Hydrologic Modeling of the Williams River with tRIBS:
Development and Testing of a Novel User Interface for the TIN-Based Hydrologic Model

By

Eric Martin Lau

S.B. Biology
Massachusetts Institute of Technology, 2001

S.B. Civil and Environmental Engineering
Massachusetts Institute of Technology, 2001

Submitted to the Department of Civil and Environmental Engineering
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF ENGINEERING
in Civil and Environmental Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUNE 2002

Signature of Author _____
Department of Civil and Environmental Engineering
May 10, 2002

Certified by _____
Rafael L. Bras
Thesis Supervisor
Bacardi and Stockholm Water Foundations Professor

Accepted by _____
Oral Buyukozturk
Professor of Civil and Environmental Engineering
Chairman Departmental Committee on Graduate Studies

Hydrologic Modeling of the Williams River with tRIBS:
Development and Testing of a Novel User Interface for the TIN-Based Hydrologic Model

By

Eric Martin Lau

Submitted to the Department of Civil and Environmental Engineering on May 23, 2002
In Partial Fulfillment of the Requirements for the Degree of Master of Engineering
in Civil and Environmental Engineering

Abstract

The management and study of water resources becomes ever more important as populations rise and sources of clean, fresh water are exhausted. Improvements in technology will help increase the amount of data collected in the field, as well as improve its organization and analysis. This data can then be used to better understand the behavior of water systems through watershed simulations and hydrologic modeling.

The Williams River watershed, NSW, Australia, was visited in January 2002 to conduct an environmental field study focused on water quality and flow characteristics. The study was aided by the newly developed STEFS (Software Tools for Environmental Field Study) system. Data collected in the field was merged with data gathered prior to the field trip in preparation for hydrologic modeling of the watershed using tRIBS (TIN-based Real-Time Integrated Basin Simulator). The tRIBS software was enhanced by the creation of a GUI (Graphical User Interface) using Java. Further enhancements to be done to tRIBS, including porting to Windows and transformation to a web-based service, are considered and outlined.

Thesis Supervisor: Rafael L. Bras
Title: Bacardi and Stockholm Water Foundations Professor

Table of Contents

List of Figures

Acknowledgements

First and foremost, the author would like to thank his family for their emotional, financial, and spiritual support over the last 5 years of MIT. Without their support, the author would be lost.

Enrique Vivoni always made himself available to lend help or advice about tRIBS, hydrology and life. It is hoped that Enrique already knows how vitally important he has been and how appreciated he always will be.

Many thanks go out to Professor Gary Willgoose at the University of Newcastle, an MIT alumnus, provided us with facilities, guidance, and his expertise. His help, and the help of his associates, proved invaluable in preparation for and throughout the duration of the field study.

Thanks go out to two MIT students who aided in the attempt to port tRIBS to Windows. Omprakash D Gnawali (MIT SB 2001, MIT MEng 2002) and Joshua Baratz (MIT c/o2003) devoted significant amounts of time looking into solving the errors encountered in this process. Although the goals were not reached, their efforts are greatly appreciated.

Thanks also go out to all the students in the 2001-2002 MEng program. Their support and friendship made everyday a fun, exiting, and unpredictable test of my English grammar skills. To other MIT students who the author refers to as "Java superheros" gave programming assistance and support, the author is extremely indebted. Thanks also go out to the wealth of MIT professors and lecturers who have shaped the author's life and education. The author's education went far beyond the subjects outlined in the class syllabi.

1 Introduction

Water resource management is quickly becoming a critical area of research; populations are growing exponentially while clean, potable water supplies remain limited. The practice of monitoring and collecting data remains particularly inefficient. The data obtainable is limited by the cost of man-power to carry out field testing or monitoring programs. Streamlining these processes would allow for more complete datasets in terms of both spatial coverage and continuity over time.

STEFS (Software Tools for Environmental Field Study) is an innovative system of handheld computers connected wirelessly to enable streamlined collection and analysis of environmental variables. The system was deployed in January 2002 in the Williams River watershed, Newcastle, New South Wales, Australia. Elements of the research conducted before the trip, as well as aspects of the trip itself, are outlined. The STEFS system is also introduced and discussed briefly.

Data relevant to the watershed was collected in preparation of execution of tRIBS, the TIN-based Real-time Integrated Basin Simulator. tRIBS allows the user to construct and simulate catchment level hydrologic processes over a Triangular Irregular Network (TIN). This hydrologic model is quite new; version 1 was released in September 2001.

The C++ files that make up tRIBS were manipulated in efforts to port tRIBS from UNIX to Windows in order to expand the potential user base. Much

work was also done in developing the model from a purely command-line executable to a user-friendly model with a graphical user interface (GUI). The goal was to integrate both the software that runs the hydrologic model with tools used to analyze the inputs and outputs of the model.

2 STEFS

Software Tools for Environmental Field Study (STEFS) is an initiative to develop a system of wirelessly connected field sampling equipment. This PocketPC based integrated wireless system allows the user to collect environmental data, correlate it with a geopositional location, and enter that data through the use of a graphical user interface. That data is then sent wirelessly to a mobile field computer where it is processed and transmitted both to the field researcher for analysis and to a web server, making it available on the World Wide Web. The STEFS system is designed to innovate upon the traditional environmental data collection process, which includes writing data with paper and pencil then later transcribing the data into a computer for analysis and data sharing. It was with this STEFS system that field tests were performed and data was collected for analysis and testing of tRIBS v2.0. This chapter will give an overview of the technology involved, the field testing campaign, and relevant data acquired in the process of working on the STEFS project.

2.1 Concept

Traditional field data collection methods are cumbersome and error prone. The field researcher must write down measurements with pen and paper, then transcribe those data into a computer once back in the laboratory. Only then can the data be analyzed and visualized. Furthermore, the sharing of this data is

dependent on submission and publication of scientific journals, delaying access to the data.

The system that STEFS envisioned and realized revolutionizes this process. From step one of data recording all the way to the sharing of the results, the process is digitized and available to the world in real time.

The data is collected using a PocketPC; in the STEFS case, these were Compaq Ipaqs running the WindowsCE operating system. The STEFS software allows the user to enter relevant data regarding field testing equipment. This feature is included to allow recording of what field instruments were used, including the identification number of the specific field testing apparatus. This knowledge enables the researcher to identify origins of problems that may arise from faulty, damaged, or mis-calibrated equipment.

The environmental data itself is entered directly into the PocketPC and is loaded directly into a database on the PocketPC. When the data collection session is complete, the user sends the database stored on the PocketPC to the field laptop. Only portions of the database that are new and updated are transmitted to the field laptop to be stored; information already held in the field laptop database is not transferred. This allows for shorter transmission times and faster updating of the database.

Field teams can update their PocketPC databases with information transferred to the field laptop by other teams. This synchronization feature allows multiple teams to be deployed in the field simultaneously. And each team, while still in the field, can access the updated database with all results from all teams.

That knowledge can enable each team to survey the data collected and perform a quick analysis to understand trends in the results while still in the field. Understanding of the results could motivate the field team to alter their sampling campaign and/or change their sampling techniques.

The database compiled on the field laptop is then updated, via mobile phone, to the web server, which then makes the data available to anyone via the World Wide Web. This feature allows for data analysis to be performed in the office while the field teams are still out in the field. The office could then contact the team and have them alter their sampling or investigate interesting trends in the data, making a return trip to the same location unnecessary.

2.2 Technology

The innovative technologies that were organized in the development of STEFS are beyond the scope of this paper. For a thorough discussion of the STEFS system, including the specific technologies involved, the reader is referred to the paper presented by the STEFS Masters of Engineering student group. (ENVIROCOM Final Report, 2002).

2.3 Research and Data Gathering

Data about the study area was gathered before visiting the field. Indeed, the data gathered prior to the field trip helped determine which area would be studied. Of the many sites considered, the Williams River Watershed was

selected based on its many favorable characteristics: it is situated just 30 kilometers north of Newcastle and its wealth of amenities.

Extremely valuable were the excellent GIS coverage maps of the Williams river watershed provided by Professor Willgoose of the University of Newcastle. GIS layers of land use, soil landscape, vegetation communities, and canopy density coverages were provided, as were digital elevation models at 1:25000 and 1:100000 resolutions. Watershed boundaries and river network files were also provided. These were obtained in ArcView Export (*.e00) format; this allowed for easy transfer and manipulation of the files.

These data layers were analyzed in order to determine potential locations for the field sampling campaign. Factors such as topography, density of vegetation and proximity to stream network were taken into account when determining sampling locations.

These GIS coverages led to insight into the specific issues relevant for the Williams basin. Knowledge of these issues helped shape the targets of the field study and, in turn, the design and architecture of the database, "look and feel" of the software applications, and system integration. The ultimate benefit of these data layers is in their use in watershed modeling.

2.4 Field Sampling Campaign

2.4.1 Study Area

The Williams River Watershed is a 1200 square kilometer catchment situated just between 32° and 33° South Latitude along the western coast of

southern Australia (see Figure 1). It lies just north of Newcastle, which itself is an approximately 2 hour drive from Sydney. The significance of the watershed lies in the fact that it supplies the city of Newcastle with 70%

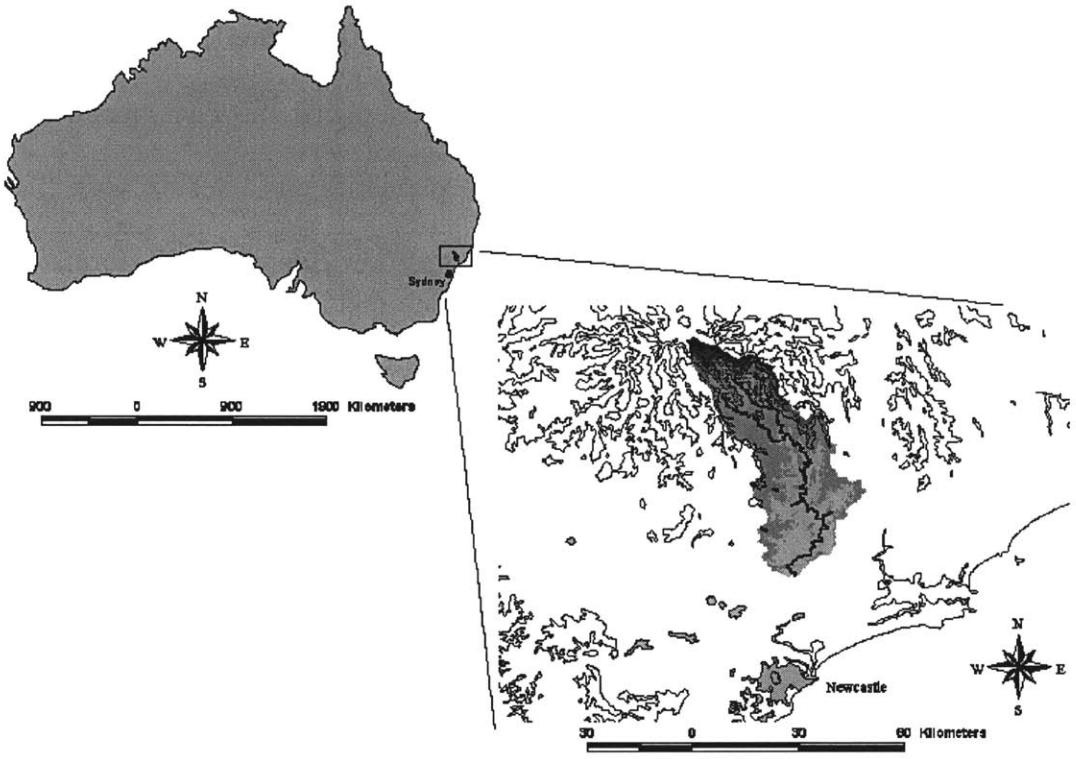

Figure 1 - The Williams River watershed, NSW, Australia. (courtesy K. Richards)

of its municipal water; thus, water quality within the watershed is vitally important. A source of contaminants is surface runoff, which increases considerably in the summer months in the Newcastle area. The increased rainfall can raise the surface water runoff to levels as high as 30,000 Mega Liters per Day (about 300 m^3/s). Large amounts of sediment, nutrients, and bacteria wash down

through the system from the surrounding land as a result of the surface runoff. This can lead to periods where the water quality parameters exceed health guidelines, especially near the middle and bottom reaches of the watershed. (http://HITS.nsw.gov.au/).

The field campaign intended to measure Nitrate, Nitrite, Ammonia, Ortho-Phosphate, and the presence of Escherichia coli and fecal coliform. Aside from the water quality parameters to be measured, it was also intended to determine flow characteristics such as channel geometry and velocity profile at a cross section.

2.4.2 Field Study

STEFS was deployed with the intent of investigating the water quality and hydrology impact due to extensive land use by the cattle grazing industry. The deployment took place in January, which is the middle of summer and the wet season for New South Wales.

Three sets of water quality sensing equipment, Hydrolabs, were distributed to the three teams on the Williams River field test. These Hydrolabs were each capable of measuring Dissolved Oxygen, pH, Temperature, Depth, and

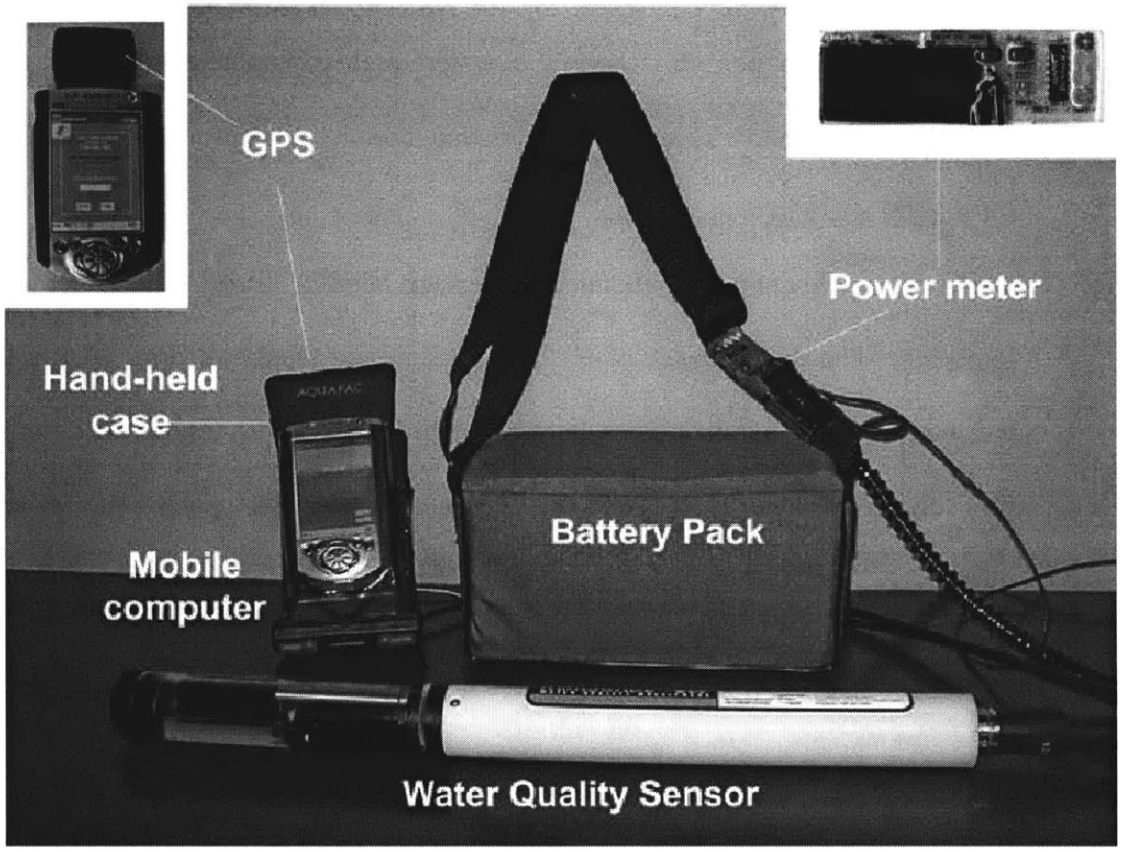

Figure 2 - STEFS. Field Data Collection System. Courtesy Vivoni, E. 2002

Turbidity. One of the Hydrolabs had the additional ability to measure chlorophyll. This Hydrolab was shared amongst the groups as the field sampling progressed. The STEFS field data collection system is shown in Figure 2.

Each team was equipped with a tape measure, to allow the determination of channel geometry. Each team also had a stream flow meter used to determine flow velocity profiles across the stream cross section.

Biological and Chemical measurements were taken at each location at various points along the cross section. The Chemical kits included materials that enabled the detection of Nitrate, Nitrite, Ammonia, and Ortho-phosphate. The

Biology kits included materials that allowed for testing of the presence and abundance of Escherichia coli as well as fecal coliform.

The Hydrolab parameters were entered into the handheld computer immediately, as were the water flow parameters and flow velocity and channel geometry. The biology samples, however, had to be incubated overnight before they were ready for analysis. At that point, the plated cells were counted and the relevant biological results were entered into the system after-the-fact.

2.5 GIS data for Williams River

The GIS data layers collected for the Williams River watershed are presented here. The collection of data layers aided the selection of sampling locations on the field trip. The layers were found to be in an AUSTRALIANNATIONAL projection. They were reprojected into decimal degrees and UTM zone 56 projections for analysis. Some of the layers, soils, land use, and digital elevation model (DEM), in particular, will be directly used in the watershed simulation using tRIBS.

The land use layer is shown in Figure 3. It shows that the majority of the watershed is categorized as "grazing or grassland", while the northern regions are comprised of timber. This was confirmed upon arrival in the study area. Much of the area is used for raising cattle and is primarily covered with grass. This land

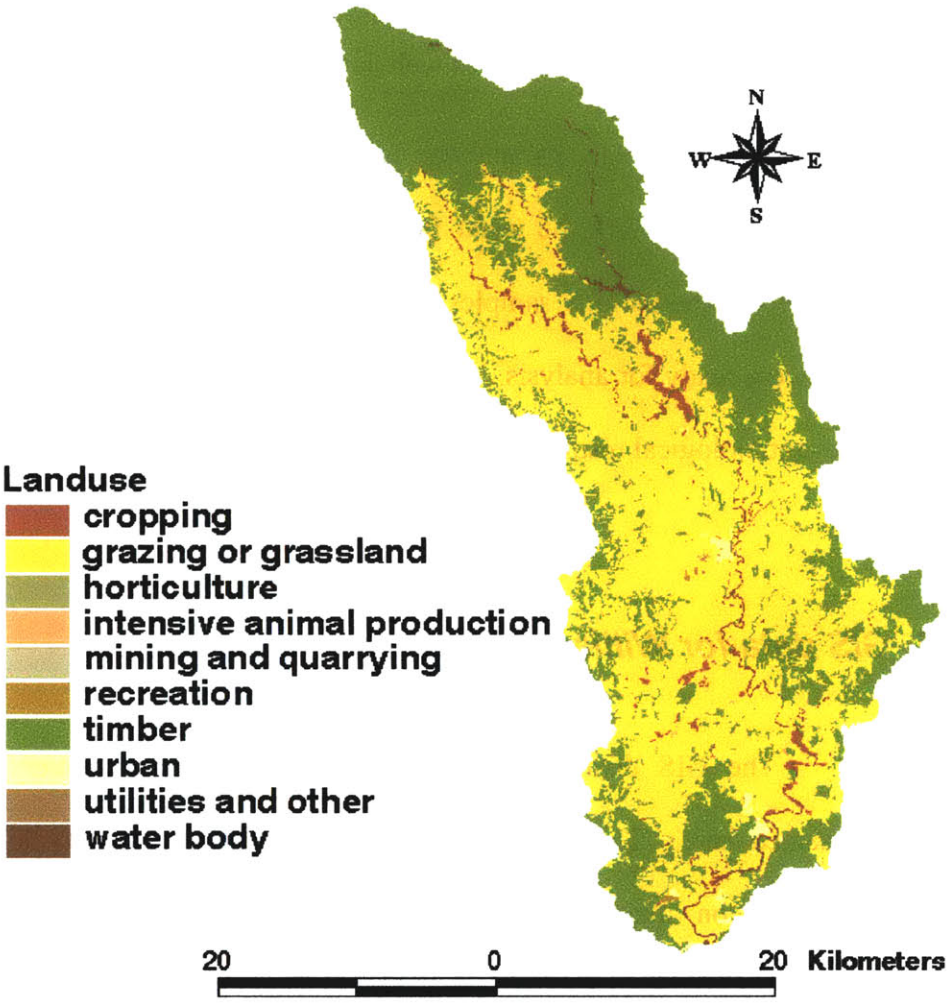

Figure 3 - GIS data layer of land use of the Williams River watershed

use GIS layer will, upon further analysis and manipulation, contribute to the parts of the tRIBS model dealing with evapotranspiration.

Figure 4 shows the stream network derived from the digital elevation model. It was more useful than the hand digitized stream network layer provided by the University of Newcastle. A stream network derived from elevation, similar to Figure 4 is what is used by tRIBS.

Figure 4 - Derived GIS layer of the Stream Network of the Williams River watershed

The GIS layer of canopy coverage (Figure 5) contains information about the density of canopy cover within the watershed area. This data does not include information about ground cover. The layer shows that the northern regions contain dense (>50%) canopy cover, while the majority of the central region of

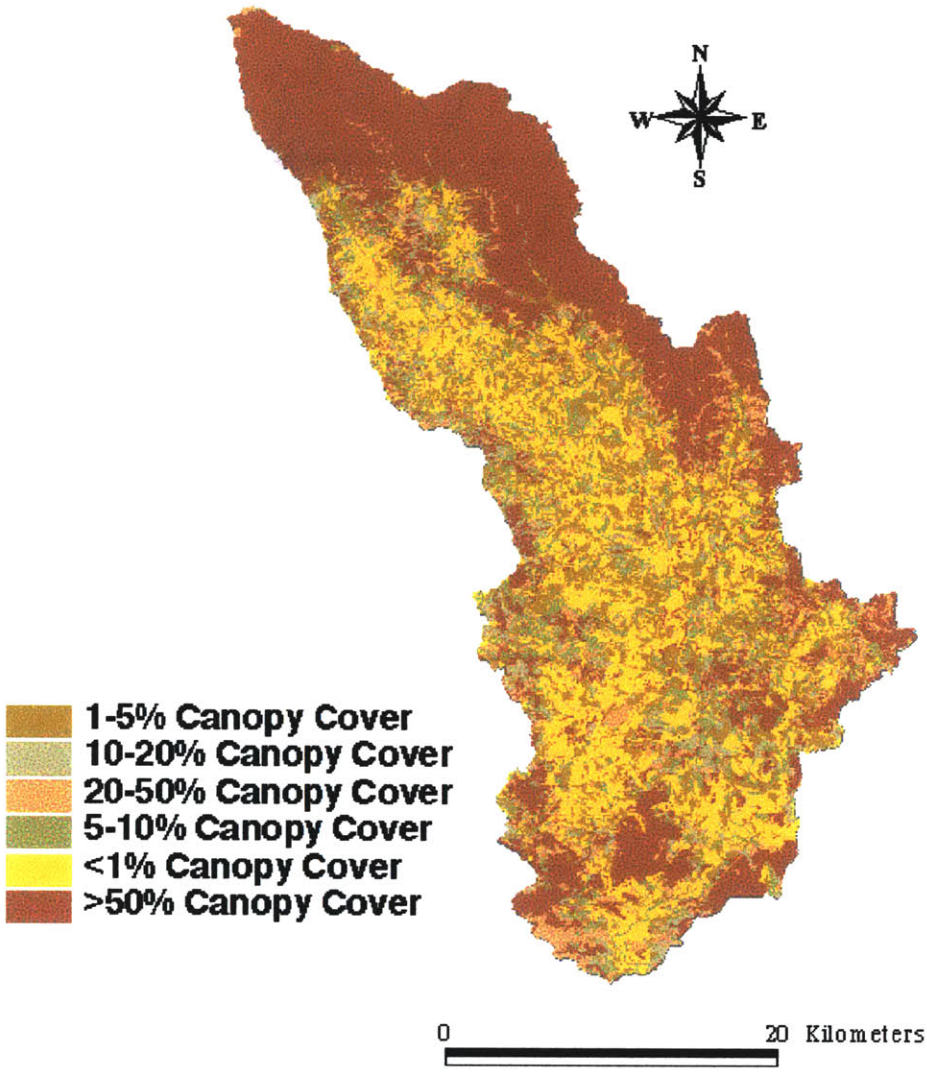

Figure 5 - Canopy Cover Density of the Williams River watershed

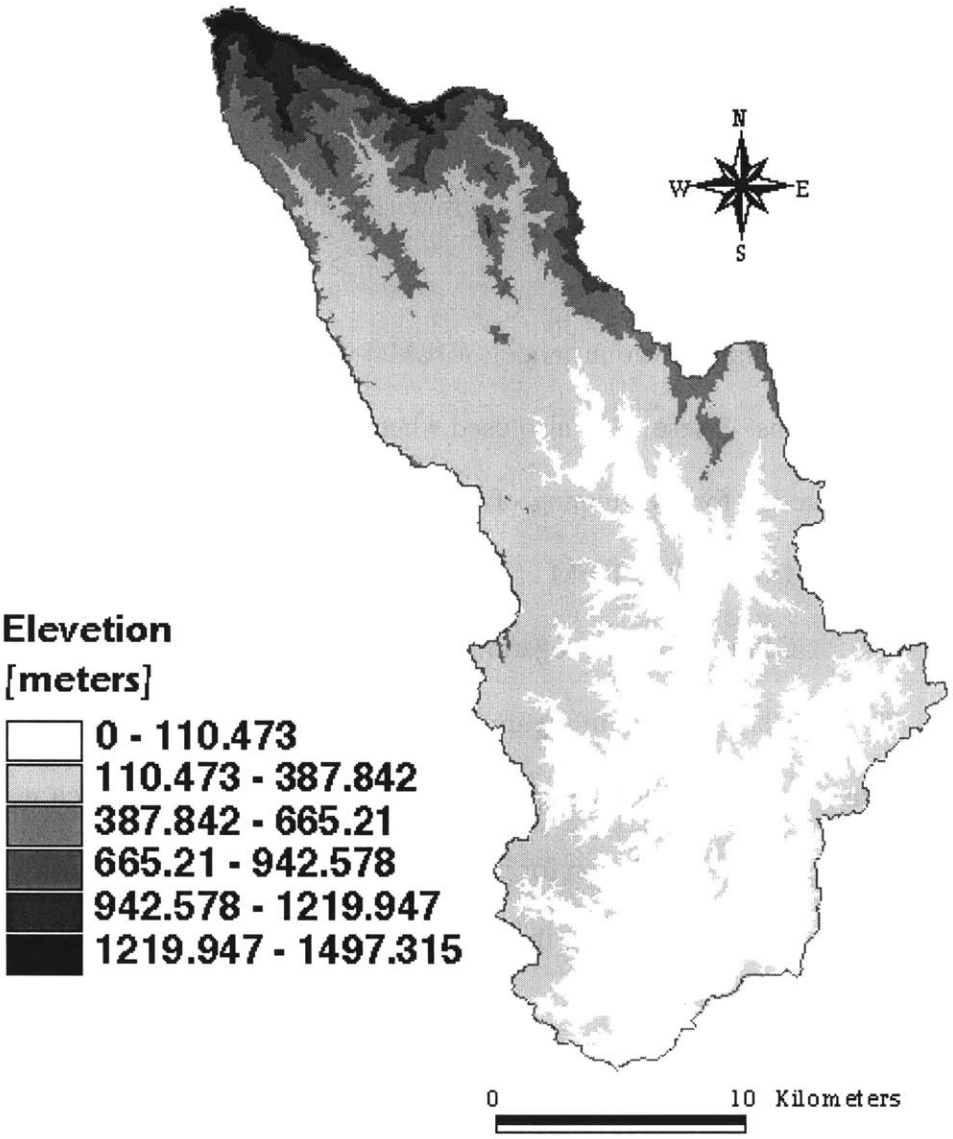

Figure 6 - Digital Elevation Model of the Williams River watershed, 25m resolution

the watershed is far less densely covered. Again, it will not be used in watershed simulation, but was utilized in the analysis of the watershed prior to arrival in the field. This was of particular relevance when considering line-of-sight requirement of the wireless transmission of data using the STEFS field data collection system.

Digital elevation models (DEMs) were available at 25, 50, and 100-meter resolutions. These were also used when preparing for the field trip. The DEM is also needed for the running of tRIBS; the stream network as well as the TIN is derived from it. The DEM of 25-meter resolution is shown in Figure 6.

GIS information regarding soils is very important in watershed simulation. The soils layer shown in Figure 7 shows the very detailed map of soil types. The data originally came formatted with a three-letter soil code, which was then translated to a description for presentation purposes. Soils data is a very important spatial input for the tRIBS model.

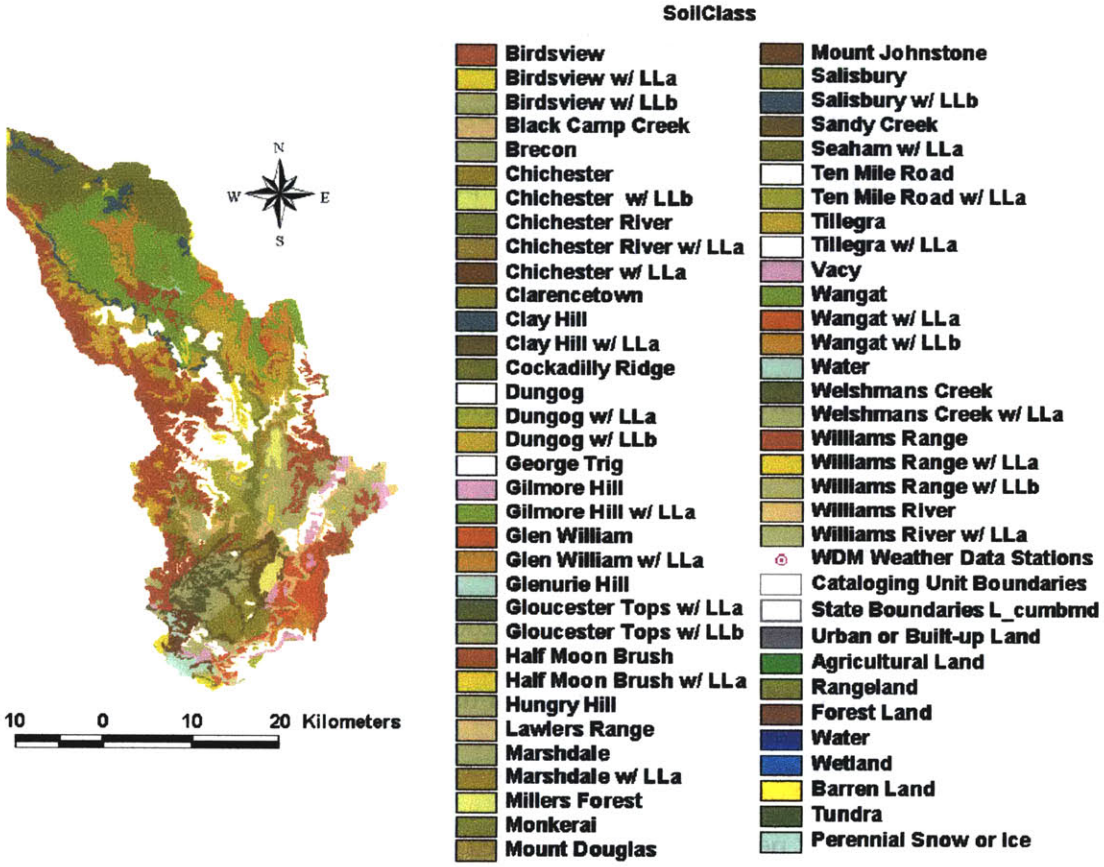

Figure 7 - Soil Map of Williams River watershed

3 tRIBS

The TIN-based Real-Time Integrated Basin Simulator (tRIBS) Distributed Hydrologic model is a set of C++ programs that allow one to construct and simulate catchment level hydrologic processes over a Triangular Irregular Network (TIN). The model is a product of hydrologic modeling and software development by members of the Ralph M. Parsons Laboratory in the Department of Civil and Environmental Engineering at the Massachusetts Institute of Technology.

The users manual, *tRIBS User Manual for Release 1.0 (September, 2001)*, written by Enrique R. Vivoni, gives a good background of the computer model, inputs, outputs and overall directions for use of the software. It also describes the software in terms of object class and workflow diagrams. This chapter will discuss the most pertinent points included in Vivoni's paper. The User's Manual can be found on the web at http://hydrology.mit.edu/tRIBS/tribs/userManual.html. (Vivoni, 2001).

3.1 Background

Distributed hydrologic models resolve physical processes of fluid flow in the land components of the hydrologic cycle in both space and time. This leads to numerous arithmetic manipulations that need to be performed, generally by large computer systems. Because of the complexity of these physically based, distributed models, their execution times are commonly long. This typically

limits the size of areas analyzed by the models, and introduces problems in calibration and validation. Computational demands also make ensemble averaging and probabilistic forecasting difficult. A primary issue contributing to these long run times is the conventional raster grid representation of the landscape. Though this method of organizing and representing terrain data is most prevalent in industry and commercial practices, it proves to be less efficient than other methods. tRIBS utilizes Triangular Irregular Networks (TINs) to ameliorate some of these problems. (Vivoni 2001).

