILS value.
 159     WRITE(*,160)
 160     FORMAT(' ENTER THE ASSOCIATED SLOPE VALUE ',/,
   +           ' CHOICES: (<5% 5-30% >30% WATER) ',/,
   +           ' ==>')
        READ(*,170) CHOICE
 170     FORMAT(A5)
        IF (CHOICE .EQ. '<5%') THEN
         SLS(SOILS,2) = 1
        ELSEIF (CHOICE .EQ. '5-30%') THEN
         SLS(SOILS,2) = 2
        ELSEIF (CHOICE .EQ. '>30%') THEN
         SLS(SOILS,2) = 3
        ELSEIF (CHOICE .EQ. 'WATER') THEN
         SLS(SOILS,2) = 4
        ELSEIF (CHOICE .EQ. ' ') THEN
         SLS(SOILS,2) = 0
        ELSE
         GOTO 159
        ENDIF
C
C Asks the user to enter the FLOODING overlay value
C which is associated with the input SOILS value.
 199     WRITE(*,200)
 200     FORMAT(' ENTER THE ASSOCIATED FLOODING VALUE ',/,
   +           ' CHOICES: (NONE FREQ RARE OCC WATER) ',/,
   +           ' ==>')
        READ(*,210) CHOICE
 210     FORMAT(A5)
        IF (CHOICE .EQ. 'NONE') THEN
         SLS(SOILS,3) = 1
        ELSEIF (CHOICE .EQ. 'FREQ') THEN
         SLS(SOILS,3) = 2
        ELSEIF (CHOICE .EQ. 'RARE') THEN
         SLS(SOILS,3) = 3
        ELSEIF (CHOICE .EQ. 'OCC') THEN
         SLS(SOILS,3) = 4
        ELSEIF (CHOICE .EQ. 'WATER') THEN
         SLS(SOILS,3) = 5
        ELSEIF (CHOICE .EQ. ' ') THEN
         SLS(SOILS,3) = 0
        ELSE
         GOTO 199
        ENDIF
C
C Asks the user to enter the PLASTICITY overlay value
C which is associated with the input SOILS value.
 239     WRITE(*,240)
 240     FORMAT(' ENTER THE ASSOCIATED PLASTICITY VALUE ',/,
   +           ' CHOICES: (<10 10-20 20-30 30-40 >40 WATER) ',/,
   +           ' ==>')
        READ(*,250) CHOICE
 250     FORMAT(A5)
        IF (CHOICE .EQ. '<10') THEN
         SLS(SOILS,4) = 1
        ELSEIF (CHOICE .EQ. '10-20') THEN
         SLS(SOILS,4) = 2
```

```
          ELSEIF (CHOICE .EQ. '20-30') THEN
           SLS(SOILS,4) = 3
          ELSEIF (CHOICE .EQ. '30-40') THEN
           SLS(SOILS,4) = 4
          ELSEIF (CHOICE .EQ. '>40') THEN
           SLS(SOILS,4) = 5
          ELSEIF (CHOICE .EQ. 'WATER') THEN
           SLS(SOILS,4) = 6
          ELSEIF (CHOICE .EQ. ' ') THEN
           SLS(SOILS,4) = 0
          ELSE
           GOTO 239
          ENDIF
 400      CONTINUE
 430      CONTINUE
C
C Saves the definition file.
         WRITE(4,440,ERR=870,IOSTAT=IOE) ((SLS(I,J),I=1,101),J=1,4)
 440      FORMAT(404(I1))
         ENDFILE(4)
        ENDIF
        CLOSE(4)
C
C Since a composite file is not being used the user is asked
C if they have the necessary non-composite files.
       ELSE
        WRITE(*,450)
 450    FORMAT(' DO YOU HAVE (PLASTICITY FLOODING SLOPE KARST) FILES ',
   +           ' (Y|N) = = > ')
        READ(*,460) ANS
 460    FORMAT(A1)
C
C Since there are non-composite files the definition file
C  name is asked for and whether or not the file already exists.
