THE MODULAR MODELING SYSTEM (MMS): USER'S MANUAL

Updated for software version 1.1 (March 1998)

By G.H. Leavesley, P.J. Restrepo, S.L. Markstrom, M. Dixon, and L.G. Stannard

U.S. GEOLOGICAL SURVEY

Open-File Report 96-151

Prepared in cooperation with the

UNIVERSITY OF COLORADO AT BOULDER, CENTER FOR ADVANCED DECISION SUPPORT FOR WATER AND ENVIRONMENTAL SYSTEMS (CADSWES)

Denver, Colorado

1996

THE MODULAR MODELING SYSTEM (MMS): USER'S MANUAL

By G.H. Leavesley, P.J. Restrepo, S.L. Markstrom, M. Dixon, and L.G. Stannard

ABSTRACT

The Modular Modeling System (MMS) is an integrated system of computer software that has been developed to provide the research and operational framework needed to support development, testing, and evaluation of physical-process algorithms and to facilitate integration of user-selected sets of algorithms into operational physical-process models. MMS uses a module library that contains modules for simulating a variety of water, energy, and biogeochemical processes. A model is created by selectively coupling the most appropriate modules from the library to create a "suitable" model for the desired application. Where existing modules do not provide appropriate process algorithms, new modules can be developed. The MMS user's manual provides installation instructions and a detailed discussion of system concepts, module development, and model development and application using the MMS graphical user interface.

PREFACE

The development of the Modular Modeling System (MMS) began with the establishment of a 3-year cooperative agreement between the U.S. Geological Survey and the University of Colorado at Boulder's Center for Advanced Decision Support for Water and Environmental Systems (CADSWES) in September 1989. The major components of MMS were identified at that time but limited funding required that component development be prioritized, with lower priority components to be developed later, as funding became available.

The problems of limited funding for system development were addressed by the addition of several collaborators who saw the potential of MMS for their own applications and research. Contributions to the system by these groups were made in terms of funding and in manpower to develop selected components of the system. Those contributing resources were the U.S. Bureau of Reclamation, U.S. Forest Service, National Aeronautics and Space Administration, Agricultural Research Service, and the Terrestrial Ecology Regional Research and Analysis (TERRA) Laboratory. International contributions were made from several groups in Germany whose research funding was supported by the federal Deutscheforschungsgemeinschaft (DFG). Contributing to MMS development were the Rheinisch-Westfälische Technische Hochschule in Aachen, the University of Bonn in Bonn, the Friedrisch Schiller University in Jena, and the Potsdam Institute for Climate Impact Analysis in Potsdam.

While the concepts and components of MMS were conceived by the senior author, and the initial basic structure of MMS was designed, coded, and implemented by P.J. Restrepo and M. Dixon, bringing these ideas and code to fruition entailed the knowledge, skills, resources, and labor of a large number of people from a variety of groups and agencies. Thus, while this user's manual is authored by those who played the major part in writing it, the contents reflect the efforts of many. It is not possible to list all the individuals who have contributed their time and talents to the design, development and testing of MMS, but the authors wish to give their wholehearted thanks to all who have made this system possible.

1. INTRODUCTION

1.1 Purpose of MMS

To clearly define what the Modular Modeling System (MMS) is, it may be best to begin with a statement of what it is not. MMS is not a model. Rather, it is a framework for modeling that can be used to develop, support, and apply any dynamic model. A major focus of the development of MMS has been for models used in the environmental and natural-resource management disciplines, but MMS could feasibly be used in any number of other scientific disciplines.

Thus, MMS is an integrated system of computer software developed to (1) provide the research and operational framework needed to enhance development, testing, and evaluation of physical-process algorithms; (2) facilitate integration of user-selected algorithms into operational physical-process models; and (3) provide a common framework in which to apply historic or new models and analyze their results. MMS uses a module library that contains modules for simulating a variety physical processes. In the context of current development work, these are water, energy, chemical, and biological processes. A model is created by selectively coupling appropriate modules from the library to create a suitable model for a desired application. When existing modules do not provide appropriate process algorithms, new modules can be developed.

The interdisciplinary nature and increasing complexity of environmental and water-resource problems require the use of modeling approaches that can incorporate knowledge from a broad range of scientific disciplines. Selection of a model to address these problems is difficult given the large number of available models and the potentially wide range of applications, data constraints, and spatial and temporal scales of application. Coupled with these issues are the problems of study area characterization and parameterization after a model is selected. Guidelines for parameter estimation are sparse, and the user commonly has to make decisions based on an incomplete understanding of the relation between parameter values and physical measures of watershed characteristics. MMS addresses these issues by enabling the development of models that are problem and scale specific and by providing the capability to evaluate alternative modeling and parameter estimation approaches.

MMS provides a common framework in which to focus multidisciplinary research and operational efforts. Researchers in a variety of disciplines can work cooperatively on multidisciplinary problems. Each can develop and test model components in their own areas of expertise and combine these modules with those of the other researchers to develop a complete system model. In addition, as research provides improved model components, these can be used to modify or enhance existing operational models by inserting or replacing process modules.

A geographic information system (GIS) interface is provided to facilitate model development, application, analysis, and visualization utilizing map information for a spatially distributed system. This interface permits application of a variety of GIS tools to

characterize the physical, hydrological, chemical, and biological features of a physical system for use in a variety of lumped- and distributed-parameter modeling approaches. The GIS currently being used is the Geographical Resources Analysis Support System (GRASS) developed by the U.S. Army Corps of Engineers (1991). MMS display capabilities permit visualization of the spatial distribution of model parameters and of the spatial and temporal variation of simulated state variables during a model run. An alternative display feature presents more comprehensive two- and three- dimensional views at the end of a run.

Continued advances in physical and biological sciences, GIS technology, computer technology, and data resources will expand the need for a dynamic set of tools to incorporate these advances in a wide range of interdisciplinary research and operational applications. MMS is being developed as a flexible framework in which to integrate these activities.

1.2 Overview

MMS has three major components: pre-process, model, and post-process (fig. 1.1). The pre-process component includes tools used to input, analyze, and prepare spatial and time-series data for use in model applications. The model component includes the tools to develop and apply models. The post-process component provides tools to display and analyze model results, and to pass results to management models or other types of software. A system supervisor, in the form of an X-window graphical user interface (GUI), provides users access to all components and features of MMS. MMS was developed for UNIX-based workstations and uses X-windows and Motif for the GUI.

1.2.1 Pre-process component

Pre-process component functions include all the data preparation, parameterization, and analysis functions needed to run the selected model. Spatial data analysis is accomplished using a GRASS-based GIS toolset developed for MMS or other user-selected GIS toolsets. In the GRASS-based system, the user is provided access to frequently used GRASS tools and other system GIS tools developed specifically for a variety of modeling tasks. Available spatial data analyses include segmentation and characterization of a watershed into subareas for distributed-parameter modeling. Digital elevation model (DEM) data and selected digital databases that include information on soils, vegetation, geology, and other pertinent physical features provide the data on which such characterizations are developed.

Databases used to store spatial and time-series data provide the interface between the pre-process and model components. The current (1995) time-series database structure used to interface with MMS is an ASCII flat file.

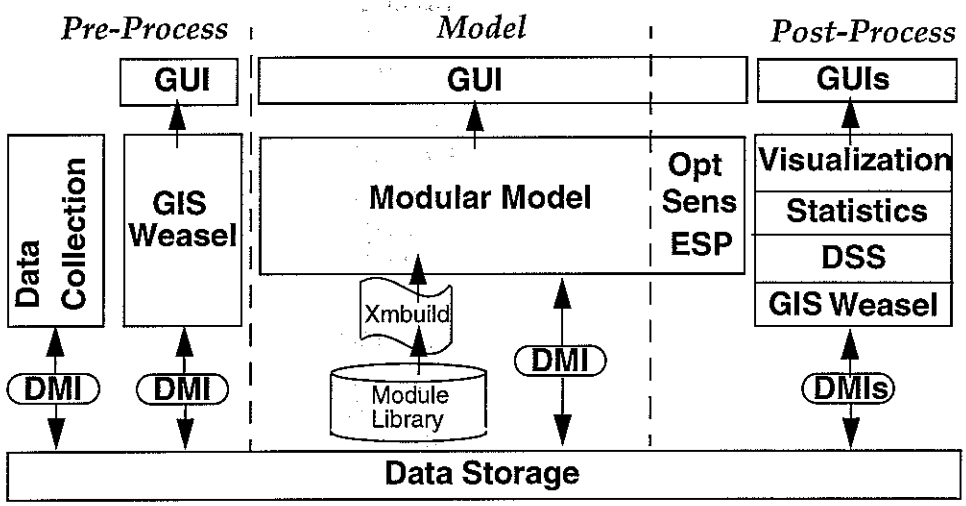

Figure 1.1. A schematic diagram of the conceptual components of the Modular Modeling System (MMS).

1.2.2 Model component

The model component is the core of the system and includes the tools to selectively link process modules from the module library to build a model, and to interact with this model to perform a variety of simulation and analysis tasks. These interactions are provided using a variety of X-window and graphical techniques.

A major feature of the model component is the module library which contains a variety of modules for simulating water, energy, chemical, and biological processes. For a given process, the library may contain several modules, each representing an alternative conceptualization or approach to simulating that process. The user, through an interactive model builder interface (*xmbuild*), selects and links modules to create a specific model. Once a model has been built, it may be saved for future use without repeating the *xmbuild* step. This capability allows complete versions of models to be provided to end users.

User interaction with model preparation and execution is accomplished using an X-windows GUI. A series of pull-down menus in the GUI provides the links to a variety of system features. These include the ability to (1) select and edit parameter files and data

files; (2) select a number of model execution options such as a standard run, a run with a graphical and/or a spatial visualization output, a prediction run, an optimization, or a sensitivity analysis; and (3) select a variety of statistical and graphical analyses of the simulation output.

1.2.3 Post-process component

The post-process component provides the tools to analyze and apply model results. These include a variety of statistical and graphical tools as well as the ability to interface with user-developed tools. Statistical and graphical analysis procedures provide a common basis for comparing module and model performance and can be used to aid in making decisions regarding the most appropriate modeling approach for a given set of study objectives, data constraints, and temporal and spatial scales.

One graphical tool enables the user to display output in up to four windows during a model run. Any variable that has been declared in a module can be plotted. As many as ten variables can be displayed in each window, and display results can be output, in HPGL or PostScript formats, to a digital file or to a printer. After a model run, the user has the option of using enhanced graphics, statistics, and three-dimensional plotting capabilities to analyze model results. A GIS interface also provides the visualization and analysis tools to display spatially distributed model results and to analyze results within and among different simulation runs.

Two post-processing tools interact with the model component. Parameter-optimization and sensitivity-analysis tools are provided to optimize selected model parameters and evaluate the extent to which uncertainty in model parameters affects uncertainty in simulation results. A third tool enables the user to use the assembled model in an extended prediction mode. At the present time, this consists of a modified version of the National Weather Service's Extended Streamflow Prediction Program (ESP) (Day, 1985) that provides forecasting capabilities using historic or synthesized meteorological data.

1.3 Manual structure

For the purposes of this manual, three levels of MMS use have been identified: (1) run existing models, (2) build and run new models using existing modules, and (3) write new modules for use in building and running models. As one moves from the first through the third levels, an increasing amount of knowledge is required about the structure and concepts of MMS and about the basic structure and design of modules. This more detailed information is presented in successive chapters.

Documentation for the various components of MMS will be released later. This volume focuses on the features of the model component (fig. 1.1). It is the basic User's Manual for MMS and *xmbuild* and includes information for all levels of use.

Future documentation will focus on the pre-process and post-process components of MMS (fig. 1.1) and selected special features of MMS. Component features to be documented include (1) *xmdoc*, an aid to users for converting existing code to modules

for inclusion in the MMS modular library, (2) an enhanced graphics package that can be implemented after an MMS run to study two- and three-dimensional aspects of the output, (3) *GIS Weasel*, the GIS interface tool that uses ARC/INFO to assist in the characterization and parameterization of spatially distributed watershed features and in the visualization of distributed-model inputs and outputs, and (4) feedback mechanisms that enable two or more processes to iterate, within a time step, when the solution to the system of algorithms, which reside in more than one module, is dependent upon mutual state variables.

1.4 Font conventions used in this manual

Helvetica italics font is used for:

UNIX pathnames, filenames, program names, user command names, and options for user commands.
New terms where they are defined.

`Courier` font is used for:

Examples of source code and text on the screen.
Queries and comments provided by system programs and setup scripts.
Names of subroutines of example programs.
Names of variables in library functions.

`Courier bold` font is used for:

User keyboard responses to system generated queries.

Times italic bold font is used for:

Mouse button selection for user-selectable options.
To show keystrokes used to selected various system options.

2. INSTALLATION

2.1 Obtaining MMS

MMS is available via the MMS internet homepage at *http://wwwbrr.cr.usgs.gov/mms.* MMS has been pre-compiled for a wide variety of platforms. Each platform is clearly identified as part of the tar file name. If the desired platform tar file is not present, MMS may be compiled from the *mms-x.x.tar.gz* version. All tar files contain the full source code regardless of any additional pre-compiled files. Software and hardware requirements are discussed in Attachment 1.

2.2 Installation Overview

The MMS directory structure is based on the use of two distinct directory areas. One is the MMS *master directory* area and the second is the MMS *user workspace directory* area (fig. 2.1). The master directory area is a read-only area that contains the basic system programs and files. This area also contains those modules of the module library, and selected models, that have received some level of quality control as defined by the using organization. All users share the files, libraries, modules, models, and other executables in the master directory area. Changes or additions to this area should be made only by persons familiar with system structure. New modules and models may be added as they receive the required testing and approval.

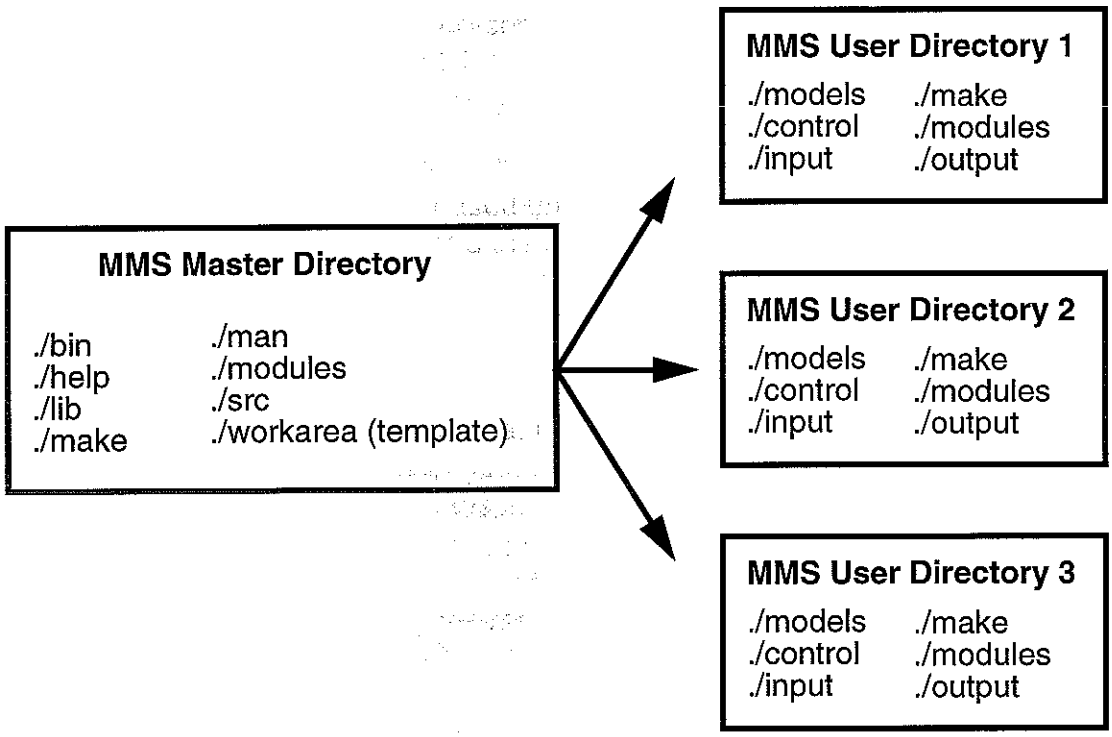

Figure 2.1 MMS master and user directories for a system of three users.

The user workspace directory area is where a user stores and manages individual data and parameter files, modules, models, and selected modifications to the MMS graphical user interface. A user workspace directory can be created for each individual using the system. The user workspace directory should not be created under the master directory because the master directory and its subdirectories are read-only.

The installation procedures involve the creation of one master directory and as many user workspace directories as needed. A set of scripts are provided for the installation of each directory type to assist the user in installing MMS. The installation scripts and procedures for each directory type are discussed in detail in the following sections.

2.3 Creating the MMS Master Directory

The MMS master directory may be created in any parent directory on the workstation system. However, all MMS users must be able to read from that location. Put the tar file in the desired parent directory. The tar file must be uncompressed before it can be extracted. Tar files with the extension ".Z" need the utility *uncompress*, whereas those with a ".gz" extension need *gzip -d*. MMS should be extracted from the tar file with the Unix command *tar -xvf <mms.platform>.tar* where <mms.platform> is the manufacturer name of the workstation hardware. For example, *SUN4.tar* is the precompiled version of MMS for a SUN workstation using the SUN OS4.1.x operating system. The directories should be extracted such that all MMS users have read access.

The major directories in the master directory area and their contents are:

The *bin* directory:

- the model builder *xmbuild*
- the workspace installation script *mms_workspace*
- the post-processing programs *XMGR, pdraw,* and *GISanimator*
- the file conversion scripts *control_update, param_update,* and *data_update*
- the executable code for selected models that have received some level of quality control as defined by the using organization.

The *help* directory:

- the MMS system documentation.

The *lib* directory:

- the MMS libraries *libXbae.a, libUNtil.a, libmms.a, libutil.a,* and *libPlotW.a.*

The *make* directory:

- the *makelist* file
- test C programs
- test FORTRAN programs
- scripts called by the *setup* script.

The *man* directory:

- Unix style *man* pages for the MMS library functions.

The *modules* directory:

- all modules that are selected to be read-only. A subdirectory is defined for each process type.:

The *src* directory:

- subdirectories with the source code for the components of MMS.

The *workarea* directory:

- a template of the directories that are copied to the user's work area by the *mms_workspace* script.

The *src* directory contains all the source code subdirectories for MMS. Each subdirectory has a *makefile* that is used to compile the source code in that directory and place the results into either the *bin* or *lib* directories. If changes are made to the source code in one of these subdirectories, only the *makefile* in that subdirectory needs to be executed to update MMS.

Also located in the *src* directory is an *include* subdirectory that contains a number of include files that are used by source code in one or more of the *src* subdirectories. If a change is made to any of these include files, then all the *src* subdirectories whose source code use the modified include files must be remade using the appropriate *makefiles*.

There is a *makefile* in the *mms* directory that will execute the *make* command in all of the *src* subdirectories. Running this *makefile* will build the entire MMS system. Prior to using this *make* option, a *make clean* command should be executed from the *mms* directory. This command removes all object code, libraries, and executables.

The *modules* directory is a read-only directory in order to provide a method to distribute tested or quality assured module code that is intended to remain constant within the system. However, if modification of one or more of these modules is desired, this is still possible. The user can copy these modules to a user workspace, modify them as desired, and construct a new model using any combination of modules from the master directory and the user workspace directory.

2.4 The MMS Setup Script

The MMS *setup* script is located in the *mms* directory. The primary function of this script is to create the *makelist* file in the *mms/make* directory. This file is included by every *makefile* in MMS; thus, it will be referenced every time an MMS library, executable, or model is compiled. The purpose of the *makelist* file is to provide a common, consistent, and unambiguous set of instructions for compilation of MMS regardless of specific hardware, operating system, or window system installations.

This *setup* script should be run whenever (1) MMS is initially installed, (2) the MMS master directory is moved, (3) there is a change in the FORTRAN or C compilers, or (4) there is an update in the X11 or Motif libraries.