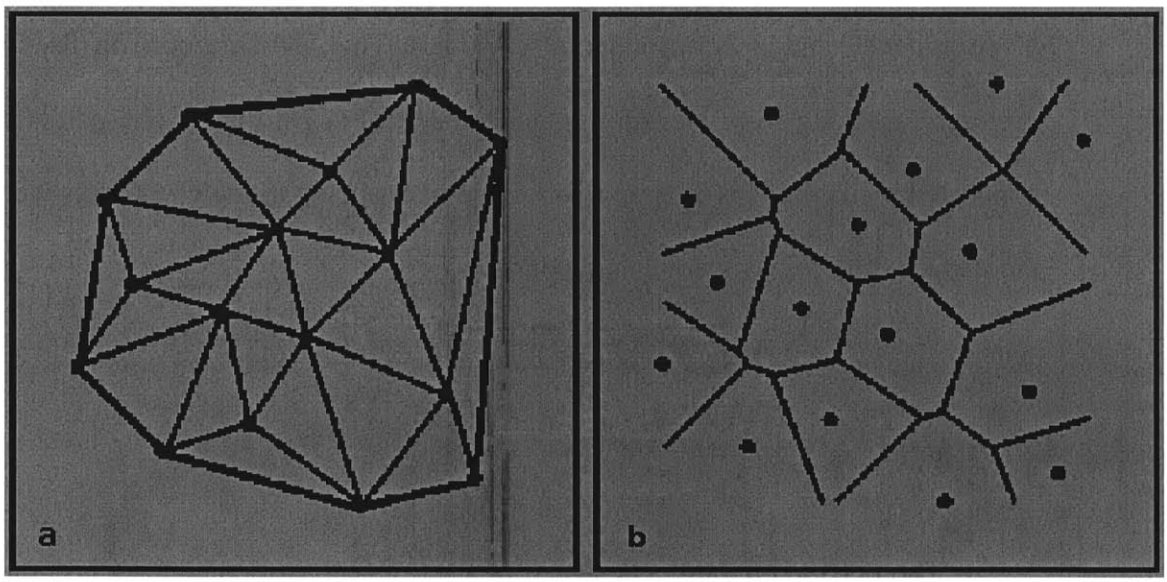

Figure 8 - TIN mesh (a) and Voronoi diagram (b) for a set of 16 nodes in a plane (www.geosoft.com)

A TIN is a data structure that defines geographic space as a set of contiguous, disjoint (non-overlapping) triangles, which may vary in size and orientation. This is quite different than raster grid format, where the data structure is based on square data cells. Figure 8a shows a schematic of a TIN

mesh of 16 nodes; its corresponding voronoi diagram is also shown (Figure 8b). The TIN nodes can often be irregularly spaced, thus allowing for dense information in areas of interest and sparse information in areas of higher homogeneity. The TIN mesh is constructed by connecting the nodes to make triangular areas. The voronoi diagram, in contrast, is constructed such that each region, or cell, consists of the part of the plane nearest to that node (http://www.geosoft.com).

A TIN dataset includes topological relationships both between points and between neighboring triangles. Each point, or node, has associated with its X and Y coordinates and a corresponding Z-Value. This Z-Value can be used to represent anything from elevation to soil properties to land use. The points are connected by their edges to form a set of non-overlapping triangles that represents the surface (http://www.geoplan.ufl.edu/).

3.2 Language

The tRIBS software is designed as a set of object-oriented C++ programs designed for distributed modeling of small- to mid-sized catchments. The object-oriented nature of the C++ programming language lends several advantages over traditional programming languages. Most notably, it allows methods and properties to be separated and grouped along logical and convenient lines. For instance, the hydrological processes that take place along the TIN can be separated from the processes that created the TIN (Tucker, et. al, 2001).

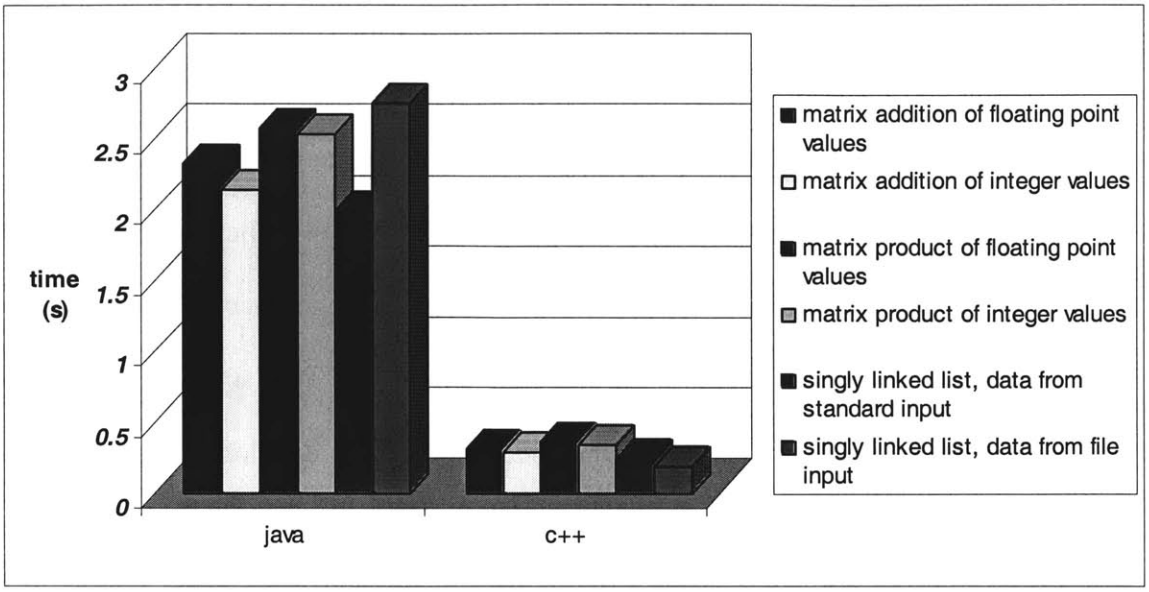

Figure 9 - Performance Comparison, Java vs. C++

Another undeniable advantage of writing the code in C++ is its speed and performance. When compared to Java, C++ has been found to be between 7 and 14 times as fast, depending on the type of operation being compared. When comparing matrix multiplication and addition, something very integral to hydrologic modeling of this type, C++ is over seven times as fast as Java (Gaylon, 4/11/02). Figure 9 shows the relative rates of operation for various operations performed by code written in Java and C++. It is clear that Java is many times slower than C++; an average of approximately 7 times as slow when dealing with matrix manipulation and close to 15 times slower when inputting and outputting to files. Taking these operation times into consideration, it is clear that the choice to engineer tRIBS in C++ was the correct one.

3.3 Architecture

The software architecture of tRIBS is an application of the principles of an object oriented programming language, in this case, C++. The classes constructed utilize and support such object oriented concepts as inheritance, virtual functions, and polymorphism, linked-lists, and class templates. The last two of these were of particular importance within the code. For a more detailed explanation of the program architecture, including class diagrams and a list of the files and directories contained within tRIBS, the reader is directed to Section 2 of the users manual, "tRIBS User Manual for Release 1.0" (Vivoni, 2001).

3.4 Inputs/Outputs

The tRIBS model is designed to be able to accept a variety of input data formats including raster grid data, TIN data files, point data, and text table files. The grid data can be time dependent, as in the case of rainfall or weather, or time-invariant, as in the case of land use or soil parameters. TIN data is inputted into the model using several possible methods, which is primarily determined by the particular application of the data. The point data inputs represent time-variant parameters such as rainfall, which are georeferenced to specific points within the watershed being modeled. Lastly, the text table files are used to correlate specific physical points in the watershed with relevant parameter values such as soil properties and/or land use. (Vivoni, 2001).

The file must conform to certain format specifications in order for the tRIBS model to run correctly. Error checking the file format for errors is placed in the hands of the user. These errors in formats may lead to failure in program execution, and the cause may not be readily apparent. The user must search through the inputs to determine which file, if any, is formatted incorrectly.

The outputs of the tRIBS hydrologic model include numerous files that represent both time series and spatial data. The output files are divided into two directories which are created: *hyd* and *voronoi*. Files placed in the *voronoi* directory pertain to the spatial data outputs whereas the contents of the *hyd* directory are hydrograph related. The locations of these two files are designated in the Input File (.in) under the keywords OUTFILENAME and OUTHYDROFILENAME. The files within these two directories are accessed when analyzing the results of a completed tRIBS model run. (Vivoni, 2001).

For a detailed description of required input file formats, the reader is again referred to the tRIBS users manual, *tRIBS User Manual for Release 1.0 (September, 2001)*. The user's manual also includes a discussion of output file hierarchy and a detailed assessment of standard analysis methods when running tRIBS 1.0.

3.5 Installation on SUN Solaris

Installation of the TIN-based Real-Time Integrated Basin Simulator is a fairly simple process. For the Sun Solaris operating system, one must follow four simple steps.

1. Download the tRIBS.tar file.
2. Untar the file using "tar -xvf tRIBS.tar"
3. Create the folder "_Objects_" using "mkdir _Objects_"
4. Open the makeSUN file with emacs or other text editor.
   Edit line "VP=/var/vivoni/tRIBS..." to read the appropriate path name.
   In the writer's case, this reads "VP=/mit/ericlau/tRIBS". It is the directory that holds all the tRIBS code, and is where the tribs executable will be stored.
5. Make the executable using "make –f makeSun"

This will create the tRIBS.exe executable which is invoked when the model is run.

3.6 Running tRIBS

Correct running of the tRIBS code relies on the correct formatting of the input (*.in) file and that file's configuration with the various files the model will use as inputs. A template of the input file can be seen in Appendix I. This shows the general format and requirements in terms of files needed and syntax to use while creating an input file.

Once the input file is created correctly and the corresponding input files are formatted appropriately, then the model is ready to be run. This is performed by using the following syntax in the command line:

```
% tribs inputfile.in [options]
```

where the "inputfile" is replaced with the name of the input file that has been created, and the [options] is replaced with the desired run options available within tRIBS. For a full discussion and description of the model run options, the reader is referred to the tRIBS User's Manual (Vivoni, 2001). This will invoke the tRIBS code and the model will run. The output files discussed above, *hyd* and *voronoi*, will be created and will be ready for analysis and visualization.

4 Porting

The term "porting" refers to the process of moving a program across operating systems and having it work correctly on the new operating system. In the case of tRIBS, it is currently running only on the UNIX operating systems (Sun Solaris and SGI Irix 6.5), and it was desired to get it working on a Windows operating system.

Moving a program between operating systems involved more than copying files and recompiling or reinstalling it on the new operating system. The porting issue becomes even more involved due to the fact that tRIBS v1.0 is written in C++. The various compilers on the different operating systems require slightly different syntax and/or different libraries included. There are also problems regarding internal methods and functions. In the case of tRIBS v1.0, a significant issue was the MAKE utility, which links the various objects into a single executable. This obstacle and others will be discussed below. This chapter will take you through the motivation behind porting to Windows, and through much of the process involved in the quest to make tRIBS run on a Windows machine.

4.1 Motivation

The motivation behind porting to windows lies in the overall use of tRIBS. Because tRIBS is such a newly developed hydrologic model, there are only a select few who have used it. The amount of research performed with the model and its presence in scientific publication is correspondingly limited. An overall goal is to develop tRIBS into a hydrologic model used by a wide range of people in the scientific community. Ideally, the model will win the favor of hydrologists and it will become their model of choice. This widespread use and support will lend considerable credibility and validation to the model as a hydrologic tool.

A major obstacle to achieving this goal regards outside access to the model. tRIBS only runs on UNIX operating systems supported by MIT, Sun Solaris and SGI Irix 6.5. This arrangement severely cuts down on potential users, specifically users that run a Windows operating system on their computer. A market survey conducted by International Data Corporation (IDC) concluded that Microsoft Windows holds 92% of the desktop computer market (Shankland, 2001). The population of potential users is indeed significant.

While there remains uncertainty in the estimate of Microsoft's market share, its dominance in the computing market is unquestionable. And it is clear that tRIBS solely running on UNIX greatly handicaps its ability to have its user base quickly and easily expand.

4.2 Steps Taken / Issues Confronted

The ultimate goal is to have a working tRIBS computer model working on Win32 machines, including error/exception handling and the rest of the functionality found in the UNIX versions. In the short term, however, the goal was to get any version of the model successfully ported to Windows.

For compiling tRIBS on Windows, DJGPP's GNU compiler was used (available at http://www.delorie.com/djgpp/). This choice of compiler was made because it was most similar to the compiler used to compile tRIBS on the UNIX machines. This compiler allows the use of certain UNIX functionality on a Windows system. The GNU compiler is a command line compiler, which did not have any graphical user interface or other inbuilt help functions. Separate documentation was also available for download, yet was difficult to navigate through and confusing when perusing.

The first errors that were resolved involved error checking and exception handling tags included in the original UNIX version of the code. These were commented out rather than fully translated to successfully ported syntax. This decision was made because the error handling aspects of the code are not essential to the running of the model. These aspects of functionality will likely be made compatible with Windows syntax at a later date.

Though there remain warnings upon compiling regarding declaration of variables such as PI (π) and other warnings regarding casting of variables (e.g. float to double), each of the 31 object files are created and stored in the

"_Objects_" directory through the compiling of their corresponding C++ files (*.cpp).

The final obstacle involved the linking of the object files into a single executable. The make utility is described as "smart enough" to know that the objects need to be linked. The following command was used to attempt to create the *tribs* executable:

%make –f makeWIN

where *makeWIN* is the name of the makefile that was used to compile on windows. This command compares the timestamps on the C++ files and their corresponding Object files. If the timestamp on the C++ file is more recent than the timestamp on the Object file, it recompiles the C++ to create an updated Object file that overwrites the previous Object file. The make utility should then link the 31 Object files into a single executable.

The makefile for compiling on Windows, *makeWIN* (see Appendix II), is adapted from *makeSUN*, the makefile for compiling on Sun Solaris. The primary adaptations involve the correcting of file path names so that they match the Window syntax for file directory navigation.

4.3 Future work

While much work has been put into porting tRIBS from Solaris to Windows, that process is not yet compete. Investigations should be performed into issues of linking the compiled object files as well as clearing up the warnings presented upon creating the object files. The assumption explaining the failure here is that the link.exe utility used in UNIX does not operate the same in Windows. Thus, errors arise in this final step in compilation of tRIBS on Windows.

Another approach the problem would be to investigate the use of different compilers. Here, only a UNIX-like compiler was used. One might also attempt to use an Individual Development Environment (IDE) to accomplish the job. These types of tools often integrate help functions and debugging aids, which could prove useful in finally compiling tRIBS on Windows.

5 Graphical User Interface

tRIBS version 1.0 is a complex C++ program running on Unix through the command prompt. To run the model, one must create and alter files, the input file for example, which lists all the input parameters and files that the model needs. This must be performed in a separate text editor such as Emacs or WordPad. Then, once the model is successfully run, the user must analyze the outputs with outside software applications; this might be GIS software like ArcView for GIS data viewing and analysis, and possibly MATLAB for time series data analysis.

Version 2.0 attempts to streamline and simplify the entire process, creating a single application that eases parameter inputs, necessary file creation, execution of the model, as well as options for visualizing the results of model execution. The main addition to tRIBS v1.0 is a graphical user interface (GUI) that will make use and analysis using tRIBS clearer and easier. The GUI provides the ease of use that is associated with graphical navigation while continuing to allow full functionality available in tRIBS version 1.

5.1 Motivation

While tRIBS v1.0 is fairly easy to use for the experienced user, it takes quite a long time to become familiar with its inputs and outputs, not to mention the options available for running the model. It is also cumbersome due to the

need for the user to also become proficient in an outside mathematics engineering software package, such as MATLAB, and a GIS software package, such as ArcView.

The ultimate goal is to make tRIBS a more user friendly hydrologic model. Ease of use will help encourage more users to use tRIBS as a hydrologic model. And as tRIBS is still a new model, it would be valuable for its core user base to grow in size. This increase in user base will lend it credibility as a valid and useful model within the hydrologic community. The larger user base will also aid in testing and improvements to the hydrologically relevant sections of the model, as well as the features and ease of use of the graphical user interface itself.

5.2 Technology

The decision regarding which technologies to use in making the GUI for tRIBS 2.0 was a difficult one. The decision of which programming language to develop the GUI in required much investigation and debate. The decision was between Java and Microsoft's new language, C#. While each language has its positives and negatives, Java was chosen to be the language that the GUI would be written in.

As discussed earlier, Java's performance pales when compared to C++. The benefit of Java, however, lies in the overall architecture of Java and its implementation through the Java Virtual Machine (JVM). When compiled, the written Java code, in .java files, is converted to a byte code, which is the run by

the JVM. The unique thing about Java is that a given Java program can be run on any machine that has a JVM running on it. This allows platform and operating system independence. Something that is written in Java, therefore, would be able to work for users regardless of whether they are on a UNIX workstation or a Windows desktop.

Microsoft response to this concept of platform independence is suggested in their newest programming language, C#. Upon compilation, code written in C# is converted into intermediate language (IL) with is run by an abstraction layer. In the Microsoft case, this is the Common Language Runtime (CLR). The CRL is the analogy to Java's JVM, allowing code written in C# to be run on any machine that has a running CLR and the Microsoft .NET framework installed. The problem, however, is that there is currently no CLR available for UNIX. While these are being developed in the software community, and are planned for release in the near future, it remains to be seen how well the CLR will ultimately work.

The choice of Java as the language to build the GUI for tRIBS is based primarily on the platform independence issue. A central goal of tRIBS v2.0 is increasing the potential for increased user base. And while tRIBS is yet to be successfully ported to Windows, it can be easily integrated into tRIBS v2.0 when it eventually is released. Thus, the GUI built with Java will be generic enough to be used as the front end for both the current UNIX version as well as the future Windows version.

The performance issue was determined to be less significant because the GUI will not run computationally extensive operations. The data previously presented relating to Java's performance pertains mainly to matrix manipulation and file input and output. The GUI does relatively little reading and writing, and only performs matrix manipulation during the initial steps of visualizing the data layers. These operations are not constantly performed and will only cause a negligible and momentary decrease in performance.

5.3 Layout

The layout for tRIBS v2.0 is loosely designed after GIS software (Figure 10). The menu bar allows you access to all of the functionality of the software. The left panel is where the visualized layers are listed. The images themselves

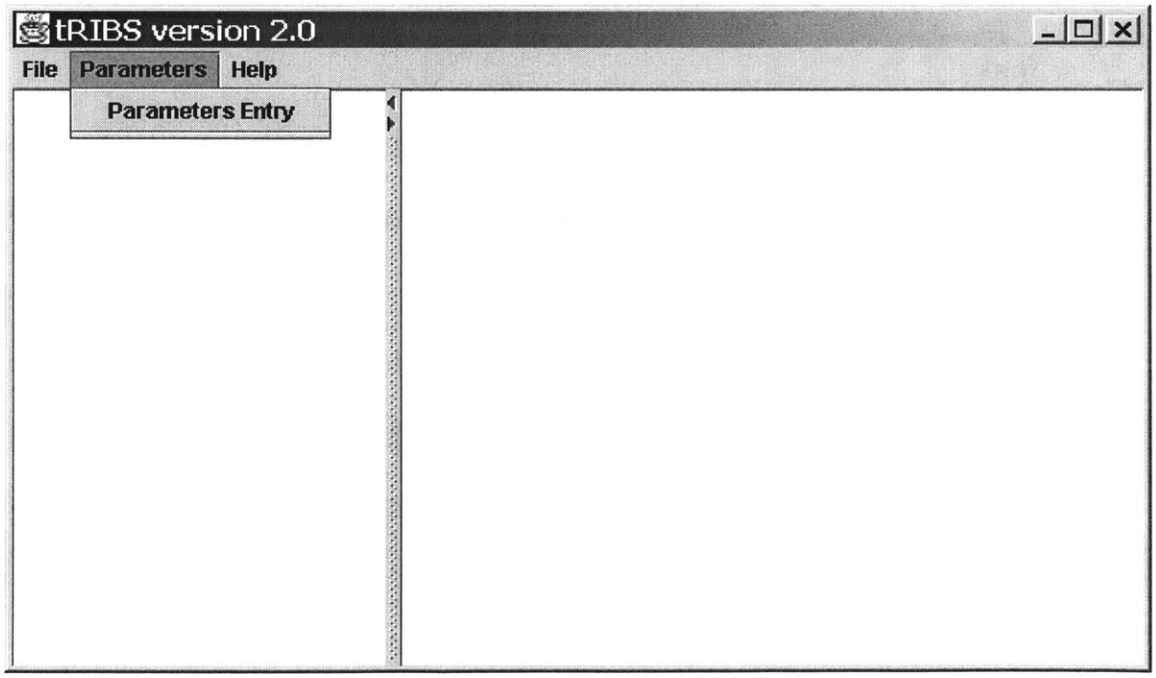

Figure 10 - tRIBS GUI upon startup

are displayed in the larger right panel. This layout allows the user full use of all aspects of the tRIBS model as well as the ability to view the inputs to and outputs from the model.

5.4 Features

tRIBS v2.0 includes a number of features that make using the hydrologic model much simpler. These features range from enhanced inputting of files to visualization of the inputs. All improve the model's ease of use and appeal to the common user.

5.4.1 Inputting of Parameters

A new feature of tRIBS v2.0 is the simplified creation of the input (.in) file. The user can choose to create an input file by selecting *Parameters* → *Parameters Entry* (Figure 10). This opens up a "New Input File" dialog window which prompts the user to enter the information required by the input file (Figure 11). The user is prompted to enter variables such as STARTDATE and RUNTIME, as well as filenames and paths. When entering filenames and paths, the user can choose between typing in that information and selecting "*choose a file...*" allowing the user to browse for a specific file (Figure 12). This

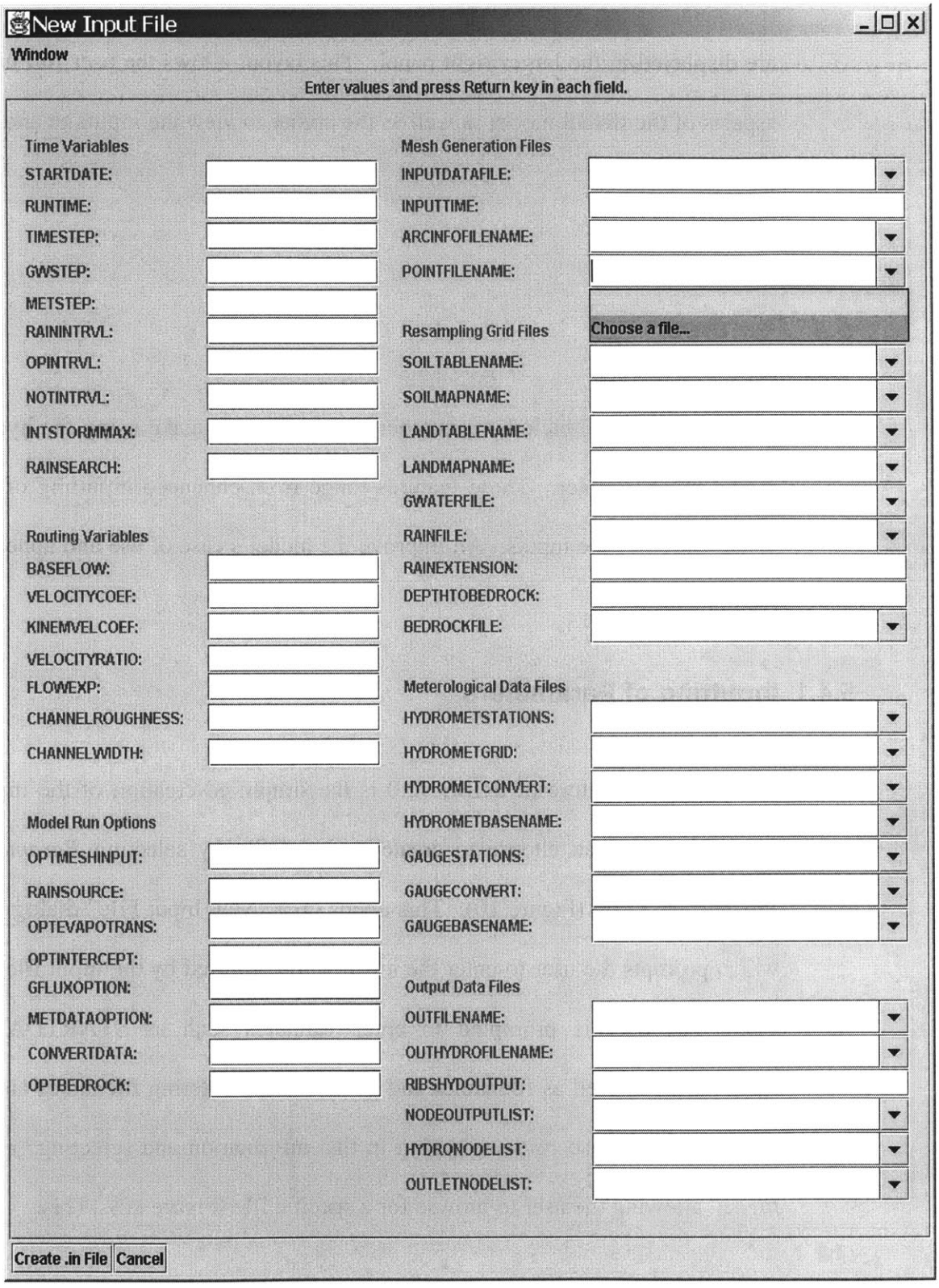

Figure 11 - New Input File dialog window used to create input (*.in) files

is instantiated by the *FileDialog*() object in Java. It takes up to three arguments:

```
Public FileDialog(Frame parent,
                  String title,
                  int mode)
```

"parent" is the Frame that is the owner of the dialog, "title" is the string that will serve as the title for the dialog window, and "mode" specifies whether the window will be of type LOAD or SAVE. A LOAD specification is used here, since we are loading the filename into the input file window. The SAVE distinction will be used later when the user is requested to save the newly created input file. The *FileDialog*() object allows the user to navigate through the folders and directories on their machine to select the desired file. Upon navigating to the desired file location and selecting *Open*, the designated filename and path is entered into the active text field.

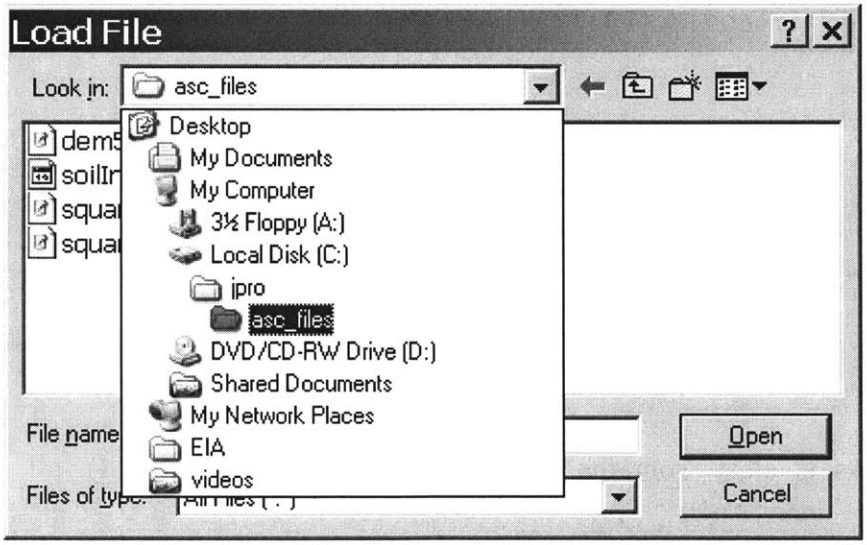

Figure 12 - Load File dialog window. Used for inputting filenames into input file window.

When the user has entered all the necessary data into the "New Input File" dialog window (Figure 11), the *Create .in File* button should be clicked. This opens another dialog window, named "Save File", which prompts the user to name the new Input file as well as designate where the new file will be stored (Figure 13). This is where the SAVE designation of the *FileDialog*() is implemented. Clicking *Save* in this dialog window triggers the creation of the output file. The variables are stored as variables within an object, and the object is called when writing out to the file.

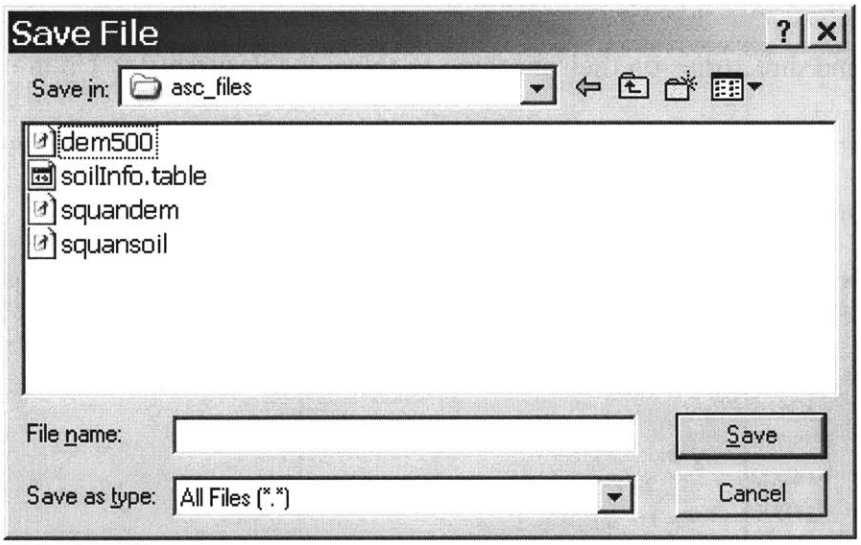

Figure 13 - Save File dialog window. Used to save the newly created Input file to disk.

As is clear from the example file created (Appendix I), all inputted variables are written to the file. The code ensures that the formatting of the input file is appropriate for use in running tRIBS. This eliminates a possible point of entry for errors that may cause the model to fail, making tRIBS more resistant to incorrect use.

Another option for the user is to use an existing input file. This is accessed by selecting *File* → *Import .in file* (Figure 14). This opens up a load *FileDialog()* window that allows the user to navigate to the desired input file, one identical to Figure 12. Upon clicking *Open*, the input file is read and parsed, and each element of the file is stored in an

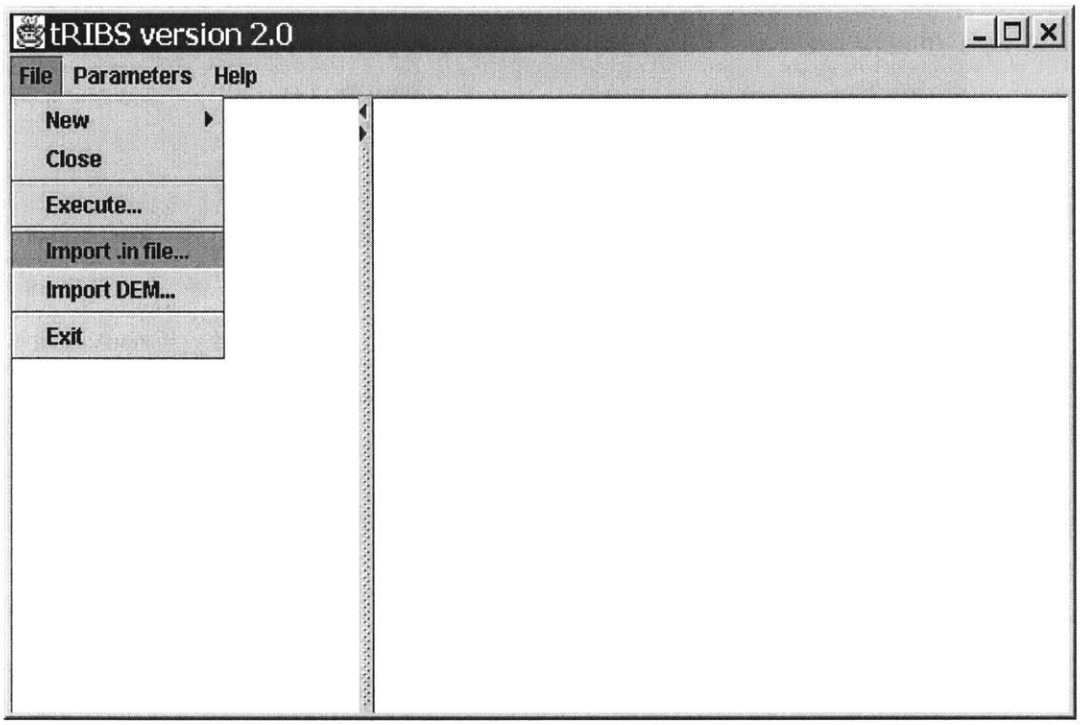

Figure 14 - tRIBS v2.0 main window. Shows location of *Import .in file...* is located

object. The user may view the input file by selecting *Parameters* → *Parameters Entry*. This will once again open the "New Input File" window, but now the fields will be populated with the contents of the imported input file (Figure 15).