        IF(ANS .EQ. 'Y') THEN
         WRITE(*,470)
 470     FORMAT(' ENTER COMPLETE PATH NAME OF THE DEFINITION FILE ',/,
   +           ' = = > ')
         READ(*,480) FNAME
 480     FORMAT(A30)
         WRITE(*,490)
 490     FORMAT(' DOES THE DEFINITION FILE ALREADY EXIST (Y|N) = = > ')
         READ(*,500) ANS
 500     FORMAT(A1)
C
C The definition file is opened and read.
         IF(ANS .EQ. 'Y') THEN
          OPEN(UNIT=4,FILE=FNAME,ERR=875,IOSTAT=IOE,STATUS='OLD',
   +          ACCESS='SEQUENTIAL',FORM='FORMATTED')
          READ(4,510,ERR=880,IOSTAT=IOE) (ARRAY4(I),I=1,6)
 510       FORMAT(I3,I3,I3,I3,I3,I3)
          READ(4,520,ERR=880,IOSTAT=IOE) (ARRAY3(I),I=1,5)
 520       FORMAT(I3,I3,I3,I3,I3)
          READ(4,530,ERR=880,IOSTAT=IOE) (ARRAY2(I),I=1,4)
 530       FORMAT(I3,I3,I3,I3)
          READ(4,540,ERR=880,IOSTAT=IOE) (ARRAY1(I),I=1,3)
 540       FORMAT(I3,I3,I3)
C
C The definition file is created and saved.
         ELSE
          OPEN(UNIT=4,FILE=FNAME,ERR=875,IOSTAT=IOE,STATUS='NEW',
   +          ACCESS='SEQUENTIAL',FORM='FORMATTED')
C
C Asks the user to set the integer values associated
C with the given KARST choices.
```

```
        WRITE(*,140)
 140     FORMAT(' ENTER THE INTEGER VALUES WHICH REPRESENT',/,
   +          ' THE KARST CHOICES: ',/,
   +          ' (YES NO WATER) ',/,
   +          ' (USE FORMAT = = = >XXX XXX XXX)',/,
   +          ' = = > ')
        READ(*,150) (ARRAY1(I),I = 1,3)
 150     FORMAT(I3,1X,I3,1X,I3)
C
C Asks the user to set the integer values associated
C with the given SLOPE choices.
        WRITE(*,180)
 180     FORMAT(' ENTER THE INTEGER VALUES WHICH REPRESENT',/,
   +          ' THE SLOPE CHOICES: ',/,
   +          ' (< 5% 5-30% > 30% WATER) ',/,
   +          ' (USE FORMAT = = = >XXX XXX XXX)',/,
   +          ' = = > ')
        READ(*,190) (ARRAY2(I),I = 1,4)
 190     FORMAT(I3,1X,I3,1X,I3,1X,I3)
C
C Asks the user to set the integer values associated
C with the given FLOODING choices.
        WRITE(*,220)
 220     FORMAT(' ENTER THE INTEGER VALUES WHICH REPRESENT',/,
   +          ' THE FLOODING CHOICES: ',/,
   +          ' (NONE FREQ RARE OCC WATER) ',/,
   +          ' (USE FORMAT = = = >XXX XXX XXX)',/,
   +          ' = = > ')
        READ(*,230) (ARRAY3(I),I = 1,5)
 230     FORMAT(I3,1X,I3,1X,I3,1X,I3,1X,I3)
C
C Asks the user to set the integer values associated
C with the given PLASTICITY choices.
        WRITE(*,260)
 260     FORMAT(' ENTER THE INTEGER VALUES WHICH REPRESENT',/,
   +          ' THE PLASTICITY CHOICES: ',/,
   +          ' (< 10 10-20 20-30 30-40 > 40 WATER) ',/,
   +          ' (USE FORMAT = = = >XXX XXX XXX)',/,
   +          ' = = > ')
        READ(*,270) (ARRAY4(I),I = 1,6)
 270     FORMAT(I3,1X,I3,1X,I3,1X,I3,1X,I3,1X,I3)
        WRITE(4,510,ERR = 885,IOSTAT = IOE) (ARRAY4(I),I = 1,6)
        WRITE(4,520,ERR = 885,IOSTAT = IOE) (ARRAY3(I),I = 1,5)
        WRITE(4,530,ERR = 885,IOSTAT = IOE) (ARRAY2(I),I = 1,4)
        WRITE(4,540,ERR = 885,IOSTAT = IOE) (ARRAY1(I),I = 1,3)
        ENDFILE(4)
       ENDIF
       CLOSE(4)
C
C Since neither the composite file or all of the
C  non-composite files are available an error message
C  is output and processing is stopped.