The *setup* script is written in the Korn shell. For operating systems that do not support the Korn shell, the script may be run under the Bourne shell by changing the first line in the file from:

```
#!/bin/ksh
```

to

```
#!/bin/sh
```

The script will ask questions about compilers and libraries. It will verify responses and insure that they are valid before proceeding onto the next question. Once the script is running, it may be terminated using a **control-C** command at any time without any adverse affects, if the proper response is not known or an incorrect response has been provided. The *makelist* file is written (or overwritten) only after all questions are answered satisfactorily. In addition, once all questions are answered, the script will inquire about compiling the MMS libraries and executables. Versions of MMS with pre-compiled libraries and executables do not need to be remade. The following is an example application of the *setup* script:

```
(ksh) runoff.pts% setup
This is the MMS system level installation script.
Installing to MMS master directory /home1/mms.
```

If this is not the correct location for MMS, exit the script now using a **control-C** command and move the MMS directory to the correct location. Restart the script.

```
You are running SunOS release 5.6
Please check your path for the placement of '/usr/ucb'.

This directory must come after the other important
solaris directories for proper installation of MMS.
```

```
ARC flag has been set to SOLARIS
```

The script will then determine the machine type and operating system. Environment variables and default values for subsequent questions will be set based on the system architecture.

```
C compiler to use [gcc]: cc
C compiler looks good.
```

A C compiler is required for MMS. The system is written in portable C code, but some of the C library include files may vary from compiler to compiler. The compiler used to develop MMS was the Gnu C compiler *gcc.* However, in the example, the compiler was switched to the solaris C compiler to take advantage of debugging tools.

```
FORTRAN compiler to use [f77]: <return>
make/main.f:
 MAIN test:
FORTRAN compiler looks good.
```

FORTRAN is not necessarily required for MMS, but the demonstration models included in the tar file and described in the help files all require a FORTRAN compiler. However, modules written entirely in C may be compiled into models without the need of a FORTRAN compiler.

```
Checking your compiler for ANSI compliance.

You are OK.

Found X11 include path -- /usr/openwin/share/include

X include path to use [/usr/openwin/share/include]: <return>
XINCPATH=/usr/openwin/share/include
Found Motif Include path -- /usr/dt/include

Motif Include path to use [/usr/dt/include]: <return>
XMINCPATH=/usr/dt/include
Found The X11 library -- /usr/openwin/lib

X11 library path to use [/usr/openwin/lib]: <return>
Found The Intrinsics library -- /usr/openwin/lib
```

```
Intrinsics library path to use [/usr/openwin/lib]: <return>
Found The Motif library -- /usr/dt/lib

Motif library path to use [/usr/dt/lib]: <return>
```

X11 and Motif must be installed on the system. If the system does not have these include files, it is not possible to recompile the MMS system executables. If the system does not have these libraries, it is not possible to make any models with MMS. If the files are not located in the standard locations, it is necessary to supply the paths to the script.

```
Specify the flags to be passed to the C-compiler.
RETURN - to accept defaults.

COMPILE_FLAGS [-O -DSOLARIS -DSYSV -DUSE_DIRENT]: <return>
```

Change the "-O" to "-g" for a debuggable version of MMS.

```
I will check the compile flags now
Please watch for error messages
Check complete
Any problems? n
COMPILE_FLAGS = -O -DSOLARIS -DSYSV -DUSE_DIRENT
```

The script will compile a test file with the specified C flags. If there is an error, correct it by resetting the flags to the proper values for the target system.

```
Specify the flags to be passed to the FORTRAN compiler.
RETURN - to accept defaults.

FORTRAN_FLAGS [-u]: <return>

I will check the FORTRAN flags now
Please watch for error messages
12522.f:
MAIN test:

Check complete
Any problems? n
COMPILE_FLAGS = -u
```

The script will compile a test file with the specified FORTRAN flags. If there is an error, correct it by resetting the flags to the proper values for the target system.

```
Specify the flags to be passed to the loader (ld).
RETURN - to accept defaults.

LDFLAGS [-s]:<return>
LDFLAGS = -s

Specify the flags to be passed to the compiler that are
required specifically for X Window-based programs. Some systems
require extra compiler flags to increase the symbol table size,
when compiling for X Windows.
RETURN - to accept defaults.

XCFLAGS []: <return>


I will check the loader flags now
Please watch for error messages

Check complete
Any problems? n
XCFLAGS =

Specify the flags to be passed to the loader (ld) that are
required specifically for X Window-based programs. Some systems
require extra loader flags when compiling for X Windows.
RETURN - if no flags are to be specified.

XLDFLAGS []: <return>
XLDFLAGS =

Some system require additional libraries when compiling X Window pro-
grams.

Specify the additional libraries needed when compiling X Window pro-
grams.
```

```
Consult your installation guide if you are unsure of the need for these
libraries.
RETURN - if there are no extra libraries needed.

XEXTRALIBS [-lsocket -lnsl]: <return>
Checking -lsocket -lnsl. Please watch for any errors

Check complete
Any problems? n
```

Default values for these "extra" X Window flags and libraries will be set when the operating system type is set. Always try the default values first. If additional compiler options are necessary, specify them here.

```
Specify the math library.
RETURN - to accept defaults.

MATHLIB [-lm]: <return>
MATHLIB = -lm

Some system require additional libraries when compiling FORTRAN code.
Specify the additional libraries needed when compiling FORTRAN.
Consult your FORTRAN manual for necessary libraries.
RETURN - to accept defaults.

FORTRANLIBS [-L/opt/SUNWspro/SC3.0.1/lib -lM77 -lF77 -lsunmath -lsocket
-lnsl -L/usr/local/ccs/lib -lgen -lm]: <return>

Checking -L/opt/SUNWspro/SC3.0.1/lib -lM77 -lF77 -lsunmath -lsocket
-lnsl -L/usr/local/ccs/lib -lgen -lm. Please watch for any errors

Check complete
Any problems? n
FORTRANLIBS = /usr/lang/SC2.0.1/values-Xt.o -L/usr/lang/SC2.0.1 -lM77
-lF77 -lansi

Now that you have defined everything, please name the file
to store this information in.

Makelist file [makelist]: <return>
```

```
HEADER = makelist
Creating makelist file
Makelist file /home/markstro/mms/make/makelist created
```

The *makelist* file is written at this point - compilation follows.

```
Updating the scripts...

cp mms_workspace /home/markstro/mms/bin
cp control_update /home/markstro/mms/bin
cp control.var /home/markstro/mms/bin
cp control.param /home/markstro/mms/bin
cp data_update /home/markstro/mms/bin
cp param_update /home/markstro/mms/bin
cp param.param /home/markstro/mms/bin
cd /home/markstro/mms/bin; sed -e "s#<MMSMD>#/home/markstro/mms#"
mms_workspace > tmp; mv -f tmp mms_workspace; chmod 755 mms_workspace;
cd /home/markstro/mms/bin; sed -e "s#<MMSMD>#/home/markstro/mms#"
control_update > tmp; mv -f tmp control_update; chmod 755
control_update;
cd /home/markstro/mms/bin; sed -e "s#<MMSMD>#/home/markstro/mms#"
data_update > tmp; mv -f tmp data_update; chmod 755 data_update;
cd /home/markstro/mms/bin; sed -e "s#<MMSMD>#/home/markstro/mms#"
param_update > tmp; mv -f tmp param_update; chmod 755 param_update;

The MMS libraries seem to exist.
Remake them anyway [no] -->n

Not making MMS libraries now.
You can always make them later by running make in /home/markstro/mms
The MMS executables seem to exist.
Remake them anyway [no] -->n
```

If the script finds that the libraries and executables already exist, the default will be not to make them. If the *mms/bin* and *mms/lib* directories are empty, the default will be to make them.

```
Not making MMS executables now.
You can always make them later by running make in /home/markstro/mms
MMS Setup completed
```

2.5 Creating the MMS User Workspace Directory

Before any user-created models can be built or run, a workspace directory must be created. Separate directory areas for each user help to keep work organized and separate from other users. The user workspace directory area must be in place before any of the MMS programs can be run.

The user workspace directory area can be located in almost any user-specified directory and is set up by executing the *mms_workspace* script. The user directory is created from a template directory in the master directory located in *mms/workarea*. A workspace should never be created anywhere in the *mms/workarea* directory as this will corrupt all workspaces created thereafter.

The directories in the user directory area and their contents are:

The *models* directory:

- the executable models that the user has created
- the model schematic files
- the model html file

The *control* directory:

- the environment file (*mms.env*)
- the control file(s) (*mms.control*)
- the resource script file (*setmms*)
- the *app-defaults* files
- the *workspace.path* file

The *input* directory:

- the *data* directory that contains the simulation data files
- the *params* directory that contains the parameter files
- the *vars* directory that contains variable files that were saved at the end of user-selected model runs.

The *make* directory:

- the files generated by the *xmbuild* program

The *modules* directory:

- FORTRAN and C module source code;

The *output* directory:

- model output files
- variable and parameter print files
- optimization and sensitivity analysis files
- ESP output files
- statistical analysis work file *statvar.dat*

The *mms_workspace* script is located in the *mms/bin* directory in the master directory. The script has two main functions. These are to copy the template workspace directories to the user's workspace directory and to update the paths in the workspace files. This script should be run whenever the user wants to create a new workspace.

The *mms_workspace* script is written in the Korn shell. For operating system which do not support the Korn shell, the script may be run under the Bourne shell by changing the first line in the file from:

```
#!/bin/ksh
```

to

```
#!/bin/sh
```

The script will ask questions about a location and system type. Once the questions are finished, the script starts copying the directories. For successful installation, it should not be interrupted. If the need arises to start over, all contents of the workspace directory should be removed and the *mms_workspace* script should be re-run. The script must be allowed to run all the way through uninterrupted to insure that all files and directories are present and not corrupted.

For best results, create the user workspace directory first, change to this directory, and then run the *mms_workspace* script. The workspace directory may be named anything. For the following example, it is called *work*:

```
(ksh) rose.p2% pwd
/home/markstro
(ksh) rose.p2% mkdir work
(ksh) rose.p2% cd work
```

From this directory, run the *mms_workspace* script. The script will ask several questions. The following is an annotated example run of the *mms_workspace* script:

```
(ksh) rose.p2% mms_workspace
*************************************************************
* Welcome to *
* MMS Workspace installation *
*************************************************************
```

```
Your current path is /home/markstro/work

Please type the FULL path of the user workspace directory.
RETURN - to use the current path.

<return>
```

Type ***return*** to use the current directory. The script will echo the location of the new workspace. If the path is wrong, exit the script using the **control-c** command, correct the path, and start over.

```
Installing workspace in /home/markstro/work
You are running SunOS release 5.6
Your current architecture is SOLARIS.  Using SOLARIS as target architec-
ture.
/home/markstro/work found...
Workspace directories copied to /home/markstro/work...
Updating paths...
Finished with paths...
Installation completed.
```

If everything is properly installed, listing the workspace directory contents using the *ls* command will produce the following list of subdirectories:

```
(ksh) rose.p2% ls
control/ make/ modules/
input/ models/ output/
```

The *control* directory contains a resource script file named *setmms* (or *setmms.ksh*) that must be run prior to the execution of any model or *xmbuild. setmms* merges the MMS *app-default* files with the current *app-default* file and sets several operating system environmental variables. This only needs to be done once at each login. This can be accomplished in one of two ways. One is for the user to manually execute a *source* command with the full path to the *setmms* script each time they login. An example *source* command is:

```
source /home/markstro/work/control/setmms
```

A second way is for the user to put the *source* command in their *.cshrc* or *.login* file so that it is automatically initiated at every login.

One other addition to the *.cshrc* file that may be desirable is the *<user_workspace>/models* directory path to the *path* environment variable. This will enable the execution of user-created models from any place on the system.

2.6 Moving the User Workspace

It is generally not desirable to move a user workspace once it has been created. There are many files with hard coded paths that are set at installation time. Thus, moving the directory will result in many errors caused by these incorrect paths. However, if it becomes necessary to move a workspace, there are two different strategies that can be adopted. The first option is to create a new user workspace using the procedure described above and then manually copy the necessary files from the old to the new workspace. The second option is to move the old workspace to the new location and manually edit all of the path references.

The steps for creating a new workspace using option one are as follows:

Create the new user workspace using the procedure described above.

Copy the contents of the *old_work/input/data*, *old_work/input/params*, and *old_work/input/vars* to the corresponding *new_work/input* directories.

Copy the contents of the *old_work/models* directory to the *new_work/models* directory. Any schematic files that contain references to modules that have moved must be edited to reflect their new locations.

Copy the contents of the *old_work/modules/src* and *old_work/modules/hier* directories to the corresponding *new_work/modules* directories.

Update the *new_work/control/workspace.path* file to contain the paths to the modules specified in the model schematic files.

The steps for moving a workspace to a new location using option two are as follows:

Update the control files (*mms.control*) in the *work/control* directory. Change the paths to the data files(s) and parameter file.

Update the *work/control/mms.env* file.

Update the *work/control/setmms* script.

Update the *work/control/workspace.path* file to reflect the new locations of the modules.

Update the *work/make/make.template* file.

Update all the schematic and html files in the *work/models* directory.

Generally, the first procedure works better when there has not been much work done in the workspace. As more data and model files are created, the second procedure should be given more consideration.

2.7 Installing Modules

A tar file with a fixed directory structure is used to distribute the source code modules and the module documentaton for complete models. This file can be installed in the user's workspace directory or in the master MMS directory. When this file is untared, the directory *<user's workspace>/modules/src/<model_name>* or *<MMS master directory>/modules/src/<model_name>* will be created and the source code modules will be written here. A second directory *<user's workspace>/modules/doc/<model_name>* or *<MMS master directory>/modules/doc/<model_name>* will also be created and the documetation for each module, in html format, will be written here.

In order for the documentation system to work correctly, each module source code file must have it's corresponding html documentation in the *modules/doc* directory.

3. MMS STRUCTURE, CONVENTIONS, AND DEFINITIONS

3.1 Modular Model Concepts

3.1.1 Modules

By definition, a module is a set of computer source code used to simulate one, or more, water, energy, chemical, and biological processes. A given process, or combination of processes, can have several modules in the library, each representing an alternative conceptualization or approach to simulating the process(es). Each module typically needs selected inputs to drive it and computes selected outputs that could be used as inputs to other modules.

The user, through an interactive model builder interface called *xmbuild*, may select and link modules to create a specific model. Previously generated models can be executed directly without the use of the *xmbuild* step. A detailed discussion on module construction is provide in Chapter 6. A brief overview of module concepts is given below to provide the user with a basic knowledge of MMS functionality and terminology.

The ability to link modules developed by a variety of users is provided by the use of a unique module structure. A module is composed of a minimum of four functions: *declare*, *initialize*, *run*, and *main*. The *declare* function is used to specify parameters and variables that are being declared in this module. The *initialize* function is used to initialize parameters and variables used in the module. The *run* function contains the algorithm code that simulates the specific process(es). The *main* function directs system calls to the *declare*, *initialize*, and *run* functions of a module. A module can be written in either the FORTRAN or C programming language. A fifth function, *cleanup*, may be used by some C coded modules to release dynamically allocated storage.

Communication among modules and between a module and MMS features is accomplished by using specific MMS library functions that a module developer uses in the module code (see Chapter 6 for details on MMS library functions). Parameters and variables are the primary elements communicated among modules. In MMS a parameter is defined as any value that is held constant during a model run. A variable is defined as a value that can change each time step and may be computed or read from an input data file.

Parameter and variable data structures are created by the *declparam* and *declvar* library functions respectively. The *declparam* function is used to declare each parameter used in the module. The *declvar* function is used to declare only those variables that are generated by the process(es) in the module and have a potential use by other process modules. These are termed "*public variables*." Local working variables are not normally declared using the *declvar* function unless there is a desire to display or analyze these variables using the system tools.

Arguments in the *declparam* and *declvar* functions include a definition of the parameter or variable and the units in which it is expressed. The definition is made available to the model user through a help feature during MMS execution. The definition can describe the parameter or variable, provide guidance on the estimation of its initial value, and any other pertinent information. This provides a mechanism for module developers to imbed their knowledge and expertise in a module and have this information available to all users. The units in which the parameter or variable is expressed is used in the *xmbuild* process to insure the compatible linkage of module parameters and variables.

Modules read parameter and data values from the MMS parameter and variable data structures using the MMS library functions *getparam* and *getvar*. In the *xmbuild* procedure, a comparison is made among selected modules to insure that all *getparam* and *getvar* functions are satisfied by a *declparam* or *declvar* function for the specified parameter or variable. A *declparam* function must be present for all parameters used in each module. A *declvar* function is used only in the module where the variable is computed.

When MMS is executed, the *declare* function of all modules is executed first to obtain the needed information from the *declparam* and *declvar* functions to build the MMS parameter and variable databases. During the execution of the *initialize* and *run* functions of a module, MMS functions *getparam* and *getvar* are used to read current values of parameters and variables from these databases. Values are written to the variables database automatically by the declaring module.

Successful communication between modules requires the use of consistent parameter and variable names. A dictionary of currently used terms and their definition is included in MMS to provide information to system users as well as to maintain consistency among module developers. Improved communication within the modeling community and the establishment of some consistency in process parameter and variable definitions are seen as major needs in being able to compare modeling approaches.

While there is a requirement for consistency in the external variable and parameter naming convention, there is no need to modify variable names in existing source code when converting to a module. The external and existing internal variable and parameter names can be equivalenced in the *declparam* and *declvar* functions. The MMS variable and parameter databases keep the external names as character strings and provide a reference to the locally declared name.

3.1.2 A Model

A model is created from user-selected modules using the MMS program *xmbuild*. A model can be composed of a combination of FORTRAN and C coded modules. During module compilation and linking, the MMS X-windows graphical user interface (GUI) is linked to the model to provide the MMS support functions. A series of pull-down menus in the GUI provide the links to a variety of system features. These menus and features are described in detail in Chapter 4.

Modules are executed sequentially in a time-based loop whose time step is defined by the input data stream and may be variable. When a model is executed within MMS, one pass is made through the initialize functions of all the modules to initialize all parameters and user-specified variables. Each subsequent pass through the modules is directed to the run function to execute the process algorithms.

3.2 The MMS Internal Databases

MMS uses an internal database for inter-module communication. Individual modules declare values, update values, and may read the values set by other modules. From the point of view of the module programmer, there are three memory-based databases. These are:

- the parameters database
- the public variables database
- the dimensions database

A fourth database is the control database which is built by the system and stores the current status of all the options in the MMS interface.

3.2.1 The Parameters Database

Parameters are declared in each module with the *declparam* function. When a model is initiated, MMS makes one pass through all the modules and reads all the *declparam* functions. The parameters database is constructed using the information provided in these function statements. During model execution, when modules request parameter values, they receive a copy of the parameter value array. Thus, integrity of the parameter database is maintained.

The parameters database is initially populated by reading a set of parameter values from the currently selected parameter file. A parameter file can be initialized to the default values declared in the modules or it can be modified and saved using a set of spreadsheet tools described in Chapter 4.

3.2.2 The Public Variables Database

The public variables database is intended to hold those variables which are made available to all modules in the system. This database serves a variety of purposes by making the variables available for:

- use by other modules
- reading from time series files
- display at run time
- saving after each time step for later statistical analysis
- saving at the end of a run, to use as start-up values for subsequent runs
- printing to a file for debugging

Not all the variables in a program need to be included in the public variable database. Only those variables for which there is an interest in plotting or statistically analyzing, making available to other modules, or using in an optimization or sensitivity analysis need to be included in this database.

Variables are declared in each module using the *declvar* function and these declared variables are used to construct the variables database. The declaring module provides the static memory to store the values of the variable. In this way, the module has the ability to both read and write to its own declared variables. When other modules request variable values, they receive a copy of the variable value array. It is possible, using the MMS function *putvar*, for one module to write directly into the variable space of another module. The *putvar* function is typically used only for feedback types of situations and is not normally used for standard communication of variable values among modules.

3.2.3 The Dimensions Database

A 'dimension' is a number that defines the size of a parameter or variable array. The number of sub-catchments and the number of rain gages are examples of dimensions. Rain-gage elevation is an example of a parameter that may have as its dimension the number of rain gages. Similarly, the rainfall measured at each gage would be a variable with the same dimension.

Dimensions are integer types, with a minimum value of zero and a maximum value that is set when they are declared. All dimensions are declared in a special MMS function named *setdims* using the *decldim* library function (see Chapter 6 for details on *setdims* and *decldim*). This function is compiled with the model modules and is executed before the simulation modules declare their parameters and variables.

One argument in the *decldim* function is a parameter defining the maximum size of this dimension. This parameter is defined in an include file named *fmodules.inc* for FORTRAN modules and *cmodules.h* for C modules. These are located in the *<user_directory>/modules/include* directory. An include file is included with all FORTRAN and C modules in a model.

The user can modify the size of any dimension in the dimension database, not to exceed the maximum size defined in the include file, using the spreadsheet editing tools described in Chapter 4. To increase the maximum dimension parameter, the user must manually edit the include file and recompile the model. The include file for maximum

dimension size is used because array storage cannot be dynamically allocated using FORTRAN 77.

3.2.4 The Control Database

The control database contains the status of all the features of the MMS graphical user interface (GUI). The user can save the values in this database in a control file with any name of choice. The control file is a system-created file that stores the status of the MMS GUI at the time the file is created. A new control file can be saved for each specific set of GUI conditions that the user may wish to use at some future time.

An example is the application of a given model to different basins. The user can select the appropriate data files and parameter file for a specific basin as well as select the variables to be graphically displayed and statistically analyzed. All these values are set in the control database using the menu options discussed in Chapter 4. Once all selections are made, a control file for this basin can be created and saved. A different control file can be saved for each basin. Then when this particular model is run, the user can simply select the control file for the basin of interest and all the previously defined control features for this basin will be loaded into the control database. This avoids the need to change the individual features of the GUI each time a different basin is chosen.

3.3 Parameter Files

When the user initiates MMS for a selected model, the parameter database is created by the system. Each installation of MMS has a default control file. For the first execution of MMS with any model, the parameter file listed in the control file will be loaded. The parameter file named in the default control file may not be associated with the selected model. If it is not, then only those parameters in the parameter file having the same names as parameters in the database will be loaded into the database. Database parameters not contained in the parameter file will contain the values present in the database when it was constructed. To change the default control file, see section 3.5 Control Files below.

To create a new parameter file for the model being run, the user must save the current file to a new file. The system will create the new file and save it in the *<user_directory>/input/params* directory by default or in a directory of the users choice. This new file can be reset to contain the default values declared in the model modules or it can be edited to include parameter values determined by the user. The initializing, editing, and saving of parameter files is discussed in detail in Chapter 4.