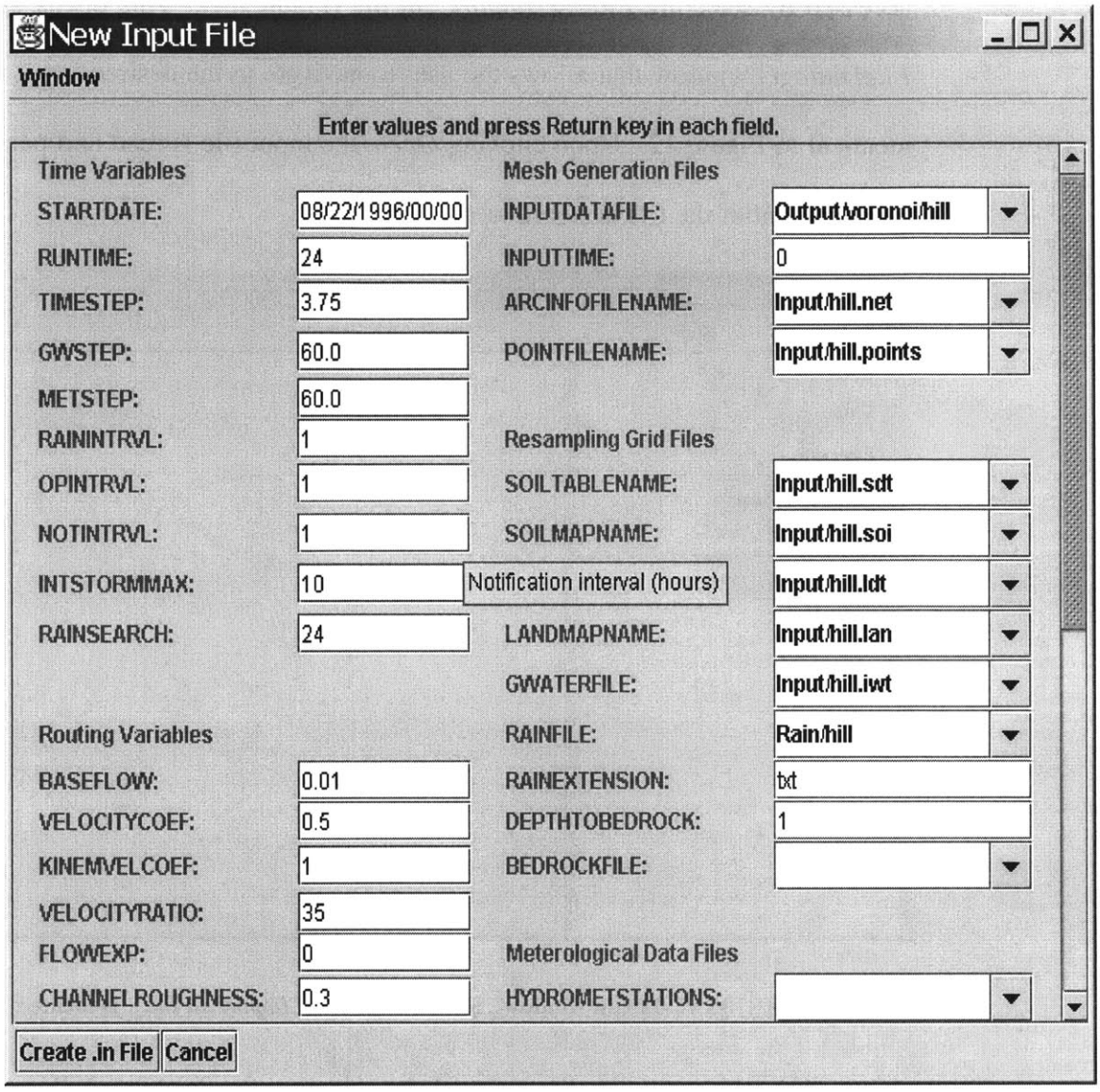

Figure 15 - Dynamically populated Input File dialog window. Populated with hill.in

The user can now edit, augment, or delete entries in the New Input File window and save the changes to a file through the same method that is used to save a newly created file.

5.4.2 Model Execution

Execution of the tRIBS C++ executable is controlled by the GUI. When the user has modified the input file to satisfaction, execution of the model occurs by selecting *File* → *Execute*.... This opens up a dialog window which prompts the user for the location of the desired input file (Figure 16). Upon the user clicking *Execute File*, tRIBS is run using the specified input file.

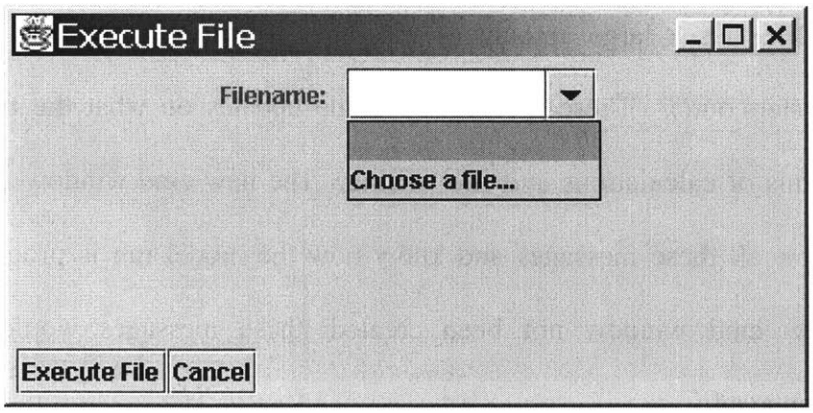

Figure 16 - Execute tRIBS dialog window.

An example of Java code used to call a C++ executable is shown below:

```
try{
      String command = ("cmd /c start /d \"c:\\jpro\" java
WatershedProgram");
      Process p = Runtime.getRuntime().exec(command);
      System.out.println("Trying to call executable using:
\n\"" + command + "\"");
      }
catch (IOException e1) {
      System.out.println("Exception Caught.\n");
      System.err.println(e1);
      System.exit(1);
      }
```

This is the code as it would appear on a Windows machine. This shows how Java, through the Runtime object, makes a call to the system line using the string *command.* Here, the *command* string lists several commands that will be called. First, a new cmd, or terminal, window is opened using "*cmd /c start /d*". Then, the local directory is changed to *c:\jpro.* Lastly, the program, "WatershedProgram", is called using "*java WatershedProgram*".

The advantage of opening up a new cmd window is that a program such a tRIBS has a large amount of messages it writes to the cmd window through System.out(). These messages include updates on what the model is doing in terms of calculations and file writing. The new cmd window allows the user to view all these messages and know how the model run is progressing. Had the new cmd window not been created, these messages would have not been displayed.

The lines of code above will not work on UNIX. While Java is known for its platform independence, as discussed earlier, it is calls to the system that are platform specific. This is because of the nature of Java's Runtime object. The *getRuntime().exec* within the *Runtime* object is the method by which the executable is called. The *Runtime* object, shown in Figure 17, interfaces the JVM and the hardware and software specifics of the host computer; including syntax and structure used within the native operating system.

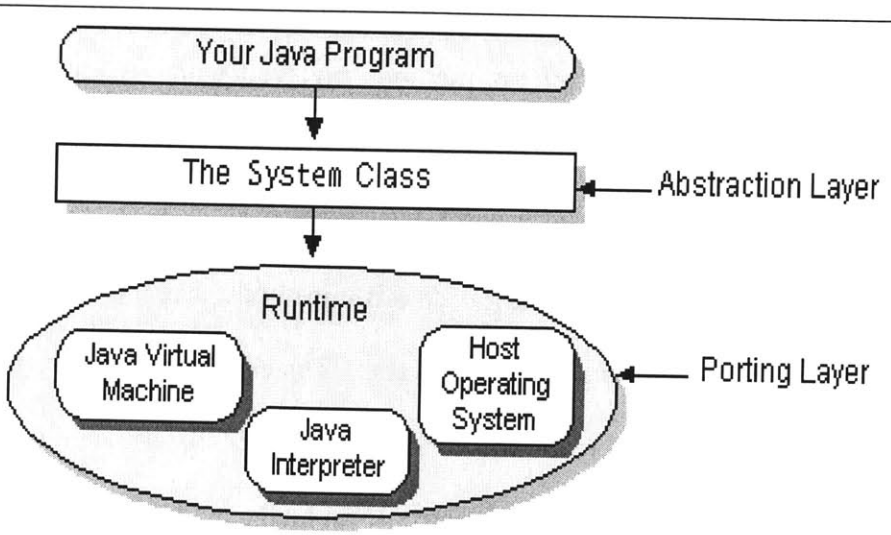

Figure 17 - Runtime Object (source java.sun.com 2002)

In the example above, only the command string would change when porting to UNIX. The commands to create a new terminal are obviously different from Windows. For example, the *cmd* of the windows command string would be replace with *xterm&* in the UNIX case. The changing of directories must be corrected for the specific location of the *tribs* executable. This must be reviewed before packaging of tRIBS v2.0 for release. Also, if the path listed in the string *command* does not match the actual location of the executable, execution of the model will be unsuccessful.

5.4.3 Visualization

A major motivating factor in the creation of tRIBS v2.0 is the integration of visualization of the inputs and outputs with the actual watershed model execution capabilities. The visualization of the inputs was accomplished here.

The majority of the methods involved with visualizing GIS data layers were modified from javaWABS, a hydrologic modeling software package developed for use in the 1.070 Introduction to Hydrology class at MIT (E. Vivoni, pers.comm., 2002). The javaWABS methods adapted for use in tRIBS v2.0 involve the plotting of GIS layers. Three methods are invoked in Vivoni's software: one that reads the GIS data from ASCII format into a matrix, one that converts the data within the matrix to an image, and one that displays the image in the desired frame. These methods, with some modification, were used to visualize the data.

Visualizing the inputs is very simple for the tRIBS user. When the user is done with the New Input File window, whether it is from the importing of an existing input file or the creating a new input file, the left panel of the tRIBS main window is populated with viewable data layers. Figure 18 shows the results when the "hill.in" example is imported. The viewable GIS layers are the soil map (Input/hill.soi), the land use map (Input/hill.lan), and the groundwater map (Input/hill.iwt). When the user clicks on a filename listed in the left panel, that map is generated and displayed in the main, right panel. This takes a few moments as the program reads the ASCII file into a matrix and converts the matrix into an image. This process may require millions of arithmetic manipulations, depending on the size of the watershed and the resolution of the data. Figure 19 shows a successfully displayed soils map.

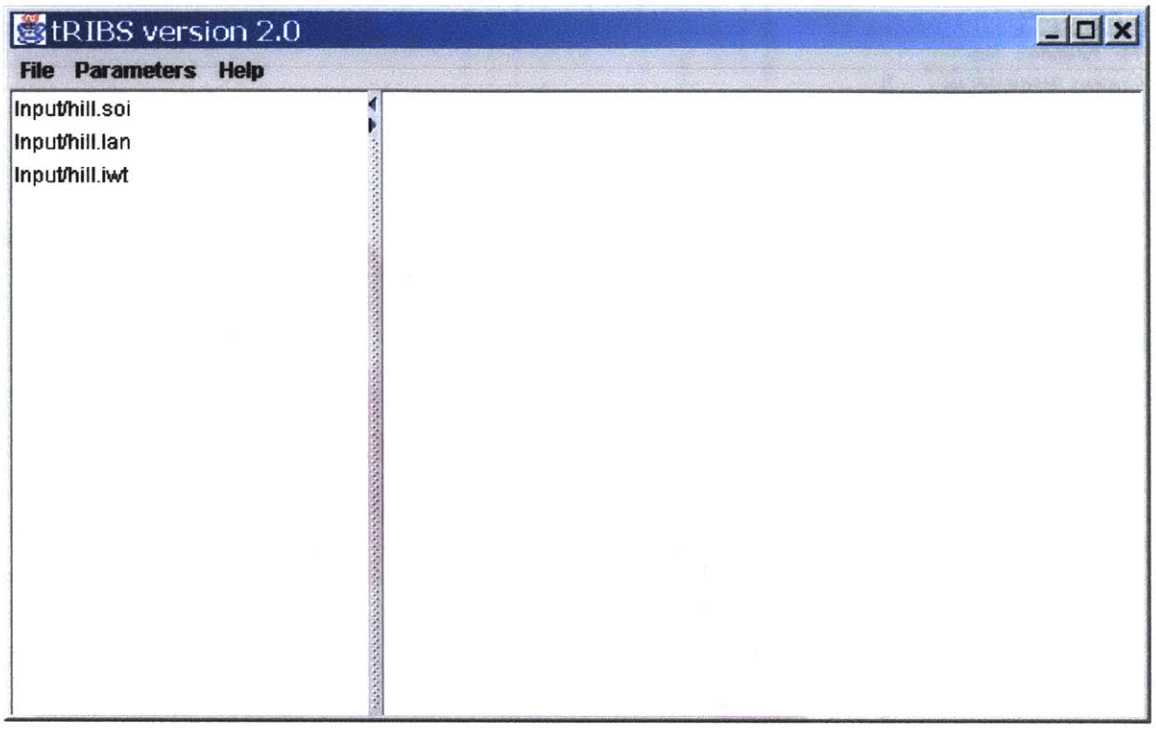

Figure 18 - tRIBS GUI left panel populated with viewable GIS layers

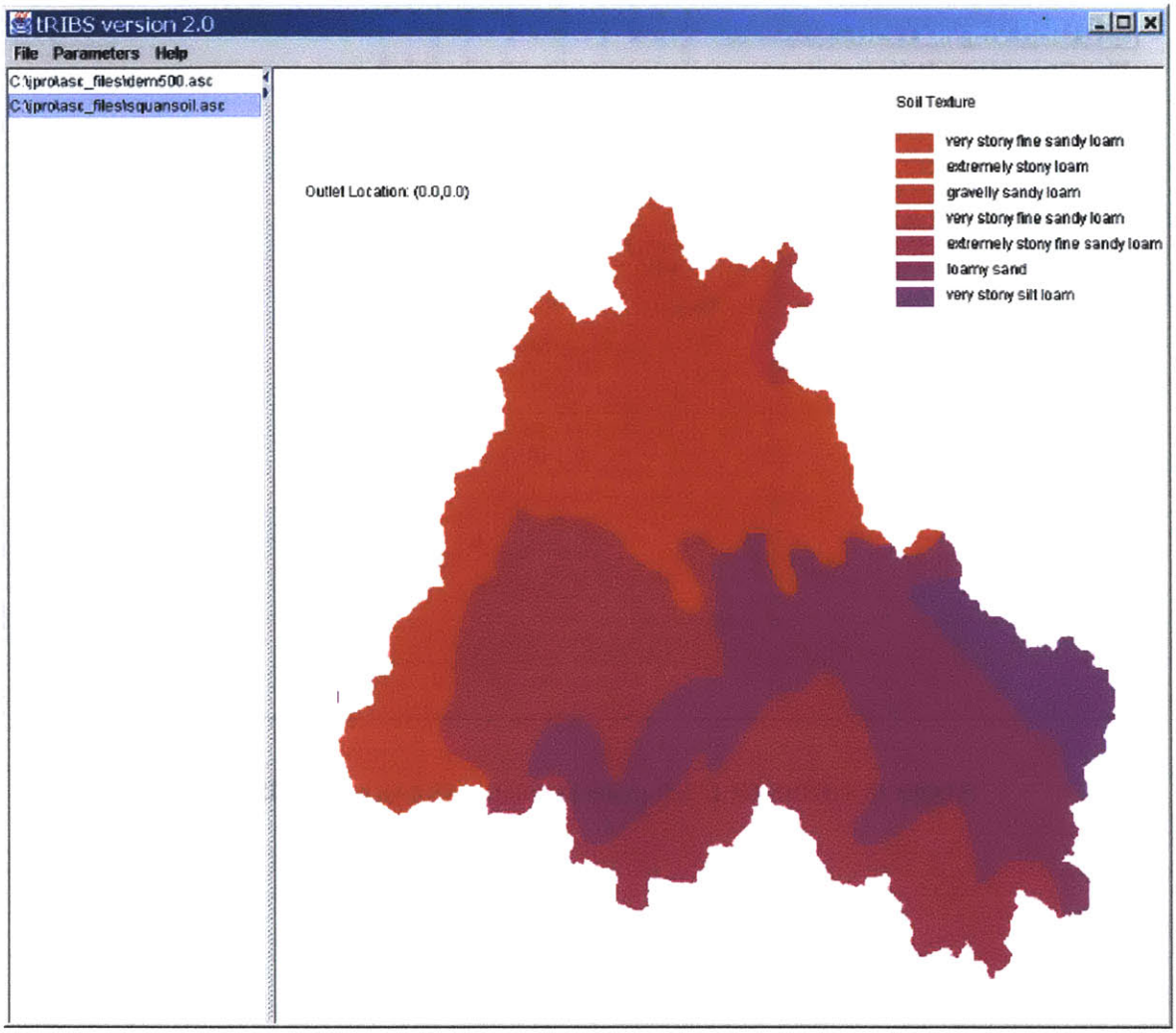

Figure 19 - tRIBS visualizing Soils GIS input

Another useful GIS layer is the DEM. While this is not directly used in the watershed simulation with tRIBS, it is useful to visualize while analyzing the study area. And since the DEM is not listed in the input file, it is not loaded into the left side panel of the main window along with the other layers.

A separate menu item was created for inputting the DEM. This can be accessed through the *File* → *Import DEM...* option. This opens up an "Import

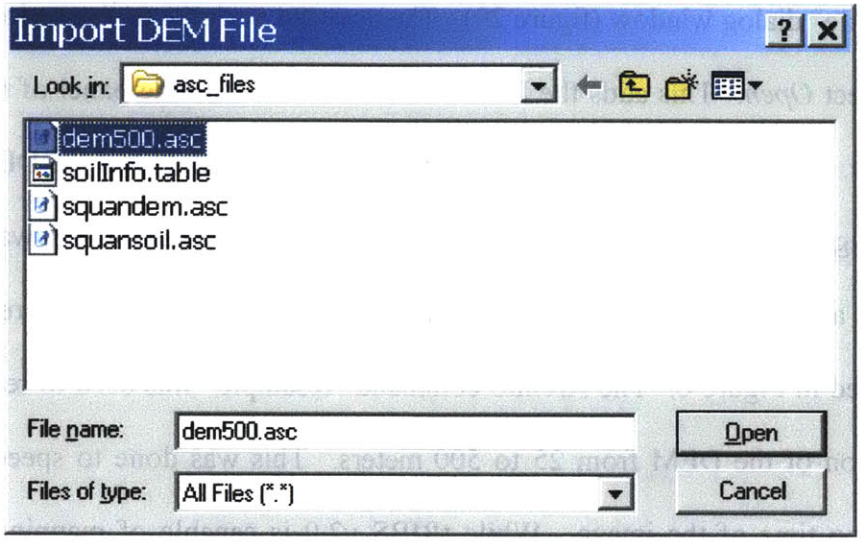

Figure 20 - Import DEM dialog window

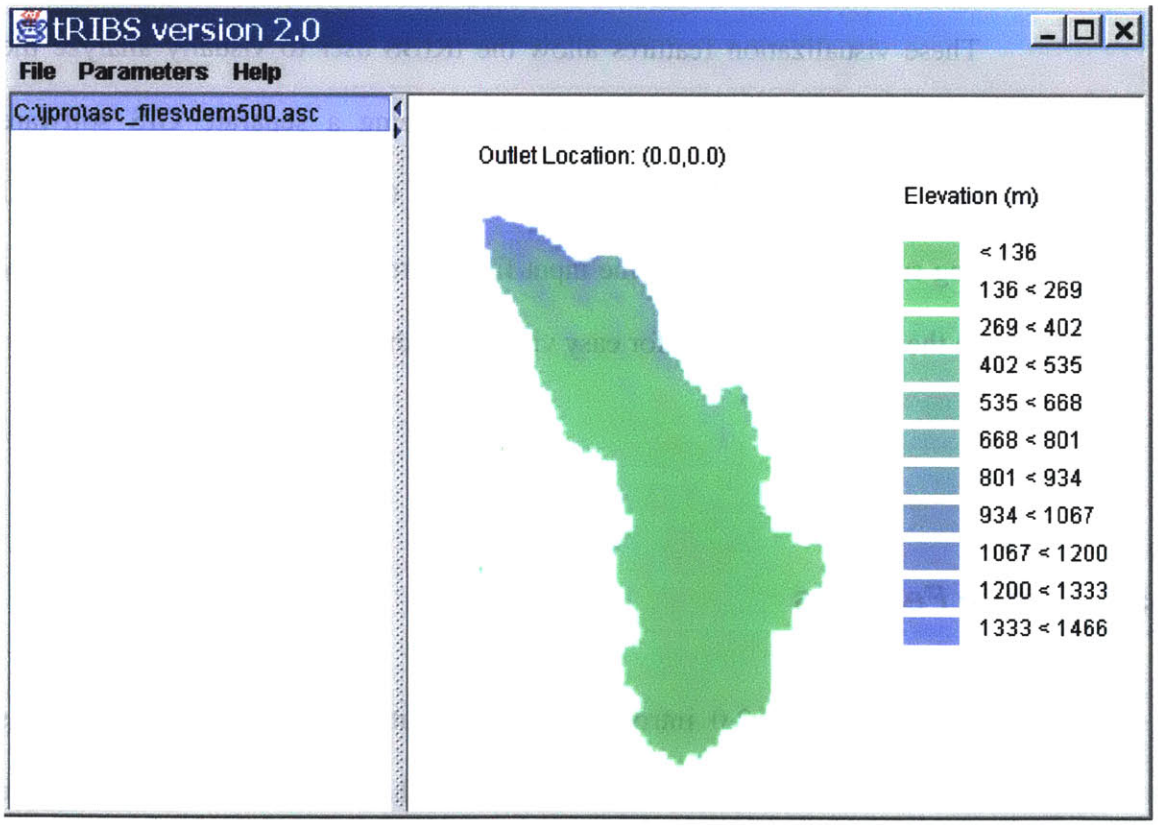

Figure 21 - tRIBS v2.0 visualizing the DEM of the Williams River watershed

DEM File" dialog window (figure 20). The user can navigate to the desired DEM and select *Open*. This adds the DEM filename to the left side panel of the main window. The user can click on the filename to render the image of the DEM.

Figure 21 shows the mapping of the DEM of the Williams River watershed. This is a DEM at 500-meter resolution rather than at the 25-meter resolution displayed in Figure 6. The Arcinfo command "resample" was used to reduce the resolution of the DEM from 25 to 500 meters. This was done to speed up the rendering time of the image. While tRIBS v2.0 is capable of mapping the 25 meter DEM, the faster rendering time of the 500 meter DEM lent efficiency in product testing of the software.

These visualization features allow the tRIBS user to visually analyze the inputs of a particular model run without utilizing a separate GIS software package. It automatically loads the viewable GIS layers in the left panel following successful loading of the input file information. The user must simply click on the desired GIS layer for easy viewing of the rendered image.

5.5 Future Features

While tRIBS v2.0 introduces several features that enhance the user's experience with the software, there still is much that can be done to improve the software. Several of these features are discussed here.

tRIBS is easily susceptible to crashing if the arrangement of files and data are not set to specification. Error checking methods could be invoked upon entry of parameters, which would ensure allowable values are inputted. Scanning methods could also be introduced that would ensure data input files were formatted correctly. Both of these methods would increase the robustness of the tRIBS software, making it more resistant to misuse.

Visualization of the input files was included in the current version of tRIBS. Visualization of the outputs was not attempted. The visualization methods used for the inputs can be used as a template for development of visualization of the outputs. The conversion must be made from visualization of rectangular grid cells to visualization of the TIN mesh and the TIN nodes' corresponding GIS data. Upon the completion of the model run, the output files available for visualization should be listed in the left side panel along with the viewable inputs. The user would then be able to select the desired file to call the visualization methods and display the corresponding image.

The ability to save images produced in tRIBS would also improve its attractiveness to potential users. Many programs contain an "export" option that allows the user to convert an image into a savable image file in JPEG, GIF, or similar format. This option could easily be added on at a later date without affecting any of the existing code.

A searchable help function would increase usability of tRIBS significantly. A user could consult the in-built help functions to learn more about the model's inputs, outputs and options. Much of this information could be

derived from the already existing *User's Manual* augmented by tips relevant to use of the software.

Another way to enhance the user experience may be to introduce the concept of a "project" to the program. Under this scheme, a user would open up a project that would contain inputted parameters, run options, as well as output files. The user could open up an existing project and have immediately available all the resources configured in the previous session with the project. As of now, the user would have to re-enter the input files or paths to output files each time he opens the tRIBS program. This would be a significant task, as many aspects from all parts of the program would be involved.

Great focus was placed on making tRIBS v2.0 scaleable and adaptable. tRIBS continues to evolve and grow as a hydrologic model; and the GUI needs to be able to accommodate those changes. Nearly all features introduced in tRIBS v2.0 are very easily updated to include new functionality. Furthermore, it was ensured that new features could easily be integrated into the current software package simply and elegantly.

5.5.1 Web Service

Web-based services are quickly becoming the latest trend in the IT world. The concept involves users accessing programs over the Internet rather than having the program on the local machine. The local machine would be able to make calls and requests to the host machine as well as being able to access the outputs from the program invoked on the host machine. This frees up the local

machine's processor and moves the computation to the web server, which is ideally much more powerful. The program is available to any user with an Internet connection and a desire to use the program.

The realization of tRIBS as a web service will be a complex and involved. The implementation and execution of the tRIBS model itself will not be extremely difficult, nor will the pure accessing and transfer of files require much effort. The things that will require much work and consideration are security issues, permissions, and data analysis options.

Microsoft's .NET framework enables easy setup of web services. The in-built documentation within Microsoft's *Visual Studio .NET* describes the .NET framework as follows:

> The .NET framework is a multi-language environment for building, deploying, and running XML Web services and applications. It consists of three main parts: Common Language Runtime, Unified Programming Classes, and ASP.NET.

The Common Language Runtime (CLR) plays a role in both runtime and development environments. It performs numerous tasks including memory allocation, security policies, and thread handling. The CLR works to dramatically ease the user's experience, reducing the amount of code that needs to be written by the user.

Unified Programming Classes combine the class libraries from various programming languages, creating a common set of extensible class libraries (APIs). This allows for cross-language inheritance, debugging and error handling.

ASP.NET is most relevant to the web services envisioned for tRIBS. It provides a Web application model that allows for simple creation of web applications. The applications run on the web server and push user interface through HTML to the local machine's browser. It also provides methods to enable developers to deliver software as a service. This is directly applicable to tRIBS.

A development tool such as *Visual Studio .NET* greatly eases the creation of web services. It decreases the amount of code written by the developer, as well as dynamically handles the connections, cross-references, and organization of files involved with creation of web services. In this way, the developer can build a web service even without a thorough knowledge of the inner workings of web services.

The architecture used to store the input parameters, creation of the input file, and execution of the model in tRIBS v2.0 would be followed in the design of the web service. The decision then must be made as to whether the tRIBS web service will return the output files to the remote user, or if the files will be stored on the host machine. The decision also must be made regarding the visualization of the outputs. One choice is to have the user receive the output files and be responsible for analysis and visualization on their own. A better choice would be for the tRIBS web service to go through the process of making the images and storing them on the host machine. These image files could then be retrieved and displayed on the remote computer.

The issue of security becomes a factor when considering how much power they user will have in terms of saving to the web server's hard drive or accessing potentially private files. Giving total permissions to every user threatens the integrity of the system, introducing potential for any number of bad things including a user deleting necessary files, even the web service itself. Limiting permissions with passwords would provide a base level of protection, but might fail under more complex situations such as handling of concurrent multiple users. Security and permissions will require considerable research and consideration, especially when the numbers of users increase.

It is very possible to make tRIBS into a web service. And while the implementation of such a web service brings along with it a wealth of permissions and security issues, these are undoubtedly manageable. The move to web services can provide tRIBS with yet another attraction by putting it on the forefront of the distributed computing movement.

6 Conclusion

Much work was done toward hydrologically modeling the Williams River watershed with tRIBS. A data collection trip to the field was taken in January 2002. The trip was successful in both terms of collecting field data as well as testing the innovative STEFS field data collection system. Data was also collected prior to site selection. The data collected pertained to both water quality and water quantity of the watershed. These data layers are available for use in modeling the watershed in the future.

Significant work was also put into development of tRIBS into a fully integrated hydrologic modeling software package. Steps were taken to port tRIBS from UNIX to Windows. And while these efforts were unsuccessful, much was learned about the porting process. Further steps can be taken with the altered C++ files make them successfully ported to Windows

The graphical user interface front end was created in Java. This software allows the user to create the input (*.in) file required by tRIBS. It also allows execution of the tRIBS model through the GUI. Finally, the GUI allows for visualization of the tRIBS model inputs, integrating the watershed simulator with data analysis capabilities. All these steps are taken to develop tRIBS into a more attractive and marketable product, increasing its user base and increasing its credibility as a hydrologic model.

There remains much to be done before tRIBS is a full functioning and complete software package. All of the work done can be easily added upon; great care was taken to make the code easily scalable and adaptable to what the model may become in the future. The evolution of the software may take a different turn altogether, becoming a Web-based service. With the recent improvements technologies he future of tRIBS is as open as the minds of the people who will work on it.

7 References

Published Papers:

Garrote, L; Bras, R. *A distributed model for real-time flood forecasting using digital elevation models.* Journal of Hydrology. 167 (1995). 279-306.

Tucker, G.E., Lancaster, S.T., Gasparini, N.M., Bras, R.L. and Rybarczyk, S.M. 2001. *An Object-oriented framework for distributed hydrologic and geomorphic modeling using triangulated irregular networks.* Computers and Geosciences. 27(8):959-973.

Vivoni, E.R., Camilli, R., Rodríguez, M.A., Sheehan, D.D. and Entekhabi, D. *Development of mobile computing applications for hydraulics and water quality field studies.* Hydraulic Engineering Software IX. WIT Press. Montreal, CA. (2002)

Walker, Jeffrey; Willgoose, Garry; Kalma, Jetse. *The Nerrigundah data set: Soil moisture patterns, soil characteristics, and hydrological flux measurements.* Water Resources Research, Volume 37, no. 11. 2853-2685.

Woolridge, Scott; Kalma, Jetse; G. Kuczera. *Parameterisation of a simple semi-distributed model for assessing the impact of land-use on hydrologic response.* Journal of Hydrology 254 (2001) 16-32.

Woolridge, Scott; Kalma, Jetse. *Regional-scale hydrological modelling using multiple-parameter landscape zones and a quasi-distributed water balance model.* Hydrology and Earth System Science, 5(1) (2001) 59-74.

Woolridge, Scott; Franks, S; Kalma, Jetse. *Hydrological implications of the Southern Oscillation: variability of the rainfall-runoff relationship.* Hydrological Sciences, 46(1), Feb. 2001. 73-88.

Woolridge, S; Kalma, J; Franks, S; Kuczera, G. *Model identification by space-time disaggregation: a case study from eastern Australia.* Hydrological Processes, 16. 2002. 459-477.

Manuals:

Vivoni, E.R., tRIBS User Manual for Release 1.0 (September, 2001) Technical Report. Department of Civil and Environmental Engineering, Massachusetts Institute of Technology. Cambridge, MA.

Reports:

Camp Dresser & McKee Inc., *Evaluation of Integrated Surface Water and Groundwater Modeling Tools*, February, 2001.

EnviroCom Final Report (2002)
Agarwal, Neeraj; Lau, Eric, Kolodziej, Kris; Richards, Kevin; Spieler, Russell; Tsou, Ching-Huei.

EnviroCom Project Proposal (2001)
Agarwal, Neeraj; Lau, Eric, Kolodziej, Kris; Richards, Kevin; Spieler, Russell; Tsou, Ching-Huei.

Presentations:

EnviroCom Project Proposal Presentation, M.I.T., 2001
Agarwal, Neeraj; Lau, Eric, Kolodziej, Kris; Richards, Kevin; Spieler, Russell; Tsou, Ching-Huei.

EnviroCom Final Presentation, M.I.T., 2002
Agarwal, Neeraj; Lau, Eric, Kolodziej, Kris; Richards, Kevin; Spieler, Russell; Tsou, Ching-Huei.

EnviroCom Presentation to Environmental Systems Research Institute (ESRI), Inc., Redlands, CA, 2002
Agarwal, Neeraj; Kolodziej, Kris; Richards, Kevin; Spieler, Russell; Tsou, Ching-Huei

Vivoni, E., December, 2001. Software Tools for Environmental Field Study (STEFS), MIT/Microsoft iCampus Project – Technical Report 1.

Master of Engineering Theses:

Kolodziej, Krzysztof (Kris). *Integrating Real-Time Geo-Referenced Data Streaming into Interoperable GIS Web Services.* M.Eng./M.C.P. thesis, Massachusetts Institute of Technology, 2002.

Richards, Kevin. *Hydrolic and Water Modeling with HSPF: Utilization of Data from a Novel Field Study and Historical Documents.* M.Eng. thesis, Massachusetts Institute of Technology, 2002.

Spieler, Russel. *Real-Time Wireless Data Streaming in a PDA-Based Geographic Information System.* M.Eng. thesis, Massachusetts Institute of Technology, 2002.

Tsou, Ching-Huei. *GIS Web Services using .NET Framework.* M.Eng. thesis, Massachusetts Institute of Technology, 2002.

Books and Articles:

Block, Joshua. Effective Java Programming Language Guide, 1st edition. Addison Wesley Professional. June 5, 2001

Deitel,Deitel; Nieto,Yaeger; and Zlatkina. C# How to Program, Introducing .NET, 2002.

Djokic, D; Maidment, D. Hydrologic and Hydraulic Modeling Support with Geographic Information Systems. ESRI Press. April 2000.

Flanagan, David. Java in a Nutshell, 4th edition. O'Reilly and Associates. April 2002.