      ELSE
       WRITE(*,570)
 570   FORMAT(' CANNOT PRODUCE A GEM FILE ',/,
   +          ' -- CRITICAL DATA FILES ARE NOT AVAILABLE.')
       GOTO 1000
      ENDIF
     ENDIF
C
C Opens the input and output files.
     IF (COMP) THEN
      WRITE(*,580)
 580  FORMAT(' ENTER THE SOILS FILE NAME = = > ')
      READ(*,590) FNAME
 590  FORMAT(A30)
        OPEN(UNIT=3,FILE=FNAME,ERR=900,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=4)
C
C Reads the number of cols and rows of input that
C  need to be processed.
        READ(3,REC=5,ERR=905,IOSTAT=IOE) COLS
        READ(3,REC=6,ERR=905,IOSTAT=IOE) ROWS
        CLOSE(3)
        OPEN(UNIT=3,FILE=FNAME,ERR=900,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
        WRITE(*,600)
 600  FORMAT(' ENTER THE GEM FILE NAME ==> ')
        READ(*,610) FNAME2
 610  FORMAT(A30)
      WRITE(*,620)
 620 FORMAT(' DOES THE GEM FILE ALREADY EXIST (Y|N) ==> ')
      READ(*,630) ANS
 630 FORMAT(A1)
C
C The GEM file is opened.
      IF(ANS .EQ. 'Y') THEN
        OPEN(UNIT=11,FILE=FNAME2,ERR=910,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
      ELSE
        OPEN(UNIT=11,FILE=FNAME2,ERR=910,IOSTAT=IOE,STATUS='NEW',
     +       ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
      ENDIF
C
C Reads the header information from the SOILS file
C  and writes it to the output file.
        READ(3,REC=1,ERR=905,IOSTAT=IOE) HEADER
        WRITE(11,REC=1,ERR=915,IOSTAT=IOE) HEADER
        CLOSE(3)
        CLOSE(11)
        OPEN(UNIT=3,FILE=FNAME,ERR=900,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
        OPEN(UNIT=11,FILE=FNAME2,ERR=910,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
C
C Calulates the number of bytes of input data.
        NUMBER = INT(COLS * ROWS) + 128
C
C Opens the input and output files.
      ELSE
        WRITE(*,640)
 640  FORMAT(' ENTER THE PLASTICITY FILE NAME ==> ')
        READ(*,650) FNAME
 650  FORMAT(A30)
        OPEN(UNIT=7,FILE=FNAME,ERR=920,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
        WRITE(*,660)
 660  FORMAT(' ENTER THE FLOODING FILE NAME ==> ')
        READ(*,670) FNAME
 670  FORMAT(A30)
        OPEN(UNIT=8,FILE=FNAME,ERR=925,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
        WRITE(*,680)
 680  FORMAT(' ENTER THE SLOPE FILE NAME ==> ')
        READ(*,690) FNAME
 690  FORMAT(A30)
        OPEN(UNIT=9,FILE=FNAME,ERR=930,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
        WRITE(*,700)
 700  FORMAT(' ENTER THE KARST FILE NAME ==> ')
```

```
      READ(*,710) FNAME
  710  FORMAT(A30)
      OPEN(UNIT=10,FILE=FNAME,ERR=935,IOSTAT=IOE,STATUS='OLD',
    +      ACCESS='DIRECT',FORM='UNFORMATTED',RECL=4)
C
C Reads the number of cols and rows of input that
C  need to be processed.
      READ(10,REC=5,ERR=940,IOSTAT=IOE) COLS
      READ(10,REC=6,ERR=940,IOSTAT=IOE) ROWS
      CLOSE(10)
      OPEN(UNIT=10,FILE=FNAME,ERR=935,IOSTAT=IOE,STATUS='OLD',
    +      ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
      WRITE(*,720)
  720  FORMAT(' ENTER THE GEM FILE NAME ==> ')
      READ(*,730) FNAME2
  730  FORMAT(A30)
     WRITE(*,740)
  740 FORMAT(' DOES THE GEM FILE ALREADY EXIST (Y|N) ==> ')
     READ(*,750) ANS
  750 FORMAT(A1)
C
C The GEM file is opened.