The system generated parameter file has three sections. These are: (1) the header, (2) the dimension data, and (3) the parameter data. An example of the header information from the demonstration parameter file is:

```
East Fork Carson River, CA
Version: 1.7
```

The first line is a description of the parameter file. It has a limit of 80 characters. The second line is the version line. This number refers to the version of the function in the MMS library which reads the file. If this line is not present, the file is version 1.0.

After the header information, the dimensions and their sizes follow:

```
** Dimensions **
####
nac
1
####
nchan
1
####
ndepl
1
####
ndeplval
11
```

The first line signifies to the model that the dimension definitions are beginning. The next three lines are repeated for every dimension definition. First comes a separator, then the dimension name as declared in the *setdims* function, and finally, the size of the dimension used. This size must be smaller than or equal to the size declared in the module include files *fmodules.inc* or *cmodules.h*.

The parameter values come after the dimension values:

```
** Parameters **
####
basin_area 4
1
one
1
2
1.851000000000e+05
####
hru_area 0
1
nhru
17
2
1.277000000000e+04
8.590000000000e+03
1.460000000000e+04
1.169000000000e+04
8.040000000000e+03
8.170000000000e+03
1.077000000000e+04
9.110000000000e+03
1.087000000000e+04
1.047000000000e+04
```

```
8.320000000000e+03
6.220000000000e+03
8.120000000000e+03
1.433000000000e+04
2.231000000000e+04
9.770000000000e+03
1.095000000000e+04
```

The first line signifies the beginning of the parameter definitions. The next series of lines are repeated for every parameter definition. First comes a separator. The next line specifies the parameter name and column width in the spreadsheet editor. A zero column width indicates the default width. The next line indicates the number of dimensions of the parameter (i.e. 1-dimensional, 2-dimensional,...). The dimension name(s) which the parameter is declared over is specified on the next line(s). The dimension name is specified as *one* if the parameter is a scaler. Next comes the number of values. This should match the dimension size declared in the dimension section of this file. The next line determines the type of the parameter values. The codes are:

- *1* for *long integer*
- *2* for *float* or *real*
- *3* for *double precision float* or *real*
- *4* for *character string*.

Finally, the parameter values follow, one per line.

3.4 Data Files

Data files have an ASCII flat-file format and are created by the user. The input variables in these data files are read by a model module written by the model developer for that purpose. The input variables are declared in this module and the values of the input variables are copied to the public variables database at each time step using the *readvar* library function. The time-series data files used for model runs are typically placed in the *<user_directory>/input/data* directory but may be placed in any directory of the user's choosing.

The following is an example data file:

```
MMS test data file - Carson Basin
runoff 1
rainfall 5
tminf 2
tmaxf 2
solrad 0
pan_evap 0
form_data 1
############################################################
1980 10 1 24 0 0 84 0 0 0 0 0 44 51 78 86
1980 10 2 24 0 0 82 0 0 0 1.5 0 51 48 78 84
1980 10 3 24 0 0 80 0 0 0 0 0 44 52 77 86
```

```
1980 10 4 24 0 0 80 0 0 0 0 0 42 50 74 84
1980 10 5 24 0 0 80 0 0 0 0 0 45 46 73 80
1980 10 6 24 0 0 79 0 0 0 0 0 42 46 74 79
1980 10 7 24 0 0 78 0 0 0 0 0 42 49 74 81
1980 10 8 24 0 0 77 0 0 0 0 0 41 47 72 83
1980 10 9 24 0 0 92 0 0.1 0 0 0 39 45 70 81
1980 10 10 24 0 0 95 0 0 0 0 0 41 47 69 76
```

There is a short multi-line header, a separator line, and then the data. The first line of the header contains a description of the data file. This description has a limit of 80 characters. The remainder of the header describes the data fields in each row. Each line contains the variable name and the number of values for that variable in each row. The number of values must match the current dimension of that variable. The order of the variables reflects the order of occurrence in each row.

A separator line indicates the end of the header information and the beginning of the data. This line must consist of at least four pound symbols (####).

The data lines start after the separator line. Fields in the data line are separated by white space. The first six fields of the data line are reserved for the time stamp. The fields are year, month, day, hour, minute, and second respectively (yyyy-mm-dd hh:mm:ss). The remaining columns must correspond to the order and number of values specified in the header section of the file.

In the above data-file example, one value is read for the *runoff* variable, five for *rainfall*, two for *tminf*, two for *tmaxf*, and none for *solrad* for each time step. An error is reported if the data requested by the modules do not match the header. Extra values on the line will be ignored.

3.4.1 Time Steps

MMS makes the distinction between two different time step modes. These are a daily mode having a 24 h time step and an incremental mode whose time step is less than 24 h. Use of either or both of these modes is a function of the modules that comprise a model. For models that can use both modes, MMS will switch between these modes automatically as dictated by the data file.

By convention, daily mode values have zeros in the hour, minute, and second fields of the time stamp (yyyy-mm-dd 00:00:00). Incremental mode times run from yyyy-mm-dd 00:00:01 to yyyy-mm-dd 24:00:00 (midnight). Time is handled on a fractional Julian day basis. The year, month, and day fields of the time stamp are converted to an absolute Julian day. Then the hour, minute, and second fields are converted to a decimal fraction of a day.

3.4.2 Multiple Data Files

MMS allows specification of multiple input data files. This allows for multiple daily-mode data files and multiple incremental-mode data files. For example, a model could be run with one daily data file and several incremental files, each containing short time step data for different individual storms to be simulated at the shorter time steps. This allows users to easily change the data set without the need to edit a single large data file.

Multiple input data files in MMS are read according to the following rules:

- The data lines within a file must be in order from earliest to latest.
- When processing multiple files, the first data line from each file is read. MMS chooses the one with the earliest time stamp and uses it as the current data. When MMS is ready for data for the next time step, the next line is read from the current file and is compared with the other files, looking for the earliest time step. This procedure repeats until there are no more data lines in any of the specified data files.
- Incremental mode data may have a variable time step.
- Incremental mode data must cover a full day (the first time step is from yyyy-mm-dd 00:00:00 to the hh:mm:ss of the first data value for that yyyy-mm-dd and the last value for the day is for the time increment that ends on yyyy-mm-dd 24:00:00).
- Incremental data take precedence over daily data. In the case where both data types are present for the same day, the daily value will be disregarded and the model will run in incremental mode for that day.
- An error will occur if multiple input files have the same yyyy-mm-dd-hh:mm:ss values and non-identical data.

3.5 Environment File

All models built in MMS reference a special environment file. This file specifies directory paths, system files, and MMS related utilities. The most important of these specifications is the control file, text editor, and HTML viewer. The default environment file is *mms.env* and is located in the *control* directory of the user workspace directory. The default environment file is specified by setting the *mms_env_file* environment variable to the full path to this file. This is done automatically by the setmms script in the *control* directory of the user workspace directory. The user can select another environment file to be used as the default file by including the -E option at the time the MMS model is initiated. An example of the -E option is:

```
xprms -E<user_directory>/control/mms.env
```

When the model *xprms* is executed, the control file *mms.env* will be substituted for the default control file. All environment files are located in the *<user_directory>/control* directory.

An example of the first few lines of an environment file are:

```
#environment file for MMS
#
#These variables are added to the environment by the routine 'append_env'
#unix path delimiter
mms_path_delim=/
mms_editor=vi
mms_html_viewer=mosaic

#directories

mms_work_dir=/home/markstro/mms_work
mms_control_dir=/home/markstro/mms_work/control
mms_data_dir=/home/markstro/mms_work/input/data
mms_inc_dir=/home/markstro/mms/src/include
mms_make_dir=/home/markstro/mms_work/make
mms_sys_make_dir=/home/markstro/mms/make
mms_sys_modules_dir=/home/markstro/mms/modules
mms_modules_dir=/home/markstro/mms_work/modules
mms_files_dir=/home/markstro/mms/modules/files
mms_output_dir=/home/markstro/mms_work/output
mms_param_dir=/home/markstro/mms_work/input/params
mms_setdims_dir=/home/markstro/mms_work/modules/setdims
mms_var_dir=/home/markstro/mms_work/input/vars
mms_user_out_dir/home/markstro/mms_work/output
mms_bin_dir=/home/markstro/mms_work/models
mms_object_dir=/home/markstro/mms_work/make/obj
mms_mlib_dir=/home/markstro/mms/lib
mms_help_dir=/home/markstro/mms/help/mms
mms_hier_dir=/home/markstro/mms_work/modules/hier

#files

mms_control_file=cane.control
mms_make_template=make.template
mms_sens_file=sens.dat
mms_opt_file=opt.dat
```

This file is created automatically by the *mms_workspace* installation script. Generally, users should not have to make any modifications except to specify a different text editor than *vi* or a different HTML viewer than *mosaic*. Also, the paths must be changed if either the master MMS directory or the current *workspace* directory is moved.

3.6 Control Files

When a selected model is run, the default control file will be loaded into the control database. The default control file is *mms.control* and is located in the *control* directory of the user workspace directory. The user can select another control file to be used as the default file by including the -C option at the time the MMS model is initiated. An example of the -C option is:

```
xprms -Cprms.control
```

When the model *xprms* is executed, the control file *prms.control* will be substituted for the default control file. All control files are located in the *<user_directory>/control* directory.

An example of the first few lines of a control file are:

```
####
data_file
1
4
/models/mms_work/input/data/carson.test
####
param_file
1
4
/models/mms_work/input/params/efcarson.params
####
var_save_file
1
4
vars.save1
####
```

The file is similar in structure and format to the parameter file described above. The file begins with a delimiter followed by the tag name of an item in the GUI. The first tag name is *data_file* which represents the path name to the data file currently being used. The next line shows that there is one data file and the next line shows that the data type that is used to name the file is type 4 or a character data type. Type specifications are 1-4 and are the same as those listed above for parameter files (Section 3.3). The next line is the actual character string for the data file.

The next sets of items between the delimiters (####) specify the path to the parameter file and the file name for the file that will be used to save all declared variables at the end of a model run. The control file continues on in this manner for all tag names of the GUI features.

3.7 Model Execution

Executable models that have been developed from the module library typically reside in one of two places. Models that have received some form of testing and approval by the agency using MMS will be located in the *bin* directory of the master directory. Models developed by the user will be located in the *model* directory of the user workspace directory. Other MMS executable programs also reside in the *bin* directory of the master direc-

tory. Putting the paths to this *bin* directory and the *model* directory of the user workspace directory in the *PATH* environmental variable will permit model execution from any directory on the system.

Each model name begins with the letter x and is followed by the remaining letters used to name the model. A specific model is executed by typing its full name at the operating system prompt. Upon execution, a GUI, based on the Motif X-windows toolkit, is initiated and provides the user access to all model initialization, run, and analysis tools in the system. The functions of the GUI are described in Chapter 4.

4. MMS GRAPHICAL USER INTERFACE

4.1 Interface Conventions

The interface is mouse driven. The user moves the mouse pointer over a desired option and clicks the *left* mouse button to execute. The other buttons have special functions which are explained below.

Some of the options require the user to choose between a number of sub-options. When the user clicks on one of those, a small square or diamond next to the choice will take on a different appearance than those next to the items not selected. Some of the options are mutually exclusive (called "radio buttons"), while other options can be simultaneously selected.

4.1.1 Menus

MMS menu choices in the main system window appear on the upper part of the window. All menus are of the "pull down" type. Sub-menus are displayed beside the parent menu, so that the user is aware of the current menu hierarchy. However, "action" windows are placed over their parent menu to indicate to the user that they require action before any other function may be selected. When traversing the menu hierarchy, those options higher in the hierarchy will be disabled. This is indicated by a "graying" of the color of the labels in the menu choices.

To pull down a menu, position the mouse pointer over the menu, and press the *left* button. The user may find that it is easier to release the mouse button immediately. While the pulled-down menu is displayed, move the mouse pointer to the desired menu option, again press the *left* mouse button, and release it immediately. Some menu options produce "cascading" menus. These options are identified by small triangular arrows on the right hand side of the option. Although it is possible to traverse the cascading menus with a *left* mouse button pressed, it is easier to release the button after the choice is made.

In many cases a menu choice will cause a small window to "pop-up." This window may contain input fields, action buttons, or choice buttons.

4.1.2 Input Fields

Some of the interface options require editing of displayed text. This may be done with the keyboard, or with both the mouse and the keyboard.

The mouse is used to position the text cursor, which looks like a tall 'I', in the text. The arrow keys may then be used to move the cursor to the left or right. Typing will add characters to the right of the cursor. The *backspace* key will delete the character to the left, and the *delete* key will delete the character to the right. Some keyboards do not have

a ***backspace*** key, so move the cursor to the left of the character to be deleted and use the ***delete*** key.

Highlight text by clicking the ***left*** button at one point in the text, and then drag the cursor across the text while holding the ***left*** button down. Releasing the button completes the operation. Typing after highlighting will replace the highlighted text with what is typed. Pressing the ***delete*** key after highlighting text will delete that text. Select an entire word by "double clicking." Select the entire text field by "triple clicking."

4.1.3 Slide Bars

Slide bars appear either on the right hand side or at the bottom of some pop-up windows. Slide bars on the right hand side of the window are used to scroll the window up and down, while slide bars on the bottom are used to scroll the window right or left. Slide bars have three components: two triangular arrows at either end of the slide bar, and a rectangular 'slider.' Clicking the ***left*** mouse button on either arrow causes a slow scroll in the respective direction. Clicking on the rectangular bar and dragging the bar causes a fast scroll. The relative location of the slider on the scroll is an indicator of the relative location of the visual portion of the window with respect to the complete window. Also, the size of the slider relative to the size of the slide bar gives an indication of the size of the visual portion of the window with respect to the complete window.

4.1.4 Action Buttons

Some pop up windows have 'action buttons.' These buttons are similar in function to menu choices, in the sense that they allow the user to select a particular option. Examples of these action buttons are the *Quit, Cancel,* and *Yes* buttons and the spreadsheet option buttons.

4.1.5 Standard Motif File Selection Dialog

File selection for some components of the system is accomplished using the standard Motif file selection dialog (fig. 4.1). The *Directories* and *Files* scrollable lists show the currently selected directory and the files contained in that directory. To select a file from this list, one can click on the file name using the ***left*** mouse button. The full path of the selected file also appears on the *Selection* text entry window. Alternatively, one can simply enter the full path of the desired file in the *Selection* text entry window.

The *Directory* list contains a list of the subdirectories of the current directory and the directory immediately above the current directory. To change to a subdirectory, click on the subdirectory name using the ***left*** mouse button. The names of the files in the newly selected subdirectory will appear in the *Files* list. To move to next higher directory, click on its name which ends in the characters "***/..***" From this directory one can move to any other directory in the system by repeating the directory selection procedure using the ***left*** mouse button.

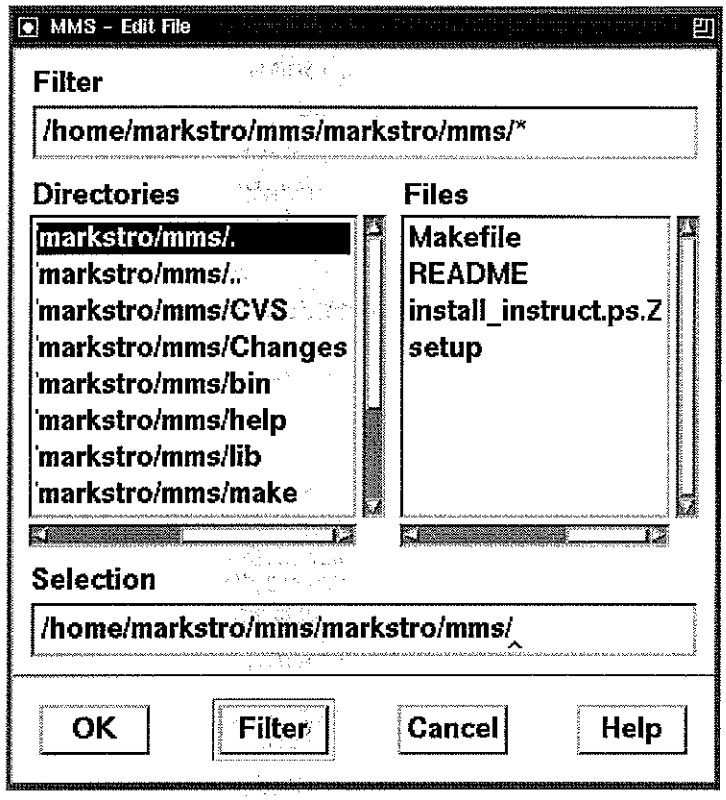

Figure 4.1 Motif File Selector window.

The *Filter* text entry window provides a way to selectively list files in the *Files* list. The path to the current directory ending with the characters "/*" will list all files in the directory. A path ending in the characters "/*.dat" would only list those files ending with a ".dat" extension.

The selector buttons located at the bottom of the window are executed using the *left* mouse button. The *OK* button is used to identify the file listed in the *Selection* text entry window as the desired file. After clicking on *OK*, control returns to the previous menu. The *Filter* button executes the filter in the *Filter* text entry window. The *Cancel* button cancels this window and returns to the previous menu. The *Help* button provides information on the use of this window.

4.2 Main MMS Interface Window

The main MMS interface window (fig. 4.2) provides information on the current model being run and the associated data, parameter, and control files being used with the model. A schematic representation showing the modules that have been linked to create the model and the information flow among the modules is presented on the right half of the window. Clicking on one of the module icons with the *left* mouse button opens a help window with detailed information about the parameters and variables associated with the

module as well as a detailed description of the equations and assumptions used to simulate processes in the module.

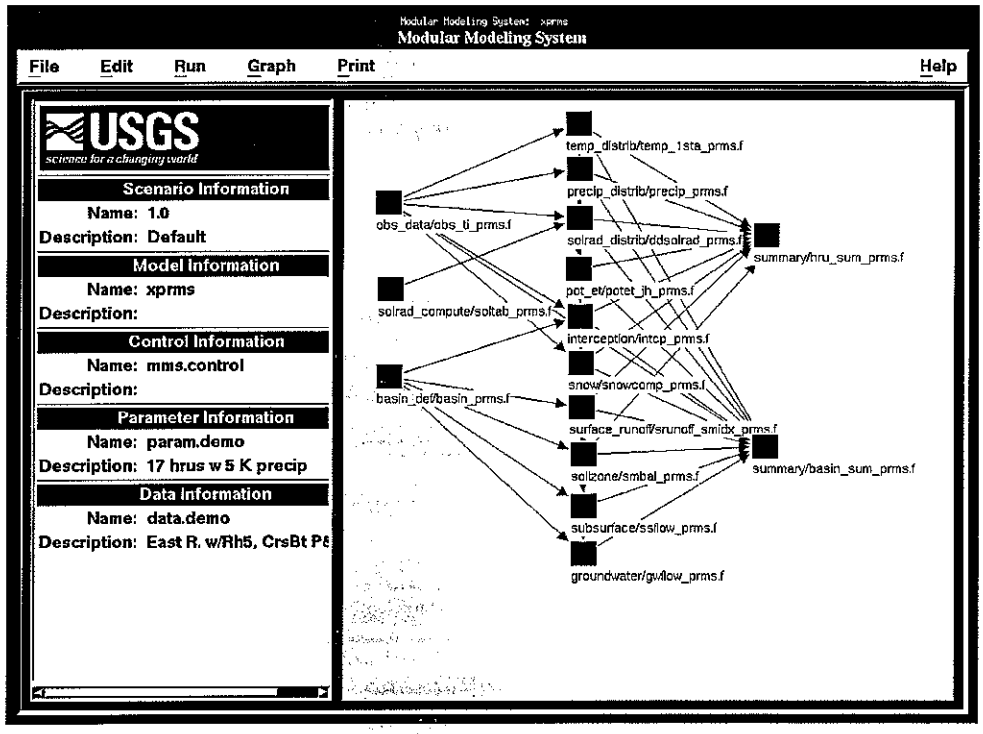

Figure 4.2 MMS Main Interface window.

Across the top of the interface window is a menu bar that provides a number of pull-down menu options to select data and parameter files, edit parameter files, and select and execute various system options. The menu bar options are:

- *File*
- *Edit*
- *Run*
- *Graph*
- *Print*
- *Help*

These menu options are discussed in detail in the following sections.

4.3 File Menu

The *File* function allows the user to change data, control, or parameter files, to save new control and parameter files, to specify the names of output files, and to exit MMS. All options under the *File* menu are selected using the ***left*** mouse button unless otherwise

noted in the text description of the function. The file pull-down menu contains the following options.

4.3.1 Load Data Files

Clicking on this option produces a *Data Files* window (fig. 4.3) containing the names of the data files currently selected for application with this model. To remove a data file from the list, click on the name. To add a file, click on the *add* button. This opens a *Load Data File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *< user_workspace>/input/data*. A data file can be selected from this directory or another directory of the user's choice. The path for the selected data file is added to the *Current Data File* list in the *Data Files* window.