Gaylon, Eric. C++ vs. Java Performance. http://www.cs.colostate.edu/~cs154/PerfComp/ April 11, 2002.

Horton, Ivor. Beginning Java 2SDK 1.4 Edition, 1st edition. Wrox Press Inc. March 2002.

Josuttis, Nicolai, M. The C++ Standard Library: A Tutorial and Reference, 1st edition. Addison-Wesley Publishing Company, August 1999.

Moore, Jeffrey; and Weatherford, Larry. Decision Modeling with Microsoft Excel, 2001.

Peck, Jerry D. Learning Unix Operating System, 5th edition. O'Reilly and Associates. January 15, 2002.

Reynolds, Alan. The Monopoly Myth. Wall Street Journal. April 9, 1999. http://interactive.wsj.com/archive/retrieve.cgi?id=SB923609170125696 77.djm

Robbins, Arnold, Gilley, Daniel. UNIX in a Nutshell: A Desktop Quick Reference for SVR4 and Solaris 7, 3rd Edition. O'Reilly and Associates. November 15, 1999.

Shankland, Stephen. Linux growth underscores threat to Microsoft. CNET News.com. February 28, 2001. http://news.com.com/2100-1001-253320.html?legacy= cnet.

Stroustrup, Bjarne. The C++ Programming Language Special Edition, 3rd edition. Addison-Wesley Publishing Company. February 15, 2000

Web Sites:

http://www.geosoft.com/

http://java.sun.com

http://www.jguru.com

http://www.javaworld.com

http://envitweb.mit.edu

http://HITS.nsw.gov.au/

http://msdn.microsoft.com

http://www.dell.com

http://www.esri.com

http://www.gpsscales.com/

http://www.hydrolab.com

http://www.geoplan.ufl.edu/

http://www.microsoft.com

http://web.mit.edu/envit/www

http://www.oriconowireless.com

http://www.geo.ed.ac.uk/

http://london-luton.com/java/zine/97_06/Runtime_exec.html.

http://www.javaworld.com

http://www.delorie.com/djgpp/doc/pitfalls.html

Appendix I – tRIBS Input File

Input File Generated by tRIBS v2.0 through filling out the "New Input File" dialog window.

```
STARTDATE:
08/22/1996/00/00

RUNTIME:
24

TIMESTEP:
3.75

GWSTEP:
60.0

METSTEP:
60.0

RAININTRVL:
1

OPINTRVL:
1

NOTINTRVL:
1

INTSTORMMAX:
10

RAINSEARCH:
24

BASEFLOW:
0.01

VELOCITYCOEF:
0.5

KINEMVELCOEF:
1

VELOCITYRATIO:
35

FLOWEXP:
0

CHANNELROUGHNESS:
0.3

CHANNELWIDTH:
```

```
35


OPTMESHINPUT:
2

RAINSOURCE:
1

OPTEVAPOTRANS:
0

OPTINTERCEPT:
0

GFLUXOPTION:
0

METDATAOPTION:
0

CONVERTDATA:
0

OPTBEDROCK:
0

INPUTDATAFILE:
Output/voronoi/hill

INPUTTIME:
0

ARCINFOFILENAME:
Input/hill.net

POINTFILENAME:
Input/hill.points

SOILTABLENAME:
Input/hill.sdt

SOILMAPNAME:
Input/hill.soi

LANDTABLENAME:
Input/hill.ldt

LANDMAPNAME:
Input/hill.lan

GWATERFILE:
Input/hill.iwt

RAINFILE:
Rain/hill
```

```
RAINEXTENSION:
txt

DEPTHTOBEDROCK:
1

OUTFILENAME:
Output/voronoi/hill

OUTHYDROFILENAME:
Output/hyd/hill

RIBSHYDOUTPUT:
0

NODEOUTPUTLIST:
Input/pNodes.dat

HYDRONODELIST:
Input/hNodes.dat

OUTLETNODELIST:
Input/oNodes.dat
```

Appendix II – tRIBS makefile

```
#############################################################################
##
##                          tRIBS for Windows
##
##          TIN-based Real-time Integrated Basin Simulator
##                        Ralph M. Parsons Laboratory
##                   Massachusetts Institute of Technology
##
##
##
##   Makefile for tRIBS on Windows DJGPP GNU compiler
##
##   Run with make -f makeWIN
##
#############################################################################

OBJS = Objects/main.o \
Objects/tHydroModel.o \
Objects/meshElements.o \
Objects/mathutil.o \
Objects/tArray.o \
Objects/tMesh.o \
Objects/tMeshList.o \
Objects/tInputFile.o \
Objects/tListInputData.o \
Objects/tCNode.o \
Objects/tList.o \
Objects/tOutput.o \
Objects/tPtrList.o \
Objects/tRunTimer.o \
Objects/tSimul.o \
Objects/tControl.o \
Objects/globalFns.o \
Objects/tFlowNet.o \
Objects/tKinemat.o \
Objects/tFlowResults.o \
Objects/tRainGauge.o \
Objects/tRainfall.o \
Objects/tResample.o \
Objects/tInvariant.o \
Objects/tVariant.o \
Objects/tIntercept.o \
```

```
Objects/tHydroMet.o \
Objects/tHydroMetConvert.o \
Objects/tEvapoTrans.o \
Objects/tWaterBalance.o \
Objects/predicates.o

tribs: $(OBJS)
        gcc -g $(OBJS) -o tribs -lm


Objects/tHydroModel.o: tHydro/tHydroModel.cpp tHydro/tHydroModel.h
        gcc.exe -g -c tHydro/tHydroModel.cpp -o Objects/tHydroModel.o

Objects/meshElements.o: tMeshElements/meshElements.cpp tMeshElements/meshElements.h
        gcc.exe -g -c tMeshElements/meshElements.cpp -o Objects/meshElements.o

Objects/mathutil.o: Mathutil/mathutil.h Mathutil/mathutil.cpp
        gcc.exe -g -c Mathutil/mathutil.cpp -o Objects/mathutil.o

Objects/tArray.o: tArray/tArray.h tArray/tArray.cpp
        gcc.exe -g -c tArray/tArray.cpp -o Objects/tArray.o

Objects/tMesh.o: tMesh/tMesh.h tMesh/tMesh.cpp
        gcc.exe -g -c tMesh/tMesh.cpp -Objects/tMesh.o

Objects/tMeshList.o: tMeshList/tMeshList.h tMeshList/tMeshList.cpp
        gcc.exe -g -c tMeshList/tMeshList.cpp -o Objects/tMeshList.o

Objects/tInputFile.o: tInOut/tInputFile.h tInOut/tInputFile.cpp
        gcc.exe -g -c tInOut/tInputFile.cpp -o Objects/tInputFile.o

Objects/tListInputData.o: tListInputData/tListInputData.h tListInputData/tListInputData.cpp
        gcc.exe -g -c tListInputData/tListInputData.cpp -o Objects/tListInputData.o

Objects/tCNode.o: tCNode/tCNode.h tCNode/tCNode.cpp
        gcc.exe -g -c tCNode/tCNode.cpp -o Objects/tCNode.o

Objects/tList.o: tList/tList.h tList/tList.cpp
        gcc.exe -g -c tList/tList.cpp -o Objects/tList.o

Objects/tOutput.o: tInOut/tOutput.h tInOut/tOutput.cpp
        gcc.exe -g -c tInOut/tOutput.cpp -o Objects/tOutput.o

Objects/tPtrList.o: tPtrList/tPtrList.h tPtrList/tPtrList.cpp
        gcc.exe -g -c tPtrList/tPtrList.cpp -o Objects/tPtrList.o

Objects/tRunTimer.o: tSimulator/tRunTimer.h tSimulator/tRunTimer.cpp
```

```
	gcc.exe -g -c tSimulator/tRunTimer.cpp -o Objects/tRunTimer.o

Objects/tSimul.o: tSimulator/tSimul.h tSimulator/tSimul.cpp
	gcc.exe -g -c tSimulator/tSimul.cpp -o Objects/tSimul.o

Objects/tControl.o: tSimulator/tControl.h tSimulator/tControl.cpp
	gcc.exe -g -c tSimulator/tControl.cpp -o Objects/tControl.o

Objects/globalFns.o: Headers/globalFns.h Headers/globalFns.cpp
	gcc.exe -g -c Headers/globalFns.cpp -o Objects/globalFns.o

Objects/tFlowNet.o: tFlowNet/tFlowNet.h tFlowNet/tFlowNet.cpp
	gcc.exe -g -c tFlowNet/tFlowNet.cpp -o Objects/tFlowNet.o

Objects/tKinemat.o: tFlowNet/tKinemat.h tFlowNet/tKinemat.cpp
	gcc.exe -g -c tFlowNet/tKinemat.cpp -o Objects/tKinemat.o

Objects/tFlowResults.o: tFlowNet/tFlowResults.h tFlowNet/tFlowResults.cpp
	gcc.exe -g -c tFlowNet/tFlowResults.cpp -o Objects/tFlowResults.o

Objects/tRainGauge.o: tRasTin/tRainGauge.h tRasTin/tRainGauge.cpp
	gcc.exe -g -c tRasTin/tRainGauge.cpp -o Objects/tRainGauge.o

Objects/tRainfall.o: tRasTin/tRainfall.h tRasTin/tRainfall.cpp
	gcc.exe -g -c tRasTin/tRainfall.cpp -o Objects/tRainfall.o

Objects/tResample.o: tRasTin/tResample.h tRasTin/tResample.cpp
	gcc.exe -g -c tRasTin/tResample.cpp -o Objects/tResample.o

Objects/tInvariant.o: tRasTin/tInvariant.h tRasTin/tInvariant.cpp
	gcc.exe -g -c tRasTin/tInvariant.cpp -o Objects/tInvariant.o

Objects/tVariant.o: tRasTin/tVariant.h tRasTin/tVariant.cpp
	gcc.exe -g -c tRasTin/tVariant.cpp -o Objects/tVariant.o

Objects/tIntercept.o: tHydro/tIntercept.h tHydro/tIntercept.cpp
	gcc.exe -g -c tHydro/tIntercept.cpp -o Objects/tIntercept.o

Objects/tHydroMet.o: tHydro/tHydroMet.h tHydro/tHydroMet.cpp
	gcc.exe -g -c tHydro/tHydroMet.cpp -o Objects/tHydroMet.o

Objects/tHydroMetConvert.o: tHydro/tHydroMetConvert.h tHydro/tHydroMetConvert.cpp
	gcc.exe -g -c tHydro/tHydroMetConvert.cpp -o Objects/tHydroMetConvert.o

Objects/tEvapoTrans.o: tHydro/tEvapoTrans.h tHydro/tEvapoTrans.cpp
	gcc.exe -g -c tHydro/tEvapoTrans.cpp -o Objects/tEvapoTrans.o
```

```
Objects/tWaterBalance.o: tHydro/tWaterBalance.h tHydro/tWaterBalance.cpp
        gcc.exe -g -c tHydro/tWaterBalance.cpp -o Objects/tWaterBalance.o

Objects/predicates.o: Mathutil/predicates.h Mathutil/predicates.cpp
        gcc.exe -g -c Mathutil/predicates.cpp -o Objects/predicates.o

Objects/main.o: main.cpp
        gcc.exe -g -c main.cpp -o Objects/main.o

#clean:
#       del -f $(OBJS) tribs
#       del -f $(OBJS)
#all:
#       make -f makeWIN clean
#       make -f makeWIN
#all:tribs.exe
```