     IF(ANS .EQ. 'Y') THEN
      OPEN(UNIT=11,FILE=FNAME2,ERR=945,IOSTAT=IOE,STATUS='OLD',
    +      ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
     ELSE
      OPEN(UNIT=11,FILE=FNAME2,ERR=945,IOSTAT=IOE,STATUS='NEW',
    +      ACCESS='DIRECT',FORM='UNFORMATTED',RECL=128)
     ENDIF
C
C Reads the header information from the KARST file
C  and writes it to the output file.
      READ(10,REC=1,ERR=940,IOSTAT=IOE) HEADER
      WRITE(11,REC=1,ERR=950,IOSTAT=IOE) HEADER
      CLOSE(10)
      CLOSE(11)
      OPEN(UNIT=10,FILE=FNAME,ERR=935,IOSTAT=IOE,STATUS='OLD',
    +      ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
      OPEN(UNIT=11,FILE=FNAME2,ERR=945,IOSTAT=IOE,STATUS='OLD',
    +      ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
C
C Calulates the number of bytes of input data.
      NUMBER = INT(COLS * ROWS) + 128
     ENDIF
     DO 770 I = 129,NUMBER
C
C Reads the input from the SOILS file and converts
C  it into seperate overlay values.
      IF (COMP) THEN
       READ(3,550,REC=1,ERR=890,IOSTAT=IOE) PIXEL
  550   FORMAT(A1)
       SOILS = ICHAR(PIXEL)
       KARST = SLS(SOILS,1)
       SLOPE = SLS(SOILS,2)
       FLOODING = SLS(SOILS,3)
       PLASTICITY = SLS(SOILS,4)
C
C Reads the input from the non-composite files.
      ELSE
       READ(7,560,REC=1,ERR=895,IOSTAT=IOE) PIXEL
       PLASTICITY = ICHAR(PIXEL)
       READ(8,560,REC=1,ERR=895,IOSTAT=IOE) PIXEL
       FLOODING = ICHAR(PIXEL)
       READ(9,560,REC=1,ERR=895,IOSTAT=IOE) PIXEL
       SLOPE = ICHAR(PIXEL)
      READ(10,560,REC=I,ERR=895,IOSTAT=IOE) PIXEL
      KARST = ICHAR(PIXEL)
560     FORMAT(A1)
    ENDIF
    IF( PLASTICITY .EQ. ARRAY4(5) ) THEN
      GEM = 13
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(1) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(1) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 12
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(2) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(2) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 11
    ELSEIF( SLOPE .EQ. ARRAY2(3) ) THEN
      GEM = 10
    ELSEIF( KARST .EQ. ARRAY1(1) ) THEN
      GEM = 9
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(2) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(4) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 8
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(2) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(1) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 7
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(2) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(3) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 6
    ELSEIF((( FLOODING .EQ. ARRAY3(2) )
  +         .OR. ( FLOODING .EQ. ARRAY3(4) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(1) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(3) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 5
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(1) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(3) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 4
    ELSEIF((( FLOODING .EQ. ARRAY3(1) )
  +         .OR. ( FLOODING .EQ. ARRAY3(3) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(1) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(2) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 3
    ELSEIF((( FLOODING .EQ. ARRAY3(2) )
  +         .OR. ( FLOODING .EQ. ARRAY3(4) ))
  +         .AND. ( SLOPE .EQ. ARRAY2(1) )
  +         .AND. ( PLASTICITY .EQ. ARRAY4(1) )
  +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
      GEM = 2
    ELSEIF((( FLOODING .EQ. ARRAY3(2) )
     +         .OR. ( FLOODING .EQ. ARRAY3(4) ))
     +         .AND. ( SLOPE .EQ. ARRAY2(1) )
     +         .AND. ( PLASTICITY .EQ. ARRAY4(2) )
     +         .AND. ( KARST .EQ. ARRAY1(2) )) THEN
          GEM = 1
        ELSE
          GEM = 0
        ENDIF
        PIXEL = CHAR(GEM)
        WRITE(11,760,REC=I,ERR=955,IOSTAT=IOE) PIXEL
  760   FORMAT(A1)
  770 CONTINUE
      CLOSE(11)
      WRITE(*,780)
  780 FORMAT(' FINISHED CREATING RESULT.')
C
C Opens and starts the Trailer file.