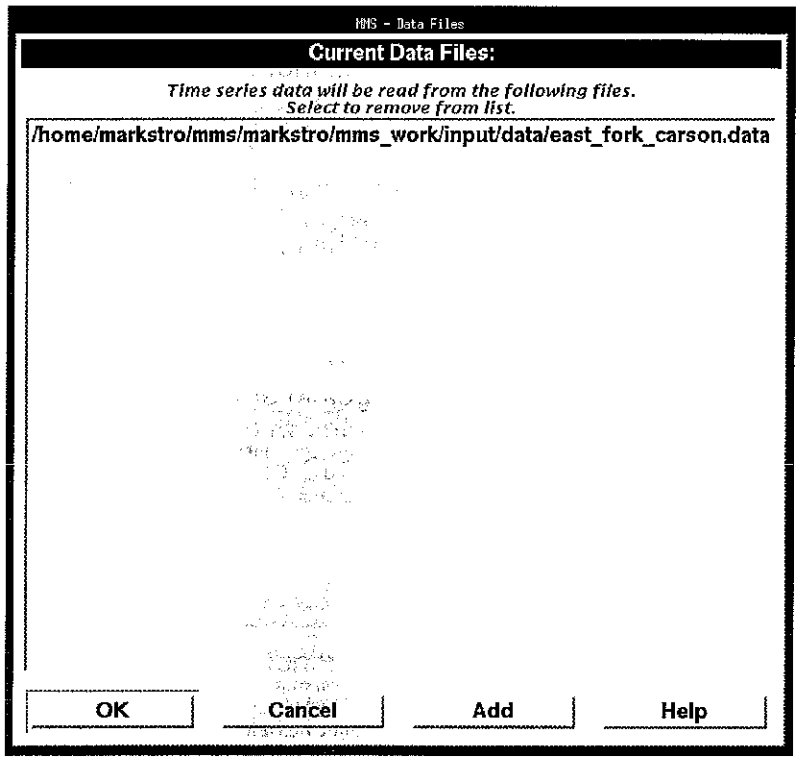

Figure 4.3 Data Files window.

The number of data files that can be selected is a function of the model being run. Models using two or more data files are normally those that support variable time step applications. In these cases, typically one data file contains data having a fixed time step, for example a daily time step. The additional data file(s) would contain data for selected periods of time within the time period of the first data file but for shorter time steps than the daily time step. An example of such an application would be a model that simulated streamflow in a daily time step but could also simulate shorter time step stormflow hydrographs. The model starts by reading all data files and uses the data file

with the earliest year, month, and day for each time step until a match is obtained for the year, month, and day in a second data file. At this point, data are read from the second data file until the year, month, and day are no longer equal for both data files. Data input then returns to the first data file until another date match is obtained.

4.3.2 Load Parameters

Clicking on this option produces a *Load Parameter File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *< user_workspace>/input/params*. A parameter file can be selected from this directory or another directory of the users choice.

4.3.3 Load Control File

Clicking on this option produces a *Load Control File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *< user_workspace>/control*. A control file can be selected from this directory or another directory of the user's choice.

4.3.4 Save Parameters

Clicking on this option produces a *Save Parameters File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *< user_workspace>/input/params*. A parameter file can be saved to this directory or another directory of the user's choice. A parameter file can be saved using an existing file name by double clicking on the name in the *Files* list.

4.3.5 Save Control

Clicking on this option produces a *Save Control File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *< user_workspace>/control*. The current control variables can be saved as a control file in this directory or another directory of the users choice. A control file can be saved using an existing file name by double clicking on the name in the *Files* list.

4.3.6 Set File Names

Clicking on this option produces a *Set File Names* window (fig. 4.4) This window allows specification of output file names where the results of model simulations are written. The directories where these files are written are specified in the MMS "environment file" (*mms.env*) which is created when the user workspace is created and is located in the *< user_workspace>/control* directory.

Output file names to be set are:

- *model output* - output generated by any modules during a model run
- *optimization* - summary of the results of an optimization run (see Run Menu)
- *sensitivity* - summary of the results of a sensitivity analysis (see Run Menu)
- *statistics* - summary statistics for user-selected variables (see Run Menu)
- *statvar* - intermediate file for storing user-selected variables to be used in the statistical analysis and for post processing graphical analysis
- *parameter print* - summary of the parameters (see Print Menu)
- *variable print* - summary of the variables (see Print Menu)
- *variable save* - save variables file

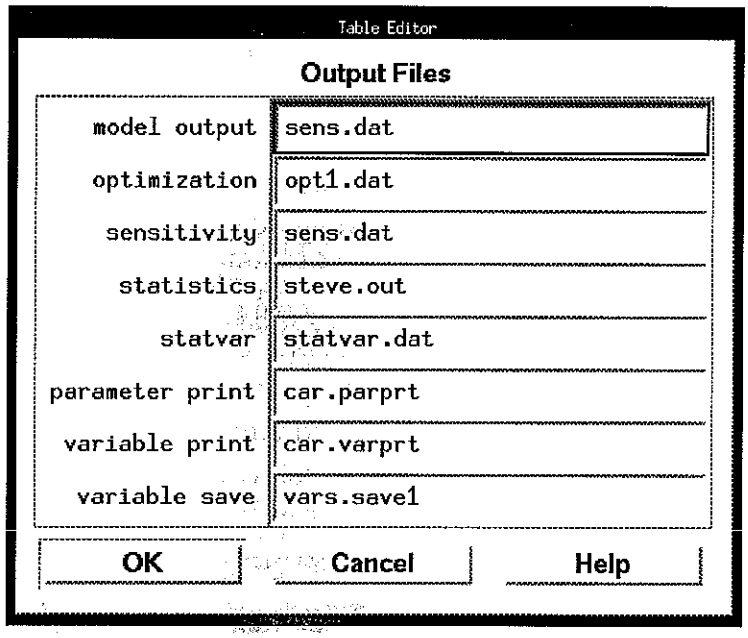

Figure 4.4 Set File Names window.

4.3.7 Exit

This option is used to exit MMS. A window will appear to verify the exit command. If parameters have been modified and not saved, a warning window will appear. When this window is dismissed, the user may save the modified parameter file or exit without saving.

4.4 Edit Menu

The *Edit* function allows the user to edit the dimensions database, the index names database or the model parameter file. Values in the parameter and dimension files may be edited individually or within a spreadsheet. The edit pull-down menu contains the following options.

4.4.1 Dimension Size

Clicking on the *Edit Dimension Size* option will produce a menu (fig. 4.5) containing the names of all of the dimension variables used in the model and the current value assigned to each of these variables. Any of the values may be changed by modifying or overwriting the current value. After changes have been made, they are applied by clicking on the *OK* button. If no changes are desired or changes are to be ignored, then exit by clicking on the *Cancel* button. Clicking on any dimension name with the ***right*** mouse button produces a window containing the definition of the dimension name and the maximum size permitted for this dimension.

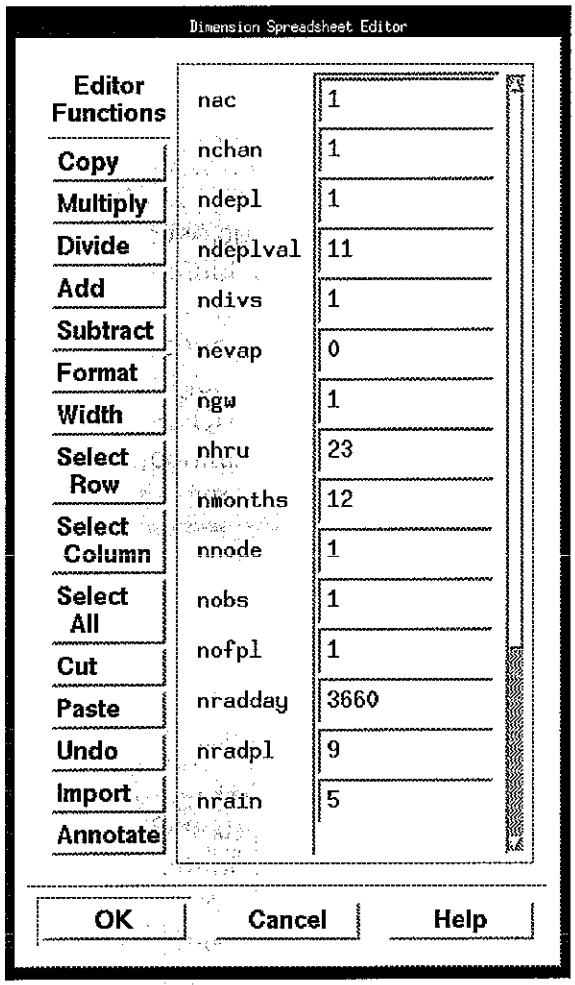

Figure 4.5 Edit Dimension Size window.

Because FORTRAN 77 does not support dynamic storage allocation, the maximum sizes of dimensions are specified in an include file which is *fmodules.inc* for FORTRAN or *cmodules.h* for C. Values entered using the *Edit Dimension Size* option cannot exceed the maximum size specified in these include files. However, a user can modify these

maximum size values as needed. The *fmodules.inc* and *cmodules.h* include files are discussed in more detail in Chapter 6.

4.4.2 Dimension Index Names

This option allows the user to assign names and a text description to the indexes defined by the dimension variables. Clicking on this option will produce a menu containing the names of all of the dimension variables used in the model. Clicking on a dimension variable and then clicking on the *Select* button or simply double clicking on the dimension variable will produce a *Dimension Index Names Editor* window (fig. 4.6). In this window, up to a ten character name and a text description can be entered for each index value. For example, if *nmonths* is a dimension variable with index values of 1 to 12, the values 1 to 12 can be given the names of their associated months and these names rather than the numbers will appear in all spreadsheet and tabular presentations that use the *nmonths* dimension. Clicking on the index value name in one of these presentations using the ***right*** mouse button will provide the user with the text description entered for that value.

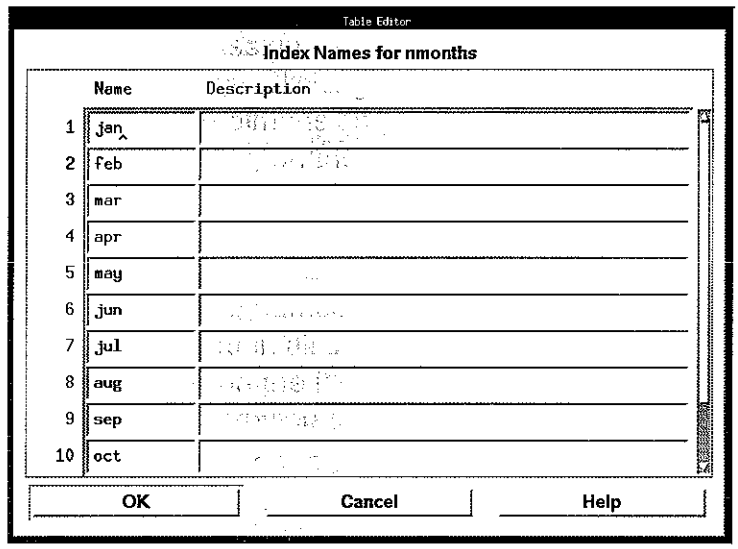

Figure 4.6 Dimension Index Names Editor window.

4.4.3 Parameters

This option enables the editing of model parameters using a number of approaches. Clicking on the *Parameters* option will produce a menu that enables the editing of parameters *by Module* and *by Dimensions,* or the resetting of the *Default Values* specified in the modules of the model.

by Module

Clicking on this option will produce a window listing all of the modules within the selected model. Clicking on one of the modules will produce a window containing a list of all the parameters declared within that module. Then, clicking on the parameter of interest will produce a *Parameter Editing by Module* window which lists the current value(s) of this parameter and allows all the values of this parameters to be edited (fig. 4.7). Values can be entered or changed by modifying or overwriting the current value. This window also has a *Help* button, which when pushed will display more detailed information about the parameter, including the maximum, minimum and default values, parameter units, a parameter definition, and possibly information regarding selection or estimation of parameter values.

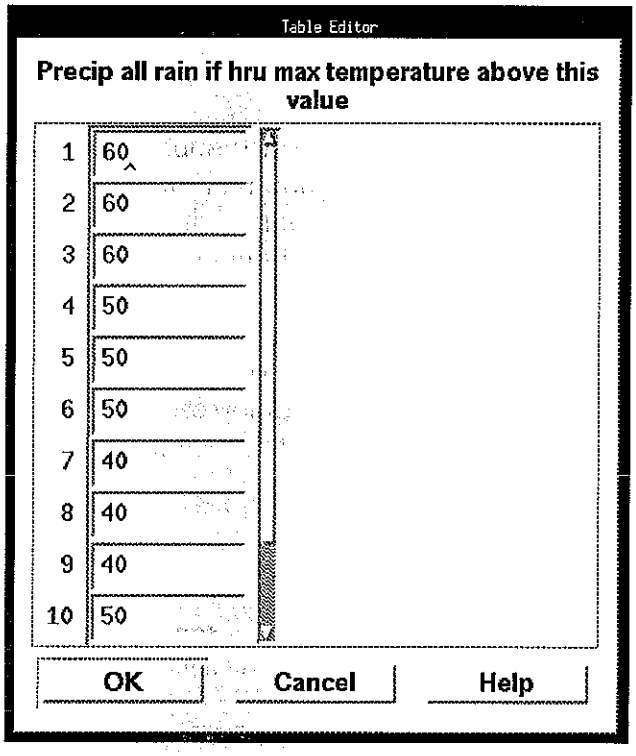

Figure 4.7 Parameter Editing by Module window.

by Dimension

Selecting this option produces a window for selecting the options of editing *Scalar*, *Single Dimension*, or *Multiple Dimension* parameters. These are based on the type of dimension used to define the parameter.

For each of these options, editing by dimension is accomplished using a *Spreadsheet* window. The *Spreadsheet* window offers the user the ability to enter or change individual values or groups of values for any of the model parameters. Individual cells are selected by clicking on the cell using the ***left*** mouse button. A group of cells in one or more rows and columns are selected by pressing and holding the ***middle*** mouse button and then dragging the pointer across the desired group of cells. The selected cells are highlighted. When all desired cells are selected, release the ***middle*** button. Extended editing functions are available within each spreadsheet for operating on a selected cell or group of cells. Functions are selected using the ***left*** mouse button unless specified otherwise. The editing functions are listed and described in table 1.

Clicking on any of the parameter names using the ***left*** mouse button opens a help window that provides detailed information about the parameter. This includes the maximum, minimum and default values, parameter units, a parameter definition, and possibly information regarding selection or estimation of parameter values.

Scalar - Clicking on the *Scalar* option opens the *Scalar Spreadsheet Editor* window (fig. 4.8) which contains the single-value model parameters. Values can be entered or changed by modifying or overwriting the current value.

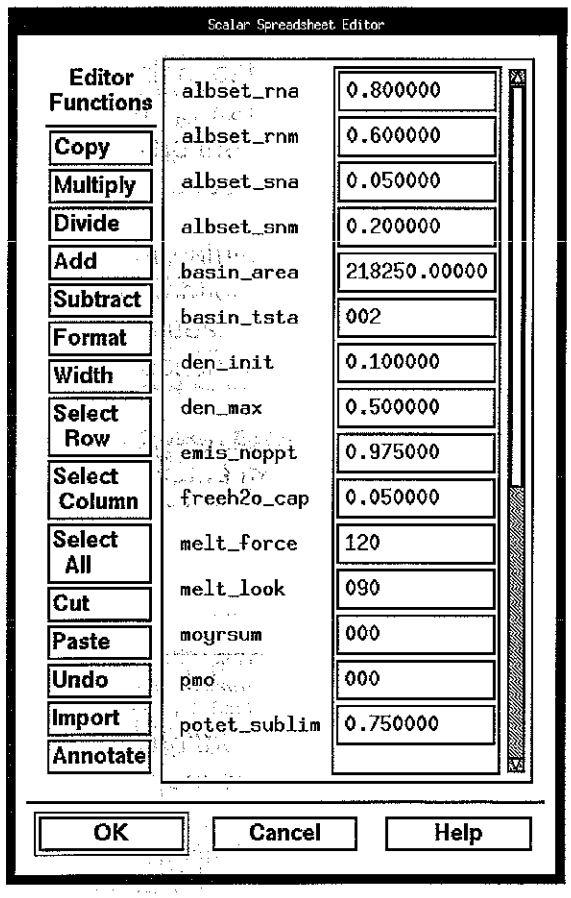

Figure 4.8 Scalar Spreadsheet Editor window.

Single Dimension - Clicking on the *Single Dimension* option opens a window containing a list of all of the one-dimensional indexes defined in the model. Selecting a dimension index and clicking on the *Select* button opens a spreadsheet whose columns are the parameters and whose rows are the dimension index values. For example, the monthly values for all of the parameters dimensioned by month are displayed in the spreadsheet shown in figure 4.9. In this example, the *Dimension Index Names* option has been applied to change the numeric values of the months (1-12), which would appear by default on the left margin of the spreadsheet, to alphabetic names.

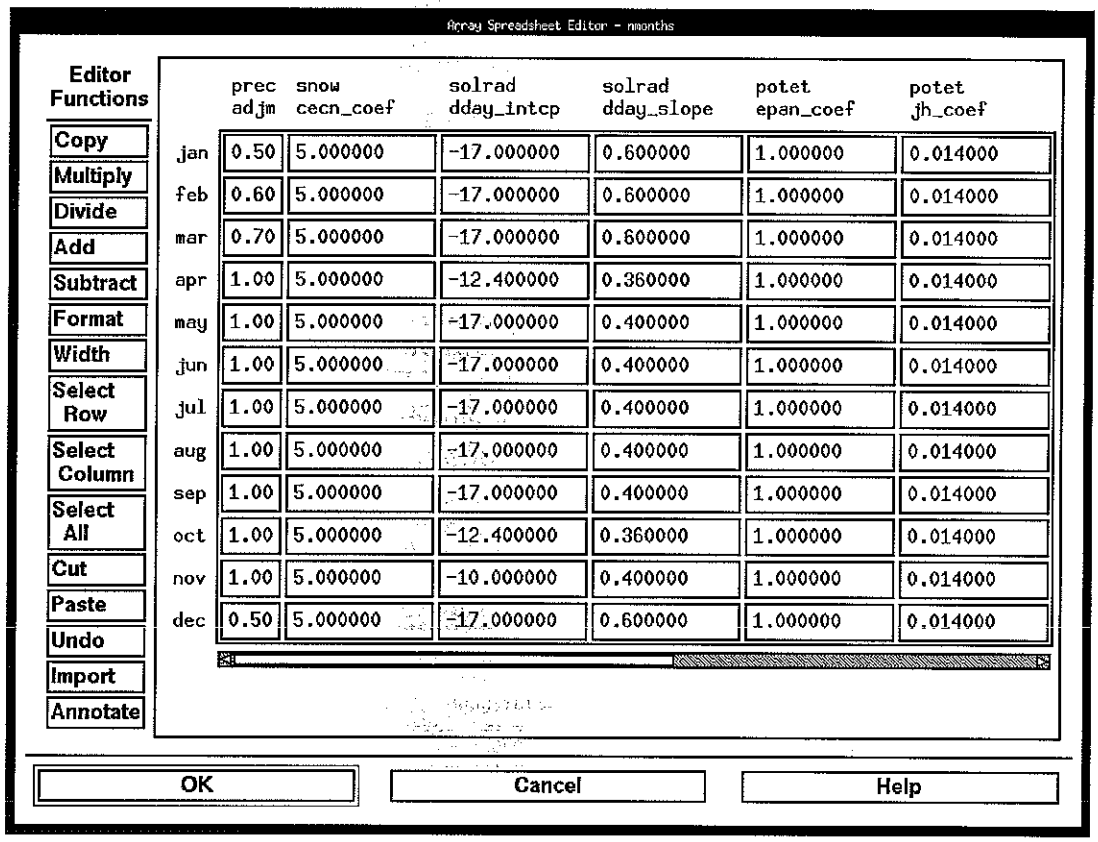

Figure 4.9 Single-Dimension Spreadsheet Editor window.

Multiple Dimension - Choosing this option will produce another menu whose options define the types of 2-D array spreadsheets available. They are:

Whole

This option will produce a menu which includes all of the 2-D parameters defined within the model. Choosing a parameter will produce a spreadsheet where the first dimension defines the columns of the spreadsheet and the second dimension defines the rows. The example in figure 4.10 shows monthly precipitation correction factors

by months (columns) for each hydrologic response unit defined within the basin (rows).

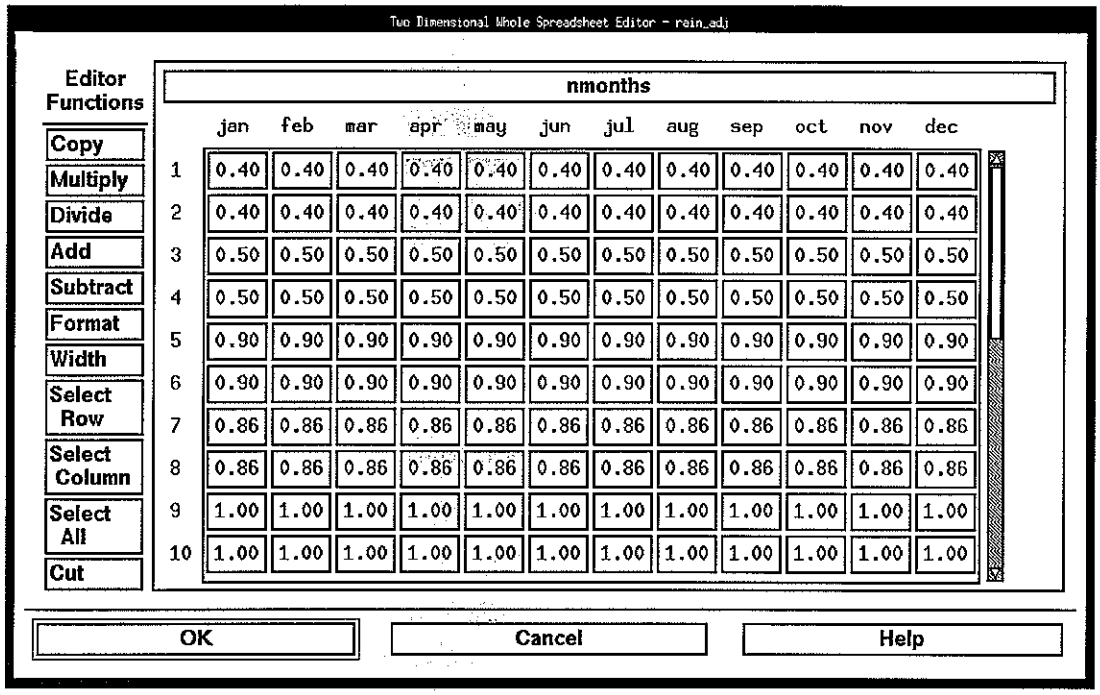

Figure 4.10 Two-Dimensional - Whole Spreadsheet Editor window.

by Column

This option produces a menu of the various 2-D index pairs that have been created in the modules. Choosing an index pair produces a columnar listing of all of the parameters defined by the chosen index combination. An example in figure 4.11 shows the parameters with the 2-D index pair of months and hydrologic response units. The parameters having this index combination are the precipitation adjustment factors *rain_adj* and *snow_adj*. They are displayed one month at a time. The Index Name "jan" and the Index Value "1" shown at the top of the window indicate that the values shown are for the index *nmonth* which is January and has the value of 1. Values for other months are displayed by clicking on the left and right arrows at the top of the window using the ***left*** mouse button. Each row has *nhru* or the number of hydrologic response units as its index and these values are shown on the left side of the columns.