Appendix III – tRIBS v2.0 Source Code – GUI

```
//***********************************
//
//   BasinPlotter.java
//
//   Basin Data GUI Definition
//
//***********************************
package tRIBS;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import java.awt.geom.*;
import java.text.*;
import java.net.*;
import java.io.*;
import java.util.*;


public class BasinPlotter extends JPanel implements ImageObserver{
    private String elevURL, soilURL, landURL, smURL;
    private String soilTable, landTable;
    private String[] DescriptorsSoil, DescriptorsLand, Descriptors;
    private double[] Values;
    private int[] HeaderData, GridData, GridValues;
    private URL asciiFile, tableFile;
    private int mapFlag, colorFlag, gradeFlag;
    private int numCat, numProp;
    private int paintFlag = -1;
    private int simDay, numColorElev, numColorSoil, numColorLand,
numColorSm, currentNumColors;
    private int dayFlag = 0;
    private RasterGrid elevGrid, soilGrid, landGrid, smGrid,
currentGrid;
    private Image elevImg, soilImg, landImg, smImg, currentImg;
    private String basinName;
    private double basinLat, basinLong;
    private int rainOpt, drainOpt, evapOpt, interOpt;
    private int[] elevInterval, smInterval, currentInterval;
    private Color[] currentColor, elevColor, landColor, soilColor,
smColor;

    protected static final String on = "on";
    protected static final String off = "off";

    public static final int ELEVATION     = 0;
    public static final int SOIL_TEXTURE  = 1;
    public static final int LAND_USE      = 2;
    public static final int SOIL_MOISTURE = 3;

    public BasinPlotter(){
      super();
```

```
    setBackground(Color.white);
    currentImg = null;
    currentGrid = null;
  }

  public void setImage(int flag, String url, String table){
    mapFlag = flag;
    paintFlag = flag;

    if(mapFlag == ELEVATION){

        gradeFlag = 0;
        if(GridData!=null){
          GridData = null;
          HeaderData = null;
        }

        elevURL = url;
        readAsciiFile(elevURL);
        elevGrid = new RasterGrid(HeaderData[0],HeaderData[1]);
        elevGrid.setAllPixels(GridData);

        setCurrentGrid(elevGrid);

        elevGrid.fillElev(gradeFlag);
        elevColor = elevGrid.getColorMap();
        currentColor = elevColor;
        elevInterval = elevGrid.getIntervals();
        currentInterval = elevInterval;
          elevImg = elevGrid.toImage(this);
        numColorElev = 11;
        currentNumColors = numColorElev;
        currentImg = elevImg;
        repaint();
    }
    if(mapFlag == SOIL_TEXTURE){

        if(GridData!=null){
          GridData = null;
          HeaderData = null;
          numCat = 0;
          numProp = 0;
          Values = null;
          GridValues = null;
          Descriptors = null;
        }

        soilURL = url;
        soilTable = table;
        readAsciiFile(soilURL);
        readTableFile(soilTable);
        soilGrid = new RasterGrid(HeaderData[0],HeaderData[1]);
        soilGrid.setAllPixels(GridData);

        setCurrentGrid(soilGrid);

        colorFlag = 0;
```

```
          DescriptorsSoil = new String[Descriptors.length];
          DescriptorsSoil = Descriptors;
          soilGrid.fillMap(DescriptorsSoil.length, GridValues,
colorFlag);
          soilColor = soilGrid.getColorMap();
          currentColor = soilColor;
            soilImg = soilGrid.toImage(this);
          numColorSoil = DescriptorsSoil.length;
          currentNumColors = numColorSoil;
          currentImg = soilImg;
          repaint();
      }
      if(mapFlag == LAND_USE){

          if(GridData!=null){
            GridData = null;
            HeaderData = null;
            numCat = 0;
            numProp = 0;
            Values = null;
            GridValues = null;
            Descriptors = null;
          }

          landURL = url;
          landTable = table;
          readAsciiFile(landURL);
          readTableFile(landTable);
          landGrid = new RasterGrid(HeaderData[0],HeaderData[1]);
          landGrid.setAllPixels(GridData);

          setCurrentGrid(landGrid);

          colorFlag = 1;
          DescriptorsLand = new String[Descriptors.length];
          DescriptorsLand = Descriptors;
          landGrid.fillMap(DescriptorsLand.length, GridValues,
colorFlag);
          landColor = landGrid.getColorMap();
          currentColor = landColor;
          landImg = landGrid.toImage(this);
          numColorLand = DescriptorsLand.length;
          currentNumColors = numColorLand;
          currentImg = landImg;
          repaint();
      }
      if(mapFlag == SOIL_MOISTURE){

          gradeFlag = 1;
          if(GridData!=null){
            GridData = null;
            HeaderData = null;
          }

          smURL = url;
          readAsciiFile(smURL);
          smGrid = new RasterGrid(HeaderData[0],HeaderData[1]);
```

```
          smGrid.setAllPixels(GridData);

          setCurrentGrid(smGrid);

          smGrid.fillElev(gradeFlag);
          smColor = smGrid.getColorMap();
          currentColor = smColor;
          smInterval = smGrid.getIntervals();
          currentInterval = smInterval;
            smImg = smGrid.toImage(this);
          numColorSm = 11;
          currentNumColors = numColorSm;
          currentImg = smImg;
          repaint();
      }
    }

    public void readAsciiFile(String url){
        File asciiFile;
//      try{
          //asciiFile = new URL(url);
          asciiFile = new File(url);
  //    }
    //  catch(MalformedURLException u){
    //      System.out.println("URLException in AsciiFile: " +
u.getMessage());
    //  }

      try{
        FileReader buffReader = new FileReader(asciiFile);
//          BufferedReader buffReader = new BufferedReader(new
  //                    InputStreamReader(asciiFile.openStream()));
          StreamTokenizer st = new StreamTokenizer(buffReader);
          int headerLines = 6;

          try{
            HeaderData = new int[headerLines];

            for(int nL=0; nL<headerLines; nL++){
                st.nextToken();
                st.nextToken();
                HeaderData[nL] = (int)st.nval;
            }
            st.nextToken();

            int cols = HeaderData[0];
            int rows = HeaderData[1];
            int noData = -9999;
            GridData = new int[cols*rows];

            for(int nL=0; nL<(cols*rows);nL++){
                st.nextToken();
                if((int)st.nval == noData)
                  GridData[nL] = 0;
                else{
                  if(mapFlag!=SOIL_MOISTURE){
                      GridData[nL] = (int)st.nval;
```

```
                    }
                    else{
                        GridData[nL] = (int)(1000*st.nval);
                    }
                }
            }

            buffReader.close();
          }

            catch(IOException e){
            System.out.println("IOException: "+e.getMessage());
          }
      }
      catch(IOException f){
          System.out.println("IOException: " +f.getMessage());
      }
    }

    public void readTableFile(String url){
      File tableFile;
//      try{
        tableFile = new File(url);//URL(url);
  //    }
    //  catch(MalformedURLException u){
      //    System.out.println("URLException in tableFile: " +
u.getMessage());
      //}
      try{
        FileReader buffReader = new FileReader(tableFile);
//          BufferedReader buffReader = new BufferedReader(new
  //                    InputStreamReader(tableFile.openStream()));
        StreamTokenizer st = new StreamTokenizer(buffReader);

          try{
            st.nextToken();
            numCat = (int)st.nval;
            st.nextToken();
            numProp = (int)st.nval;

            Descriptors = new String[numCat];
            Values = new double[numCat*(numProp-1)];
            GridValues = new int[numCat];

            for(int nC=0;nC<numCat;nC++){
                for(int nP=0;nP<numProp;nP++){
                  st.nextToken();
                  if(nP == 0)
                      Descriptors[nC] = st.sval;
                  else
                      Values[nC*(numProp-1)+(nP-1)] = st.nval;
                }
            }
            int count = 0;
            for(int ct=0;ct<Values.length;ct=ct+(numProp-1)){
                GridValues[count] = (int)Values[ct];
                count++;
```

```
            }
            buffReader.close();
          }
          catch(IOException e){
            System.out.println("IOException: "+e.getMessage());
          }
      }
      catch(IOException f){
          System.out.println("IOException: " +f.getMessage());
      }
    }

    public void paintComponent(Graphics g){
      Graphics2D g2 = (Graphics2D) g;
      super.paintComponent(g);
      double recWidth = 30.0;
      double recHeight = 15.0;
      double startX = 500.0;
      double startY = 50.0;

      try {
      if(paintFlag!=-1){
          //g2.drawImage(currentImg,50,100,this);
          g.drawImage(currentImg,50,100,this);
//          System.out.println("paint basinName 1a");
          g.drawString(basinName, 25, 80);
//          System.out.println("paint basinName 1b");
          g.drawString("Outlet Location:
("+basinLat+","+basinLong+")",25,100);

          if(paintFlag == ELEVATION){
            g.drawString("Elevation (m)",(int)(startX),(int)(startY-
20));
          }
          if(paintFlag == SOIL_TEXTURE){
            g.drawString("Soil Texture",(int)(startX),(int)(startY-
20));
          }
          if(paintFlag == LAND_USE){
            g.drawString("Land Use",(int)(startX),(int)(startY-20));
          }
          if(paintFlag == SOIL_MOISTURE){
            g.drawString("Soil Moisture",(int)(startX),(int)(startY-
20));
          }

          double count = 0.0;
          double gap = 5.0;
          double hgap = 10;
          for(int ct = 0;ct<currentNumColors;ct++){
            g2.setPaint(currentColor[ct]);
            g2.fill(new
Rectangle2D.Double(startX,(startY+count),recWidth,recHeight));
            if(paintFlag == SOIL_TEXTURE || paintFlag == LAND_USE){
                g2.setPaint(Color.black);
                if(paintFlag == SOIL_TEXTURE){
//                    System.out.println("paint descSoil 1a");
```

```
g.drawString(DescriptorsSoil[ct],(int)(startX+recWidth+hgap),(int)(star
tY+count+10));
//                    System.out.println("paint descSoil 1b");
                  }
                  if(paintFlag == LAND_USE){
//                      System.out.println("paint descSoil 2a");

g.drawString(DescriptorsLand[ct],(int)(startX+recWidth+hgap),(int)(star
tY+count+10));
//                    System.out.println("paint descSoil 2b");
                  }
              }
              if(paintFlag == ELEVATION){
                  g2.setPaint(Color.black);
                  if(ct == 0){
                    g.drawString("<
"+currentInterval[ct],(int)(startX+recWidth+hgap),(int)(startY+count+10
));
                  }
                  else{
                    g.drawString(currentInterval[ct-1]+" <
"+currentInterval[ct],

(int)(startX+recWidth+hgap),(int)(startY+count+10));
                  }
              }
              if(paintFlag == SOIL_MOISTURE){
                  g2.setPaint(Color.black);
                  if(ct == 0){
                    g.drawString("<
"+((double)(currentInterval[ct]))/1000,

(int)(startX+recWidth+hgap),(int)(startY+count+10));
                  }
                  else{
                    g.drawString(((double)(currentInterval[ct-1]))/1000+"
< "+
                                  ((double)(currentInterval[ct]))/1000,
(int)(startX+recWidth+hgap),
                                   (int)(startY+count+10));
                  }
              }
              count=count+recHeight+gap;
            }

            if(dayFlag ==1){
              g.drawString("Simulation Day = "+simDay,25,120);
              if(rainOpt == 1){
                  g.drawString("Precipitation Modeling (on)",25,630);
              }
              else
                  g.drawString("Precipitation Modeling (off)",25,630);
              if(interOpt == 1){
                  g.drawString("Interception Modeling (on)",25,645);
              }
              else
```

```
                g.drawString("Interception Modeling (off)",25,645);
            if(evapOpt == 1){
                g.drawString("Evapotranspiration Modeling
(on)",25,660);
            }
            else
                g.drawString("Evapotranspiration Modeling
(off)",25,660);
            if(drainOpt == 1){
                g.drawString("Drainage Modeling (on)",25,675);
            }
            else
                g.drawString("Drainage Modeling (off)",25,675);
          }
      }
      } catch (Exception e) {
              System.out.println("paint exceptin - " +e.getMessage());
      }
    }

    public void resetCurrentImage() {
        setBackground(Color.white);
        currentImg = null;
        currentGrid = null;
    }

    public void setCurrentImage(int flag){

      if(flag==ELEVATION){
          currentImg = elevImg;
          currentColor = elevColor;
          currentInterval = elevInterval;
          paintFlag = flag;
          currentNumColors = numColorElev;
          dayFlag = 0;
          repaint();
      }
      if(flag==SOIL_TEXTURE){
          currentImg = soilImg;
          currentColor = soilColor;
          paintFlag = flag;
          currentNumColors = numColorSoil;
          dayFlag = 0;
          repaint();
      }
      if(flag==LAND_USE){
          currentImg = landImg;
          currentColor = landColor;
          currentNumColors = numColorLand;
          dayFlag = 0;
          paintFlag = flag;
          repaint();
      }
      if(flag==SOIL_TEXTURE){
          currentImg = smImg;
          currentColor = smColor;
          currentInterval = smInterval;
```

```
            currentNumColors = numColorSm;
            dayFlag = 0;
            paintFlag = flag;
            repaint();
        }
    }

    public void setAnimImage(Image img, Color[] smcolor, int[] smint){
        currentImg = img;
        currentColor = smcolor;
        currentInterval = smint;
        dayFlag = 1;
        repaint();
    }


    public boolean imageUpdate(Image image, int flags, int x, int y,
                               int width, int height){
        if((flags & ALLBITS)!=0){
            repaint();
            return false;
        }
        return true;
    }

    public void setCurrentGrid(RasterGrid currentgrid){
        currentGrid = new
RasterGrid(currentgrid.getX(),currentgrid.getY());
        currentGrid.setAllPixels(currentgrid.getPixelArray());
    }

    public RasterGrid getRasterGrid(){
        return currentGrid;
    }

    public double[] getValues(){
        return Values;
    }

    public int getNumCat(){
        return numCat;
    }

    public int getNumProp(){
        return numProp;
    }

    public void setBasinName(String name){
        basinName = name;
    }

    public void setDay(int day){
        simDay = day;
    }

    public void setBasinLat(double latitude){
        basinLat = latitude;
```

```
    }

    public void setBasinLong(double longitude){
      basinLong = longitude;
    }

    public void setRainOption(String rain){
      if(rain.compareTo(on)==0)
          rainOpt = 1;
      else
          rainOpt = 0;
    }

    public void setDrainOption(String drain){
      if(drain.compareTo(on)==0)
          drainOpt = 1;
      else
          drainOpt = 0;
    }

    public void setInterOption(String inter){
      if(inter.compareTo(on)==0)
          interOpt = 1;
      else
          interOpt = 0;
    }

    public void setEvapOption(String evap){
      if(evap.compareTo(on)==0)
          evapOpt = 1;
      else
          evapOpt = 0;
    }

}
```

```
//**********************************
//
//   DecimalField.java
//
//   Number Input Class Definition
//
//**********************************
package tRIBS;


import javax.swing.*;
import javax.swing.text.*;
import java.text.*;

public class DecimalField extends JTextField {
    private DecimalFormat format;

    public DecimalField(int columns, DecimalFormat f) {
        super(columns);
      format = f;
    }

    public double getValue() {
        double retVal = 0.0;

        try {
            retVal = format.parse(getText()).doubleValue();
        } catch (ParseException e) {
            System.err.println("getValue: could not parse: " +
getText());
        }
        return retVal;
    }
}
```

```
//************************************
//
//   ExecuteView.java
//
//   File Execution Class Definition
//
//************************************
package tRIBS;

import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.io.*;


public class ExecuteView extends JFrame implements ActionListener{

    private JButton okButton, cancelButton;
    private JToolBar toolBar;
    private JComboBox filenameComboBox;
    private JLabel filenameLabel;

    public ExecuteView(){
        super("Execute File");
        setSize(400, 200);
        setToolBar();
        setVisible(true);

        JLabel filenameLabel = new JLabel("Filename:");

filenameLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

        String[] filenameComboBoxLabel = {"",
            "Choose a file..."};
        filenameComboBox = new JComboBox(filenameComboBoxLabel);
        filenameComboBox.setEditable(true);
        filenameComboBox.addActionListener(this);
        filenameComboBox.setToolTipText("Type in a complete pathname or
choose a file from your computer");
        filenameComboBox.setActionCommand("INPUTDATAFILE");
        filenameComboBox.setAlignmentY(JComboBox.CENTER_ALIGNMENT);
//        filenameComboBox.setSize(100,2);

        JPanel inputPanel = new JPanel();
        inputPanel.add(filenameLabel,"West");
        inputPanel.add(filenameComboBox,"Center");
        inputPanel.setAlignmentY(JPanel.CENTER_ALIGNMENT);

        /*
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        inputPanel.setLayout(gridbag);
        addLabelTextRows(leftSideTitleLabels, leftSideLabels,
                         rightSideTitleLabels, rightSideLabels,
```

```
                        leftSideFields, rightSideFields,
                        gridbag, inputPanel);

        */
        this.getContentPane().add(inputPanel,"Center");
    //    this.getContentPane().add(filenameLabel,"West");
    }

    public void setToolBar(){
      toolBar = new JToolBar();

      okButton = new JButton("Execute File");
      okButton.setToolTipText("Executes specified File");
      okButton.addActionListener(this);
      toolBar.add(okButton);
      okButton.setEnabled(true);

      cancelButton = new JButton("Cancel");
      cancelButton.setToolTipText("Exit File Execution View");
      cancelButton.addActionListener(this);
      toolBar.add(cancelButton);
      cancelButton.setEnabled(true);

      toolBar.setFloatable(false);
      toolBar.setAlignmentX(JToolBar.CENTER_ALIGNMENT);
      this.getContentPane().add(toolBar,"South");
    }

    public void actionPerformed(ActionEvent evt){
      Object src = evt.getSource();

      if(src == okButton){
           try {
               //TODO - execute file
               dispose();
           } catch (Exception e) {
               JOptionPane.showMessageDialog(this,
                    "Run failed\n\n" + e.getMessage(),
                    "File Execution Error",
                    JOptionPane.ERROR_MESSAGE);
           }

      }
      else if(src == cancelButton){
          System.out.println("File execution cancelled.");
          dispose();
      }
      else if(evt.getActionCommand().equals("INPUTDATAFILE")) {
        if(filenameComboBox.getSelectedItem().equals("Choose a
file...")){
            FileDialog FileLoadDialog = new FileDialog(this,
                       "Load File ", FileDialog.LOAD);
            FileLoadDialog.show();

filenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
```

```
        }

    }
}
```

```
//***********************************
//
//   InfoView.java
//
//   Help Window Program
//
//***********************************
package tRIBS;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class InfoView extends JFrame implements ActionListener{
    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem menuItem1;
    private int infoFlag;


    public InfoView(int flag){
      super("Information View");
      setSize(550,375);
      setMenuBar();

      infoFlag = flag;
      if(infoFlag == -1){

          System.out.println("let the games begin...");
          dispose();
      }

      if(infoFlag == 0){
          JTextArea textArea = new JTextArea("");
          textArea.setLineWrap(true);
          textArea.setWrapStyleWord(true);
          JScrollPane areaScrollPane = new JScrollPane(textArea);
          areaScrollPane.setVerticalScrollBarPolicy(
                       JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
          areaScrollPane.setPreferredSize(new Dimension(500, 300));
          areaScrollPane.setBorder(
              BorderFactory.createCompoundBorder(
                BorderFactory.createCompoundBorder(
                              BorderFactory.createTitledBorder("About
javaWABS"),

BorderFactory.createEmptyBorder(5,5,5,5)),
                areaScrollPane.getBorder())));
          JPanel contentPane = new JPanel();
          contentPane.add(areaScrollPane);
          setContentPane(contentPane);

      }

      if(infoFlag == 1){
```

```
          JTextArea textArea = new JTextArea();
          textArea.setLineWrap(true);
          textArea.setWrapStyleWord(true);
          JScrollPane areaScrollPane = new JScrollPane(textArea);
          areaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
          areaScrollPane.setPreferredSize(new Dimension(500, 300));
          areaScrollPane.setBorder(
              BorderFactory.createCompoundBorder(
                BorderFactory.createCompoundBorder(

BorderFactory.createTitledBorder("Instructions for tRIBS"),

BorderFactory.createEmptyBorder(5,5,5,5)),
                     areaScrollPane.getBorder()));
          JPanel contentPane = new JPanel();
          contentPane.add(areaScrollPane);
          setContentPane(contentPane);
      }

      setVisible(true);
      addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
            System.out.println("Exiting Help");
            dispose();
          }
      });
    }

    public void setMenuBar(){
      menuBar = new JMenuBar();
      fileMenu = new JMenu("Window");
      fileMenu.setToolTipText("Window Exit");
      menuBar.add(fileMenu);
      menuItem1 = new JMenuItem("Exit");
      menuItem1.addActionListener(this);
      fileMenu.add(menuItem1);
      setJMenuBar(menuBar);
    }

    public void setFlag(int flag){
      infoFlag = flag;
      repaint();
    }

    public void actionPerformed(ActionEvent evt){
      Object src = evt.getSource();

      if(src == menuItem1){
           System.out.println("Exiting Information View");
           dispose();
      }
    }
}
```

```
//************************************
//
//   InputView.java
//
//   Parameter Input Class Definition
//
//************************************
package tRIBS;

import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.io.*;



public class InputView extends JFrame implements ActionListener{

    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem menuItem1;
    private JToolBar toolBar;
    private JButton okButton, cancelButton;
    private FileDialog inputDataFileLoadDialog;

    private JTextField startDateField, runtimeField, timeStepField,
            gwStepField, metStepField, rainIntrvlField, opIntrvlField,
            notIntrvlField,intStormMaxField,rainSearchField;
    private JTextField baseFlowField, velocityCoefField,
kinemVelCoefField,
            velocityRatioField, flowExpField, channelRoughnessField,
            channelWidthField;
    private JTextField optMeshInputField, rainSourceField,
optEvapoTransField,
            optInterceptField, gFluxOptionField, metDataOptionField,
            convertDataField, optBedrockField;
    private JTextField inputTimeField, rainExtensionField,
depthToBedrockField,
            ribsHydOutputField;
    private JComboBox inputDataFileComboBox, arcInfoFilenameComboBox,
            pointFilenameComboBox;
    private JComboBox soilTableNameComboBox, soilMapNameComboBox,
            landTablenameComboBox, landMapNameComboBox,
gWaterFileComboBox,
            rainFileComboBox, bedrockFileComboBox;
    private JComboBox hydrometStationsComboBox, hydrometGridComboBox,
            hydrometConvertComboBox, hydrometBaseNameComboBox,
            gaugeStationsComboBox, gaugeConvertComboBox,
gaugeBaseNameComboBox;
    private JComboBox outFilenameComboBox, outHydroFilenameComboBox,
            nodeOutputListComboBox, hydroNodeListComboBox,
            outletNodeListComboBox;
    private TRIBSFileIO fileGenerator;
```

```
//Time Variable Strings
private static final String STARTDATE = "STARTDATE";
private static final String RUNTIME = "RUNTIME";
private static final String TIMESTEP = "TIMESTEP";
private static final String GWSTEP = "GWSTEP";
private static final String METSTEP = "METSTEP";
private static final String RAININTRVL = "RAININTRVL";
private static final String OPINTRVL = "OPINTRVL";
private static final String NOTINTRVL = "NOTINTRVL";
private static final String INTSTORMMAX = "INTSTORMMAX";
private static final String RAINSEARCH = "RAINSEARCH";
//Routing Variable Strings
private static final String BASEFLOW = "BASEFLOW";
private static final String VELOCITYCOEF = "VELOCITYCOEF";
private static final String KINEMVELCOEF = "KINEMVELCOEF";
private static final String VELOCITYRATIO = "VELOCITYRATIO";
private static final String FLOWEXP = "FLOWEXP";
private static final String CHANNELROUGHNESS = "CHANNELROUGHNESS";
private static final String CHANNELWIDTH = "CHANNELWIDTH";
//Model Run Options Strings
private static final String OPTMESHINPUT = "OPTMESHINPUT";
private static final String RAINSOURCE = "RAINSOURCE";
private static final String OPTEVAPOTRANS = "OPTEVAPOTRANS";
private static final String OPTINTERCEPT = "OPTINTERCEPT";
private static final String GFLUXOPTION = "GFLUXOPTION";
private static final String METDATAOPTION = "METDATAOPTION";
private static final String CONVERTDATA = "CONVERTDATA";
private static final String OPTBEDROCK = "OPTBEDROCK";
//Mesh Generation Strings
private static final String INPUTDATAFILE = "INPUTDATAFILE";
private static final String INPUTTIME = "INPUTTIME";
private static final String ARCINFOFILENAME = "ARCINFOFILENAME";
private static final String POINTFILENAME = "POINTFILENAME";
//Resampling Grid Strings
private static final String SOILTABLENAME = "SOILTABLENAME";
private static final String SOILMAPNAME = "SOILMAPNAME";
private static final String LANDTABLENAME = "LANDTABLENAME";
private static final String LANDMAPNAME = "LANDMAPNAME";
private static final String GWATERFILE = "GWATERFILE";
private static final String RAINFILE = "RAINFILE";
private static final String RAINEXTENSION = "RAINEXTENSION";
private static final String DEPTHTOBEDROCK = "DEPTHTOBEDROCK";
private static final String BEDROCKFILE = "BEDROCKFILE";
//Meterological Data Strings
private static final String HYDROMETSTATIONS = "HYDROMETSTATIONS";
private static final String HYDROMETGRID = "HYDROMETGRID";
private static final String HYDROMETCONVERT = "HYDROMETCONVERT";
private static final String HYDROMETBASENAME = "HYDROMETBASENAME";
private static final String GAUGESTATIONS = "GAUGESTATIONS";
private static final String GAUGECONVERT = "GAUGECONVERT";
private static final String GAUGEBASENAME = "GAUGEBASENAME";
//Output Data Strings
private static final String OUTFILENAME = "OUTFILENAME";
private static final String OUTHYDROFILENAME = "OUTHYDROFILENAME";
private static final String RIBSHYDOUTPUT = "RIBSHYDOUTPUT";
private static final String NODEOUTPUTLIST = "NODEOUTPUTLIST";
private static final String HYDRONODELIST = "HYDRONODELIST";
```

```
    private static final String OUTLETNODELIST = "OUTLETNODELIST";

    public InputView(TRIBSFileIO tribsIO){
        super("New Input File");
        setSize(800, 600);
        setMenuBar();
        setToolBar();
        setVisible(true);

        fileGenerator = tribsIO;

        /***leftside layout items***/
        //time fields
        startDateField = new JTextField(fileGenerator.getStartDate());
//*1
        startDateField.setToolTipText("Starting time
(MM/DD/YYYY/HH/MM)");
        startDateField.setActionCommand(STARTDATE);
        startDateField.addActionListener(this);

        runtimeField = new JTextField(fileGenerator.getRuntime());//*2
        runtimeField.setToolTipText("Run Duration (hours)");
        runtimeField.setActionCommand(RUNTIME);
        runtimeField.addActionListener(this);

        timeStepField = new
JTextField(fileGenerator.getTimeStep());//*3
        timeStepField.setToolTipText("Unsaturated Zone computational
timestep (minutes)");
        timeStepField.setActionCommand(TIMESTEP);
        timeStepField.addActionListener(this);

        gwStepField = new JTextField(fileGenerator.getGWStep());//*4
        gwStepField.setToolTipText("Saturated Zone computational
timestep (minutes)");
        gwStepField.setActionCommand(GWSTEP);
        gwStepField.addActionListener(this);

        metStepField = new JTextField(fileGenerator.getMetStep());//*5
        metStepField.setToolTipText("Meterological data time step
(minutes)");
        metStepField.setActionCommand(METSTEP);
        metStepField.addActionListener(this);

        rainIntrvlField = new
JTextField(fileGenerator.getRainIntrvl());//*6
        rainIntrvlField.setToolTipText("Time interval in Rainfall input
(hours)");
        rainIntrvlField.setActionCommand(RAININTRVL);
        rainIntrvlField.addActionListener(this);

        opIntrvlField = new
JTextField(fileGenerator.getOpIntrvl());//*7
        opIntrvlField.setToolTipText("Output Interval (hours)");
        opIntrvlField.setActionCommand(OPINTRVL);
        opIntrvlField.addActionListener(this);
```

```
        notIntrvlField = new
JTextField(fileGenerator.getNotIntrvl());//*8
        notIntrvlField.setToolTipText("Notification interval (hours)");
        notIntrvlField.setActionCommand(NOTINTRVL);
        notIntrvlField.addActionListener(this);

        intStormMaxField = new
JTextField(fileGenerator.getIntStormMax());//*9
        intStormMaxField.setToolTipText("Interstorm Interval (hours)");
        intStormMaxField.setActionCommand(INTSTORMMAX);
        intStormMaxField.addActionListener(this);

        rainSearchField = new
JTextField(fileGenerator.getRainSearch());//*10
        rainSearchField.setToolTipText("Rainfall search interval
(hours)");
        rainSearchField.setActionCommand(RAINSEARCH);
        rainSearchField.addActionListener(this);

        JTextField[] timeFields = {
            startDateField,
            runtimeField,
            timeStepField,
            gwStepField,
            metStepField,
            rainIntrvlField,
            opIntrvlField,
            notIntrvlField,
            intStormMaxField,
            rainSearchField
        };

        //time Labels
        JLabel startDateLabel = new JLabel(STARTDATE + ":");//*1
        startDateLabel.setLabelFor(startDateField);
        startDateLabel.setToolTipText("Starting time
(MM/DD/YYYY/HH/MM)");

        JLabel runtimeLabel = new JLabel(RUNTIME + ":");//*2
        runtimeLabel.setLabelFor(runtimeField);
        runtimeLabel.setToolTipText("Run duration (hours)");

        JLabel timeStepLabel = new JLabel(TIMESTEP + ":");//*3
        timeStepLabel.setLabelFor(timeStepField);
        timeStepLabel.setToolTipText("Unsaturated Zone Time Step
(minutes)");

        JLabel gwStepLabel = new JLabel(GWSTEP + ":");//*4
        gwStepLabel.setLabelFor(gwStepField);
        gwStepLabel.setToolTipText("Saturated Zone Time Step
(minutes)");

        JLabel metStepLabel = new JLabel(METSTEP + ":");//*5
        metStepLabel.setLabelFor(metStepField);
        metStepLabel.setToolTipText("Meteorological data time step
(minutes)");
```

```
        JLabel rainIntrvlLabel = new JLabel(RAININTRVL + ":");//*6
        rainIntrvlLabel.setLabelFor(rainIntrvlField);
        rainIntrvlLabel.setToolTipText("Time interval in Rainfall input
(hours)");

        JLabel opIntrvlLabel = new JLabel(OPINTRVL + ":");//*7
        opIntrvlLabel.setLabelFor(opIntrvlField);
        opIntrvlLabel.setToolTipText("Output interval (hours)");

        JLabel notIntrvlLabel = new JLabel(NOTINTRVL + ":");//*8
        notIntrvlLabel.setLabelFor(notIntrvlField);
        notIntrvlLabel.setToolTipText("Notification interval (hours)");

        JLabel intStormMaxLabel = new JLabel(INTSTORMMAX + ":");//*9
        intStormMaxLabel.setLabelFor(intStormMaxField);
        intStormMaxLabel.setToolTipText("Interstorm interval (hours)");

        JLabel rainSearchLabel = new JLabel(RAINSEARCH + ":");//*10
        rainSearchLabel.setLabelFor(rainSearchField);
        rainSearchLabel.setToolTipText("Rainfall Search Interval
(hours)");

        JLabel[] timeLabels = {
                 startDateLabel,
                 runtimeLabel,
                 timeStepLabel,
                 gwStepLabel,
                 metStepLabel,
                 rainIntrvlLabel,
                 opIntrvlLabel,
                 notIntrvlLabel,
                 intStormMaxLabel,
                 rainSearchLabel
        };

        //routing fields
        baseFlowField = new JTextField(fileGenerator.getBaseFlow());
//*1
        baseFlowField.setToolTipText("Baseflow discharge (m3/s)");
        baseFlowField.setActionCommand(BASEFLOW);
        baseFlowField.addActionListener(this);

        velocityCoefField = new
JTextField(fileGenerator.getVelocityCoef());//*2
        velocityCoefField.setToolTipText("Discharge-velocity
coefficient");
        velocityCoefField.setActionCommand(VELOCITYCOEF);
        velocityCoefField.addActionListener(this);

        kinemVelCoefField = new
JTextField(fileGenerator.getKinemVelCoef());//*3
        kinemVelCoefField.setToolTipText("                ");
        kinemVelCoefField.setActionCommand(KINEMVELCOEF);
        kinemVelCoefField.addActionListener(this);

        velocityRatioField = new
JTextField(fileGenerator.getVelocityRatio());//*4
```

```
        velocityRatioField.setToolTipText("Stream to hillslope velocit
coefficient");
        velocityRatioField.setActionCommand(VELOCITYRATIO);
        velocityRatioField.addActionListener(this);

        flowExpField = new JTextField(fileGenerator.getFlowExp());//*5
        flowExpField.setToolTipText("Nonlinear coefficient");
        flowExpField.setActionCommand(FLOWEXP);
        flowExpField.addActionListener(this);

        channelRoughnessField = new
JTextField(fileGenerator.getChannelRoughness());//*6
        channelRoughnessField.setToolTipText("Uniform channel roughness
value");
        channelRoughnessField.setActionCommand(CHANNELROUGHNESS);
        channelRoughnessField.addActionListener(this);

        channelWidthField = new
JTextField(fileGenerator.getChannelWidth());//*7
        channelWidthField.setToolTipText("Uniform channel width
(meters)");
        channelWidthField.setActionCommand(CHANNELWIDTH);
        channelWidthField.addActionListener(this);

        JTextField[] routingFields = {
            baseFlowField,
            velocityCoefField,
            kinemVelCoefField,
            velocityRatioField,
            flowExpField,
            channelRoughnessField,
            channelWidthField
        };

        //routing labels
        JLabel baseFlowLabel = new JLabel(BASEFLOW + ":");//*1
        baseFlowLabel.setToolTipText("Baseflow discharge (m3/s)");

        JLabel velocityCoefLabel = new JLabel(VELOCITYCOEF + ":");//*2
        velocityCoefLabel.setToolTipText("Discharge-velocity
coefficient");

        JLabel kinemVelCoefLabel = new JLabel(KINEMVELCOEF + ":");//*3
        kinemVelCoefLabel.setToolTipText("          ");

        JLabel velocityRatioLabel = new JLabel(VELOCITYRATIO +
":");//*4
        velocityRatioLabel.setToolTipText("Stream to hillslope velocity
coefficient");

        JLabel flowExpLabel = new JLabel(FLOWEXP + ":");//*5
        flowExpLabel.setToolTipText("Nonlinear discharge coefficient");

        JLabel channelRoughnessLabel = new JLabel(CHANNELROUGHNESS +
":");//*6
        channelRoughnessLabel.setToolTipText("Uniform channel rouchness
value");
```

```
        JLabel channelWidthLabel = new JLabel(CHANNELWIDTH + ":");//*7
        channelWidthLabel.setToolTipText("Uniform channel width
(meters)");

        JLabel[] routingLabels = {
            baseFlowLabel,
            velocityCoefLabel,
            kinemVelCoefLabel,
            velocityRatioLabel,
            flowExpLabel,
            channelRoughnessLabel,
            channelWidthLabel
        };


        //model run fields
        optMeshInputField = new
JTextField(fileGenerator.getOptMeshInput());
        optMeshInputField.setToolTipText("Mesh Input data option");
        optMeshInputField.setActionCommand(OPTMESHINPUT);
        optMeshInputField.addActionListener(this);

        rainSourceField = new
JTextField(fileGenerator.getRainSource());     //2**
        rainSourceField.setToolTipText("Rainfall data source option");
        rainSourceField.setActionCommand(RAINSOURCE);
        rainSourceField.addActionListener(this);

        optEvapoTransField = new
JTextField(fileGenerator.getOptEvapoTrans());//**3
        optEvapoTransField.setToolTipText("Option for
evapoTranspiration scheme");
        optEvapoTransField.setActionCommand(OPTEVAPOTRANS);
        optEvapoTransField.addActionListener(this);

        optInterceptField = new
JTextField(fileGenerator.getOptIntercept());//**4
        optInterceptField.setToolTipText("Option for interception
scheme");
        optInterceptField.setActionCommand(OPTINTERCEPT);
        optInterceptField.addActionListener(this);

        gFluxOptionField = new
JTextField(fileGenerator.getGFluxOption());//**5
        gFluxOptionField.setToolTipText("Option for ground heat flux");
        gFluxOptionField.setActionCommand(GFLUXOPTION);
        gFluxOptionField.addActionListener(this);

        metDataOptionField = new
JTextField(fileGenerator.getMetDataOption());//**6
        metDataOptionField.setToolTipText("Option for meterological
data");
        metDataOptionField.setActionCommand(METDATAOPTION);
        metDataOptionField.addActionListener(this);
```

```
        convertDataField = new
JTextField(fileGenerator.getConvertData());//**7
        convertDataField.setToolTipText("Option to convert met data
format");
        convertDataField.setActionCommand(CONVERTDATA);
        convertDataField.addActionListener(this);

        optBedrockField = new
JTextField(fileGenerator.getOptBedrock());//**8
        optBedrockField.setToolTipText("Option for uniform or variable
depth");
        optBedrockField.setActionCommand(OPTBEDROCK);
        optBedrockField.addActionListener(this);

        JTextField[] modelRunFields = {
            optMeshInputField,
            rainSourceField,
            optEvapoTransField,
            optInterceptField,
            gFluxOptionField,
            metDataOptionField,
            convertDataField,
            optBedrockField
        };

        //model run labels
        JLabel optMeshInputLabel = new JLabel(OPTMESHINPUT + ":");//*1
        optMeshInputLabel.setToolTipText("Mesh Input data option");
        optMeshInputLabel.setLabelFor(optMeshInputField);

        JLabel rainSourceLabel = new JLabel(RAINSOURCE + ":");//*2
        rainSourceLabel.setToolTipText("Rainfall data source option");
        rainSourceLabel.setLabelFor(rainSourceField);

        JLabel optEvapoTransLabel = new JLabel(OPTEVAPOTRANS +