      WRITE(*,790)
  790 FORMAT(' ENTER COMPLETE PATH NAME OF THE GIS TRAILER FILE ',
     +       ' = = > ')
      READ(*,800) FNAME
  800 FORMAT(A30)
      WRITE(*,810)
  810 FORMAT(' DOES THE TRAILER FILE ALREADY EXIST (Y|N) = = > ')
      READ(*,820) ANS
  820 FORMAT(A1)
C
C The trailer file is opened.
      IF(ANS .EQ. 'Y') THEN
        OPEN(UNIT=12,FILE=FNAME,ERR=960,IOSTAT=IOE,STATUS='OLD',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
      ELSE
        OPEN(UNIT=12,FILE=FNAME,ERR=960,IOSTAT=IOE,STATUS='NEW',
     +       ACCESS='DIRECT',FORM='FORMATTED',RECL=1)
      ENDIF
C
C  Blank out the file first.
      DO 830 I = 1,2048
        WRITE(12,835,REC=I,ERR=965,IOSTAT=IOE)
  835   FORMAT('0')
  830 CONTINUE
      CLOSE(12)
      OPEN(UNIT=12,FILE=FNAME,ERR=960,IOSTAT=IOE,STATUS='OLD',
     +     ACCESS='DIRECT',FORM='UNFORMATTED',RECL=72)
      LINE1 = 'TRAILER'
      WRITE(12,REC=1,ERR=965,IOSTAT=IOE) LINE1
      LINE1 = 'GEOLOGICAL ENGINEERING MAP°'
      WRITE(12,REC=2,ERR=965,IOSTAT=IOE) LINE1
      CLOSE(12)
      OPEN(UNIT=12,FILE=FNAME,ERR=960,IOSTAT=IOE,STATUS='OLD',
     +     ACCESS='DIRECT',FORM='UNFORMATTED',RECL=32)
C
C Puts the values in the trailer file.
      IF (TYPE .EQ. 'GENERAL ') THEN
        LINE2 = 'Background°'
        WRITE(12,REC=65,ERR=965,IOSTAT=IOE) LINE2
        LINE2 = 'CLASS IA°'
        WRITE(12,REC=66,ERR=965,IOSTAT=IOE) LINE2
        LINE2 = 'CLASS IB°'
        WRITE(12,REC=67,ERR=965,IOSTAT=IOE) LINE2
        LINE2 = 'CLASS ID°'
        WRITE(12,REC=68,ERR=965,IOSTAT=IOE) LINE2
        LINE2 = 'CLASS IE°'
        WRITE(12,REC=69,ERR=965,IOSTAT=IOE) LINE2
        LINE2 = 'CLASS IF°'
      WRITE(12,REC=70,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IG°'
      WRITE(12,REC=71,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IIA°'
      WRITE(12,REC=72,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IIB°'
      WRITE(12,REC=73,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IIC°'
      WRITE(12,REC=74,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IID°'
      WRITE(12,REC=75,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IIE°'
      WRITE(12,REC=76,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS IIF°'
      WRITE(12,REC=77,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'CLASS XC°'
      WRITE(12,REC=78,ERR=965,IOSTAT=IOE) LINE2
      WRITE(12,REC=79,ERR=965,IOSTAT=IOE)
     +  '°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°'
      WRITE(12,REC=80,ERR=965,IOSTAT=IOE)
     +  '°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°'
     ELSEIF (TYPE .EQ. 'LANDFILL') THEN
      LINE2 = 'Background°'
      WRITE(12,REC=65,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'NO WAY°'
      WRITE(12,REC=66,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'VERY POOR°'
      WRITE(12,REC=67,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'POOR°'
      WRITE(12,REC=68,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=69,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=70,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=71,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=72,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'VERY GOOD°'
      WRITE(12,REC=73,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'NO WAY°'
      WRITE(12,REC=74,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=75,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=76,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'GOOD°'
      WRITE(12,REC=77,ERR=965,IOSTAT=IOE) LINE2
      LINE2 = 'POOR°'
      WRITE(12,REC=78,ERR=965,IOSTAT=IOE) LINE2
      WRITE(12,REC=79,ERR=965,IOSTAT=IOE)
     +  '°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°'
      WRITE(12,REC=80,ERR=965,IOSTAT=IOE)
     +  '°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°'
     ELSE
      CLOSE(12)
      WRITE(*,840)
840   FORMAT(' ERROR -- MAP TYPE INCORRECT,
    MUST BE GENERAL|LANDFILL',/,
    +       ' TRAILER FILE COULD NOT BE CREATED.')