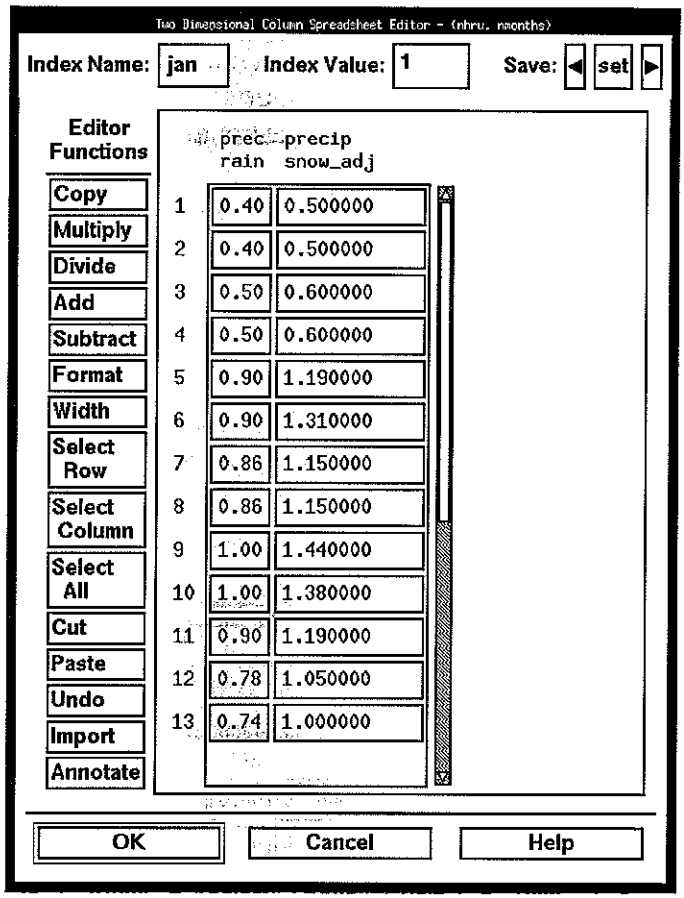

Figure 4.11 Two-Dimensional by Column Spreadsheet window.

4.4.4 Default Values

Selecting this option enables the user to reset all parameter values to their default values as defined in the modules of the model being used. When this option is selected, a *Warning* window is produced with the message "*Do you want to set all parameters to default values?*". Pressing the *OK* button will overwrite all parameter values in the current parameter work file. Pressing the *Cancel* button will exit this option without resetting any parameter values. The user must save the parameter work file after exercising this option in order to retain these parameter changes for future use. **It is important to remember to save the current parameter work file before selecting this option if the user wants to retain the current parameter values for future use.**

4.4.5 Parameter Information

Selecting this option opens a text entry window in which the *Description* field of the *Parameter Information* area in the *MMS Main Interface* window (fig. 4.2) can be entered.

4.4.6 Control Information

Selecting this option opens a text entry window in which the *Description* field of the *Control Information* area in the *MMS Main Interface* window (fig. 4.2) can be entered.

4.4.7 File

Selecting this option opens a *File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The user is free to move to any directory on their system and to view and edit any ASCII file on their system. Any file selected using the dialog will be displayed using the default editor defined by the MMS environment file (*mms.env*) located in the *< user_workspace>/control* directory.

4.5 Run Menu

The *R*un function enables the user to run a model in a number of different modes. The options available in the run pull-down menu are:

- *Single Run*
- *Sensitivity*
- *Optimization*
- *ESP*

4.5.1 Single Run

This option enables the user to run the model for a selected period of time using the parameter and data files shown in the *MMS Main Interface* menu (fig. 4.2). The *Single Run Control* window (fig. 4.12) contains a number of features that can be used to define the period of the run, select a variety of analyses to be conducted on the run output, and specify file names where output is to be written.

Time Info

The upper left quadrant of the window contains the *Time Info* entry box. Here one can set the start and end times (year, month, day, hour, minute, and second) and the initial time step, in hours, for this run.

File Info

In the lower left quadrant is the *File Info* entry box. The option to write model output to a file is selected by pressing the *Model Output* toggle button "on" and entering a file name in the text entry window. If an output file was named using the *Set File Names* window (fig. 4.4), that file name will automatically be placed in the text entry window. Changing the output file name here will also change it in the *Set File Names* file. The output file will be written to the *< user_workspace>/output* directory.

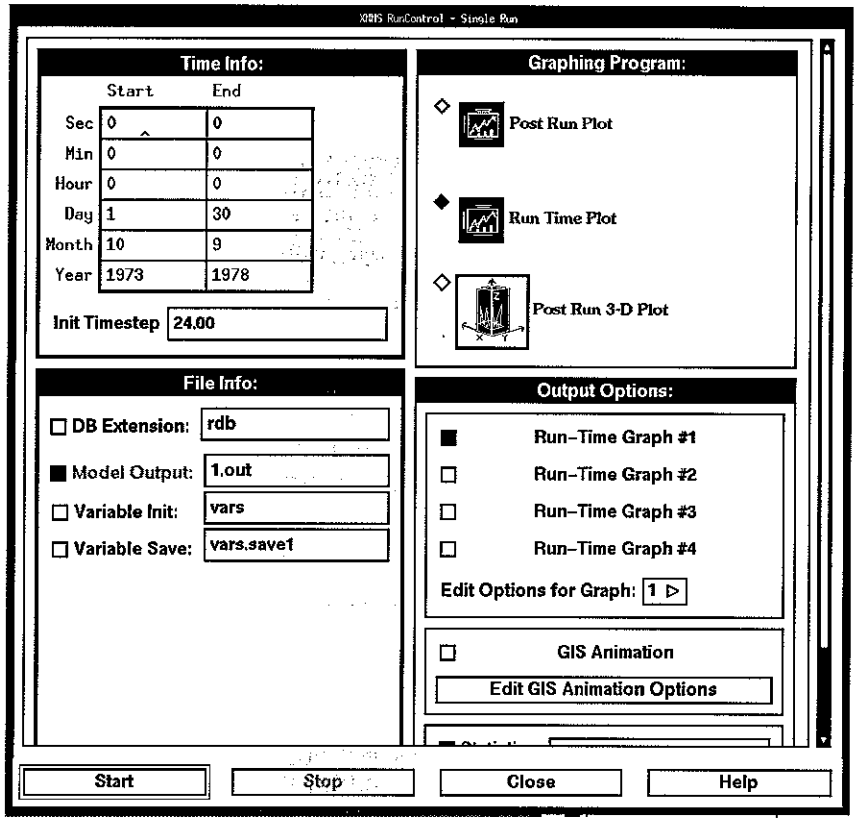

Figure 4.12 Single Run Control window.

When a model is run, there are two options available for the initialization of declared model variables. One is that all declared variables are initialized according to the 'Initialize' sections of the program modules. The second is that all declared variables can be initialized from a set of variables saved at the end of a previous run. The option to save all the declared model variables at the end of a run is selected by pressing the *Variables Save* toggle button "on" and entering a file name in the text entry window. The file will be written to *< user_workspace>/input/vars* directory. To use a previously saved variables file, press the *Variables Init* toggle button "on" and enter the name of the variables file in the text entry window. The file must be located in the *<user_workspace>/input/vars* directory.

Graphing Program

In the upper right quadrant of the window three *Graphing Program* options are available and are selected by pressing the desired toggle button "on." Each option can provide up to four graphical display windows. Only one option can be selected at a time. *Run TIme Plots* are displayed as the model is running. *Post Run Plots* and *Post Run 3-D Plots* are displayed at the end of the model run.

The *Post Run Plot* option will display user-selected time series of model output variables as X-Y plots and provides a variety of interactive display and analysis capabilities. The *Post Run Plot* window (fig. 4.13) has a menu bar across the top with a number of pull-down menus to provide access to all the program features. The *Help* button at the far right side of the menu bar provides access to on-line documentation for use of all the *Post Run Plot* features.

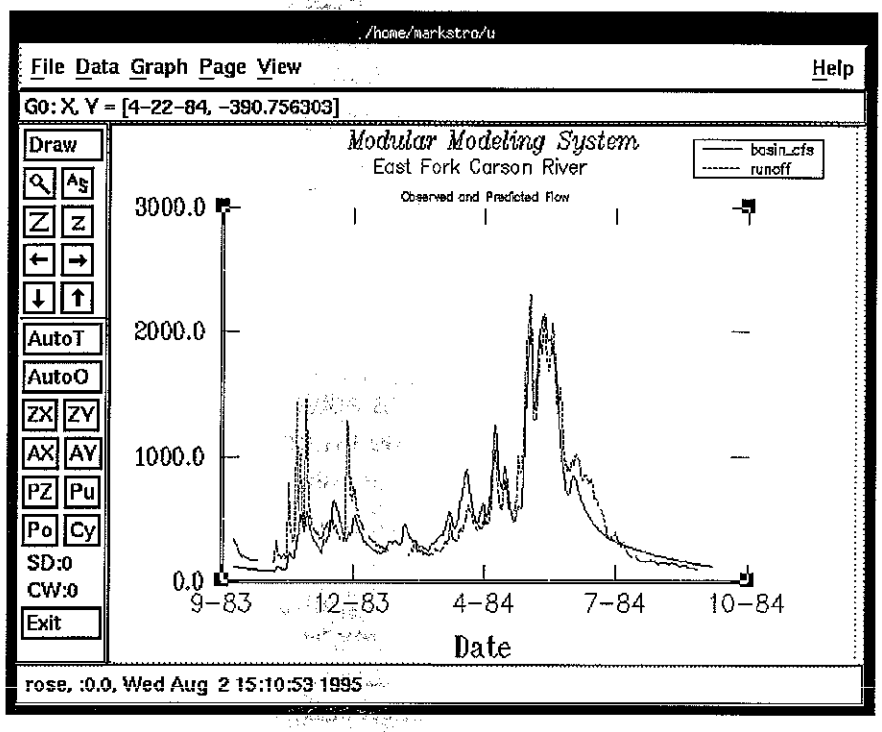

Figure 4.13 Post Run Plot window.

The *Run TIme Plot* option has only X-Y plot display capabilities. The *Run TIme Plot* window (fig. 4.14) has three buttons available. The *Options* button allows access to the *Options* window (fig 4.15) which allows the user to select a variety of scaling and printing options. Either axis can be toggled to a logarithmic scale and the legend can be toggled on or off. There are also numerous zooming options. Finally, there is a hardcopy option. The output device or file is Postscript format. The *Close* button exits the *Run TIme Plot* window. The *Help* button displays help for the *Run TIme Plot* window.

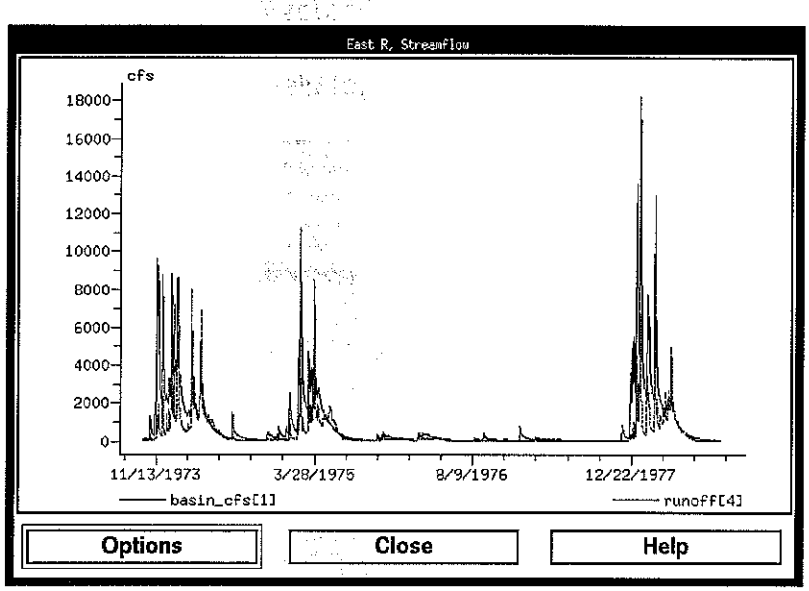

Figure 4.14 Run TIme Plot window.

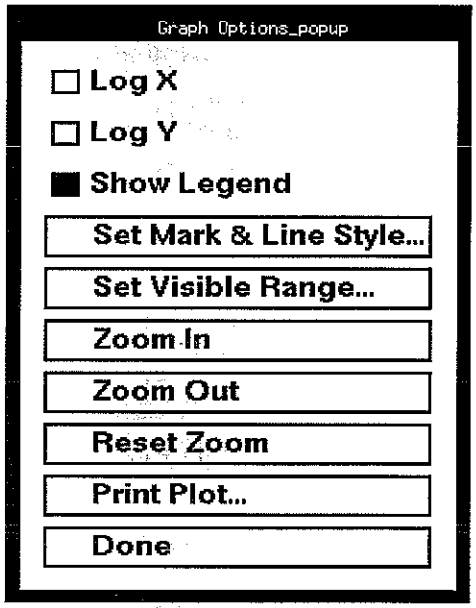

Figure 4.15 Run Time Plot Options window.

The *Post Run 3-D Plot* is a tool that provides 3-dimensional plotting capabilities. The *Post Run 3-D Plot* window (fig. 4.16) provides a 3-dimensional view of a selected variable in time and space. The time step is plotted on the x-axis, the dimension of variable is plotted on the y-axis, and the values of the variable for each dimension and time step are plotted on the z-axis. The plot can be rotated *left, right, up,* and *down* by clicking these buttons located in the second row of options. The *Theta* value in the top row of options is the degrees of rotation in the x-y plane and the *Phi* value is the degrees of rotation in the y-z plane. The upper right button allows the user to toggle between a *Wire Frame* or *Hidden Lines* display. The lower right button can be used to print the plot on the system Postscript printer. Press the *Quit* button to exit the window.

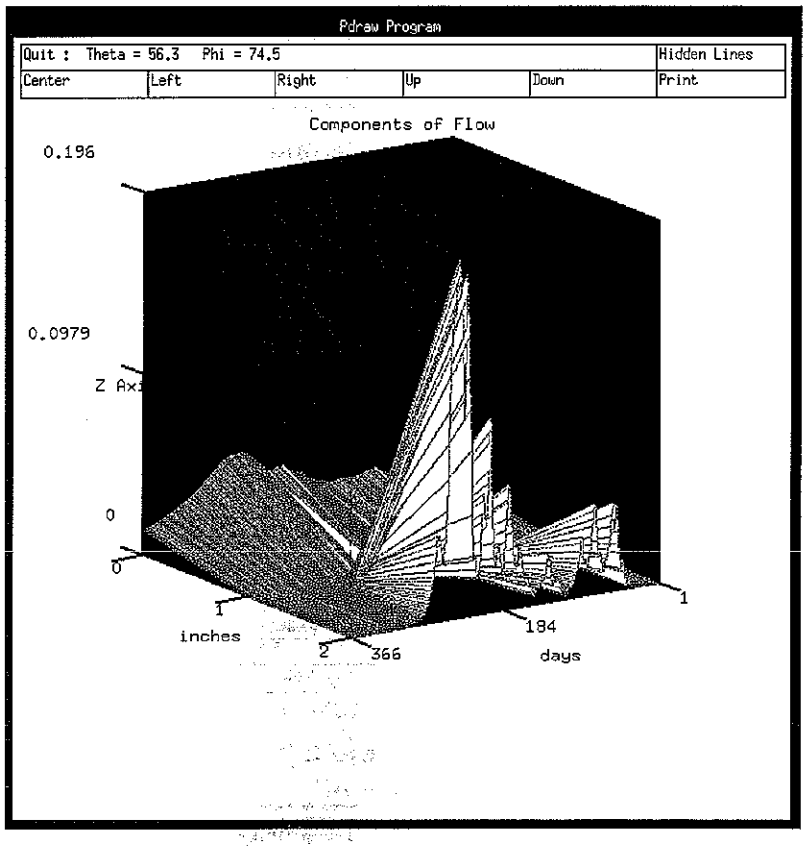

Figure 4.16 Post Run 3-D Plot window.

Output Options

Options to enable the selection of the variables to be displayed with the graphing tools and to turn selected graphs on and off are provided in the *Output Options* section in the lower right quadrant of the *Single Run Control* window (fig. 4.12).

Edit Options for Graphs - The toggle buttons for the four possible graphs must be pressed "on" to display the selected graph. The variables to be displayed in each graph

are selected using the *Edit Options for Graph* button. Clicking on the button opens a set of buttons with the numbers 1,2,3, and 4. Click on the number of the graph to be edited. For the *Post Run Plot* and *Run TIme Plot* options, the *Set Display Variables* window (fig. 4.17) will appear. In this window the variables of the selected graph and the graph titles and axes labels and dimensions are specified. The selection and editing procedure can be repeated for each of the four graphs.

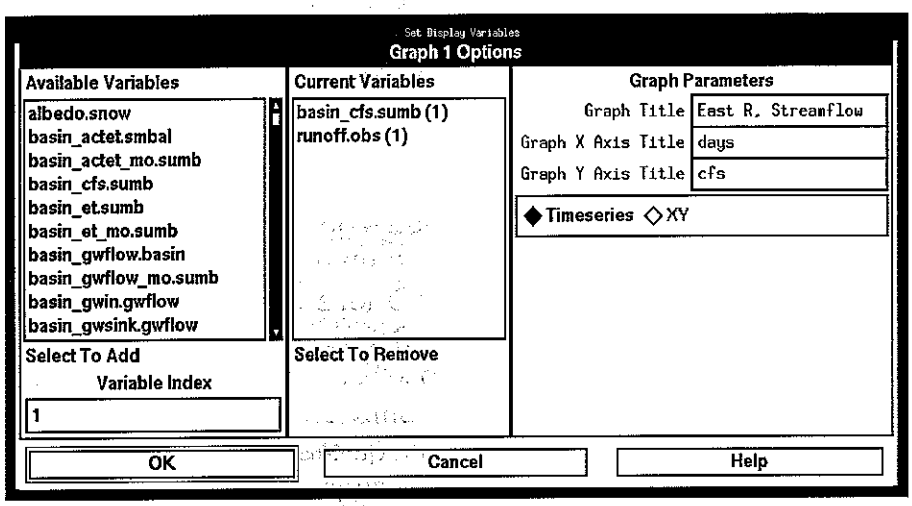

Figure 4.17 Set Display Variables window for Post Run and Run TIme Plots.

In this edit window the *Available Variables* list is a list of all the declared variables in the model. Clicking on a variable name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the variable. Clicking on a variable with the ***left*** mouse button will add the variable to the *Current Variables* list. Clicking on a variable in the *Current Variables* list with the ***left*** mouse button will remove it from the list.

For single- or multi-dimensioned variables, the *Variable Index* box may be used to specify the dimension index number of the array location of the desired variable value before it is selected. For multi-dimension variables the "i,j" dimension is entered in the *Variables Index* box. Any combination of up to 10 single or multiple dimension variables can be displayed in one graph. Changes to a selected graph are saved by pressing the *OK* button. Pressing the *Cancel* button will exit the window without saving changes.

The physical specifications of each graph may be entered in the *Graph Parameters* section of the window. The graph title, titles for the X and Y axes, and the minimum and maximum for the X and Y axes may be specified in the appropriate boxes. For all time increments, the X axis maximum will default to the time period of the run.

For the *Post Run* 3-dimensional graphic option the *Set Display Variables* window (fig. 4.18) is slightly different. The *Available Displays* list is a list of all the single dimensioned

variables and their associated dimension names. Selecting a variable places the variable name in the *Z-Axis* field and the dimension name in the *Y-Axis* field. The time step is selected from the available button options in the *X-Axis* field. The graph title and the axis titles are entered in their associated entry fields.