":");//*3
        optEvapoTransLabel.setToolTipText("Option for
evapoTranspiration scheme");
        optEvapoTransLabel.setLabelFor(optEvapoTransField);

        JLabel optInterceptLabel = new JLabel(OPTINTERCEPT + ":");//*4
        optInterceptLabel.setToolTipText("Option for interception
scheme");
        optInterceptLabel.setLabelFor(optInterceptField);

        JLabel gFluxOptionLabel = new JLabel(GFLUXOPTION + ":");//*5
        gFluxOptionLabel.setToolTipText("Option for ground heat flux");
        gFluxOptionLabel.setLabelFor(gFluxOptionField);

        JLabel metDataOptionLabel = new JLabel(METDATAOPTION +
":");//*6
        metDataOptionLabel.setToolTipText("Option for meterological
data");
        metDataOptionLabel.setLabelFor(metDataOptionField);

        JLabel convertDataLabel = new JLabel(CONVERTDATA + ":");//*7
```

```
        convertDataLabel.setToolTipText("Option to convert met data
format");
        convertDataLabel.setLabelFor(convertDataField);

        JLabel optBedrockLabel = new JLabel(OPTBEDROCK + ":");//*8
        optBedrockLabel.setToolTipText("Option for uniform or variable
depth");
        optBedrockLabel.setLabelFor(optBedrockField);

        JLabel[] modelRunLabels = {
                 optMeshInputLabel,
                 rainSourceLabel,
                 optEvapoTransLabel,
                 optInterceptLabel,
                 gFluxOptionLabel,
                 metDataOptionLabel,
             convertDataLabel,
                 optBedrockLabel
        };

        JLabel[] leftSideTitleLabels = {
                 new JLabel("Time Variables"),
                 new JLabel("Routing Variables"),
                 new JLabel("Model Run Options")
        };

        //make title labels bold
        for (int i=0; i<leftSideTitleLabels.length; i++) {
            leftSideTitleLabels[i].setFont(
              new Font(
                  leftSideTitleLabels[i].getFont().getName(),
                  Font.BOLD,
                  leftSideTitleLabels[i].getFont().getSize()));
        }

        JLabel[][] leftSideLabels = {
                 timeLabels,
                 routingLabels,
                 modelRunLabels
        };

        JComponent[][] leftSideFields = {
                        timeFields,
                        routingFields,
                        modelRunFields
        };

        /***right-side layout items***/
        //mesh generation fields
        String[] inputDataFileComboBoxLabel =
{fileGenerator.getInputDataFile(),
            "Choose a file..."};
        inputDataFileComboBox = new
JComboBox(inputDataFileComboBoxLabel);
        inputDataFileComboBox.setEditable(true);
        inputDataFileComboBox.addActionListener(this);
```

```
        inputDataFileComboBox.setToolTipText("Type in a complete
pathname or choose a file from your computer");
        inputDataFileComboBox.setActionCommand(INPUTDATAFILE);

        inputTimeField = new JTextField(fileGenerator.getInputTime());
        inputTimeField.setToolTipText("tMesh input time slice (Option
1)");
        inputTimeField.setActionCommand(INPUTTIME);
        inputTimeField.addActionListener(this);

        String[] arcInfoFilenameComboBoxLabel =
{fileGenerator.getArcInfoFilename(),
                  "Choose a file..."};
        arcInfoFilenameComboBox = new
JComboBox(arcInfoFilenameComboBoxLabel);
        arcInfoFilenameComboBox.setEditable(true);
        arcInfoFilenameComboBox.addActionListener(this);
        arcInfoFilenameComboBox.setToolTipText("Type in a complete
pathname or choose a file from your computer");
        arcInfoFilenameComboBox.setActionCommand(ARCINFOFILENAME);

        String[] pointFilenameComboBoxLabel =
{fileGenerator.getPointFilename(),
                  "Choose a file..."};
        pointFilenameComboBox = new
JComboBox(pointFilenameComboBoxLabel);
        pointFilenameComboBox.setEditable(true);
        pointFilenameComboBox.addActionListener(this);
        pointFilenameComboBox.setToolTipText("Type in a complete
pathname or choose a file from your computer");
        pointFilenameComboBox.setActionCommand(POINTFILENAME);

        JComponent[] meshGenerationFields = {
                    inputDataFileComboBox,
                inputTimeField,
                arcInfoFilenameComboBox,
                pointFilenameComboBox
        };

        //mesh generation labels
        JLabel inputDataFileLabel = new JLabel(INPUTDATAFILE + ":");
        JLabel inputTimeLabel = new JLabel(INPUTTIME + ":");
        JLabel arcInfoFilenameLabel = new JLabel(ARCINFOFILENAME +
":");
        JLabel pointFilenameLabel = new JLabel(POINTFILENAME + ":");
        JLabel[] meshGenerationLabels = {
               inputDataFileLabel,
               inputTimeLabel,
               arcInfoFilenameLabel,
               pointFilenameLabel
        };


        //resampling grid fields
        String[] soilTableNameComboBoxLabel =
{fileGenerator.getSoilTableName(),
             "Choose a file..."};
```

```
        soilTableNameComboBox = new
JComboBox(soilTableNameComboBoxLabel);
        soilTableNameComboBox.setEditable(true);
        soilTableNameComboBox.addActionListener(this);
        soilTableNameComboBox.setToolTipText("Type in a complete
pathname or choose a file from your computer");
        soilTableNameComboBox.setActionCommand(SOILTABLENAME);

        String[] soilMapNameComboBoxLabel =
{fileGenerator.getSoilMapName(),
             "Choose a file..."};
        soilMapNameComboBox = new JComboBox(soilMapNameComboBoxLabel);
        soilMapNameComboBox.setEditable(true);
        soilMapNameComboBox.addActionListener(this);
        soilMapNameComboBox.setToolTipText("Type in a complete pathname
or choose a file from your computer");
        soilMapNameComboBox.setActionCommand(SOILMAPNAME);

        String[] landTablenameComboBoxLabel =
{fileGenerator.getLandTablename(),
             "Choose a file..."};
        landTablenameComboBox = new
JComboBox(landTablenameComboBoxLabel);
        landTablenameComboBox.setEditable(true);
        landTablenameComboBox.addActionListener(this);
        landTablenameComboBox.setToolTipText("Type in a complete
pathname or choose a file from your computer");
        landTablenameComboBox.setActionCommand(LANDTABLENAME);

        String[] landMapNameComboBoxLabel =
{fileGenerator.getLandMapName(),
             "Choose a file..."};
        landMapNameComboBox = new JComboBox(landMapNameComboBoxLabel);
        landMapNameComboBox.setEditable(true);
        landMapNameComboBox.addActionListener(this);
        landMapNameComboBox.setToolTipText("Type in a complete pathname
or choose a file from your computer");
        landMapNameComboBox.setActionCommand(LANDMAPNAME);

        String[] gWaterFileComboBoxLabel =
{fileGenerator.getGWaterFile(),
            "Choose a file..."};
        gWaterFileComboBox = new JComboBox(gWaterFileComboBoxLabel);
        gWaterFileComboBox.setEditable(true);
        gWaterFileComboBox.addActionListener(this);
        gWaterFileComboBox.setToolTipText("Type in a complete pathname
or choose a file from your computer");
        gWaterFileComboBox.setActionCommand(GWATERFILE);

        String[] rainFileComboBoxLabel = {fileGenerator.getRainFile(),
            "Choose a file..."};
        rainFileComboBox = new JComboBox(rainFileComboBoxLabel);
        rainFileComboBox.setEditable(true);
        rainFileComboBox.addActionListener(this);
        rainFileComboBox.setToolTipText("Type in a complete pathname or
choose a file from your computer");
        rainFileComboBox.setActionCommand(RAINFILE);
```

```
        rainExtensionField = new
JTextField(fileGenerator.getRainExtension());
        rainExtensionField.setToolTipText("Extension for the radar
ASCII grid");
        rainExtensionField.setActionCommand(RAINEXTENSION);
        rainExtensionField.addActionListener(this);

        depthToBedrockField = new
JTextField(fileGenerator.getDepthToBedrock());
        depthToBedrockField.setToolTipText("Uniform depth to bedrock
(meters)");
        depthToBedrockField.setActionCommand(DEPTHTOBEDROCK);
        depthToBedrockField.addActionListener(this);

        String[] bedrockFileComboBoxLabel =
{fileGenerator.getBedrockFile(),
             "Choose a file..."};
        bedrockFileComboBox = new JComboBox(bedrockFileComboBoxLabel);
        bedrockFileComboBox.setEditable(true);
        bedrockFileComboBox.addActionListener(this);
        bedrockFileComboBox.setToolTipText("Type in a complete pathname
or choose a file from your computer");
        bedrockFileComboBox.setActionCommand(BEDROCKFILE);


        JComponent[] resamplingGridFields = {
                     soilTableNameComboBox,
                soilMapNameComboBox,
                landTablenameComboBox,
                landMapNameComboBox,
                gWaterFileComboBox,
                rainFileComboBox,
                rainExtensionField,
                depthToBedrockField,
                bedrockFileComboBox
        };

        //resampling grid labels
        JLabel soilTableNameLabel = new JLabel(SOILTABLENAME + ":");
        JLabel soilMapNameLabel = new JLabel(SOILMAPNAME + ":");
        JLabel landTablenameLabel = new JLabel(LANDTABLENAME + ":");
        JLabel landMapNameLabel = new JLabel(LANDMAPNAME + ":");
        JLabel gWaterFileLabel = new JLabel(GWATERFILE + ":");
        JLabel rainFileLabel = new JLabel(RAINFILE + ":");
        JLabel rainExtensionLabel = new JLabel(RAINEXTENSION + ":");
        JLabel depthToBedrockLabel = new JLabel(DEPTHTOBEDROCK + ":");
        JLabel bedrockFileLabel = new JLabel(BEDROCKFILE + ":");
        JLabel[] resamplingGridLabels = {
             soilTableNameLabel,
               soilMapNameLabel,
               landTablenameLabel,
               landMapNameLabel,
               gWaterFileLabel,
               rainFileLabel,
               rainExtensionLabel,
               depthToBedrockLabel,
```

```
                bedrockFileLabel
        };


        //meterological data fields
        String[] hydrometStationsComboBoxLabel =
{fileGenerator.getHydrometStations(),
             "Choose a file..."};
        hydrometStationsComboBox = new
JComboBox(hydrometStationsComboBoxLabel);
        hydrometStationsComboBox.setEditable(true);
        hydrometStationsComboBox.addActionListener(this);
        hydrometStationsComboBox.setActionCommand(HYDROMETSTATIONS);

        String[] hydrometGridComboBoxLabel =
{fileGenerator.getHydrometGrid(),
             "Choose a file..."};//HYDROMETGRID
        hydrometGridComboBox = new
JComboBox(hydrometGridComboBoxLabel);
        hydrometGridComboBox.setEditable(true);
        hydrometGridComboBox.addActionListener(this);
        hydrometGridComboBox.setActionCommand(HYDROMETGRID);

        String[] hydrometConvertComboBoxLabel =
{fileGenerator.getHydrometConvert(),
             "Choose a file..."};//HYDROMETCONVERT
        hydrometConvertComboBox = new
JComboBox(hydrometConvertComboBoxLabel);
        hydrometConvertComboBox.setEditable(true);
        hydrometConvertComboBox.addActionListener(this);
        hydrometConvertComboBox.setActionCommand(HYDROMETCONVERT);

        String[] hydrometBaseNameComboBoxLabel =
{fileGenerator.getHydrometBaseName(),
             "Choose a file..."};//HYDROMETBASENAME
        hydrometBaseNameComboBox = new
JComboBox(hydrometBaseNameComboBoxLabel);
        hydrometBaseNameComboBox.setEditable(true);
        hydrometBaseNameComboBox.addActionListener(this);
        hydrometBaseNameComboBox.setActionCommand(HYDROMETBASENAME);

        String[] gaugeStationsComboBoxLabel =
{fileGenerator.getGaugeStations(),
             "Choose a file..."};//GAUGESTATIONS
        gaugeStationsComboBox = new
JComboBox(gaugeStationsComboBoxLabel);
        gaugeStationsComboBox.setEditable(true);
        gaugeStationsComboBox.addActionListener(this);
        gaugeStationsComboBox.setActionCommand(GAUGESTATIONS);

        String[] gaugeConvertComboBoxLabel =
{fileGenerator.getGaugeConvert(),
             "Choose a file..."};//GAUGECONVERT
        gaugeConvertComboBox = new
JComboBox(gaugeConvertComboBoxLabel);
        gaugeConvertComboBox.setEditable(true);
        gaugeConvertComboBox.addActionListener(this);
```

```
        gaugeConvertComboBox.setActionCommand(GAUGECONVERT);

        String[] gaugeBaseNameComboBoxLabel =
{fileGenerator.getGaugeBaseName(),
             "Choose a file..."};//GAUGEBASENAME
        gaugeBaseNameComboBox = new
JComboBox(gaugeBaseNameComboBoxLabel);
        gaugeBaseNameComboBox.setEditable(true);
        gaugeBaseNameComboBox.addActionListener(this);
        gaugeBaseNameComboBox.setActionCommand(GAUGEBASENAME);


        JComponent[] metDataFields = {
                     hydrometStationsComboBox,
                hydrometGridComboBox,
                hydrometConvertComboBox,
                hydrometBaseNameComboBox,
                gaugeStationsComboBox,
                gaugeConvertComboBox,
                gaugeBaseNameComboBox
        };

        //meterological data labels
        JLabel hydrometStationLabel = new JLabel(HYDROMETSTATIONS +
":");
        JLabel hydrometGridLabel = new JLabel(HYDROMETGRID + ":");
        JLabel hydrometConvertLabel = new JLabel(HYDROMETCONVERT +
":");
        JLabel hydrometBaseNameLabel = new JLabel(HYDROMETBASENAME +
":");
        JLabel gaugeStationsLabel = new JLabel(GAUGESTATIONS + ":");
        JLabel gaugeConvertLabel = new JLabel(GAUGECONVERT + ":");
        JLabel gaugeBaseNameLabel = new JLabel(GAUGEBASENAME + ":");
        JLabel[] metDataLabels = {
             hydrometStationLabel,
             hydrometGridLabel,
             hydrometConvertLabel,
             hydrometBaseNameLabel,
             gaugeStationsLabel,
             gaugeConvertLabel,
             gaugeBaseNameLabel
        };



        //output data fields
        String[] outFilenameComboBoxLabel =
{fileGenerator.getOutFilename(),
             "Choose a file..."};
        outFilenameComboBox = new JComboBox(outFilenameComboBoxLabel);
        outFilenameComboBox.setEditable(true);
        outFilenameComboBox.addActionListener(this);
        outFilenameComboBox.setActionCommand(OUTFILENAME);

        String[] outHydroFilenameComboBoxLabel =
{fileGenerator.getOutHydroFilename(),
             "Choose a file..."};//OUTHYDROFILENAME
```

```
        outHydroFilenameComboBox = new
JComboBox(outHydroFilenameComboBoxLabel);
        outHydroFilenameComboBox.setEditable(true);
        outHydroFilenameComboBox.addActionListener(this);
        outHydroFilenameComboBox.setActionCommand(OUTHYDROFILENAME);

        ribsHydOutputField = new
JTextField(fileGenerator.getRIBSHydOutput());        //RIBSHYDOUTPUT
        ribsHydOutputField.setActionCommand(RIBSHYDOUTPUT);
        ribsHydOutputField.addActionListener(this);

        String[] nodeOutputListComboBoxLabel =
{fileGenerator.getNodeOutputList(),
             "Choose a file..."};//NODEOUTPUTLIST
        nodeOutputListComboBox = new
JComboBox(nodeOutputListComboBoxLabel);
        nodeOutputListComboBox.setEditable(true);
        nodeOutputListComboBox.addActionListener(this);
        nodeOutputListComboBox.setActionCommand(NODEOUTPUTLIST);

        String[] hydroNodeListComboBoxLabel =
{fileGenerator.getHydroNodeList(),
             "Choose a file..."};//HYDRONODELIST
        hydroNodeListComboBox = new
JComboBox(hydroNodeListComboBoxLabel);
        hydroNodeListComboBox.setEditable(true);
        hydroNodeListComboBox.addActionListener(this);
        hydroNodeListComboBox.setActionCommand(HYDRONODELIST);

        String[] outletNodeListComboBoxLabel =
{fileGenerator.getOutletNodeList(),
             "Choose a file..."};//OUTLETNODELIST
        outletNodeListComboBox = new
JComboBox(outletNodeListComboBoxLabel);
        outletNodeListComboBox.setEditable(true);
        outletNodeListComboBox.addActionListener(this);
        outletNodeListComboBox.setActionCommand(OUTLETNODELIST);

        JComponent[] outputDataFields = {
                     outFilenameComboBox,
                outHydroFilenameComboBox,
                ribsHydOutputField,
                nodeOutputListComboBox,
                hydroNodeListComboBox,
                outletNodeListComboBox
        };

        //output data labels
        JLabel outFilenameLabel = new JLabel(OUTFILENAME + ":");
        JLabel outHydroFilenameLabel = new JLabel(OUTHYDROFILENAME +
":");
        JLabel ribsHydOutputLabel = new JLabel(RIBSHYDOUTPUT + ":");
        JLabel nodeOutputListLabel = new JLabel(NODEOUTPUTLIST + ":");
        JLabel hydroNodeListLabel = new JLabel(HYDRONODELIST + ":");
        JLabel outletNodeListLabel = new JLabel(OUTLETNODELIST + ":");
        JLabel[] outputDataLabels = {
             outFilenameLabel,
```

```
            outHydroFilenameLabel,
            ribsHydOutputLabel,
            nodeOutputListLabel,
            hydroNodeListLabel,
            outletNodeListLabel
        };

        JLabel[] rightSideTitleLabels = {
                new JLabel("Mesh Generation Files"),
                new JLabel("Resampling Grid Files"),
                new JLabel("Meterological Data Files"),
                new JLabel("Output Data Files")
        };

        for (int i=0; i<rightSideTitleLabels.length; i++) {
            rightSideTitleLabels[i].setFont(
              new Font(
                  rightSideTitleLabels[i].getFont().getName(),
                  Font.BOLD,
                  rightSideTitleLabels[i].getFont().getSize()));
        }

        JLabel[][] rightSideLabels = {
                meshGenerationLabels,
                resamplingGridLabels,
                metDataLabels,
                outputDataLabels
        };

        JComponent[][] rightSideFields = {
                       meshGenerationFields,
                       resamplingGridFields,
                       metDataFields,
                       outputDataFields
        };

        JPanel inputPanel = new JPanel();
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        inputPanel.setLayout(gridbag);
        addLabelTextRows(leftSideTitleLabels, leftSideLabels,
                         rightSideTitleLabels, rightSideLabels,
                         leftSideFields, rightSideFields,
                         gridbag, inputPanel);

        JLabel actionLabel = new JLabel("Enter values and press Return
key in each field.");
        actionLabel.setForeground(Color.red);
        actionLabel.setHorizontalAlignment(JLabel.CENTER);

actionLabel.setBorder(BorderFactory.createEmptyBorder(5,5,0,5));

        this.getContentPane().add(actionLabel, "North");
        inputPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

        JScrollPane inputScrollPane = new JScrollPane(inputPanel);
        this.getContentPane().add(inputScrollPane,"Center");
```

```
    }


    public void addLabelTextRows( JLabel[] leftTitles, JLabel[][]
leftLabels,
            JLabel[] rightTitles, JLabel[][] rightLabels,
            JComponent[][] leftFields, JComponent[][] rightFields,
            GridBagLayout gridbag, Container container)
    {
        int absXPos, absYPos;
        GridBagConstraints c = new GridBagConstraints();

        int numRightLabelSections = rightTitles.length;
        int numLeftLabelSections = leftTitles.length;

        /***setup left-side of panel***/
        //setup left-side labels
        absXPos = 0;
        absYPos = 0;
        c.insets = new Insets(0,10,0,10);
        for (int i = 0; i < numLeftLabelSections; i++) {

            c.fill = GridBagConstraints.HORIZONTAL;
            c.anchor = GridBagConstraints.CENTER;
            c.weightx = 0.0;
            c.gridx = absXPos;

            c.gridy = absYPos;

            gridbag.setConstraints(leftTitles[i], c);
            container.add(leftTitles[i]);

            int numLabels = leftLabels[i].length;
            c.anchor = GridBagConstraints.WEST;


            for (int j=0; j < numLabels; j++) {
                absYPos++;

                c.gridx = absXPos;
                c.gridy = absYPos;
                c.weightx=0.0;
                gridbag.setConstraints(leftLabels[i][j],c);
                container.add(leftLabels[i][j]);

                c.gridx = absXPos + 1;
                c.weightx=0.5;                          //want text
fields to resize
                gridbag.setConstraints(leftFields[i][j],c);
                container.add(leftFields[i][j]);
            }
            absYPos += 2;
        }
        absXPos += 2;

        /***setup right-side of panel***/
        //setup right-side labels
```

```
        absYPos = 0;
        c.insets = new Insets(0,10,0,10);
        for (int i = 0; i < numRightLabelSections; i++) {
            c.fill = GridBagConstraints.HORIZONTAL;
            c.anchor = GridBagConstraints.CENTER;
            c.weightx = 0.0;
            c.gridx = absXPos;
            c.gridy = absYPos;

            gridbag.setConstraints(rightTitles[i], c);
            container.add(rightTitles[i]);

            int numLabels = rightLabels[i].length;
            c.anchor = GridBagConstraints.WEST;

            for (int j=0; j < numLabels; j++) {
                absYPos++;
                c.gridx = absXPos;
                c.gridy = absYPos;
                c.weightx=0.0;
                gridbag.setConstraints(rightLabels[i][j],c);
                container.add(rightLabels[i][j]);

                c.gridx = absXPos + 1;
                c.weightx=0.5;    //want text fields to resize
                gridbag.setConstraints(rightFields[i][j],c);
                container.add(rightFields[i][j]);
            }
            absYPos += 2;
        }

    }

    public void setMenuBar(){
      menuBar = new JMenuBar();
      fileMenu = new JMenu("Window");
      fileMenu.setToolTipText("Window Exit");
      menuBar.add(fileMenu);
      menuItem1 = new JMenuItem("Exit");
      menuItem1.addActionListener(this);
      fileMenu.add(menuItem1);
      setJMenuBar(menuBar);
    }

    public void setToolBar(){
      toolBar = new JToolBar();

      okButton = new JButton("Create .in File");
      okButton.setToolTipText("Writes inputted variables to a .in
File");
      okButton.addActionListener(this);
      toolBar.add(okButton);
      okButton.setEnabled(true);

      cancelButton = new JButton("Cancel");
      cancelButton.setToolTipText("Exit New Input View");
      cancelButton.addActionListener(this);
```

```
    toolBar.add(cancelButton);
    cancelButton.setEnabled(true);

    toolBar.setFloatable(false);
    toolBar.setAlignmentX(JToolBar.CENTER_ALIGNMENT);
    this.getContentPane().add(toolBar,"South");
  }


  public void actionPerformed(ActionEvent evt){
    Object src = evt.getSource();

    if(src == menuItem1){
        System.out.println("Exiting Input Data View");
        dispose();
    }
    else if(src == okButton){ //CHANGED
         boolean isFileStored = false;

         System.out.print("file saving...");

         FileDialog fileSaveDialog = new FileDialog(this,
                     "Save File ", FileDialog.SAVE);
           fileSaveDialog.show();

         try {
           String filename= fileSaveDialog.getDirectory()
                      + fileSaveDialog.getFile();
           if (fileSaveDialog.getFile() == null) {
              //user cancelled save
              isFileStored = false;
           } else {
              fileGenerator.exportFile(filename);
              System.out.println("Input Parameters Stored!");
              isFileStored = true;
              dispose();
           }
         } catch (Exception e) {
          JOptionPane.showMessageDialog(this,
              "Save failed\n\n" + e.getMessage(),
              "File Save Error",
              JOptionPane.ERROR_MESSAGE);
         }

    }
    else if(src == cancelButton){
        System.out.println("No Input Parameters Stored!");
        dispose();
    }


      else if(evt.getActionCommand().equals(STARTDATE)) {
         fileGenerator.setStartDate(startDateField.getText());
      }
      else if(evt.getActionCommand().equals(RUNTIME)) {
        fileGenerator.setRuntime(runtimeField.getText());
      }
```

```
        else if(evt.getActionCommand().equals(TIMESTEP)) {
          fileGenerator.setTimeStep(timeStepField.getText());
        }
        else if(evt.getActionCommand().equals(GWSTEP)) {
          fileGenerator.setGWStep(gwStepField.getText());
        }
        else if(evt.getActionCommand().equals(METSTEP)) {
          fileGenerator.setMetStep(metStepField.getText());
        }
        else if(evt.getActionCommand().equals(RAININTRVL)) {
          fileGenerator.setRainIntrvl(rainIntrvlField.getText());
        }
        else if(evt.getActionCommand().equals(OPINTRVL)) {
          fileGenerator.setOpIntrvl(opIntrvlField.getText());
        }
        else if(evt.getActionCommand().equals(NOTINTRVL)) {
          fileGenerator.setNotIntrvl(notIntrvlField.getText());
        }
        else if(evt.getActionCommand().equals(INTSTORMMAX)) {
          fileGenerator.setIntStormMax(intStormMaxField.getText());
        }
        else if(evt.getActionCommand().equals(RAINSEARCH)) {
          fileGenerator.setRainSearch(rainSearchField.getText());
        }
        else if(evt.getActionCommand().equals(BASEFLOW)) {
          fileGenerator.setBaseFlow(baseFlowField.getText());
        }
        else if(evt.getActionCommand().equals(VELOCITYCOEF)) {
          fileGenerator.setVelocityCoef(velocityCoefField.getText());
        }
        else if(evt.getActionCommand().equals(KINEMVELCOEF)) {
          fileGenerator.setKinemVelCoef(kinemVelCoefField.getText());
        }
        else if(evt.getActionCommand().equals(VELOCITYRATIO)) {
          fileGenerator.setVelocityRatio(velocityRatioField.getText());
        }
        else if(evt.getActionCommand().equals(FLOWEXP)) {
          fileGenerator.setFlowExp(flowExpField.getText());
        }
        else if(evt.getActionCommand().equals(CHANNELROUGHNESS)) {

fileGenerator.setChannelRoughness(channelRoughnessField.getText());
        }
        else if(evt.getActionCommand().equals(CHANNELWIDTH)) {
          fileGenerator.setChannelWidth(channelWidthField.getText());
        }
        else if(evt.getActionCommand().equals(OPTMESHINPUT)) {
          fileGenerator.setOptMeshInput(optMeshInputField.getText());
        }
        else if(evt.getActionCommand().equals(RAINSOURCE)) {
          fileGenerator.setRainSource(rainSourceField.getText());
        }
        else if(evt.getActionCommand().equals(OPTEVAPOTRANS)) {
          fileGenerator.setOptEvapoTrans(optEvapoTransField.getText());
        }
        else if(evt.getActionCommand().equals(OPTINTERCEPT)) {
          fileGenerator.setOptIntercept(optInterceptField.getText());
```

```
        }
        else if(evt.getActionCommand().equals(GFLUXOPTION)) {
          fileGenerator.setGFluxOption(gFluxOptionField.getText());
        }
        else if(evt.getActionCommand().equals(METDATAOPTION)) {
          fileGenerator.setMetDataOption(metDataOptionField.getText());
        }
        else if(evt.getActionCommand().equals(CONVERTDATA)) {
          fileGenerator.setConvertData(convertDataField.getText());
        }
        else if(evt.getActionCommand().equals(OPTBEDROCK)) {
          fileGenerator.setOptBedrock(optBedrockField.getText());
        }

        else if(evt.getActionCommand().equals(INPUTDATAFILE)) {
        if(inputDataFileComboBox.getSelectedItem().equals("Choose a
file...")){
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

inputDataFileComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setInputDataFile((String)
inputDataFileComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(INPUTTIME)) {
            fileGenerator.setInputTime(inputTimeField.getText());
        }

        else if(evt.getActionCommand().equals(ARCINFOFILENAME)) {
           if(arcInfoFilenameComboBox.getSelectedItem().equals("Choose
a file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
             "Load File", FileDialog.LOAD);
             FileLoadDialog.show();

arcInfoFilenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
           }
           fileGenerator.setArcInfoFilename((String)
arcInfoFilenameComboBox.getSelectedItem());
        }

       else if(evt.getActionCommand().equals(POINTFILENAME)) {
          if(pointFilenameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
             "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

pointFilenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
             }
```

```
            fileGenerator.setPointFilename((String)
pointFilenameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(SOILTABLENAME)) {
           //if user clicks on combobox, open file dialog
           if(soilTableNameComboBox.getSelectedItem().equals("Choose a
file...")) {
               FileDialog FileLoadDialog = new FileDialog(this,
                 "Load File ", FileDialog.LOAD);
                 FileLoadDialog.show();

soilTableNameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                   + FileLoadDialog.getFile());
           }
           //otherwise take what the user typed in
           fileGenerator.setSoilTableName((String)
soilTableNameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(SOILMAPNAME)) {
        if(soilMapNameComboBox.getSelectedItem().equals("Choose a
file...")) {

             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

soilMapNameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setSoilMapName((String)
soilMapNameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(LANDTABLENAME)) {
        if(landTablenameComboBox.getSelectedItem().equals("Choose a
file...")) {

             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

landTablenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setLandTablename((String)
landTablenameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(LANDMAPNAME)) {
        if(landMapNameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
             "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();
```

```
landMapNameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setLandMapName((String)
landMapNameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(GWATERFILE)) {
        if(gWaterFileComboBox.getSelectedItem().equals("Choose a
file...")) {
           FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

gWaterFileComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setGWaterFile((String)
gWaterFileComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(RAINFILE)) {
        if(rainFileComboBox.getSelectedItem().equals("Choose a
file...")) {
         FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
           FileLoadDialog.show();

rainFileComboBox.setSelectedItem(FileLoadDialog.getDirectory()
               + FileLoadDialog.getFile());
        }
        fileGenerator.setRainFile((String)
rainFileComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(RAINEXTENSION)) {

fileGenerator.setRainExtension(rainExtensionField.getText());
        }

        else if(evt.getActionCommand().equals(DEPTHTOBEDROCK)) {

fileGenerator.setDepthToBedrock(depthToBedrockField.getText());
        }

        else if(evt.getActionCommand().equals(BEDROCKFILE)) {
        if(bedrockFileComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

bedrockFileComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
```

```
        fileGenerator.setBedrockFile((String)
bedrockFileComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(HYDROMETSTATIONS)) {
        if(hydrometStationsComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

hydrometStationsComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setHydrometStations((String)
hydrometStationsComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(HYDROMETGRID)) {
        if(hydrometGridComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

hydrometGridComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setHydrometGrid((String)
hydrometGridComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(HYDROMETCONVERT)) {
        if(hydrometConvertComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

hydrometConvertComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setHydrometConvert((String)
hydrometConvertComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(HYDROMETBASENAME)) {
        if(hydrometBaseNameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

hydrometBaseNameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
```

```
        fileGenerator.setHydrometBaseName((String)
hydrometBaseNameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(GAUGESTATIONS)) {
        if(gaugeStationsComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

gaugeStationsComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setGaugeStations((String)
gaugeStationsComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(GAUGECONVERT)) {
        if(gaugeConvertComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

gaugeConvertComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setGaugeConvert((String)
gaugeConvertComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(GAUGEBASENAME)) {
        if(gaugeBaseNameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

gaugeBaseNameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setGaugeBaseName((String)
gaugeBaseNameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(OUTFILENAME)) {
        if(outFilenameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

outFilenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
```

```
        fileGenerator.setOutFilename((String)
outFilenameComboBox.getSelectedItem());
      }

        else if(evt.getActionCommand().equals(OUTHYDROFILENAME)) {
        if(outHydroFilenameComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

outHydroFilenameComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setOutHydroFilename((String)
outHydroFilenameComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(RIBSHYDOUTPUT)) {

fileGenerator.setRIBSHydOutput(ribsHydOutputField.getText());
        }


        else if(evt.getActionCommand().equals(NODEOUTPUTLIST)) {
        if(nodeOutputListComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

nodeOutputListComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setNodeOutputList((String)
nodeOutputListComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(HYDRONODELIST)) {
        if(hydroNodeListComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
             FileLoadDialog.show();

hydroNodeListComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                + FileLoadDialog.getFile());
        }
        fileGenerator.setHydroNodeList((String)
hydroNodeListComboBox.getSelectedItem());
        }

        else if(evt.getActionCommand().equals(OUTLETNODELIST)) {
        if(outletNodeListComboBox.getSelectedItem().equals("Choose a
file...")) {
             FileDialog FileLoadDialog = new FileDialog(this,
                "Load File ", FileDialog.LOAD);
```

```
                FileLoadDialog.show();

outletNodeListComboBox.setSelectedItem(FileLoadDialog.getDirectory()
                    + FileLoadDialog.getFile());
            }
            fileGenerator.setOutletNodeList((String)
outletNodeListComboBox.getSelectedItem());
            }
        }
}
```

```
//***********************************
//
//   PictureView.java
//
//   Picture View Class Definition
//
//***********************************
package tRIBS;

import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PictureView extends JFrame implements ActionListener{
    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem menuItem1;
    private JPanel PicPanel, PicPanel2;
    private ImageIcon metstation, location;

    public PictureView(int flag){
      super("Picture View");
      setMenuBar();

      if(flag == 0){
          setSize(300,235);
          PicPanel = new JPanel(){
            public void paintComponent(Graphics g){
                super.paintComponent(g);
                ImageIcon metstation = new ImageIcon("metstation.gif");
                metstation.paintIcon(this,g,0,0);
            }
          };
          setContentPane(PicPanel);
      }
      if(flag == 1){
          setSize(620,778);
          PicPanel2 = new JPanel(){
            public void paintComponent(Graphics g){
                super.paintComponent(g);
                ImageIcon location = new ImageIcon("location.jpg");
                location.paintIcon(this,g,0,0);
            }
          };
          setContentPane(PicPanel2);
      }
      setVisible(true);
    }

    public void setMenuBar(){
      menuBar = new JMenuBar();
      fileMenu = new JMenu("Window");
      fileMenu.setToolTipText("Window Exit");
      menuBar.add(fileMenu);
      menuItem1 = new JMenuItem("Exit");
```

```
        menuItem1.addActionListener(this);
        fileMenu.add(menuItem1);
        setJMenuBar(menuBar);
    }


    public void actionPerformed(ActionEvent evt){
        Object src = evt.getSource();

        if(src == menuItem1){
            System.out.println("Exiting Hydrometeorological Data View");
            dispose();
        }
    }
}
```

```
//************************************
//
//   RasterGrid.java
//
//   Raster Grid Data Type Definition
//
//************************************
package tRIBS;

import java.awt.*;
import java.awt.image.*;

public class RasterGrid{
    private int width, height;
    private int[] pixel;
    private int[] color;
    private Color[] colorMap;
    private int[] InterVals;

    public RasterGrid(int w, int h) {
      width = w;
      height = h;
      pixel = new int[w*h];
      color = new int[w*h];
    }

    public int getX(){
      return width;
    }

    public int getY(){
      return height;
    }

    public final int[] getPixelArray(){
      return pixel;
    }

    public final int size(){
      return pixel.length;
    }

    public int getPixel(int x, int y) {
      return pixel[y*width+x];
    }

    public int getMax(int[] pix){
      int s = size();
      int max = pix[0];
      for(int i = 0; i<s;i++){
          if(pix[i]>max)
            max = pix[i];
      }
      return max;
    }
```

```
public int getElevMax(){
  int s = size();
  int max = pixel[0];
  for(int i = 0; i<s;i++){
      if(pixel[i]>max)
        max = pixel[i];
  }
  return max;
}

public int getMin(int[] pix){
  int s = size();
  int min = 10000;
  for(int i = 0; i<s;i++){
      if(pix[i]<min && pix[i]!=0)
        min = pix[i];
  }
  return min;
}

public int[] setInterval(int max, int min, int num){
  int diff = max-min;
  int div = diff/num;
  int[] intervs = new int[11];

  for(int i = 0;i<num;i++){
      intervs[i] = min+div;
      min = min+div;
  }
  return intervs;
}

public void setPixel(int pix, int x, int y) {
  pixel[y*width+x] = pix;
}

public void setAllPixels(int[] pix){
  pixel = pix;
}

public void fillElev(int flag){
  Color c, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10;
  int s = size();
  int max = getMax(pixel);
  int min = getMin(pixel);
  int num = 11;
  int[] intervs = setInterval(max,min,num);

  InterVals = new int[intervs.length];
  for(int ct=0;ct<intervs.length;ct++){
      InterVals[ct] = intervs[ct];
  }

  if(flag == 0){
      c = new Color(1,255,10,100);
      c1 = new Color(1,225,20,100);
      c2 = new Color(1,195,30,100);
```

```
        c3 = new Color(1,165,40,100);
        c4 = new Color(1,135,50,100);
        c5 = new Color(1,105,60,100);
        c6 = new Color(1,75,70,100);
        c7 = new Color(1,45,80,100);
        c8 = new Color(1,15,90,100);
        c9 = new Color(1,0,100,100);
        c10 = new Color(1,0,110,100);
    }
    else{
        c = new Color(10,250,0,100);
        c1 = new Color(10,220,10,100);
        c2 = new Color(10,190,40,100);
        c3 = new Color(10,160,70,100);