      GOTO 1000
     ENDIF
     WRITE(*,850)
850 FORMAT(' FINISHED CREATING THE LEGEND FILE.')
```

```
      GOTO 1000
C
C File Error Messages.
  860 WRITE(*,862) IOE
  862 FORMAT(' ERROR OPENING SOILS ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  865 WRITE(*,867) IOE
  867 FORMAT(' ERROR READING SOILS ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  870 WRITE(*,872) IOE
  872 FORMAT(' ERROR WRITING SOILS ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  875 WRITE(*,877) IOE
  877 FORMAT(' ERROR OPENING (PLASTICITY FLOODING SLOPE KARST) ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  880 WRITE(*,882) IOE
  882 FORMAT(' ERROR READING (PLASTICITY FLOODING SLOPE KARST) ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  885 WRITE(*,887) IOE
  887 FORMAT(' ERROR WRITING (PLASTICITY FLOODING SLOPE KARST) ',/,
     +       ' DEFINITION FILE, ERROR = ',I4)
      GOTO 1000
  890 WRITE(*,892) IOE
  892 FORMAT(' ERROR READING SOILS FILE INPUT, ERROR = ',I4)
      GOTO 1000
  895 WRITE(*,897) IOE
  897 FORMAT(' ERROR READING A NON-COMPOSITE FILE, ERROR = ',I4)
      GOTO 1000
  900 WRITE(*,902) IOE
  902 FORMAT(' ERROR OPENING SOILS FILE, ERROR = ',I4)
      GOTO 1000
  905 WRITE(*,907) IOE
  907 FORMAT(' ERROR READING SOILS FILE HEADER, ERROR = ',I4)
      GOTO 1000
  910 WRITE(*,912) IOE
  912 FORMAT(' ERROR OPENING GEM FILE, ERROR = ',I4)
      GOTO 1000
  915 WRITE(*,917) IOE
  917 FORMAT(' ERROR WRITING GEM FILE HEADER, ERROR = ',I4)
      GOTO 1000
  920 WRITE(*,922) IOE
  922 FORMAT(' ERROR OPENING PLASTICITY FILE, ERROR = ',I4)
      GOTO 1000
  925 WRITE(*,927) IOE
  927 FORMAT(' ERROR OPENING FLOODING FILE, ERROR = ',I4)
      GOTO 1000
  930 WRITE(*,932) IOE
  932 FORMAT(' ERROR OPENING SLOPE FILE, ERROR = ',I4)
      GOTO 1000
  935 WRITE(*,937) IOE
  937 FORMAT(' ERROR OPENING KARST FILE, ERROR = ',I4)
      GOTO 1000
  940 WRITE(*,942) IOE
  942 FORMAT(' ERROR READING KARST FILE HEADER, ERROR = ',I4)
      GOTO 1000
  945 WRITE(*,947) IOE
  947 FORMAT(' ERROR OPENING GEM FILE, ERROR = ',I4)
      GOTO 1000
  950 WRITE(*,952) IOE
  952 FORMAT(' ERROR WRITING GEM FILE HEADER, ERROR = ',I4)
```

```
      GOTO 1000
  955 WRITE(*,957) IOE
  957 FORMAT(' ERROR WRITING GEM FILE, ERROR = ',I4)
      GOTO 1000
  960 WRITE(*,962) IOE
  962 FORMAT(' ERROR OPENING GIS TRAILER FILE',/,
     +       ' ERROR = ',I4)
      GOTO 1000
  965 WRITE(*,967) IOE
  967 FORMAT(' ERROR WRITING GIS TRAILER FILE',/,
     +       ' ERROR = ',I4)
      GOTO 1000
 1000 CONTINUE
      END
```