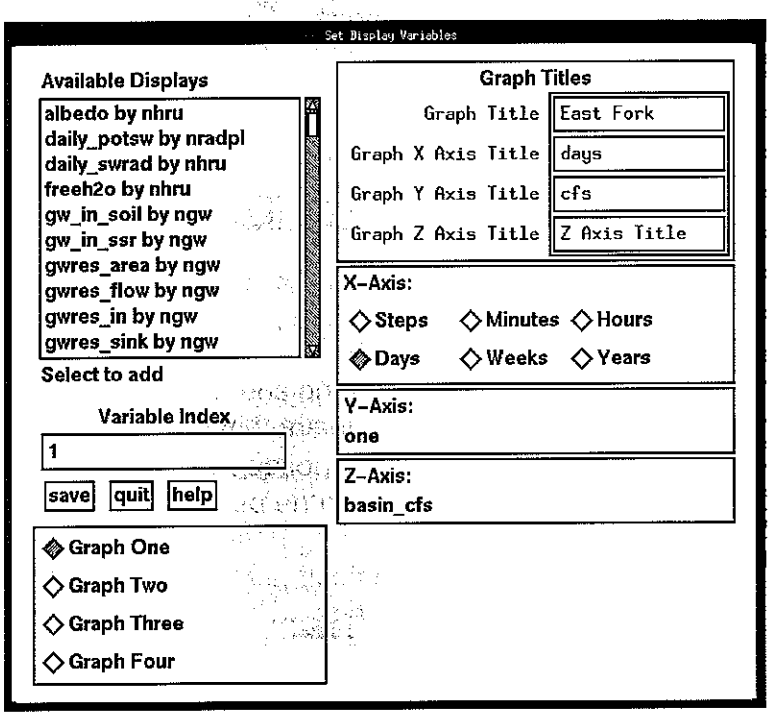

Figure 4.18 Set Display Variables window for Post Run 3-D Plot.

GIS Animation Options - This option enables a user to view the change in a selected spatially distributed variable overlaid on a GIS developed map of a watershed (fig. 4.19). The changes are displayed on the map for each time step as the model is running, thus providing an animated view of the changes.

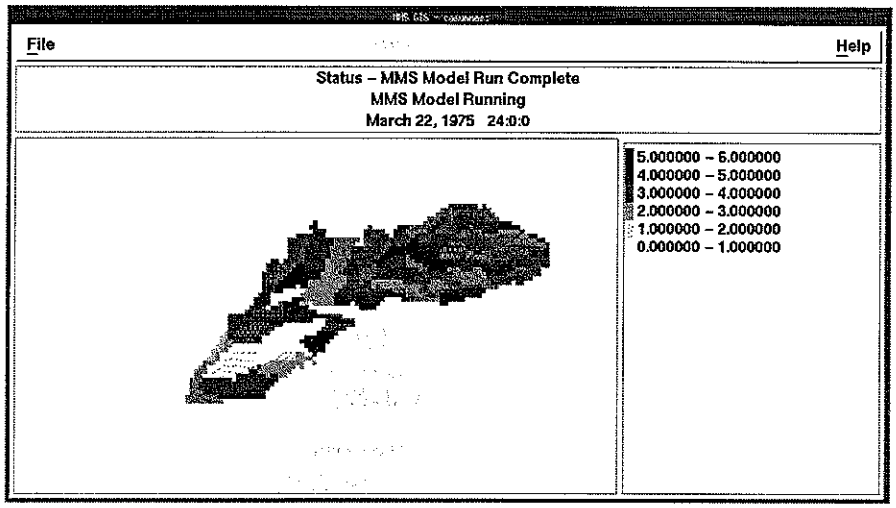

Figure 4.19 GIS Animation window.

When the *Edit GIS Animation Options* button on the *Single Run Control* window (fig. 4.12) is pushed, the *Set Animation Variable* window (fig. 4.20) will appear. The user must specify the path and name of a valid raster map in the *Map Directory* and *Map Name* fields. This is typically the map that has the basin subareas delineated on it. Next, select a variable from the Available Variables list which corresponds to the specified map. Set the variable display range in the min and max fields. These are the minimum and maximum values over which this variable may occur. Enter the number of divisions into which the range from *min* to *max* is to be divided in the # Divisions field. Finally, the format of the raster map file should be specified. These formats include Arc raster ASCII, Grass Raster, and BIL.

Animation output at specific time steps can be saved as GRASS raster maps by specifying the time information and map name in the *Save Map Options* window. Click on the *Add Save item* to construct the *Current maps to save* list. Maps for the dates selected will be saved when the model runs.

When the *GIS Animation* toggle button is activated on the *Single Run Control* window (fig. 4.12), and the *Start* button is pushed, an additional run controller will appear. Users can control the speed of the animation as well as step forward in single time increments.

Edit Statistics Options - Pushing the *Edit Statistics Variables* button on the *Single Run Control* window (fig. 4.12) will activate the *Statistics Variable Options* window (fig. 4.21). The *Available Variables* list is a list of all the declared variables in the model. Clicking on a variable name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the variable. Clicking on a variable with the ***left*** mouse button will add the variable to the *Current Variables* list. Clicking on a variable in the *Current Variables* list with the ***left*** mouse button will remove it from the list. For dimensioned variables, the *Variable Index* box may be used to specify the dimension

index number of the variable before it is selected. Variables in the *Current Variables* list will have statistics computed for them. Up to twenty variables can be selected for statistical analysis.

The statistics produced for each variable are the mean, standard deviation, skewness, minimum, maximum, and a histogram. These statistics are automatically saved in a file in the *< user_workspace>/output* directory. The name for this file is specified in the *Statistics* box above the *Edit Statistics Variables* button (fig. 4.12). A file called *statvar.dat* is also created in the *output* directory and contains the values of the variables listed in the *Current Variables* box (fig. 4.21) for each time step.

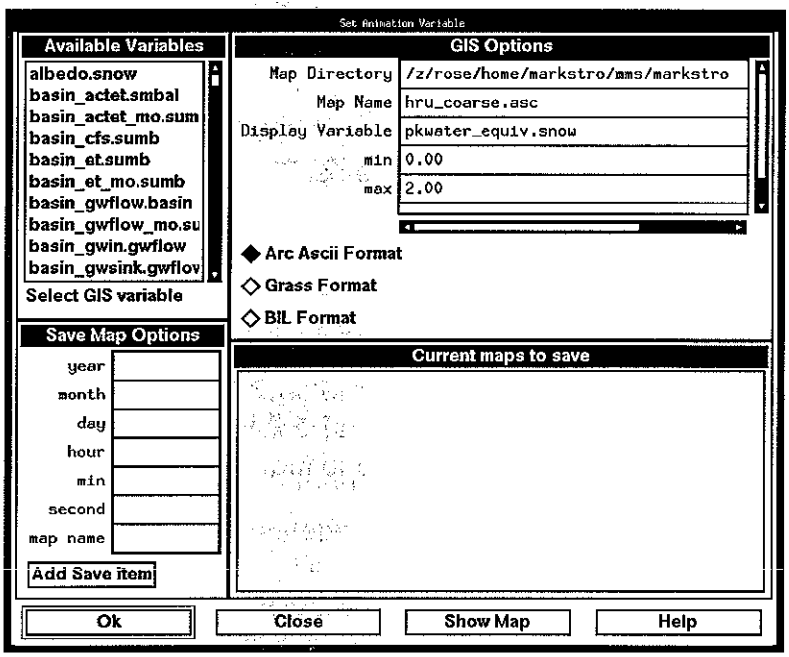

Figure 4.20 Set Animation Variables window.

Edit GIS Output Variables Option - Pushing the *Edit GIS Output Variables* button on the *Single Run Control* window (fig. 4.12) will activate the *GIS Output Options* window. The *Available Variables* list is a list of all the declared variables in the model. Clicking on a variable name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the variable. Clicking on a variable with the ***left*** mouse button will add the variable to the *Current Variables* list. Clicking on a variable in the *Current Variables* list with the ***left*** mouse button will remove it from the list. Variables in the *Current Variables* list will be output in Arc View file format in the output directory. There will be one output file for each dimension and all variables of the same dimension will be written to the same file.

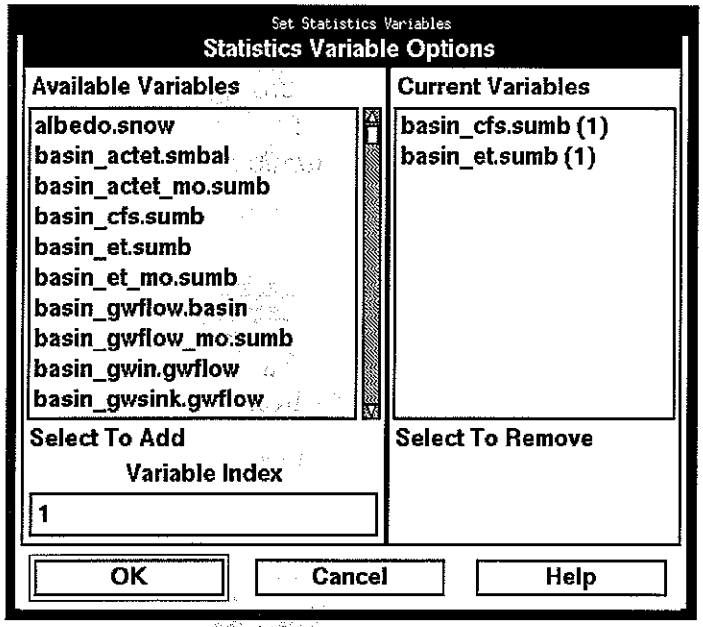

Figure 4.21 Statistics Variables Options window.

Start Button

Clicking on the *Start* button (fig. 4.12) will begin the model run.

Stop Button

The model run may be stopped by the user prior to run completion by clicking on the *Stop* button (fig. 4.12).

4.5.2 Sensitivity

The sensitivity-analysis option allows the user to determine the extent to which uncertainty in selected parameters results in uncertainty in the predicted variable. Discussions of sensitivity analysis and its interpretation are presented by Mein and Brown (1978) and Beck and Arnold (1977).

Two methods of sensitivity analysis are available. One is the method developed for use with the USGS Precipitation-Runoff Modeling System (PRMS) and is described in the PRMS User's Manual (Leavesley and others, 1983). This method allows the evaluation of up to 10 parameters at one time. The second method evaluates the sensitivity of a single parameter or any pair of parameters and develops the objective function surface for a selected range of these two parameters.

PRMS Sensitivity

Selecting the *PRMS Sensitivity* menu item will activate the *PRMS Sensitivity* window (fig. 4.22). The initialization period and the sensitivity analysis period can be set in the *Time Info* field of the window. The initialization period is a user-defined period the purpose of which is to allow the model to cycle a number of times in an attempt to minimize the effects of the user's estimate of initial values of state variables at model start up. The model will execute once for the initialization period and then be run iteratively for the sensitivity analysis period using the computed state variable values at the end of the initialization period as the initial conditions for the full sensitivity analysis. Setting the initialization period to all zeros eliminates the use of an initialization period.

A model can be run for one time period but have the objective function computed for a subset of that time period. An example might be for a snowmelt runoff model where one may be interested in the sensitivity of selected runoff parameters during the melt period of April through July. However, to evaluate this period, the model needs to be run from October to July to accumulate and melt the snowpack. The beginning and ending month for the computation of the objective function used in the sensitivity analysis are set in the *Obj Func Begin Month* and *Obj Func End Month* boxes.

The *Optimization Options* field of the window provides access to tools for defining the objective function and parameters to be used. The objective function is computed as a function of the difference between selected Observed (*O*) and Predicted (*P*) variables. The form of the objective function is determined by the settings of the *Obj Fun Transformation* and the *Obj Fun Sum* buttons in this field.

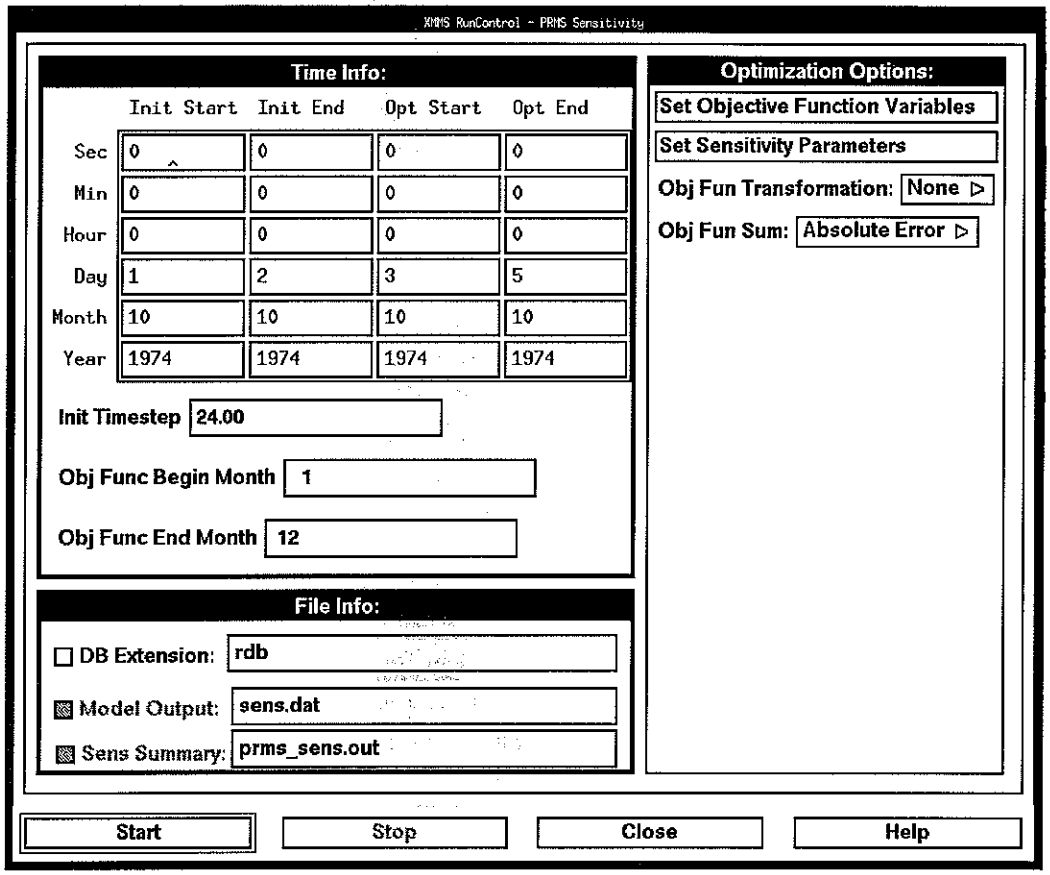

Figure 4.22 PRMS Sensitivity window.

The *Obj Fun Transformation* options are "none" or "log." The *Obj Fun Sum* options are "Absolute Error" and "Squared Error." These provide the following four possible objective functions for use in the analysis where n is the number of time steps over which the objective function is computed:

Obj Fun Transformation = none
Obj Fun Sum = Absolute Error

$$\sum_{1}^{n} |(O - P)| \qquad (1)$$

Obj Fun Transformation = none
Obj Fun Sum = Squared Error

$$\sum_{1}^{n} (O - P)^2 \qquad (2)$$

Obj Fun Transformation = log
Obj Fun Sum = Absolute Error

$$\sum_{1}^{n} |(ln(O+1) - ln(P+1))| \qquad (3)$$

Obj Fun Transformation = log
Obj Fun Sum =Squared Error

$$\sum_{1}^{n} (ln(O+1) - ln(P+1))^2 \qquad (4)$$

The *Set Objective Function Variables* button (fig. 4.22) opens the *Sensitivity Objective Function Variables* window (fig. 4.23). This window allows the user to set the observed and predicted variables to be used in the objective function equation. Pressing the *Set* button (fig. 4.23) opens the *Sensitivity Analysis Objective Function Variables Selection* window (fig. 4.24). All variables declared in the model are displayed in each of two columns, *Observed* and *Predicted*. The observed and predicted variables to be used in the objective function are selected by clicking on the desired variable in each column using the ***left*** mouse button. Clicking on any variable name with the ***right*** mouse button will provide a definition of the variable and its dimension and units. Pressing *OK* copies these variable names into the *Sensitivity Objective Function Variables* window (fig. 4.23). If the selected variables are dimensioned, the user can specify the index for the variable by clicking on the *Ind* box following the variable name and entering the value. The variables are then accepted by pressing the *OK* button in this window.

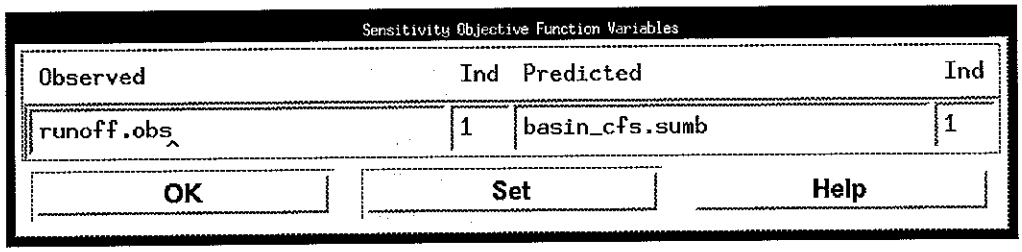

Figure 4.23 Sensitivity Analysis Objective Function Variables window.

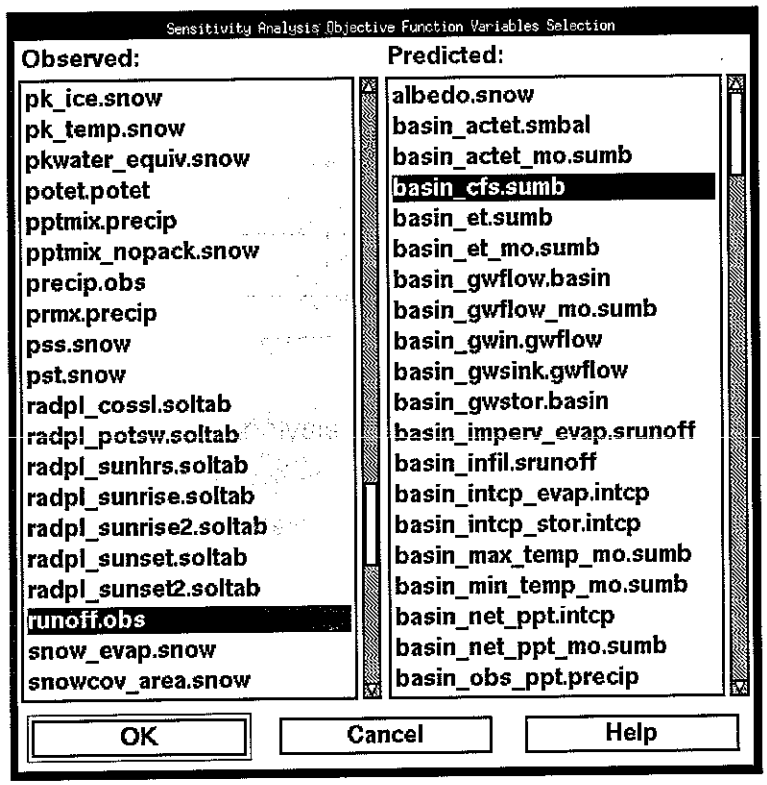

Figure 4.24 Sensitivity Analysis Objective Function Variables Selection window.

Parameters to be evaluated are selected by pushing the *Set Sensitivity Parameters* button (fig. 4.22) which opens the *Sensitivity Analysis Parameters* window (fig. 4.25). All parameters declared in the model are listed in the *Available Parameters* window. Up to 10 parameters can be selected for analysis. Clicking on a parameter name with the ***left*** mouse button opens an *Indices* window (not shown) which is a list of sequential numbers from one to the dimension size of the parameter. All index values can be selected by pressing the *Select All* button in the *Indices* window or a subset of all values can be

selected by individually clicking on the desired values with the *left* mouse button. All selected values are highlighted. For distributed parameters, the sensitivity of the selected parameter is evaluated only for the set of indices selected. Clicking *OK* moves the parameter name and selected indices to the *Selected Parameters* table (fig. 4.25).

To deselect a parameter, click on the parameter name in the *Parameter* column of the *Selected Parameters* table (fig. 4.25) using the *left* mouse button. This will open the *Indices* window (not shown) where clicking on the Cancel button with the *left* mouse button will remove the selected parameter.

In the analysis of a distributed parameter, all values, or the subset of selected values, of the parameter are treated collectively. The values are increased or decreased at the same time and are not treated individually. Thus, the change in objective function associated with a given increase or decrease in a parameter value reflects the aggregate response of that parameter or subset of parameter values. A major assumption in this approach is that the initial estimates of the values of a given distributed parameter are correct with regard to their relative differences in space or time. Two or more subsets of a distributed parameter may be evaluated by selecting the parameter multiple times, each with a different subset of values.