        c4 = new Color(10,130,100,100);
        c5 = new Color(10,100,130,100);
        c6 = new Color(10,70,160,100);
        c7 = new Color(10,40,190,100);
        c8 = new Color(10,20,220,100);
        c9 = new Color(10,10,240,100);
        c10 = new Color(10,0,255,100);
    }

    colorMap = new Color[num];
    colorMap[0] = c;
    colorMap[1] = c1;
    colorMap[2] = c2;
    colorMap[3] = c3;
    colorMap[4] = c4;
    colorMap[5] = c5;
    colorMap[6] = c6;
    colorMap[7] = c7;
    colorMap[8] = c8;
    colorMap[9] = c9;
    colorMap[10] = c10;

    for(int i = 0; i < s; i++){
        if(pixel[i]!=0){
          if(pixel[i]>=0&&pixel[i]<intervs[0]){
              int rgb = c.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[0]&&pixel[i]<intervs[1]){
              int rgb = c1.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[1]&&pixel[i]<intervs[2]){
              int rgb = c2.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[2]&&pixel[i]<intervs[3]){
              int rgb = c3.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[3]&&pixel[i]<intervs[4]){
              int rgb = c4.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[4]&&pixel[i]<intervs[5]){
              int rgb = c5.getRGB();
              color[i] = rgb;}
          if(pixel[i]>=intervs[5]&pixel[i]<intervs[6]){
```

```
                int rgb = c6.getRGB();
                color[i] = rgb;}
            if(pixel[i]>=intervs[6]&&pixel[i]<intervs[7]){
                int rgb = c7.getRGB();
                color[i] = rgb;}
            if(pixel[i]>=intervs[7]&&pixel[i]<intervs[8]){
                int rgb = c8.getRGB();
                color[i] = rgb;}
            if(pixel[i]>=intervs[8]&&pixel[i]<intervs[9]){
                int rgb = c9.getRGB();
                color[i] = rgb;}
            if(pixel[i]>=intervs[9]){
                int rgb = c10.getRGB();
                color[i] = rgb;
            }

        }
    }
}


public void fillMap(int length, int[] values, int flag){
  Color[] colors = new Color[length];
  int r,g,b,a;
  int s = size();

  if(flag==0){
      r=250;
      g=0;
      b=0;
      a=250;}
  else{
      r=0;
      g=250;
      b=0;
      a=250;}

  colorMap = new Color[length];
  for(int ct = 0; ct<length;ct++){
      if(flag == 0)
        colors[ct] = new Color((r-ct*20),g,(b+ct*20),(a-ct*10));
      else
        colors[ct] = new Color((r+ct*10),(g-ct*10),(b+ct*10),(a-
ct*10));
      colorMap[ct] = colors[ct];
  }

  for(int i = 0; i < s; i++){
       if(pixel[i]!=0){
         for(int ct = 0;ct<length;ct++){
             if(pixel[i] == values[ct]){
               int rgb = colors[ct].getRGB();
               color[i] = rgb;
             }
         }
       }
  }
```

```
    }

    public Color[] getColorMap(){
      return colorMap;
    }

    public int[] getIntervals(){
      return InterVals;
    }

    public final Image toImage(Component root) {
      return root.createImage(new MemoryImageSource(width, height,
                color, 0, width));
    }


}
```

```
//***********************************
//
//   ReadandPlotData.java
//
//   Weather Input Class Definition
//
//***********************************
package tRIBS;

import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.awt.geom.*;


public class ReadandPlotData extends JFrame implements ActionListener{
    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem menuItem1;
    private JPanel TempPanel, PrecipPanel, OtherPanel;
    private double[] Temperature, DTemperature, Precipitation, DayT,
DayDT, DayP;
    private double[] DayS, Cover, DayW, Wind;
    private double[] DayScaled, TempScaled, PrecipScaled, OtherScaled;
    private URL tempData, precipData, otherData;
    private int numDays, dataFlag;
    private String rainURL, tempURL, otherURL;
    private int Xextent = 500;
    private int Yextent = 300;


    public ReadandPlotData(int flag, String url){
      super("Hydrometeorological Data View");
      setSize(Xextent+100,Yextent+100);
      setMenuBar();
      Container cf = getContentPane();
      cf.setLayout(new BorderLayout());

      dataFlag = flag;

      if(dataFlag == 0 || dataFlag == 2){
          tempURL = url;
          getTemp(dataFlag);

          TempPanel = new JPanel(){
            public void paintComponent(Graphics g){
                Graphics2D g2 = (Graphics2D)g;
                DayScaled = new double[numDays];
                TempScaled = new double[numDays];

                super.paintComponent(g);
                setBackground(Color.white);
```

```
                int shift = 10;
                for(int ct = 0; ct<numDays;ct++){
                  if(dataFlag == 0){
                      DayScaled[ct] = DayT[ct]*(Xextent*1.05/numDays);
                      TempScaled[ct] = (Yextent-
Temperature[ct]*(Yextent/100));
                  }
                  else{
                      DayScaled[ct] = DayDT[ct]*(Xextent*1.05/numDays);
                      TempScaled[ct] = (Yextent-
DTemperature[ct]*(Yextent/100))-shift+2;
                  }
                }

                BasicStroke stroke = new BasicStroke(2.0f);
                GeneralPath polyline = new
GeneralPath(GeneralPath.WIND_EVEN_ODD,numDays);

polyline.moveTo((float)(DayScaled[0]+shift),(float)TempScaled[0]);
                for(int rt = 1; rt<numDays;rt++)

polyline.lineTo((float)DayScaled[rt]+shift,(float)TempScaled[rt]);
                g2.setStroke(stroke);
                if(dataFlag == 0){
                  g2.setColor(Color.red);}
                else{
                  g2.setColor(Color.green);}
                g2.draw(polyline);
                g2.setColor(Color.black);
                g2.draw(new Rectangle2D.Double(shift-2,shift-2,
Xextent+40-shift, Yextent-shift-5));

                float dash1[] = {3.0f};
                BasicStroke dashed = new BasicStroke(1.0f,
BasicStroke.CAP_BUTT,
                                BasicStroke.JOIN_MITER, 10.0f, dash1,
0.0f);
                g2.setStroke(dashed);
                for(int ct = 0; ct<360;ct=ct+30){
                  g2.draw(new Line2D.Double(DayScaled[ct+29], shift-2,
DayScaled[ct+29], Yextent-4));
                }
                for(int ct = 0; ct<90;ct=ct+10){
                  g2.draw(new Line2D.Double(shift-2, (shift-
2+ct)*Yextent/100,
                                Xextent+4*shift,(shift-
2+ct)*Yextent/100));
                }
                g.drawString("(0,0)",shift,Yextent-shift-5);
                g.drawString("(365,0)",Xextent-shift,Yextent-shift-5);
                g.drawString("Time (days)",Xextent/2,Yextent-shift-5);
                g.drawString("(0,100)", shift,2*shift);
                if(dataFlag == 0){
                  g.drawString("Temperature",shift, Yextent/2);}
                else{
                  g.drawString("Dew Temperature",shift, Yextent/2);}
                g.drawString("(F)",shift+25, (Yextent/2+15));
```

```
            }
          };

          JScrollPane areaScrollPane = new JScrollPane(TempPanel);
          areaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
          areaScrollPane.setPreferredSize(new Dimension(Xextent,
Yextent));

          if(dataFlag == 0){
            areaScrollPane.setBorder(
                  BorderFactory.createCompoundBorder(
                      BorderFactory.createCompoundBorder(
                                BorderFactory.createTitledBorder("Daily
Temperature Time Series (F) "),

BorderFactory.createEmptyBorder(5,5,5,5)),
                    areaScrollPane.getBorder()));
          }
          else{
            areaScrollPane.setBorder(
                  BorderFactory.createCompoundBorder(
                      BorderFactory.createCompoundBorder(
                                BorderFactory.createTitledBorder("Daily
Dew Point Temperature Time Series (F) "),

BorderFactory.createEmptyBorder(5,5,5,5)),
                    areaScrollPane.getBorder()));
          }
          cf.add(areaScrollPane);
      }

      if(dataFlag == 1){
          rainURL = url;
          getPrecip();

          PrecipPanel = new JPanel(){
            public void paintComponent(Graphics g){
                Graphics2D g2 = (Graphics2D)g;
                DayScaled = new double[numDays];
                PrecipScaled = new double[numDays];

                super.paintComponent(g);
                setBackground(Color.white);

                int shift = 10;
                for(int ct = 0; ct<numDays;ct++){
                      DayScaled[ct] = DayP[ct]*(Xextent*1.05/numDays);
                      PrecipScaled[ct] = (Yextent-
Precipitation[ct]*(Yextent/10));
                }

                BasicStroke stroke = new BasicStroke(2.0f);
                GeneralPath polyline = new
GeneralPath(GeneralPath.WIND_EVEN_ODD,numDays);
```

```
polyline.moveTo((float)(DayScaled[0]+shift),(float)PrecipScaled[0]-
shift+2);
                for(int rt = 1; rt<numDays;rt++)

polyline.lineTo((float)DayScaled[rt]+shift,(float)PrecipScaled[rt]-
shift+2);
                g2.setStroke(stroke);
                g2.setColor(Color.blue);
                g2.draw(polyline);
                g2.setColor(Color.black);
                g2.draw(new Rectangle2D.Double(shift-2,shift-2,
Xextent+40-shift, Yextent-shift-5));

                float dash1[] = {3.0f};
                BasicStroke dashed = new BasicStroke(1.0f,
BasicStroke.CAP_BUTT,
                                BasicStroke.JOIN_MITER, 10.0f, dash1,
0.0f);
                g2.setStroke(dashed);
                for(int ct = 0; ct<360;ct=ct+30){
                  g2.draw(new Line2D.Double(DayScaled[ct+29], shift-2,
DayScaled[ct+29], Yextent-4));
                }
                for(int ct = 0; ct<90;ct=ct+10){
                  g2.draw(new Line2D.Double(shift-2, (shift-
2+ct)*Yextent/100,
                                Xextent+4*shift,(shift-
2+ct)*Yextent/100));
                }
                g.drawString("(0,0)",shift,Yextent-shift-35);
                g.drawString("(365,0)",Xextent-shift,Yextent-shift-35);
                g.drawString("Time (days)",Xextent/2,Yextent-shift-35);
                g.drawString("(0,10)", shift,2*shift);
                g.drawString("Precipitation",shift, Yextent/2);
                g.drawString("(in/day)",shift+10, (Yextent/2+15));
              }
            };

            JScrollPane areaScrollPane = new JScrollPane(PrecipPanel);
            areaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
            areaScrollPane.setPreferredSize(new Dimension(Xextent,
Yextent));
            areaScrollPane.setBorder(
                    BorderFactory.createCompoundBorder(
                        BorderFactory.createCompoundBorder(
                                BorderFactory.createTitledBorder("Daily
Precipitation Time Series (in/day)"),

BorderFactory.createEmptyBorder(5,5,5,5)),
                        areaScrollPane.getBorder())));
            cf.add(areaScrollPane);
        }

        if(dataFlag == 3 || dataFlag == 4){
            otherURL = url;
```

```
          getOther(dataFlag);

          OtherPanel = new JPanel(){
            public void paintComponent(Graphics g){
                Graphics2D g2 = (Graphics2D)g;
                DayScaled = new double[numDays];
                OtherScaled = new double[numDays];

                super.paintComponent(g);
                setBackground(Color.white);

                int shift = 10;
                for(int ct = 0; ct<numDays;ct++){
                  if(dataFlag == 3){
                      DayScaled[ct] = DayW[ct]*(Xextent*1.05/numDays);
                      OtherScaled[ct] = (Yextent-
Wind[ct]*(Yextent/50));
                  }
                  else{
                      DayScaled[ct] = DayS[ct]*(Xextent*1.05/numDays);
                      OtherScaled[ct] = (Yextent-
Cover[ct]*(Yextent/50));
                  }
                }

                BasicStroke stroke = new BasicStroke(2.0f);
                GeneralPath polyline = new
GeneralPath(GeneralPath.WIND_EVEN_ODD,numDays);

polyline.moveTo((float)(DayScaled[0]+shift),(float)OtherScaled[0]-
shift+2);
                for(int rt = 1; rt<numDays;rt++)

polyline.lineTo((float)DayScaled[rt]+shift,(float)OtherScaled[rt]-
shift+2);
                g2.setStroke(stroke);
                if(dataFlag == 3){
                  g2.setColor(Color.cyan);}
                else{
                  g2.setColor(Color.magenta);}
                g2.draw(polyline);
                g2.setColor(Color.black);
                g2.draw(new Rectangle2D.Double(shift-2,shift-2,
Xextent+40-shift, Yextent-shift-5));

                float dash1[] = {3.0f};
                BasicStroke dashed = new BasicStroke(1.0f,
BasicStroke.CAP_BUTT,
                                    BasicStroke.JOIN_MITER, 10.0f, dash1,
0.0f);
                g2.setStroke(dashed);
                for(int ct = 0; ct<360;ct=ct+30){
                  g2.draw(new Line2D.Double(DayScaled[ct+29], shift-2,
DayScaled[ct+29], Yextent-4));
                }
                for(int ct = 0; ct<90;ct=ct+10){
```

```
                    g2.draw(new Line2D.Double(shift-2, (shift-
2+ct)*Yextent/100,
                                    Xextent+4*shift,(shift-
2+ct)*Yextent/100));
                  }
                  g.drawString("(0,0)",shift,Yextent-shift-5);
                  g.drawString("(365,0)",Xextent-shift,Yextent-shift-5);
                  g.drawString("Time (days)",Xextent/2,Yextent-shift-5);
                  g.drawString("(0,50)", shift,2*shift);
                  if(dataFlag == 3){
                    g.drawString("Wind",shift, Yextent/2);
                    g.drawString("(mi/hr)",shift, (Yextent/2+15));}
                  else{
                    g.drawString("Sky Cover",shift, Yextent/2);
                    g.drawString("(tenths)",shift, Yextent/2+15);
                  }
              }
            };

            JScrollPane areaScrollPane = new JScrollPane(OtherPanel);
            areaScrollPane.setVerticalScrollBarPolicy(
                        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
            areaScrollPane.setPreferredSize(new Dimension(Xextent,
Yextent));

            if(dataFlag == 3){
              areaScrollPane.setBorder(
                    BorderFactory.createCompoundBorder(
                        BorderFactory.createCompoundBorder(
                                    BorderFactory.createTitledBorder("Daily
Wind Speed Time Series (mi/hr) "),

BorderFactory.createEmptyBorder(5,5,5,5)),
                      areaScrollPane.getBorder()));
            }
            else{
              areaScrollPane.setBorder(
                    BorderFactory.createCompoundBorder(
                        BorderFactory.createCompoundBorder(
                                    BorderFactory.createTitledBorder("Daily
Sky Cover Time Series (tenths) "),

BorderFactory.createEmptyBorder(5,5,5,5)),
                      areaScrollPane.getBorder()));
            }
            cf.add(areaScrollPane);
        }

        setVisible(true);
      }

      public void getTemp(int flag){

        try{
            tempData = new URL(tempURL);
        }
        catch(MalformedURLException u){
```

```
            System.out.println("URLException: " + u.getMessage());
        }
        try{
            BufferedReader buffReader = new BufferedReader(new
                        InputStreamReader(tempData.openStream()));
            StreamTokenizer st = new StreamTokenizer(buffReader);
            int numLines=365;
            numDays = numLines;

            try{
              DayT = new double[numLines];
              DayDT = new double[numLines];
              Temperature = new double[numLines];
              DTemperature = new double[numLines];

              for(int nL=0;nL<numLines;nL++){
                  st.nextToken();
                  if(flag == 0){
                    DayT[nL] = st.nval;
                  }
                  else{
                    DayDT[nL] = st.nval;
                  }
                  st.nextToken();
                  if(flag == 0){
                    Temperature[nL] = st.nval;
                  }
                  else{
                    DTemperature[nL] = st.nval;
                  }
              }
              buffReader.close();
            }
            catch(IOException e){
              System.out.println("IOException: "+e.getMessage());
            }
        }
        catch(IOException f){
            System.out.println("IOException: " +f.getMessage());
        }
    }

    public void getPrecip(){

        try{
            precipData = new URL(rainURL);
        }
        catch(MalformedURLException u){
            System.out.println("URLException: " + u.getMessage());
        }
        try{
            BufferedReader buffReader = new BufferedReader(new
                        InputStreamReader(precipData.openStream()));
            StreamTokenizer st = new StreamTokenizer(buffReader);
            int numLines=365;
            numDays = numLines;
```

```
        try{
          DayP = new double[numLines];
          Precipitation = new double[numLines];

          for(int nL=0;nL<numLines;nL++){
              st.nextToken();
              DayP[nL] = st.nval;
              st.nextToken();
              Precipitation[nL] = st.nval;
          }
          buffReader.close();
        }
        catch(IOException e){
          System.out.println("IOException: " + e.getMessage());
        }
    }
    catch(IOException f){
        System.out.println("IOException: " + f.getMessage());
    }
}

public void getOther(int flag){

    try{
        otherData = new URL(otherURL);
    }
    catch(MalformedURLException u){
        System.out.println("URLException: " + u.getMessage());
    }
    try{
        BufferedReader buffReader = new BufferedReader(new
                    InputStreamReader(otherData.openStream()));
        StreamTokenizer st = new StreamTokenizer(buffReader);
        int numLines=365;
        numDays = numLines;

        try{
          DayW = new double[numLines];
          DayS = new double[numLines];
          Wind = new double[numLines];
          Cover = new double[numLines];

          for(int nL=0;nL<numLines;nL++){
              st.nextToken();
              if(flag == 3){
                DayW[nL] = st.nval;
              }
              else{
                DayS[nL] = st.nval;
              }
              st.nextToken();
              if(flag == 3){
                Wind[nL] = st.nval;
              }
              else{
                  Cover[nL] = st.nval;
              }
```

```
            }
            buffReader.close();
          }
          catch(IOException e){
            System.out.println("IOException: " + e.getMessage());
          }
      }
      catch(IOException f){
          System.out.println("IOException: " + f.getMessage());
      }
    }

    public void setMenuBar(){
      menuBar = new JMenuBar();
      fileMenu = new JMenu("Window");
      fileMenu.setToolTipText("Window Exit");
      menuBar.add(fileMenu);
      menuItem1 = new JMenuItem("Exit");
      menuItem1.addActionListener(this);
      fileMenu.add(menuItem1);
      setJMenuBar(menuBar);
    }

    public double[] getATemp(){
      return Temperature;
    }

    public double[] getDTemp(){
      return DTemperature;
    }

    public double[] getRPrec(){
      return Precipitation;
    }

    public double[] getWind(){
      return Wind;
    }

    public double[] getCover(){
      return Cover;
    }

    public double[] getDay(){
      return DayT;
    }

    public void actionPerformed(ActionEvent evt){
      Object src = evt.getSource();

      if(src == menuItem1){
          System.out.println("Exiting Hydrometeorological Data View");
          dispose();
      }
    }
}
```

```
//************************************
//
//   TRIBSFileIO.java
//
//   Input File Class Definition
//
//************************************
package tRIBS;

import java.io.*;
import java.lang.reflect.*;
import javax.swing.DefaultListModel;
import java.util.ArrayList;
import tRIBS.BasinPlotter;

public class TRIBSFileIO {


/*       private String startDate, runtime, timeStep, gwStep, metStep,
rainIntrvl,
                opIntrvl, notIntrvl, intStormMax, rainSearch;
        private String baseFlow, velocityCoef, kinemVelCoef,
velocityRatio,
                flowExp, channelRoughness, channelWidth;
        private String optMeshInput, rainSource, optEvapTrans,
optIntercept,
                gFluxOption, metDataOption, convertData, optBedrock;
        private String inputDataFile, inputTime, arcInfoFilename,
pointFilename;
        private String soilTableName, soilMapName, landTablename,
landMapName,
                gWaterFile, rainFile, rainExtension, depthToBedrock,
bedrockFile;
        private String hydrometStations, hyrdroMetGrid, hydrometConvert,
                hydrometBaseName, gaugeStations, gaugeConvert,
gaugeBaseName;
        private String outFilename, outHydroFilename, ribsHydOutput,
                nodeOutputList, hydroNodeList, outletNodeList;
        private File inFile, outFile;
*/
        private String[] fields;
        private boolean[] isActiveField;

        private static final int TOTALVARS=51;
        //Time Variable ints
        private static final int STARTDATE = 0;
        private static final int RUNTIME = 1;
        private static final int TIMESTEP = 2;
        private static final int GWSTEP = 3;
        private static final int METSTEP = 4;
        private static final int RAININTRVL = 5;
        private static final int OPINTRVL = 6;
        private static final int NOTINTRVL = 7;
        private static final int INTSTORMMAX = 8;
        private static final int RAINSEARCH = 9;
        //Routing Variable ints
```

```
private static final int BASEFLOW = 10;
private static final int VELOCITYCOEF = 11;
private static final int KINEMVELCOEF = 12;
private static final int VELOCITYRATIO = 13;
private static final int FLOWEXP = 14;
private static final int CHANNELROUGHNESS = 15;
private static final int CHANNELWIDTH = 16;
//Model Run Options ints
private static final int OPTMESHINPUT = 17;
private static final int RAINSOURCE = 18;
private static final int OPTEVAPOTRANS = 19;
private static final int OPTINTERCEPT = 20;
private static final int GFLUXOPTION = 21;
private static final int METDATAOPTION = 22;
private static final int CONVERTDATA = 23;
private static final int OPTBEDROCK = 24;
//Mesh Generation ints
private static final int INPUTDATAFILE = 25;
private static final int INPUTTIME = 26;
private static final int ARCINFOFILENAME = 27;
private static final int POINTFILENAME = 28;
//Resampling Grid ints
private static final int SOILTABLENAME = 29;
private static final int SOILMAPNAME = 30;
private static final int LANDTABLENAME = 31;
private static final int LANDMAPNAME = 32;
private static final int GWATERFILE = 33;
private static final int RAINFILE = 34;
private static final int RAINEXTENSION = 35;
private static final int DEPTHTOBEDROCK = 36;
private static final int BEDROCKFILE = 37;
//Meterological Data ints
private static final int HYDROMETSTATIONS = 38;
private static final int HYDROMETGRID = 39;
private static final int HYDROMETCONVERT = 40;
private static final int HYDROMETBASENAME = 41;
private static final int GAUGESTATIONS = 42;
private static final int GAUGECONVERT = 43;
private static final int GAUGEBASENAME = 44;
//Output Data ints
private static final int OUTFILENAME = 45;
private static final int OUTHYDROFILENAME = 46;
private static final int RIBSHYDOUTPUT = 47;
private static final int NODEOUTPUTLIST = 48;
private static final int HYDRONODELIST = 49;
private static final int OUTLETNODELIST = 50;


//Time Variable Strings
private static final String STARTDATELABEL = "STARTDATE";
private static final String RUNTIMELABEL = "RUNTIME";
private static final String TIMESTEPLABEL = "TIMESTEP";
private static final String GWSTEPLABEL = "GWSTEP";
private static final String METSTEPLABEL = "METSTEP";
private static final String RAININTRVLLABEL = "RAININTRVL";
private static final String OPINTRVLLABEL = "OPINTRVL";
private static final String NOTINTRVLLABEL = "NOTINTRVL";
```

```
        private static final String INTSTORMMAXLABEL = "INTSTORMMAX";
        private static final String RAINSEARCHLABEL = "RAINSEARCH";
        //Routing Variable Strings
        private static final String BASEFLOWLABEL = "BASEFLOW";
        private static final String VELOCITYCOEFLABEL = "VELOCITYCOEF";
        private static final String KINEMVELCOEFLABEL = "KINEMVELCOEF";
        private static final String VELOCITYRATIOLABEL =
"VELOCITYRATIO";
        private static final String FLOWEXPLABEL = "FLOWEXP";
        private static final String CHANNELROUGHNESSLABEL =
"CHANNELROUGHNESS";
        private static final String CHANNELWIDTHLABEL = "CHANNELWIDTH";
        //Model Run Options Strings
        private static final String OPTMESHINPUTLABEL = "OPTMESHINPUT";
        private static final String RAINSOURCELABEL = "RAINSOURCE";
        private static final String OPTEVAPOTRANSLABEL =
"OPTEVAPOTRANS";
        private static final String OPTINTERCEPTLABEL = "OPTINTERCEPT";
        private static final String GFLUXOPTIONLABEL = "GFLUXOPTION";
        private static final String METDATAOPTIONLABEL =
"METDATAOPTION";
        private static final String CONVERTDATALABEL = "CONVERTDATA";
        private static final String OPTBEDROCKLABEL = "OPTBEDROCK";
        //Mesh Generation Strings
        private static final String INPUTDATAFILELABEL =
"INPUTDATAFILE";
        private static final String INPUTTIMELABEL = "INPUTTIME";
        private static final String ARCINFOFILENAMELABEL =
"ARCINFOFILENAME";
        private static final String POINTFILENAMELABEL =
"POINTFILENAME";
        //Resampling Grid Strings
        private static final String SOILTABLENAMELABEL =
"SOILTABLENAME";
        private static final String SOILMAPNAMELABEL = "SOILMAPNAME";
        private static final String LANDTABLENAMELABEL =
"LANDTABLENAME";
        private static final String LANDMAPNAMELABEL = "LANDMAPNAME";
        private static final String GWATERFILELABEL = "GWATERFILE";
        private static final String RAINFILELABEL = "RAINFILE";
        private static final String RAINEXTENSIONLABEL =
"RAINEXTENSION";
        private static final String DEPTHTOBEDROCKLABEL =
"DEPTHTOBEDROCK";
        private static final String BEDROCKFILELABEL = "BEDROCKFILE";
        //Meterological Data Strings
        private static final String HYDROMETSTATIONSLABEL =
"HYDROMETSTATIONS";
        private static final String HYDROMETGRIDLABEL = "HYDROMETGRID";
        private static final String HYDROMETCONVERTLABEL =
"HYDROMETCONVERT";
        private static final String HYDROMETBASENAMELABEL =
"HYDROMETBASENAME";
        private static final String GAUGESTATIONSLABEL =
"GAUGESTATIONS";
        private static final String GAUGECONVERTLABEL = "GAUGECONVERT";
```

```
        private static final String GAUGEBASENAMELABEL =
"GAUGEBASENAME";
        //Output Data Strings
        private static final String OUTFILENAMELABEL = "OUTFILENAME";
        private static final String OUTHYDROFILENAMELABEL =
"OUTHYDROFILENAME";
        private static final String RIBSHYDOUTPUTLABEL =
"RIBSHYDOUTPUT";
        private static final String NODEOUTPUTLISTLABEL =
"NODEOUTPUTLIST";
        private static final String HYDRONODELISTLABEL =
"HYDRONODELIST";
        private static final String OUTLETNODELISTLABEL =
"OUTLETNODELIST";

        private String[] labels = {
                //Time Variable Labels
                STARTDATELABEL,
                RUNTIMELABEL,
                TIMESTEPLABEL,
                GWSTEPLABEL,
                METSTEPLABEL,
                RAININTRVLLABEL,
                OPINTRVLLABEL,
                NOTINTRVLLABEL,
                INTSTORMMAXLABEL,
                RAINSEARCHLABEL,
                //Routing Variable Labels
                BASEFLOWLABEL,
                VELOCITYCOEFLABEL,
                KINEMVELCOEFLABEL,
                VELOCITYRATIOLABEL,
                FLOWEXPLABEL,
                CHANNELROUGHNESSLABEL,
                CHANNELWIDTHLABEL,
                //Model Run Options Labels
                OPTMESHINPUTLABEL,
                RAINSOURCELABEL,
                OPTEVAPOTRANSLABEL,
                OPTINTERCEPTLABEL,
                GFLUXOPTIONLABEL,
                METDATAOPTIONLABEL,
                CONVERTDATALABEL,
                OPTBEDROCKLABEL,
                //Mesh Generation Labels
                INPUTDATAFILELABEL,
                INPUTTIMELABEL,
                ARCINFOFILENAMELABEL,
                POINTFILENAMELABEL,
                //Resampling Grid Labels
                SOILTABLENAMELABEL,
                SOILMAPNAMELABEL,
                LANDTABLENAMELABEL,
                LANDMAPNAMELABEL,
                GWATERFILELABEL,
                RAINFILELABEL,
                RAINEXTENSIONLABEL,
```

```
        DEPTHTOBEDROCKLABEL,
        BEDROCKFILELABEL,
        //Meterological Data Labels
        HYDROMETSTATIONSLABEL,
        HYDROMETGRIDLABEL,
        HYDROMETCONVERTLABEL,
        HYDROMETBASENAMELABEL,
        GAUGESTATIONSLABEL,
        GAUGECONVERTLABEL,
        GAUGEBASENAMELABEL,
        //Output Data Labels
        OUTFILENAMELABEL,
        OUTHYDROFILENAMELABEL,
        RIBSHYDOUTPUTLABEL,
        NODEOUTPUTLISTLABEL,
        HYDRONODELISTLABEL,
        OUTLETNODELISTLABEL
};


public TRIBSFileIO() {
     fields = new String[TOTALVARS];
     isActiveField = new boolean[TOTALVARS];
     for (int i=0; i<TOTALVARS; i++) {
       fields[i] = "";
       isActiveField[i] = false;
     }
}

/************************************
 * GET functions
 ************************************/
public String getStartDate()
{
   return fields[STARTDATE];
}

public String getRuntime()
{
   return fields[RUNTIME];
}

public String getTimeStep()
{
   return fields[TIMESTEP];
}

public String getGWStep()
{
   return fields[GWSTEP];
}

public String getMetStep()
{
   return fields[METSTEP];
}
```

```
public String getRainIntrvl()
{
   return fields[RAININTRVL];
}

public String getOpIntrvl()
{
   return fields[OPINTRVL];
}

public String getNotIntrvl()
{
   return fields[NOTINTRVL];
}

public String getIntStormMax()
{
   return fields[INTSTORMMAX];
}

public String getRainSearch()
{
   return fields[RAINSEARCH];
}

public String getBaseFlow()
{
   return fields[BASEFLOW];
}

public String getVelocityCoef()
{
   return fields[VELOCITYCOEF];
}

public String getKinemVelCoef()
{
   return fields[KINEMVELCOEF];
}

public String getVelocityRatio()
{
   return fields[VELOCITYRATIO];
}

public String getFlowExp()
{
   return fields[FLOWEXP];
}

public String getChannelRoughness()
{
   return fields[CHANNELROUGHNESS];
}

public String getChannelWidth()
{
```

```
        return fields[CHANNELWIDTH];
    }

    public String getOptMeshInput()
    {
        return fields[OPTMESHINPUT];
    }

    public String getRainSource()
    {
        return fields[RAINSOURCE];
    }

    public String getOptEvapoTrans()
    {
        return fields[OPTEVAPOTRANS];
    }

    public String getOptIntercept()
    {
        return fields[OPTINTERCEPT];
    }

    public String getGFluxOption()
    {
        return fields[GFLUXOPTION];
    }

    public String getMetDataOption()
    {
        return fields[METDATAOPTION];
    }

    public String getConvertData()
    {
        return fields[CONVERTDATA];
    }

    public String getOptBedrock()
    {
        return fields[OPTBEDROCK];
    }

    public String getInputDataFile()
    {
        return fields[INPUTDATAFILE];
    }

    public String getInputTime()
    {
        return fields[INPUTTIME];
    }

    public String getArcInfoFilename()
    {
        return fields[ARCINFOFILENAME];
    }
```

```
public String getPointFilename()
{
   return fields[POINTFILENAME];
}

public String getSoilTableName()
{
   return fields[SOILTABLENAME];
}

public String getSoilMapName()
{
   return fields[SOILMAPNAME];
}

public String getLandTablename()
{
   return fields[LANDTABLENAME];
}

public String getLandMapName()
{
   return fields[LANDMAPNAME];
}

public String getGWaterFile()
{
   return fields[GWATERFILE];
}

public String getRainFile()
{
   return fields[RAINFILE];
}

public String getRainExtension()
{
   return fields[RAINEXTENSION];
}

public String getDepthToBedrock()
{
   return fields[DEPTHTOBEDROCK];
}

public String getBedrockFile()
{
   return fields[BEDROCKFILE];
}

public String getHydrometStations()
{
   return fields[HYDROMETSTATIONS];
}
```

```
public String getHydrometGrid()
{
   return fields[HYDROMETGRID];
}

public String getHydrometConvert()
{
   return fields[HYDROMETCONVERT];
}

public String getHydrometBaseName()
{
   return fields[HYDROMETBASENAME];
}

public String getGaugeStations()
{
   return fields[GAUGESTATIONS];
}

public String getGaugeConvert()
{
   return fields[GAUGECONVERT];
}

public String getGaugeBaseName()
{
   return fields[GAUGEBASENAME];
}

public String getOutFilename()
{
   return fields[OUTFILENAME];
}

public String getOutHydroFilename()
{
   return fields[OUTHYDROFILENAME];
}

public String getRIBSHydOutput()
{
   return fields[RIBSHYDOUTPUT];
}

public String getNodeOutputList()
{
   return fields[NODEOUTPUTLIST];
}

public String getHydroNodeList()
{
   return fields[HYDRONODELIST];
}

public String getOutletNodeList()
{
```

```
    return fields[OUTLETNODELIST];
}


/************************************
 * SET functions
 ************************************/
public void setStartDate(String newStartDate)
{
 fields[STARTDATE] = newStartDate;
 if (newStartDate.equals("")) {
    this.isActiveField[STARTDATE] = false;
 } else {
    this.isActiveField[STARTDATE]= true;
 }
}
public void setRuntime(String newRuntime)
{
 fields[RUNTIME] = newRuntime;
if (newRuntime.equals("")) {
    this.isActiveField[RUNTIME] = false;
 } else {
    this.isActiveField[RUNTIME]= true;
 }
}

public void setTimeStep(String newTimeStep)
{
 fields[TIMESTEP] = newTimeStep;
 if (newTimeStep.equals("")) {
    this.isActiveField[TIMESTEP] = false;
 } else {
    this.isActiveField[TIMESTEP]= true;
 }
}

public void setGWStep(String newGWStep)
{
 fields[GWSTEP] = newGWStep;
       if (newGWStep.equals("")) {
    this.isActiveField[GWSTEP] = false;
 } else {
    this.isActiveField[GWSTEP]= true;
 }
}

public void setMetStep(String newMetStep)
{
 fields[METSTEP] = newMetStep;
       if (newMetStep.equals("")) {
    this.isActiveField[METSTEP] = false;
 } else {
    this.isActiveField[METSTEP]= true;
 }
}

public void setRainIntrvl(String newRainIntrvl)
```

```
{
 fields[RAININTRVL] = newRainIntrvl;
       if (newRainIntrvl.equals("")) {
    this.isActiveField[RAININTRVL] = false;
 } else {
    this.isActiveField[RAININTRVL]= true;
 }
}

public void setOpIntrvl(String newOpIntrvl)
{
 fields[OPINTRVL] = newOpIntrvl;
       if (newOpIntrvl.equals("")) {
    this.isActiveField[OPINTRVL] = false;
 } else {
    this.isActiveField[OPINTRVL]= true;
 }
}

public void setNotIntrvl(String newNotIntrvl)
{
 fields[NOTINTRVL] = newNotIntrvl;
       if (newNotIntrvl.equals("")) {
    this.isActiveField[NOTINTRVL] = false;
 } else {
    this.isActiveField[NOTINTRVL]= true;
 }
}

public void setIntStormMax(String newIntStormMax)
{
 fields[INTSTORMMAX] = newIntStormMax;
       if (newIntStormMax.equals("")) {
    this.isActiveField[INTSTORMMAX] = false;
 } else {
    this.isActiveField[INTSTORMMAX]= true;
 }
}

public void setRainSearch(String newRainSearch)
{
 fields[RAINSEARCH] = newRainSearch;
       if (newRainSearch.equals("")) {
    this.isActiveField[RAINSEARCH] = false;
 } else {
    this.isActiveField[RAINSEARCH]= true;
 }
}

public void setBaseFlow(String newBaseFlow)
{
 fields[BASEFLOW] = newBaseFlow;
       if (newBaseFlow.equals("")) {
    this.isActiveField[BASEFLOW] = false;
 } else {
    this.isActiveField[BASEFLOW]= true;
 }
```

```
}

public void setVelocityCoef(String newVelocityCoef)
{
 fields[VELOCITYCOEF] = newVelocityCoef;
       if (newVelocityCoef.equals("")) {
    this.isActiveField[VELOCITYCOEF] = false;
 } else {
    this.isActiveField[VELOCITYCOEF]= true;
 }
}

public void setKinemVelCoef(String newKinemVelCoef)
{
 fields[KINEMVELCOEF] = newKinemVelCoef;
       if (newKinemVelCoef.equals("")) {
    this.isActiveField[KINEMVELCOEF] = false;
 } else {
    this.isActiveField[KINEMVELCOEF]= true;
 }
}

public void setVelocityRatio(String newVelocityRatio)
{
 fields[VELOCITYRATIO] = newVelocityRatio;
       if (newVelocityRatio.equals("")) {
    this.isActiveField[VELOCITYRATIO] = false;
 } else {
    this.isActiveField[VELOCITYRATIO]= true;
 }
}

public void setFlowExp(String newFlowExp)
{
 fields[FLOWEXP] = newFlowExp;
       if (newFlowExp.equals("")) {
    this.isActiveField[FLOWEXP] = false;
 } else {
    this.isActiveField[FLOWEXP]= true;
 }
}

public void setChannelRoughness(String newChannelRoughness)
{
 fields[CHANNELROUGHNESS] = newChannelRoughness;
       if (newChannelRoughness.equals("")) {
    this.isActiveField[CHANNELROUGHNESS] = false;
 } else {
    this.isActiveField[CHANNELROUGHNESS]= true;
 }
}

public void setChannelWidth(String newChannelWidth)
{
 fields[CHANNELWIDTH] = newChannelWidth;
       if (newChannelWidth.equals("")) {
    this.isActiveField[CHANNELWIDTH] = false;
```

```
 } else {
    this.isActiveField[CHANNELWIDTH]= true;
 }
}

public void setOptMeshInput(String newOptMeshInput)
{
 fields[OPTMESHINPUT] = newOptMeshInput;
       if (newOptMeshInput.equals("")) {
    this.isActiveField[OPTMESHINPUT] = false;
 } else {
    this.isActiveField[OPTMESHINPUT]= true;
 }
}

public void setRainSource(String newRainSource)
{
 fields[RAINSOURCE] = newRainSource;
       if (newRainSource.equals("")) {
    this.isActiveField[RAINSOURCE] = false;
 } else {
    this.isActiveField[RAINSOURCE]= true;
 }
}

public void setOptEvapoTrans(String newOptEvapoTrans)
{
 fields[OPTEVAPOTRANS] = newOptEvapoTrans;
       if (newOptEvapoTrans.equals("")) {
    this.isActiveField[OPTEVAPOTRANS] = false;
 } else {
    this.isActiveField[OPTEVAPOTRANS]= true;
 }
}

public void setOptIntercept(String newOptIntercept)
{
 fields[OPTINTERCEPT] = newOptIntercept;
       if (newOptIntercept.equals("")) {
    this.isActiveField[OPTINTERCEPT] = false;
 } else {
    this.isActiveField[OPTINTERCEPT]= true;
 }
}

public void setGFluxOption(String newGFluxOption)
{
 fields[GFLUXOPTION] = newGFluxOption;
       if (newGFluxOption.equals("")) {
    this.isActiveField[GFLUXOPTION] = false;
 } else {
    this.isActiveField[GFLUXOPTION]= true;
 }
}

public void setMetDataOption(String newMetDataOption)
{
```

```
 fields[METDATAOPTION] = newMetDataOption;
       if (newMetDataOption.equals("")) {
    this.isActiveField[METDATAOPTION] = false;
 } else {
    this.isActiveField[METDATAOPTION]= true;
 }
}

public void setConvertData(String newConvertData)
{
 fields[CONVERTDATA] = newConvertData;
       if (newConvertData.equals("")) {
    this.isActiveField[CONVERTDATA] = false;
 } else {
    this.isActiveField[CONVERTDATA]= true;
 }
}

public void setOptBedrock(String newOptBedrock)
{
 fields[OPTBEDROCK] = newOptBedrock;
       if (newOptBedrock.equals("")) {
    this.isActiveField[OPTBEDROCK] = false;
 } else {
    this.isActiveField[OPTBEDROCK]= true;
 }
}

public void setInputDataFile(String newInputDataFile)
{
 fields[INPUTDATAFILE] = newInputDataFile;
       if (newInputDataFile.equals("")) {
    this.isActiveField[INPUTDATAFILE] = false;
 } else {
    this.isActiveField[INPUTDATAFILE]= true;
 }
}

public void setInputTime(String newInputTime)
{
 fields[INPUTTIME] = newInputTime;
 if (newInputTime.equals("")) {
    this.isActiveField[INPUTTIME] = false;
 } else {
    this.isActiveField[INPUTTIME]= true;
 }
       }

public void setArcInfoFilename(String newArcInfoFilename)
{