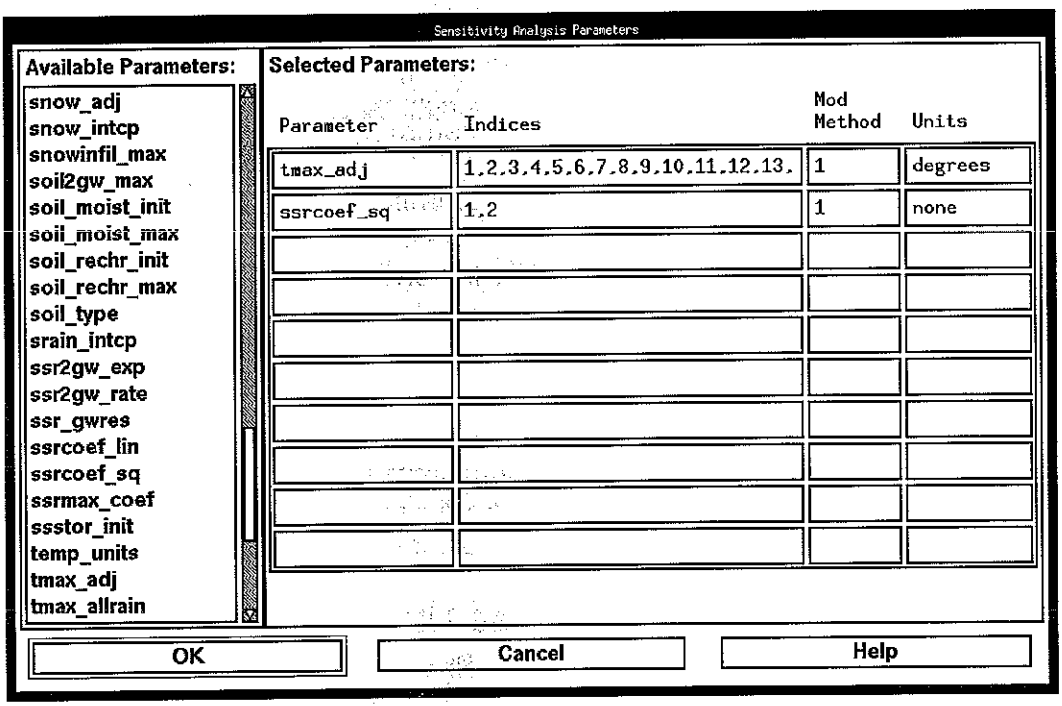

FIgure 4.25 Sensitivity Analysis Parameters window.

The user also has the option to define the manner in which changes in distributed parameter values are made. One option is to change all selected values by the same magnitude. For example, increase or decrease the selected values of a parameter by 0.5

units. The second option is to change all values of the parameter by a percentage of their current values, for example, increase or decrease all selected values of a parameter by 10 percent. These options are selected for each parameter using the *Mod Method* column of the *Selected Parameters* table (fig. 4.25). A zero "0" in this column is used for the "same magnitude" option and a one "1" is used for the "percentage of current value" option.

This sensitivity analysis will produce two output files. The names of these files can be set in the *FIle Info* field of the *PRMS Sensitivity* window (fig. 4.22). The *Model Output* file is the standard MMS output file where model results will be written. The *Sens Summary* file contains a PRMS style sensitivity summary which includes (1) a table of the relative sensitivities of each parameter for each time step, (2) an error propagation table which examines the effects of a 5, 10, 20, and 50 percent error in the estimate of each parameter on the objective function, (3) a measure of the standard error of the parameters, and (4) the diagonal elements of a HAT matrix which provide an indication of the relative influence of each individual time step on an optimization.

Two-Parameter Sensitivity

The two-parameter sensitivity option allows the user to create a map of the objective function (*z* axis) against each of the chosen parameters (*x* and *y* axis). The purpose of this sensitivity is to provide a visualization of the objective function in the region delimited by user-defined maximum and minimum values for each of the two parameters. An example of this type of sensitivity analysis is presented by Eagleson (1978). The objective function used is equation 2 in the PRMS Sensitivity analysis section above.

An special case of this option can be selected to evaluate the sensitivity of a single parameter. This produces the response function for a range of values of a single parameter varying between a user-specified minimum and maximum. The description below applies to the two-parameter sensitivity. However, selecting all settings for the one-parameter sensitivity are identical to selecting the settings for the two-parameter sensitivity, except that only one parameter is selected.

Clicking on the *Two-Parameter Sensitivity* option opens the *Two Parameter Sensitivity* window (fig. 4.26). Most features are similar to those described in the PRMS sensitivity section above. These include the *Time Info* and *File Info* fields and the *Set Objective Function Variables* option in the *Sensitivity Analysis Options* field.

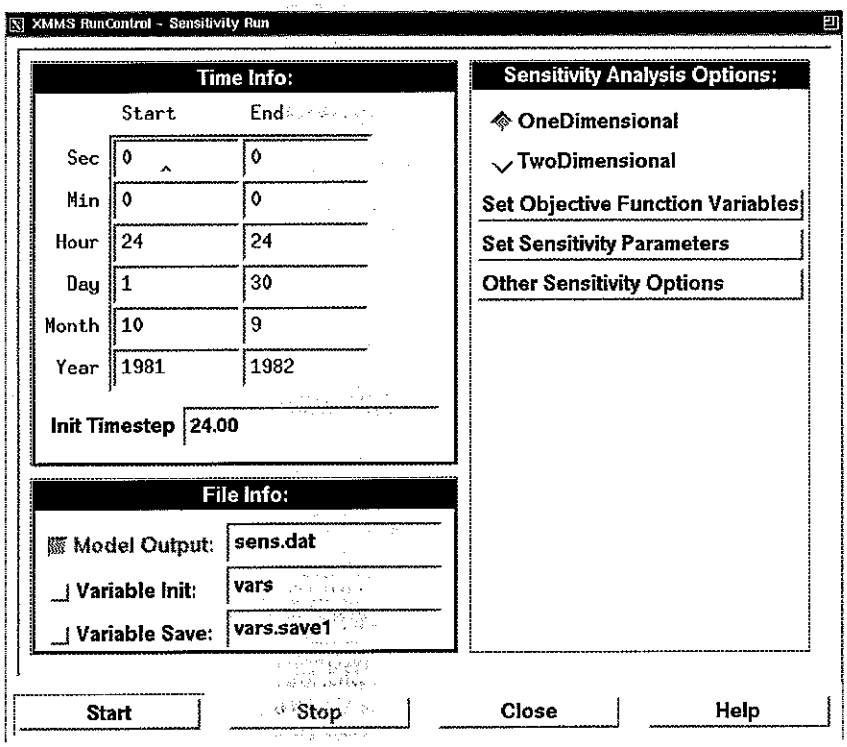

Figure 4.26 Two-Parameter Sensitivity Analysis window.

Clicking the *Set Sensitivity Parameters* button (fig. 4.26) opens the *Set Sensitivity Parameters* window (fig. 4.27). In this window the *Available Parameters* list is a list of all the declared parameters in the model. Clicking on a parameter name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the parameter. Clicking on a parameter with the ***left*** mouse button will add the parameter to the *Current Parameters* list. Clicking on a parameter in the *Selected Parameters* list with the ***left*** mouse button will remove it from the list. For distributed parameters, the *Parameter Index* box may be used to specify the index number of the parameter before it is selected. For multiple dimension parameters the "i,j" dimension is entered in the *Parameters Index* box.

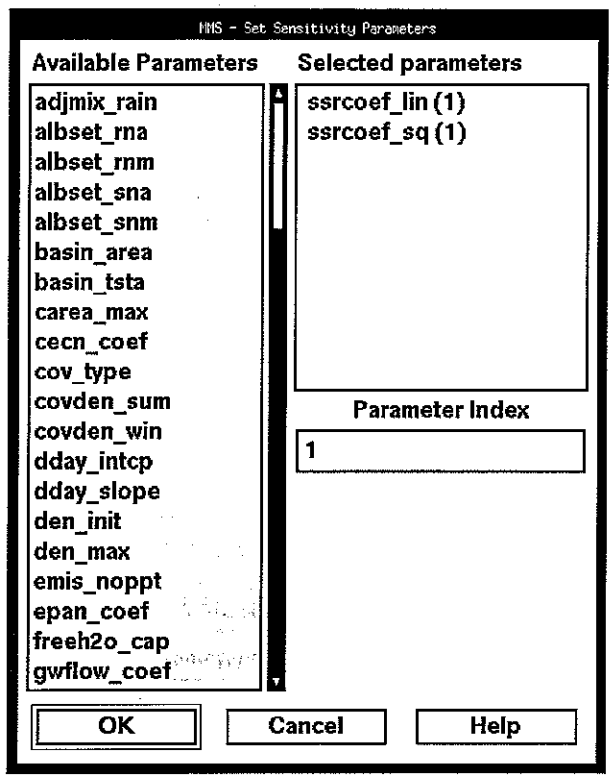

FIgure 4.27 Set Sensitivity Parameters window.

Pressing the *OK* button copies the two selected parameters and indices to input fields of the *Sensitivity Analysis Parameter Summary* window (fig. 4.28). This window is accessed by clicking on the *Other Sensitivity Options* button in the *Two Parameter Sensitivity Analysis* window (fig. 4.26). The *Sensitivity Analysis Parameter Summary* window is used for entering computational limits on the sensitivity analysis. The text entry window options for each parameter are parameter index, where the previously selected index may be revised; the minimum and maximum parameter values to be used in the analysis; and the number of intervals between the minimum and maximum parameter values to be evaluated.

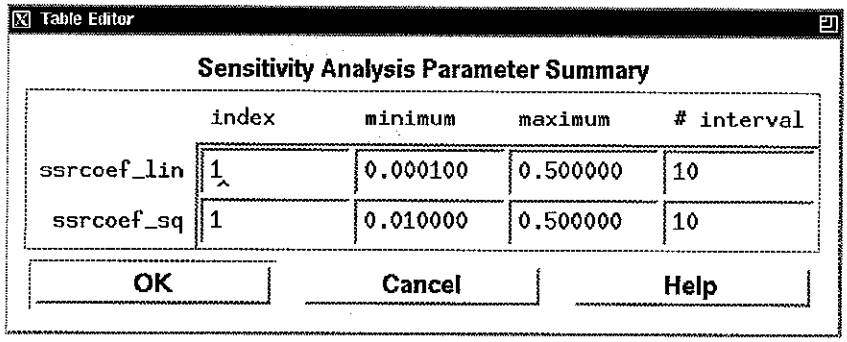

Figure 4.28 Two-Parameter Sensitivity Analysis Parameter Summary window.

For example, if the user selects 10 intervals for the first parameter, and 15 for the second, MMS will carry out 10 x 15 =150 runs of the model for the time period specified. The first parameter will be varied over 10 equal increments between the specified bounds and the second parameter will be varied over 15 equal increments. The objective function results for each pair of parameters will appear on a grid dimensioned 10 x 15.

Sensitivity results are stored in the *<user_workspace>/output* directory. The file name is specified in the *Model Output* box in the *File Info* field of the *Two Parameter Sensitivity Analysis* window (fig. 4.26). Output can be viewed graphically using the *Sensitivity Analysis* option of the *Graph* menu in the *MMS Main Interface* window (fig. 4.2). The graphical options are described below in the Graph section of this chapter.

4.5.3 Optimization

Optimization components control the automatic adjustment of model parameters to obtain better agreement between observed and predicted values. A model parameter is broadly defined as a value that is used to represent a characteristic of a process and is held constant during a simulation run. Using this definition can produce a large number of parameters; however, the availability of a large number of parameters is not meant to encourage optimization of all of them. Two optimization procedures are available to fit user-selected parameters. One is the Rosenbrock technique (Rosenbrock, 1960) as it is implemented in the USGS Precipitation-Runoff Modeling System (PRMS) (Leavesley and others, 1983). The second is a hyper-tunnel method (Restrepo and Bras, 1982).

Rosenbrock Optimization

Selecting the *Rosenbrock* menu item from the optimization choices in the *Run* menu will activate the *Rosenbrock Optimization* window (fig. 4.29). This window is similar in functionality to the *PRMS Sensitivity* window (fig. 4.22) described above. The initialization period and the optimization period can be set in the *Time Info* field. The initialization period is a user-defined period the purpose of which is to allow the model to cycle a number of times in an attempt to minimize the effects of the user's estimate of

initial values of state variables at model start up. The model will execute once for the initialization period and then be run iteratively for the optimization period using the computed state variable values at the end of the initialization period as the initial conditions for the full optimization. Setting the initialization period to all zeros eliminates the use of an initialization period.

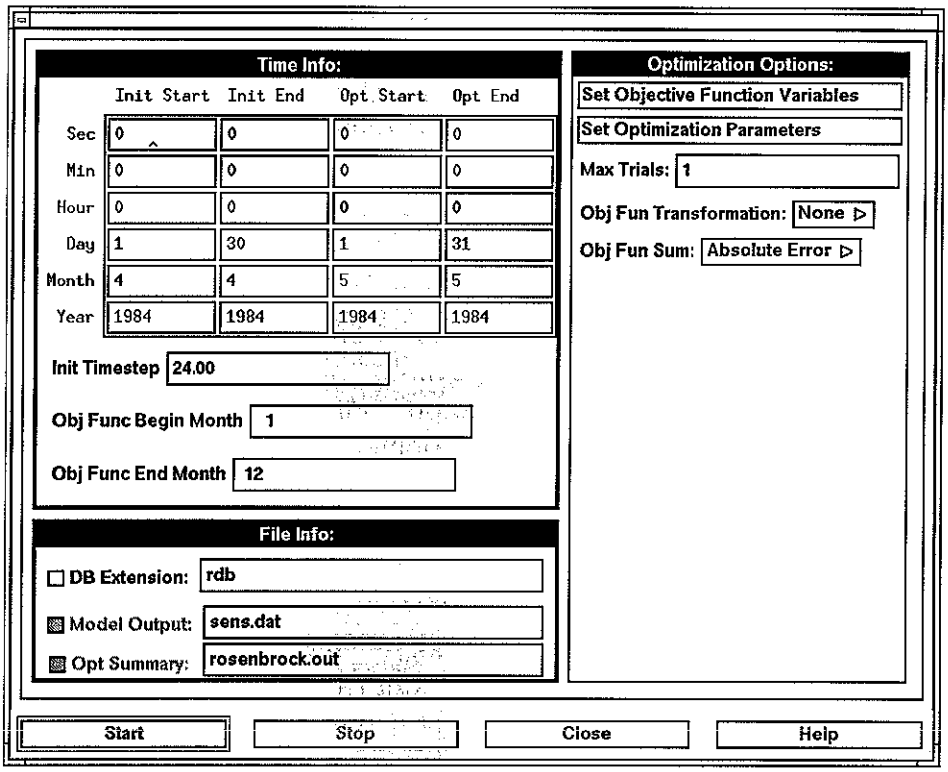

Figure 4.29 Rosenbrock Optimization window.

As described in the sensitivity analysis section, the model can be run for one time period but have the objective function computed for a subset of that time period. The beginning and ending month for the computation of the objective function used in the optimization can also be set in the *Obj Func Begin Month* and *Obj Func End Month* boxes.

The *Optimization Options* field provides access to defining the objective function and parameters to be used. The objective function is computed as a function of the difference between selected observed and predicted variables. The form of the objective function is determined by the settings of the *Obj Fun Transformation* and the *Obj Fun Sum* buttons in this window. The *Obj Fun Transformation* options are "none" or "log" and the *Obj Fun Sum* options are "Absolute Error" and "Squared Error". The objective function equations

resulting from the available combinations of these options are shown in equations 1-4 in the PRMS Sensitivity Analysis discussed above.

The *Set Objective Function Variables* button opens the *Rosenbrock Objective Function Variables* window (fig. 4.30). This window allows the user to set the observed (O) and predicted (P) variables to be used in the objective function equation. Pressing the *Set* button opens the *Rosenbrock Objective Function Variables Selection* window (fig. 4.31). All variables declared in the model are displayed in two columns, *Observed* and *Predicted*.

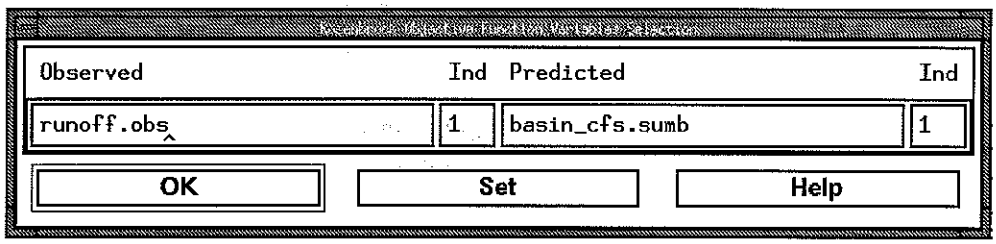

Figure 4.30 Rosenbrock Objective Function Variables window.

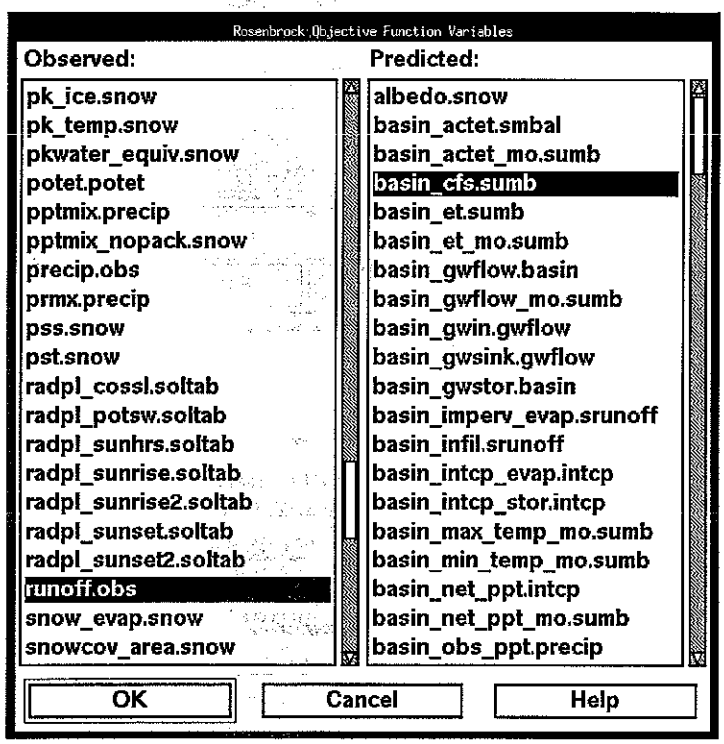

Figure 4.31 Rosenbrock Objective Function Variables Selection window.

The observed and predicted variables to be used in the objective function are selected by clicking on the desired variable in each column using the ***left*** mouse button. Clicking on any variable name with the ***right*** mouse button will provide a definition of the variable and its dimension and units. Pressing *OK* copies these variable names into the *Rosenbrock Objective Function Variables* window (fig. 4.30). If the selected variables are dimensioned, the user can specify the dimension index for the variable by clicking on the *Ind* box following the variable name and entering the value. The variables are then accepted by pressing the *OK* button in this window.

Parameters to be optimized are selected by pushing the *Set Optimization Parameters* button (fig. 4.29) which opens the *Rosenbrock Optimization Parameters* window (fig. 4.32). All parameters declared in the model are listed in the *Available Parameters* window. Up to 10 parameters can be selected for optimization at any one time. Clicking on a parameter name with the ***left*** mouse button opens an *Indices* window (not shown) which is a list of sequential numbers from one to the dimension size of the parameter. All index values can be selected by pressing the *Select All* button on the *Indices* window or a subset of all values can be selected by individually clicking on the desired values with the ***left*** mouse button. All selected values are highlighted. For distributed parameters, optimization of a selected parameter is evaluated only for the set of indices selected. Clicking *OK* moves the parameter name and the selected indices to the *Selected Parameters* table (fig. 4.32).

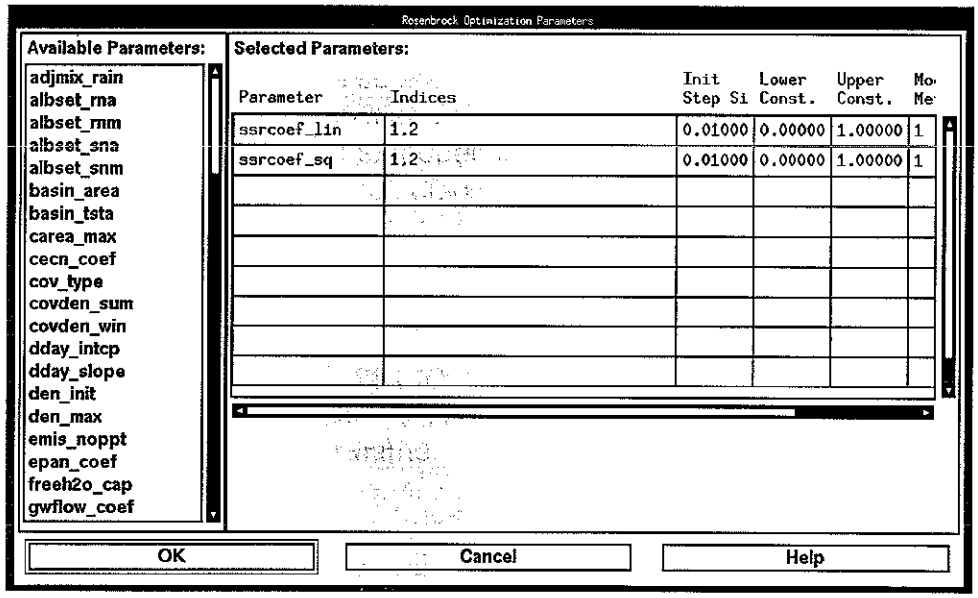

Figure 4.32 Rosenbrock Optimization Parameters window.

To deselect a parameter, click on the parameter name in the *Parameter* column of the *Selected Parameters* table using the ***left*** mouse button. This will open the *Indices*

window (not shown) where clicking on the Cancel button with the *left* mouse button will remove the selected parameter.