 fields[ARCINFOFILENAME] = newArcInfoFilename;
  if (newArcInfoFilename.equals("")) {
    this.isActiveField[ARCINFOFILENAME] = false;
 } else {
    this.isActiveField[ARCINFOFILENAME]= true;
 }
}
```

```
public void setPointFilename(String newPointFilename)
{
 fields[POINTFILENAME] = newPointFilename;
  if (newPointFilename.equals("")) {
    this.isActiveField[POINTFILENAME] = false;
 } else {
    this.isActiveField[POINTFILENAME]= true;
 }
}

public void setSoilTableName(String newSoilTableName)
{
 fields[SOILTABLENAME] = newSoilTableName;
 if (newSoilTableName.equals("")) {
    this.isActiveField[SOILTABLENAME] = false;
 } else {
    this.isActiveField[SOILTABLENAME]= true;
 }
}

public void setSoilMapName(String newSoilMapName)
{
 fields[SOILMAPNAME] = newSoilMapName;
       if (newSoilMapName.equals("")) {
    this.isActiveField[SOILMAPNAME] = false;
 } else {
    this.isActiveField[SOILMAPNAME]= true;
 }
}

public void setLandTablename(String newLandTablename)
{
 fields[LANDTABLENAME] = newLandTablename;
       if (newLandTablename.equals("")) {
    this.isActiveField[LANDTABLENAME] = false;
 } else {
    this.isActiveField[LANDTABLENAME]= true;
 }
}

public void setLandMapName(String newLandMapName)
{
 fields[LANDMAPNAME] = newLandMapName;
       if (newLandMapName.equals("")) {
    this.isActiveField[LANDMAPNAME] = false;
 } else {
    this.isActiveField[LANDMAPNAME]= true;
 }
}

public void setGWaterFile(String newGWaterFile)
{
 fields[GWATERFILE] = newGWaterFile;
       if (newGWaterFile.equals("")) {
    this.isActiveField[GWATERFILE] = false;
 } else {
```

```
    this.isActiveField[GWATERFILE]= true;
 }
}

public void setRainFile(String newRainFile)
{
 fields[RAINFILE] = newRainFile;
       if (newRainFile.equals("")) {
    this.isActiveField[RAINFILE] = false;
 } else {
    this.isActiveField[RAINFILE]= true;
 }
}

public void setRainExtension(String newRainExtension)
{
 fields[RAINEXTENSION] = newRainExtension;
       if (newRainExtension.equals("")) {
    this.isActiveField[RAINEXTENSION] = false;
 } else {
    this.isActiveField[RAINEXTENSION]= true;
 }
}

public void setDepthToBedrock(String newDepthToBedrock)
{
 fields[DEPTHTOBEDROCK] = newDepthToBedrock;
       if (newDepthToBedrock.equals("")) {
    this.isActiveField[DEPTHTOBEDROCK] = false;
 } else {
    this.isActiveField[DEPTHTOBEDROCK]= true;
 }
}

public void setBedrockFile(String newBedrockFile)
{
 fields[BEDROCKFILE] = newBedrockFile;
       if (newBedrockFile.equals("")) {
    this.isActiveField[BEDROCKFILE] = false;
 } else {
    this.isActiveField[BEDROCKFILE]= true;
 }
}

public void setHydrometStations(String newHydrometStations)
{
 fields[HYDROMETSTATIONS] = newHydrometStations;
       if (newHydrometStations.equals("")) {
    this.isActiveField[HYDROMETSTATIONS] = false;
 } else {
    this.isActiveField[HYDROMETSTATIONS]= true;
 }
}

public void setHydrometGrid(String newHydrometGrid)
{
 fields[HYDROMETGRID] = newHydrometGrid;
```

```
        if (newHydrometGrid.equals("")) {
    this.isActiveField[HYDROMETGRID] = false;
 } else {
    this.isActiveField[HYDROMETGRID]= true;
 }
}

public void setHydrometConvert(String newHydrometConvert)
{
 fields[HYDROMETCONVERT] = newHydrometConvert;
        if (newHydrometConvert.equals("")) {
    this.isActiveField[HYDROMETCONVERT] = false;
 } else {
    this.isActiveField[HYDROMETCONVERT]= true;
 }
}

public void setHydrometBaseName(String newHydrometBaseName)
{
 fields[HYDROMETBASENAME] = newHydrometBaseName;
        if (newHydrometBaseName.equals("")) {
    this.isActiveField[HYDROMETBASENAME] = false;
 } else {
    this.isActiveField[HYDROMETBASENAME]= true;
 }
}

public void setGaugeStations(String newGaugeStations)
{
 fields[GAUGESTATIONS] = newGaugeStations;
        if (newGaugeStations.equals("")) {
    this.isActiveField[GAUGESTATIONS] = false;
 } else {
    this.isActiveField[GAUGESTATIONS]= true;
 }
}

public void setGaugeConvert(String newGaugeConvert)
{
 fields[GAUGECONVERT] = newGaugeConvert;
        if (newGaugeConvert.equals("")) {
    this.isActiveField[GAUGECONVERT] = false;
 } else {
    this.isActiveField[GAUGECONVERT]= true;
 }
}

public void setGaugeBaseName(String newGaugeBaseName)
{
 fields[GAUGEBASENAME] = newGaugeBaseName;
        if (newGaugeBaseName.equals("")) {
    this.isActiveField[GAUGEBASENAME] = false;
 } else {
    this.isActiveField[GAUGEBASENAME]= true;
 }
}
```

```
public void setOutFilename(String newOutFilename)
{
 fields[OUTFILENAME] = newOutFilename;
       if (newOutFilename.equals("")) {
    this.isActiveField[OUTFILENAME] = false;
 } else {
    this.isActiveField[OUTFILENAME]= true;
 }
}

public void setOutHydroFilename(String newOutHydroFilename)
{
 fields[OUTHYDROFILENAME] = newOutHydroFilename;
       if (newOutHydroFilename.equals("")) {
    this.isActiveField[OUTHYDROFILENAME] = false;
 } else {
    this.isActiveField[OUTHYDROFILENAME]= true;
 }
}

public void setRIBSHydOutput(String newRIBSHydOutput)
{
 fields[RIBSHYDOUTPUT] = newRIBSHydOutput;
       if (newRIBSHydOutput.equals("")) {
    this.isActiveField[RIBSHYDOUTPUT] = false;
 } else {
    this.isActiveField[RIBSHYDOUTPUT]= true;
 }
}

public void setNodeOutputList(String newNodeOutputList)
{
 fields[NODEOUTPUTLIST] = newNodeOutputList;
       if (newNodeOutputList.equals("")) {
    this.isActiveField[NODEOUTPUTLIST] = false;
 } else {
    this.isActiveField[NODEOUTPUTLIST]= true;
 }
}

public void setHydroNodeList(String newHydroNodeList)
{
 fields[HYDRONODELIST] = newHydroNodeList;
       if (newHydroNodeList.equals("")) {
    this.isActiveField[HYDRONODELIST] = false;
 } else {
    this.isActiveField[HYDRONODELIST]= true;
 }
}

public void setOutletNodeList(String newOutletNodeList)
{
 fields[OUTLETNODELIST] = newOutletNodeList;
       if (newOutletNodeList.equals("")) {
    this.isActiveField[OUTLETNODELIST] = false;
 } else {
    this.isActiveField[OUTLETNODELIST]= true;
```

```
        }
       }


       /************************************
        * Import and Export functions
        ************************************/
       public void exportFile(String filename) throws Exception {
          try {
              File outputFile = new File(filename);
            System.out.println("filename=" +filename);
            FileWriter outputWriter = new FileWriter(outputFile);

            for (int i=0; i<TOTALVARS; i++) {
              if (isActiveField[i]) {
                 outputWriter.write(labels[i]);
                 outputWriter.write(":\n");
                 outputWriter.write(fields[i]);
                 outputWriter.write("\n\n");
              }
            }
            outputWriter.close();
          } catch (IOException e) {
              throw new Exception("File writing error");
          }
       }

       public void importFile(String filename) throws IOException,
Exception {
          FileReader inReader = new FileReader(filename);
          StreamTokenizer inTokenizer = new StreamTokenizer(inReader);
          Class c = this.getClass();
          int labelIndex = -1;
          boolean isLabelSet = false;
          boolean isWritten = false;
          boolean pastColon = false;

          //Treat everything - slash, backslash, numbers, dots, etc
          //as parts of the token string
          inTokenizer.resetSyntax();
          inTokenizer.wordChars(0,255);
          //ignore whitespace
          inTokenizer.whitespaceChars(0, ' ');
          //ignore comment lines
          inTokenizer.commentChar('#');
          //ignore colon
          inTokenizer.ordinaryChar(':');

          //test for empty file
          if (inTokenizer.nextToken() == StreamTokenizer.TT_EOF) {
             throw new IOException("Empty File");
          }

          //if nt empty, push back first token
          inTokenizer.pushBack();

          while (inTokenizer.nextToken() != StreamTokenizer.TT_EOF) {
```

```
                try {
                    System.out.println(inTokenizer.toString());
                    if (inTokenizer.ttype == StreamTokenizer.TT_WORD) {
                      if(!isLabelSet) {
                          if (pastColon) {
                              throw new IOException("Missing label at" +
                                    inTokenizer.sval);
                          } else {
                              //set label
                              labelIndex = ((Field)
c.getDeclaredField(inTokenizer.sval)).
                                        getInt(this);
                              System.out.println("label - " +labelIndex);
                              isLabelSet = true;
                              isWritten = false;
                          }
                      } else {
                          if (!pastColon ) {
                              throw new IOException("Missing value at" +
                                    inTokenizer.sval);
                          } else {
                              //set value
                              if (isActiveField[labelIndex]) {
                                  throw new
IOException(labels[labelIndex]
                                          + " already written");
                              } else {
                                fields[labelIndex] = inTokenizer.sval;
                                isActiveField[labelIndex] = true;

                                labelIndex = -1;
                                pastColon = false;
                                isLabelSet =false;
                                isWritten = true;
                              }
                          }
                      }
                    } else if (inTokenizer.ttype == ':') {
                          //make sure got label, but not colon or value
yet
                          if (!isLabelSet || pastColon) {
                             throw new IOException("Missing value at " +
                                    inTokenizer.toString());
                          } else {
                             pastColon = true;
                          }
                    }
                } catch (NoSuchFieldException e) {
                    throw new Exception("No Such field: " +
inTokenizer.sval);
                } catch (Exception e) {
                    throw new Exception("File reading error: " +
e.getMessage());
                }
            }
        }
```

```
        public void appendToFileListModel(DefaultListModel listModel,
                ArrayList listRenderFlags, ArrayList listSupportFiles)
        {
           if(isActiveField[SOILMAPNAME]) {
               listModel.addElement(fields[SOILMAPNAME]);
               listRenderFlags.add(new
Integer(BasinPlotter.SOIL_TEXTURE));
               listSupportFiles.add(fields[SOILTABLENAME]);
           }

           if(isActiveField[LANDMAPNAME]) {
               listModel.addElement(fields[LANDMAPNAME]);
               listRenderFlags.add(new Integer(BasinPlotter.LAND_USE));;
               listSupportFiles.add(fields[LANDTABLENAME]);
           }

           if(isActiveField[GWATERFILE]) {
               listModel.addElement(fields[GWATERFILE]);
               //TODO
           }

           if(isActiveField[BEDROCKFILE]) {
               listModel.addElement(fields[BEDROCKFILE]);
               //TODO
           }

           if(isActiveField[RAINFILE]) {
               listModel.addElement(fields[RAINFILE]);
               //TODO
           }

           if(isActiveField[HYDROMETGRID]) {
               listModel.addElement(fields[HYDROMETGRID]);
               //TODO
           }
        }

        public String toString()
        {
         for (int i=0; i<TOTALVARS; i++) {
             System.out.println(i+":" +labels[i] + " - " + fields[i]
                                      + " - " +isActiveField[i]);
         }
         return "done";
        }
}
```

```
//************************************
//
//   WatershedClass.java
//
//   Watershed Class Definition
//
//************************************
package tRIBS;


public class WatershedClass{
    private double[] Temp, DTemp, Precip, Wind, Sky, Day;
    private RasterGrid elevGrid, soilGrid, landGrid, currentState,
smGrid;
    private RasterGrid Drainage, NetPrecipitation, EvapoTrans;
    private RasterGrid SoilDepth;
    private double[] SoilValues, LandValues;
    private int numCatLand, numPropLand, numCatSoil, numPropSoil;
    private double[] cumIntercept;
    private double[] coeffA, coeffB, coeffZo, coeffRst, coeffKs,
coeffC;
    private int[] soilID, landID, soilProp, landProp, pastSoilMoist,
soilDepth;
    private int xsize, ysize;
    private int rainOpt, drainOpt, evapOpt, interOpt;

    protected static final String on = "on";
    protected static final String off = "off";

    public WatershedClass(){}

    public RasterGrid updateState(int day, RasterGrid pastState){
      int sizeLand = (int)((LandValues.length)/numPropLand);
      int sizeSoil = (int)((SoilValues.length)/numPropSoil);
      double[] soilMoisture = new double[pastState.size()];
      int[] intSoilMoist = new int[pastState.size()];
      double porosity = 0.40;
      double factor = 0.03937;

      if(day==1){
          int size = (int)((LandValues.length)/(numPropLand-1));

          coeffA = new double[size];
          coeffB = new double[size];
          coeffA = extractData(LandValues,numCatLand,numPropLand,0);
          coeffB = extractData(LandValues,numCatLand,numPropLand,1);

          landID = new int[size];
          landID = extractID(LandValues,numCatLand,numPropLand);
          landProp = new int[landGrid.size()];
          landProp = landGrid.getPixelArray();

          soilID = new int[size];
          soilID = extractID(SoilValues,numCatSoil,numPropSoil);
          soilProp = new int[soilGrid.size()];
          soilProp = soilGrid.getPixelArray();
```

```
            coeffZo = new double[size];
            coeffRst = new double[size];
            coeffZo = extractData(LandValues,numCatLand,numPropLand,2);
            coeffRst = extractData(LandValues,numCatLand,numPropLand,3);

            coeffKs = new double[size];
            coeffC = new double[size];
            coeffKs = extractData(SoilValues,numCatSoil,numPropSoil,0);
            coeffC  = extractData(SoilValues,numCatSoil,numPropSoil,1);

            SoilDepth = new RasterGrid(elevGrid.getX(),elevGrid.getY());
            SoilDepth = getSoilDepth();
            soilDepth = new int[elevGrid.size()];
            soilDepth = SoilDepth.getPixelArray();
        }

        System.out.println("Day = " +day);

        int[] currentDrain, currentEvap, currentPrec;

        NetPrecipitation = new
RasterGrid(landGrid.getX(),landGrid.getY());
        NetPrecipitation = getInterception(day);
        currentPrec = new int[NetPrecipitation.size()];
        currentPrec = NetPrecipitation.getPixelArray();

        EvapoTrans = new RasterGrid(landGrid.getX(),landGrid.getY());
        EvapoTrans = getEvapoTranspiration(day);
        currentEvap = new int[EvapoTrans.size()];
        currentEvap = EvapoTrans.getPixelArray();

        Drainage = new RasterGrid(elevGrid.getX(),elevGrid.getY());
        Drainage = getDrainage(pastState);
        currentDrain =  new int[Drainage.size()];
        currentDrain = Drainage.getPixelArray();

        pastSoilMoist = pastState.getPixelArray();

        for(int i=0; i<NetPrecipitation.size();i++){
            if(rainOpt == 1 && evapOpt == 1 && drainOpt == 1){
              soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*
                  (currentPrec[i] - currentDrain[i] - currentEvap[i]);
            }
            if(rainOpt == 1 && evapOpt == 1 && drainOpt == 0){
              soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*
                  (currentPrec[i] - currentEvap[i]);
            }
            if(rainOpt == 1 && evapOpt == 0 && drainOpt == 1){
              soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*
                  (currentPrec[i] - currentDrain[i]);
            }
            if(rainOpt == 1 && evapOpt == 0 && drainOpt == 0){
```

```
            soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*(currentPrec[i]);
          }
          if(rainOpt == 0 && evapOpt == 0 && drainOpt == 0){
            soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]));
          }
          if(rainOpt == 0 && evapOpt == 1 && drainOpt == 0){
            soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*(-currentEvap[i]);
          }
          if(rainOpt == 0 && evapOpt == 1 && drainOpt == 1){
            soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*(-currentEvap[i]-
currentDrain[i]);
          }
          if(rainOpt == 0 && evapOpt == 0 && drainOpt == 1){
            soilMoisture[i] =
pastSoilMoist[i]+(1/(porosity*factor*soilDepth[i]))*(-currentDrain[i]);
          }
          if(soilMoisture[i] < 0.0)
            soilMoisture[i] = 0.0;
          intSoilMoist[i] = (int)(soilMoisture[i]);
      }

      currentState = new RasterGrid(elevGrid.getX(),elevGrid.getY());
      currentState.setAllPixels(intSoilMoist);

      soilMoisture = null;
      intSoilMoist = null;
      currentPrec = null;
      currentDrain = null;
      currentEvap = null;
      NetPrecipitation = null;
      EvapoTrans = null;
      Drainage = null;
      SoilDepth = null;
      System.gc();

      return currentState;
    }


    public RasterGrid getInterception(int day){
      int size = (int)((LandValues.length)/(numPropLand-1));
      RasterGrid NetPrecipitation = new
RasterGrid(landGrid.getX(),landGrid.getY());
      double[] intercept = new double[size];
      double[] netPrecip = new double[landGrid.size()];
      int[] NPrec = new int[landGrid.size()];
      double beta = 0.25;

      double[] rain = new double[Precip.length];
      rain = getPrecip();

      if(day == 1 || rain[day-1]==0){
          if(day==1)
```

```
            cumIntercept = new double[size];
          for(int ct = 0;ct<coeffA.length;ct++){
            cumIntercept[ct] = 0;
          }
      }

      for(int ct = 0; ct<coeffA.length; ct++){
          if(cumIntercept[ct] <= coeffA[ct] && rain[day-1]<0.05){
            intercept[ct] = rain[day-1];
          }
          else{
            intercept[ct] = beta*coeffB[ct]*rain[day-1];
          }
      }

      for(int ct = 0;ct<coeffA.length;ct++){
          cumIntercept[ct] = cumIntercept[ct]+intercept[ct];
      }

      for(int i = 0;i<landGrid.size();i++){
          if(landProp[i]!=0){
            for(int ct = 0;ct<size;ct++){
                if(landProp[i] == landID[ct]){
                  if(interOpt == 1){
                      netPrecip[i] = rain[day-1]-intercept[ct];
                  }
                  else{
                      netPrecip[i] = rain[day-1];
                  }
                }
            }
          }
          else{
            netPrecip[i] = 0.0;
          }
      }

      for(int ct=0;ct<landGrid.size();ct++){
          NPrec[ct] = (int)(1000*netPrecip[ct]);
      }

      NetPrecipitation.setAllPixels(NPrec);

      rain = null;
      netPrecip = null;
      NPrec = null;
      System.gc();

      return NetPrecipitation;

    }

    public RasterGrid getEvapoTranspiration(int day){
      int size = (int)((LandValues.length)/(numPropLand-1));
      RasterGrid EvapoTrans = new
RasterGrid(landGrid.getX(),landGrid.getY());
      double[] newET = new double[landGrid.size()];
```

```
      int[] evapoTranspiration = new int[landGrid.size()];
      double[] coeffRa;
      double DTempK, TempK, Vel, Rain;
      double vPresTD, vPresTS, PresTD, PresTS, qTa, qTs;
      double zref = 2;
      double vonK = 0.41;
      double eps = 0.622;
      double Rd = 287.0;
      double Rv = 461.0;
      double rho = 1.3;
      double lambda = 2250000.0;
      double eo = 611.0;
      double To = 273.15;

      double[] wind = new double[Wind.length];
      wind = getWind();
      double[] temp = new double[Temp.length];
      temp = getTemp();
      double[] dtemp = new double[DTemp.length];
      dtemp = getDTemp();
      double[] rain = new double[Precip.length];
      rain = getPrecip();

      Rain = rain[day-1];
      Vel = wind[day-1]*0.44704;
      DTempK = 273.15+((5.0/9.0)*(dtemp[day-1]-32.0));
      TempK = 273.15+((5.0/9.0)*(temp[day-1]-32.0));

      vPresTD = eo*Math.exp((lambda/Rv)*((1/To)-(1/DTempK)));
      PresTD = rho*Rd*TempK + vPresTD;
      qTa = eps*vPresTD/PresTD;

      vPresTS = eo*Math.exp((lambda/Rv)*((1/To)-(1/TempK)));
      PresTS = rho*Rd*TempK + vPresTS;
      qTs = eps*vPresTS/PresTS;

      coeffRa = new double[size];
      for(int ct =0;ct<size;ct++){
          coeffRa[ct] =
Math.pow((Math.log(zref/coeffZo[ct])),2)/(vonK*vonK*Vel);
      }

      for(int i = 0;i<landGrid.size();i++){
          if(landProp[i]!=0){
            for(int ct = 0;ct<size;ct++){
                if(landProp[i] == landID[ct]){
                  if(Rain != 0.0){
                      newET[i] = (0.001)*(39.37)*(86400)*rho*(qTs-
qTa)*(1/(coeffRst[ct]+coeffRa[ct]));
                  }
                  else{
                      newET[i] = 0.0;
                  }
                }
            }
          }
          else{
```

```
            newET[i] = 0.0;
          }
        }
        for(int ct=0;ct<landGrid.size();ct++){
            evapoTranspiration[ct] = (int)(1000*newET[ct]);
        }
        EvapoTrans.setAllPixels(evapoTranspiration);

        newET = null;
        evapoTranspiration = null;
        wind = null;
        temp = null;
        dtemp = null;
        System.gc();

        return EvapoTrans;
    }


    public RasterGrid getDrainage(RasterGrid pastState){
        int size = (int)((SoilValues.length)/(numPropSoil-1));
        RasterGrid Drainage = new
RasterGrid(pastState.getX(),pastState.getY());
        int[] soilMoisture = new int[pastState.size()];
        int[] pastMoisture = new int[pastState.size()];
        double[] newMoisture = new double[pastState.size()];
        double beta = 0.01;

        pastMoisture = pastState.getPixelArray();
        for(int i = 0;i<pastState.size();i++){
            if(soilProp[i]!=0){
              for(int ct = 0;ct<size;ct++){
                  if(soilProp[i] == soilID[ct]){
                    if(pastMoisture[i]>1000){
                        newMoisture[i] = beta*coeffKs[ct];
                    }
                    else{
                        newMoisture[i] =
beta*coeffKs[ct]*Math.pow(((0.001)*pastMoisture[i]),coeffC[ct]);
                    }
                  }
              }
            }
            else{
              newMoisture[i] = 0.0;
            }
        }
        for(int ct=0;ct<pastState.size();ct++){
            soilMoisture[ct] = (int)(1000*newMoisture[ct]);
        }
        Drainage.setAllPixels(soilMoisture);

        soilMoisture = null;
        newMoisture = null;
        pastMoisture = null;
        System.gc();
```

```
      return Drainage;
    }

    public RasterGrid getSoilDepth(){
      RasterGrid SDepth = new
RasterGrid(elevGrid.getX(),elevGrid.getY());
      double percent;
      double[] depth = new double[elevGrid.size()];
      int[] elevation = new int[elevGrid.size()];
      int[] intdepth = new int[elevGrid.size()];
      int maxElev;
      double maxDepth;

      maxDepth = 2.0;
      maxElev = elevGrid.getElevMax();
      percent = maxDepth/maxElev;
      elevation = elevGrid.getPixelArray();

      for(int ct = 0; ct<elevGrid.size();ct++){
          depth[ct] = percent*elevation[ct];
          intdepth[ct] = (int)(1000*depth[ct]);
      }

      SDepth.setAllPixels(intdepth);

      depth = null;
      elevation = null;
      intdepth =null;
      System.gc();

      return SDepth;
    }


    public double[] extractData(double[] values, int numCat, int
numProp, int flag){
      double[] parameter = new double[numCat];
      int count = 0;

      if(flag == 0){
          for(int ct = 1;ct<values.length;ct=ct+(numProp-1)){
            parameter[count] = values[ct];
            count++;
          }
      }
      else if(flag == 1){
          for(int ct = 2;ct<values.length;ct=ct+(numProp-1)){
            parameter[count] = values[ct];
            count++;
          }
      }
      else if(flag == 2){
        for(int ct = 3;ct<values.length;ct=ct+(numProp-1)){
            parameter[count] = values[ct];
            count++;
          }
      }
```

```
    else if(flag == 3){
      for(int ct = 4;ct<values.length;ct=ct+(numProp-1)){
          parameter[count] = values[ct];
          count++;
        }
    }

    return parameter;
  }

  public int[] extractID(double[] values, int numCat, int numProp){
    int[] id = new int[numCat];
    int count = 0;

    for(int ct = 0;ct<values.length;ct=ct+(numProp-1)){
          id[count] = (int)values[ct];
          count++;
        }

    return id;
  }


  public void setElevGrid(RasterGrid gridata){
    elevGrid = gridata;
    xsize = elevGrid.getX();
    ysize = elevGrid.getY();
  }

  public void setSMGrid(RasterGrid gridata){
    smGrid = gridata;
  }

  public void setSoilGrid(RasterGrid gridata){
    soilGrid = gridata;
  }

  public void setLandGrid(RasterGrid gridata){
    landGrid = gridata;
  }

  public void setSoil(double[] tableValues){
    SoilValues = tableValues;
  }

  public void setNumCatSoil(int numCat){
    numCatSoil = numCat;
  }

  public void setNumPropSoil(int numProp){
    numPropSoil = numProp;
  }

  public void setLand(double[] tableValues){
    LandValues = tableValues;
  }
```

```
public void setNumCatLand(int numCat){
  numCatLand = numCat;
}

public void setNumPropLand(int numProp){
  numPropLand = numProp;
}

public void setTemp(double[] temp){
  Temp = temp;
}

public void setDTemp(double[] dtemp){
  DTemp = dtemp;
}

public void setPrec(double[] precip){
  Precip = precip;
}

public void setWind(double[] wind){
  Wind = wind;
}

public void setCover(double[] cover){
  Sky = cover;
}

public void setDay(double[] day){
  Day = day;
}

public void setRainOption(String rain){
  if(rain.compareTo(on)==0)
      rainOpt = 1;
  else
      rainOpt = 0;
}

public void setDrainOption(String drain){
  if(drain.compareTo(on)==0)
      drainOpt = 1;
  else
      drainOpt = 0;
}

public void setInterOption(String inter){
  if(inter.compareTo(on)==0)
      interOpt = 1;
  else
      interOpt = 0;
}
public void setEvapOption(String evap){
  if(evap.compareTo(on)==0)
      evapOpt = 1;
  else
      evapOpt = 0;
```

```
    }

    public RasterGrid getSMGrid(){
      return smGrid;
    }

    public int getX(){
      return xsize;
    }

    public int getY(){
      return ysize;
    }

    public double[] getPrecip(){
      return Precip;
    }

    public double[] getWind(){
      return Wind;
    }

    public double[] getTemp(){
      return Temp;
    }

    public double[] getDTemp(){
      return DTemp;
    }

    public void setNull(){
      Drainage = null;
      NetPrecipitation = null;
      EvapoTrans = null;
      System.gc();
    }
}
```

```
//************************************
//
//   WatershedProgram.java
//
//   Main Window Program
//
//************************************
package tRIBS;

public class WatershedProgram{
    public static void main(String args[]){
      WatershedView BasinView = new WatershedView();
      //InfoView Introduction = new InfoView(-1);
    }
}
```

```
//***********************************
//
//   WatershedView.java
//
//   GUI Window Program
//
//***********************************
package tRIBS;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.ArrayList;

public class WatershedView extends JFrame implements Runnable,
ActionListener,
       SwingConstants, ListSelectionListener
{
    //UI Elements
    private JMenuBar menuBar;
    private JMenu Prep, Parameters, Inputs,  MExecute, MView, Compile,
MHelp;
    private JMenuItem prep_1, prep_2, prep_3, prep_4;
    private JMenuItem MNew, Print, Export, Close, MExit;
    private JMenuItem Table_p, Table_i;
    private JMenu Elev_i, Rain_i;
    private JMenuItem e_dem, e_tin, r_rain, r_nex, r_wsi;
    private JMenuItem Soil_i, Land_i, Bed_i, Channel_i, Gauge_i;
    private JMenuItem Opt_p, Flow_p, Rain_p, Hydro_p, Output_p;
    private JMenuItem Check, MRun, Time, Layer;
    private JMenuItem Animate, Pixel, TBD, MAbout;
    private InputView in_data, para_data;
    //private GoView goExecute;
    private JMenu File, Import, Run, Help, Basin, Case, TEST;
    private JMenuItem Water, Exit, Weather, Temp, Precip, Execute,
Params;
    private JMenuItem Elev, Land, Soil, DTemp, SCover, Wind, SM;
    private JMenuItem Instruct, About, MetPic, Location;
    private JMenuItem execute, importFile, importDEM;
    private JPanel contentPane;
    private JLabel picture, list;
    private JScrollPane pictureScrollPane, listScrollPane;
    private JSplitPane splitPane;
    //required: keep elements in listModel in sync with arraylists
    private DefaultListModel listModel;
    private ArrayList listRenderFlags, listSupportFiles,
listItemIsRendered;
    private JList fileList;
    private Border emptyBorder;
//    private JToolBar listToolBar;
//    private JButton listOkButton, listCancelButton;
```

```
//    private JButton elevButton, soilButton, landButton, startButton,
smButton;
//    private JButton rainButton, tempButton, dtempButton, windButton,
skyButton;

    private FileDialog importInFileLoadDialog;
    private InfoView aboutInfo, instructInfo;
    private ReadandPlotData PrecipData, TempData, SkyData, DTempData,
WindData;
    private InputView weatherData, basinData, runData;
    private BasinPlotter basinPlot;
    private PictureView MetPhoto, LocPhoto;
    private WatershedClass watershed;

    private int infoFlag, dataFlag, inputFlag, mapFlag, imageFlag,
picFlag, runFlag;
    private String rainURL, tempURL, dtempURL, skyURL, windURL,
elevURL, smURL;
    private String soilTable, landTable, soilURL, landURL, basinName;
    private double modelStart, modelEnd, basinLat, basinLong;
    private String rainOption, evapOption, drainOption, interOption;
    private TRIBSFileIO tribsIO;

    private ExecuteView executeView;

    private String tableFilePath =
"/Users/eklee/Desktop/forbones/asc_files/soilInfo.table";
    private String ascFilePath =
"/Users/eklee/Desktop/forbones/asc_files/squansoil.asc";
//    private String tableFilePath = "soilInfo.table";
//    private String ascFilePath = "squansoil.asc";


    Thread AnimThread, currentThread;

    public WatershedView(){
      super("tRIBS version 2.0");
      JPopupMenu.setDefaultLightWeightPopupEnabled(true);
      setSize(1000,750);
      rainURL = null;
      tempURL = null;
      tribsIO = new TRIBSFileIO();
      listRenderFlags = new ArrayList();
      listSupportFiles = new ArrayList();
      listItemIsRendered = new ArrayList();

      setMenuBar();
//      setToolBar();

      contentPane = new JPanel();
      contentPane.setLayout(new BorderLayout());

      //     setSideToolBar();
      //setBasinPlot();
      //      setWatershed();
      listModel = new DefaultListModel();
      fileList = new JList(listModel);
```

```
      fileList.addListSelectionListener(this);

      basinPlot = new BasinPlotter();

      listScrollPane = new JScrollPane(fileList);
      pictureScrollPane = new JScrollPane(basinPlot);

//      basinPlot.setImage(1, ascFilePath, tableFilePath);

      //set empty borders on the scroll panes
      emptyBorder = BorderFactory.createEmptyBorder();
      listScrollPane.setBorder(emptyBorder);
      pictureScrollPane.setBorder(emptyBorder);

//      listPane.add(listScrollPane,"Center");
//      listPane.add(listToolBar,"North");

      splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                listScrollPane, pictureScrollPane);
                //listPane, pictureScrollPane);
      splitPane.setOneTouchExpandable(true);
      splitPane.setDividerLocation(150);

      //Provide minimum sizes for the two components in the split pane
      Dimension minimumSize = new Dimension(200, 100);
      listScrollPane.setMinimumSize(minimumSize);
      pictureScrollPane.setMinimumSize(minimumSize);

      //Provide a preferred size for the split pane
      splitPane.setPreferredSize(new Dimension(400, 200));

      setContentPane(splitPane);
      setVisible(true);

      addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
            dispose();
            System.out.println("exiting tRIBS2.0");
            System.exit(0);
          }
      });//end addWindowListener
    }//end WatershedView()


    public void setMenuBar(){
        menuBar = new JMenuBar();

        File = new JMenu("File");
        File.setToolTipText("Program Functions");
        menuBar.add(File);
//      Case = new JMenu("Case Study");
//      Case.addActionListener(this);
//      Location = new JMenuItem("Watershed Location Map");
//      Location.addActionListener(this);
//      MetPic = new JMenuItem("Meteorological Station Photo");
//      MetPic.addActionListener(this);
//    Case.add(Location);
```

```
//    Case.add(MetPic);
//    File.add(Case);
//    File.addSeparator();


      //added-->
//      File.addSeparator();
        MNew = new JMenu("New");
        MNew.addActionListener(this);
        File.add(MNew);
//      Print = new JMenuItem("Print");
//      Print.addActionListener(this);
//      File.add(Print);
//      Export = new JMenuItem("Export");
//      Export.addActionListener(this);
//      File.add(Export);
        Close = new JMenuItem("Close");
        Close.addActionListener(this);
        File.add(Close);

        File.addSeparator();

        execute = new JMenuItem("Execute...");
        execute.addActionListener(this);
        File.add(execute);

        File.addSeparator();

        importFile = new JMenuItem("Import .in file...");
        importFile.addActionListener(this);
        File.add(importFile);

        importDEM = new JMenuItem("Import DEM...");
        importDEM.addActionListener(this);
        File.add(importDEM);

        File.addSeparator();

        Exit = new JMenuItem("Exit");
        Exit.addActionListener(this);
        File.add(Exit);

      //****Parameters****
      Parameters = new JMenu("Parameters");
      Parameters.addActionListener(this);
      menuBar.add(Parameters);
      //##
      Table_p = new JMenuItem("Parameters Entry");
      Table_p.addActionListener(this);
      Parameters.add(Table_p);
      Parameters.addSeparator();

      //****Help****
      MHelp = new JMenu("Help");
      MHelp.addActionListener(this);
      menuBar.add(MHelp);
      MAbout = new JMenuItem("About tRIBS v2.0");
```

```
      MAbout.addActionListener(this);
      MHelp.add(MAbout);

      setJMenuBar(menuBar);
    }

    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) {
            return;
        }

        JList theList = (JList)e.getSource();
        if (theList.isSelectionEmpty()) {

        } else {
            System.out.println("picked value"
+theList.getSelectedValue());
            //todo:
            int listIndex = theList.getSelectedIndex();

            if (((Boolean)
listItemIsRendered.get(listIndex)).booleanValue() == false) {
               basinPlot.setBasinName("     ");
               System.out.println("basinPlot.setImage(" +
                   ((Integer)
listRenderFlags.get(listIndex)).intValue() + ","
                   + (String) theList.getSelectedValue() + ","
                   +(String) listSupportFiles.get(listIndex) +")");
               basinPlot.setImage(((Integer)
listRenderFlags.get(listIndex)).intValue(),
                                                  (String)
theList.getSelectedValue(),
                                                  (String)
listSupportFiles.get(listIndex));
               listItemIsRendered.set(listIndex, new Boolean(true));
            } else {
                System.out.println("basinPlot.setCurrentImage("
                    + ((Integer)
listRenderFlags.get(listIndex)).intValue() + ")");
                basinPlot.setCurrentImage(((Integer)
                    listRenderFlags.get(listIndex)).intValue());
            }
        }
    }

    public void setWatershed(){
     watershed = new WatershedClass();
    }

    public int getRunFlag(){
      return runFlag;
  }

    public void setRunFlag(int rFlag){
      runFlag = rFlag;
    }
```

```
    public void run(){
      Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
      Thread currentThread = Thread.currentThread();
      int flag = 1;
      int day;
      RasterGrid currentState, pastState;
      pastState = new RasterGrid(watershed.getX(),watershed.getY());
      Image currentImg;
      int startDay, endDay;

      startDay = (int)modelStart;
      endDay = (int)modelEnd;
      day = startDay;

    }
    public void actionPerformed(ActionEvent evt){
      Object src = evt.getSource();

      if(src == Exit){
          System.out.println("Exiting Watershed Simulator");
          System.exit(0);
      }
      else if(src == execute) {
          executeView = new ExecuteView();
          //TODO - do something here
      }
      //TODO - rename Table_p;  this imports .in file
      else if(src == Table_p){
          in_data = new InputView(tribsIO);
//          tribsIO.toString();
            System.out.println("table_p");

tribsIO.appendToFileListModel(listModel,listRenderFlags,listSupportFile
s);
          listItemIsRendered.add(new Boolean(false));
          fileList.repaint();
      }
      else if(src == importDEM) {
          importInFileLoadDialog = new FileDialog(this, "Import DEM
File",
                                      importInFileLoadDialog.LOAD);

          importInFileLoadDialog.show();

          if(importInFileLoadDialog.getFile() != null) {
              try {

listModel.addElement(importInFileLoadDialog.getDirectory() +

importInFileLoadDialog.getFile());
                    listRenderFlags.add(new
Integer(BasinPlotter.ELEVATION));
                    listItemIsRendered.add(new Boolean(false));
                    listSupportFiles.add(null);
                    System.out.println("File imported from: " +

importInFileLoadDialog.getFile());
```

```
                    fileList.repaint();
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(this,
                        "Import failed\n\n" + e.getMessage(),
                        "File Import Error",
                        JOptionPane.ERROR_MESSAGE);
                }
            } else {
                System.out.println("Import Cancelled");
            }
        }
        else if(src == importFile) {
            importInFileLoadDialog = new FileDialog(this, "Import .in
File",
                                    importInFileLoadDialog.LOAD);

            importInFileLoadDialog.show();

            if(importInFileLoadDialog.getFile() != null) {
                try {
                    tribsIO = new TRIBSFileIO();

tribsIO.importFile(importInFileLoadDialog.getDirectory() +

importInFileLoadDialog.getFile());
                    //tribsIO.toString();
                    //Fixme...UI update
                    System.out.println("File imported from: " +
                       importInFileLoadDialog.getFile());
                    tribsIO.appendToFileListModel(listModel,
listRenderFlags,

listSupportFiles);
                    listItemIsRendered.add(new Boolean(false));
                    fileList.repaint();

                    tribsIO.toString();
                } catch (Exception e) {
                    e.printStackTrace();
                    JOptionPane.showMessageDialog(this,
                        "Import failed\n\n" + e.getMessage(),
                        "File Import Error",
                        JOptionPane.ERROR_MESSAGE);
                }
            } else {
                System.out.println("Import Cancelled");
            }
        }

    }//end ActionPerformed

}
```