Additional items to be filled in the *Selected Parameters* table are the *Initial Step Size* for parameter adjustment and the *Lower Constraint* and *Upper Constraint* for the parameter. The *Initial Step Size* is expressed as a decimal fraction of parameter magnitude. A value of 0.1 would cause the selected parameter to be increased by 10 percent in the first iteration of the fitting process. The *Lower and Upper Constraints* are expressed in the units of the parameter and define the region over which the parameter will be fitted.

As with the PRMS Sensitivity Analysis described above, in the optimization of a distributed parameter, all values, or the subset of selected values, of the parameter are treated collectively. The values are increased or decreased at the same time and are not treated individually. The user has the same options to define the manner in which changes in distributed parameter values are made. One option is to change all selected values by the same magnitude. These options are selected for each parameter using the *Mod Method* column of the *Selected Parameters* table. A zero ("0") in this column is used for the "same magnitude" option and a one ("1") is used for the "percentage of current value" option.

The *Max Trials* input option (fig. 4.29) is used to set the number of iterations over which the optimization is to be run. One pass is made through all the selected parameters for each iteration.

The optimization will produce two output files. The names of these files can be set in the *FIle Info* window. The *Model Output* file is the standard MMS output file where model results will be written. The *Opt Summary* file contains a PRMS style optimization summary which includes initial and final parameter values and the objective function value for each iteration of the optimization procedure.

The optimized parameter values reside in the parameter database at the end of the optimization run. To keep these optimized values, the parameter file must be saved using the *Save Parameters* option of the *File* pull-down menu in the *MMS Main Interface* window (fig. 4.2) before making any changes to the file or before exiting MMS.

Hyper-tunnel Optimization

The Hyper-tunnel optimization (Restrepo and Bras, 1982) is a variation on the Davidon Fletcher Powell (DFP) optimization procedure (Davidon, 1959; Fletcher and Powell, 1963). Two modifications were made to the DFP method. The first is in the use of constraints for enforcing maximum and minimum parameter values. The second selectively excludes those parameters to which the objective function is less sensitive. The rest of the algorithm is identical to the DFP algorithm.

The calculation of the Hessian is re-started after a number of major iterations. The search direction is computed based on the current gradient and Hessian matrix, and a

minor iteration is conducted along this direction. At this point the Hessian is recalculated and a new direction of search is determined. The set of parameters is revised to exclude those parameters having no sensitivity and to include those with high sensitivity. Then a new major iteration is performed.

The objective function used is equation 2 in the PRMS Sensitivity Analysis section above. The algorithm finds the minimum value of the objective function by performing a quadratic interpolation between the best three values of the objective function. The convergence criterion is based on the change in the value of the objective function between major iterations.

Clicking on the *Hyper-tunnel* option opens the *Hyper-tunnel Optimization* window (fig. 4.33) which is similar to the *Rosenbrock Optimization* window. All features are similar to those described in the Rosenbrock Optimization section above. These include the *Time Info* and *File Info* fields and the *Set Objective Function Variables* option in the *Optimization Options* field.

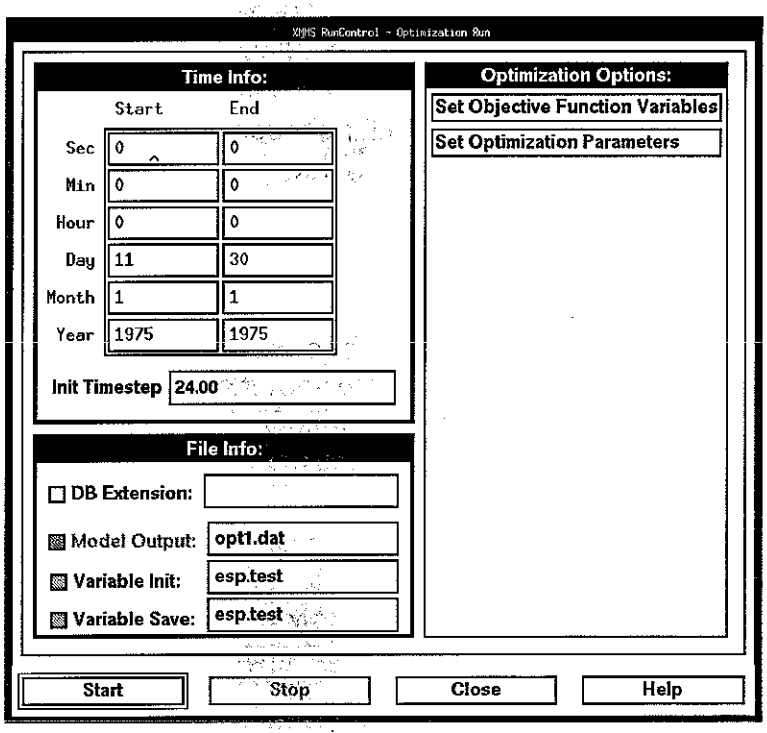

Figure 4.33 Hyper-tunnel Optimization window.

Clicking on the *Set Optimization Parameters* button opens the *Hyper-tunnel Optimization Parameter Selection* window (fig. 4.34). In this window the *Available Parameters* list is a list of all the declared parameters in the model. Clicking on a parameter name with the ***right*** mouse button opens a help window that provides the

definition, units, and dimensions of the parameter. Clicking on a parameter with the *left* mouse button will add the variable to the *Selected Parameters* list. Clicking on a parameter in the *Selected Parameters* list with the *left* mouse button will remove it from the list. For distributed parameters, the *Parameter Index* box may be used to specify the index number of the parameter before it is selected. For multiple dimension parameters, the "i,j" dimension is entered in the *Parameters Index* box.

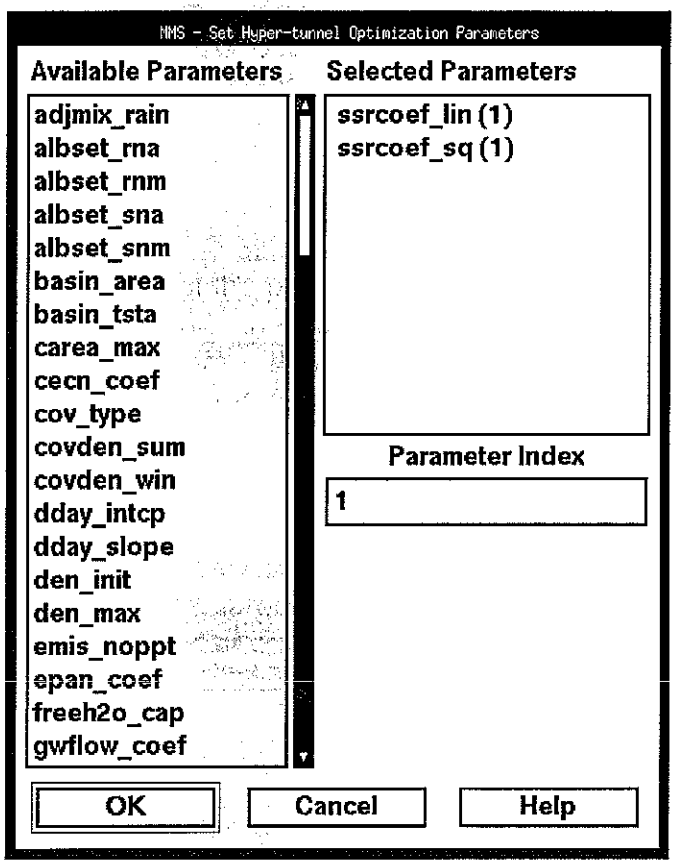

Figure 4.34 Hyper-tunnel Optimization Parameter Selection window.

Up to ten parameters can be selected for optimization. Pressing the *OK* button accepts the selected parameters and indices. Pressing *Start* in the *Hyper-tunnel Optimization* window (fig. 4.33) initiates the optimization. The optimized parameter values reside in the parameter database at the end of the optimization run. To keep these optimized values, the parameter file must be saved using the *Save Parameters* option of the *File* pull-down menu in the *MMS Main Interface* window (fig. 4.2) before making any changes to the file or before exiting MMS.

4.5.4 ESP

A modified version of the U.S. Weather Service's Extended Streamflow Prediction (ESP) program (Day, 1985) has been coupled to MMS to provide forecasting capabilities which include short-term and seasonal forecasting for floods and water supply and evaluation of the effects of land-use and climate changes on hydrologic response. The ESP procedure uses historic or synthesized meteorologic data to forecast future streamflow, given the simulated hydrologic conditions for a watershed at a specified point in time. When historic data are used, the procedure assumes that past meteorological events are representative of future meteorological events. Alternative assumptions about future meteorological conditions can be made with the use of synthesized meteorological data.

The current implementation of ESP is directed to forecasting streamflow. The user defines a forecast period, which can vary from a few days to an entire year. Typically, a model is run up to the start of the forecast period and all the model state variables are saved. A streamflow hydrograph is then simulated for a user-defined forecast period for each year in the meteorological database. The model is reinitialized at the start of each iteration of the forecast period using the saved set of state variables that are assumed to be representative of the conditions on the first day of the forecast period. The simulation results obtained from this iterative procedure are termed the conditional simulation.

Streamflow variables of interest are extracted from each forecast hydrograph and stored in an ESP file. The variables that can be extracted are the maximum daily flow, flow volume, and the dates that the flow decreases to less than as many as three selected threshold values. One variable, or any combination of variables, can be selected for a given model run.

The ESP procedure is implemented by selecting the *ESP* option in the *Run* menu (fig. 4.2) which opens the *ESP* window (fig. 4.35). The *Time Info* part of the window is where the forecast period is defined. The forecast period can vary from a few days to an entire water year.

Two options are available to set the initial values of the state variables for the first day of the forecast period. One is to use the initial conditions that are defined by the model at start up and assume that these conditions are representative of the first day of the forecast period.

A second option is to run the model, in the *Single Run* mode, up to the day prior to the first day of the forecast period using historic data. The *Save Variables* option described in the *Single Run* mode discussion above is selected to save the variables to a file. This saved file can be specified in the *Variable Init* field of the *File Info* part of the *ESP* window. Turning this option on will cause the file specified to be read into the variables database prior to model execution. The *Model Output* option of *File Info* specifies the standard MMS output file where model results will be written.

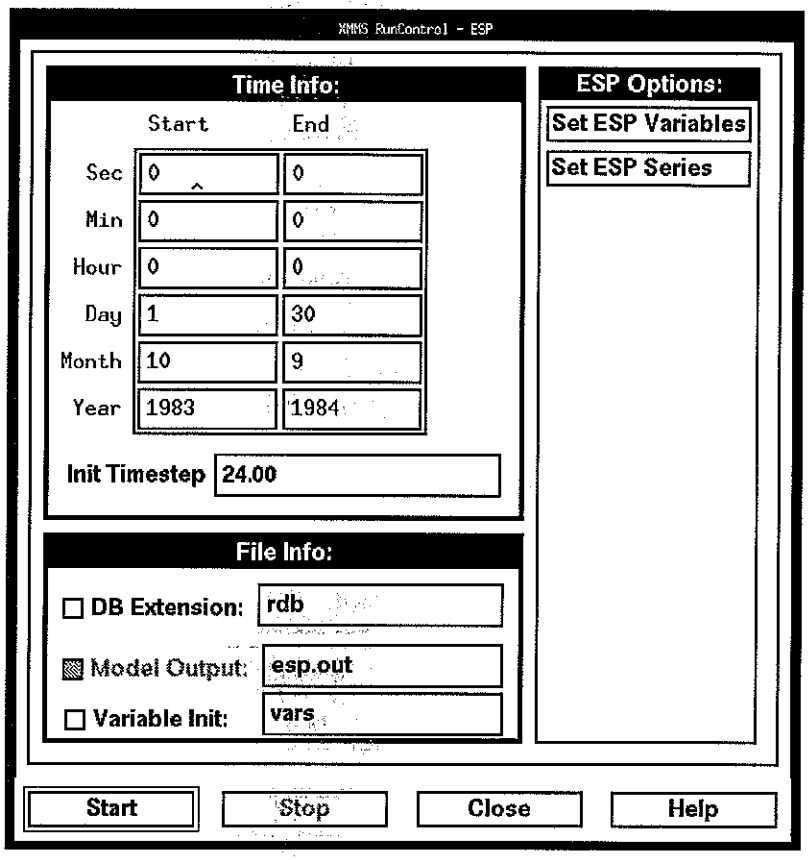

Figure 4.35 ESP window.

Selecting the *Set ESP Variables* button opens the *Set ESP Variables* window (fig. 4.36). In this window the *Variables* list is a list of all the declared variables in the model. Clicking on a variable name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the variable. Clicking on a variable with the ***left*** mouse button will place the variable in the *Selection* box. Clicking on another variable in the *Variables* list with the ***left*** mouse button will overwrite the *Selection* box. For distributed variables, the *Variable Index* box may be used to specify the index number of the variable before it is selected. For the current implementation, this variable must be a streamflow variable.

The other *Selection* options are used to define a variety of measures of streamflow that will be used in the frequency analysis. *Flows* is used to select cubic feet per second (*cfs*) or cubic meters per second (*cms*) for the measure of peak flow. *Volumes* is used to select cubic feet per second per day (*cfsd*), acre-feet (*acft*), or cubic meters (*m3*) for the measure of flow volume over the forecast period. *Thresholds* permit the selection of up to three different flow values for frequency analysis computation of the date that the flow drops below the selected threshold values.

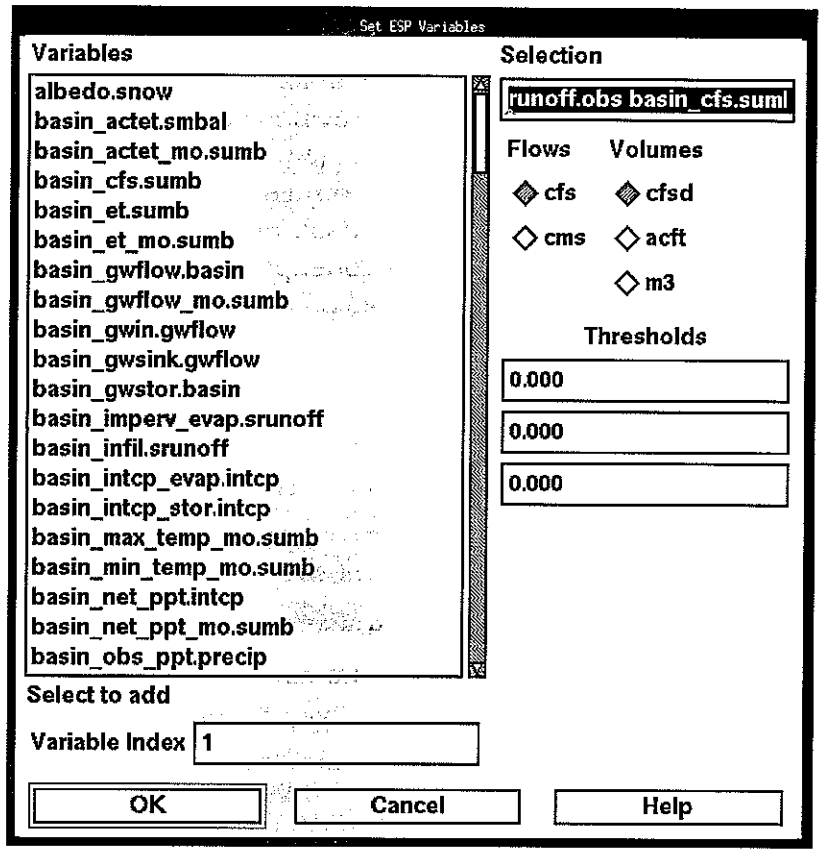

Figure 4.36 Set ESP Variables window.

Selecting the *Set ESP Series* button (fig. 4.35) opens the *Set ESP Series* window (fig. 4.37). This window allows the user to select an additional set of variables whose values will be written to the file named in the *Output file name* field of the window. An output time series for each variable and for each forecast period is written to this file. In the *Available Variables* column of this window is a list of all the declared variables in the model. Clicking on a variable name with the ***right*** mouse button opens a help window that provides the definition, units, and dimensions of the variable. Clicking on a variable with the ***left*** mouse button will place the variable in the *Selected Variables* column. Clicking on a variable in the *Selected Variables* column with the ***left*** mouse button will remove it from the list. For dimensioned variables, the *Variable Index* box may be used to specify the dimension index number of the variable before it is selected.

The ESP program reads the file generated for the ESP variable, performs a frequency analysis on each of the selected measures of the ESP variable, and produces a probabilistic forecast for each of the variables in the file. These may include:

* Probability distribution of peak flow
* Probability distribution of total volume

* Probability distribution of time to peak flow
* Probability distribution of time to each of the three low flows

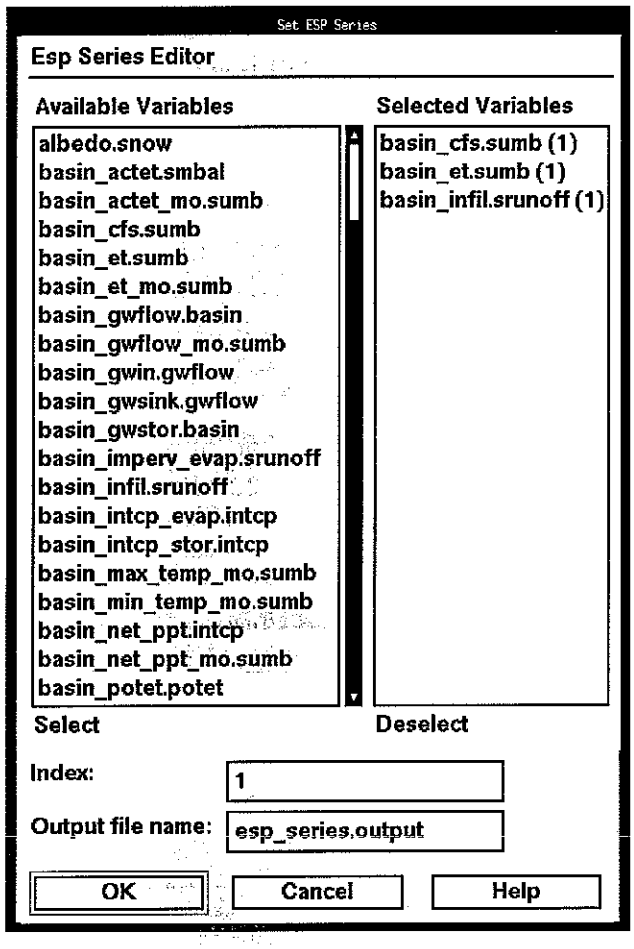

Figure 4.37 Set ESP Series window.

The Log-Pearson Type III probability distribution is currently (1996) supported, but other distributions could be added. Frequency analysis results for the conditional simulation are presented in tabular form and as a graphical plot. The graphical plot options are discussed in the *Graph Menu* section below.

The ESP time series selected in the *Set ESP Series* window (fig. 4.37) are saved to provide these values for use in a variety of external user applications. One example is the selection and use of a specific flow hydrograph for a user-defined probability of occurrence for application in a management model to assess the probable impacts of the selected flow on management options.

4.6 Graph Menu

The *Graph* pull-down menu (fig. 4.2) allows the user to obtain plots of sensitivity and ESP variables output. The graph pull-down menu contains the following options:

- *Sensitivity Analysis*
- *ESP*

4.6.1 Sensitivity Analysis

This option produces a menu of sensitivity analysis plots that were generated using the *Two-Parameter* sensitivity analysis procedure described above. Five different types of plots are available. These are:

- **Display Contours in xgcontour-** Simple contour plot in which the maximum and minimum values of the objective function are highlighted.
- **Display plots of families, variable 1-** Graph of the objective function values for all values of one variable, while considering the values of the other variable constant along each curve.
- **Display plots of families, variable 2-** Same as above for second variable
- **Display contours - Alternative-** Contour plot of the objective function. By pressing the *left* mouse button and dragging the cursor along the contours, the system displays the values of the objective function, and of each of the parameters.
- **Display Objective function in 3-D-** Plot of the objective function in a simple 3D view.

4.6.2 ESP

This option produces a menu of the four plots that are available for the *ESP Variable* selected for output. These are:

- **Maximum Flow Probability Distribution Curve-** Plot of the probability distribution of the maximum values of the *ESP Variable*.
- **Volume Probability Distribution Curve-** Plot of the probability distribution of the computed volume of the *ESP Variable*.
- **Time to Max. Probability Distribution Curve-** Plot of the probability distribution of the time to the maximum value of the *ESP Variable*.
- **Time to Threshold Probability Distribution Curves-** Plots of the probability distribution of the time for the *ESP Variable* to fall below each of the three

user-defined thresholds.

4.7 Print Menu

The *Print* pull-down menu (fig. 4.2) allows the user to create a print file of either the parameters file or the variables file. The options available are:

- *Parameters*
- *Variables*

4.7.1 Parameters

Selecting the *Parameters* option allows the user to create a print file listing the current parameter values. The parameters are saved in the file name that is specified in the *parameter print* field of the *Set File Names* window (fig. 4.4) which is accessed from the *File* menu of the MMS Main Interface window (fig. 4.2).

4.7.2 Variables

Selecting the *Variables* option allows the user to create a print file listing the state variables at the end of a model run. The variables are saved in the file that is specified in the *variable print* field of the *Set File Names* window (fig. 4.4) that is accessed from the *File* menu of the MMS Main Interface window (fig. 4.2).

4.8 Help Menu

The *Help* pull-down menu (fig. 4.2) allows users to get information on various aspects of the MMS interface, modules, and models. Installation of a Hypertext Markup Language (HTML) viewer, such as **Mosaic** (fig. 4.38), is required for use of the MMS help system. Specification of this viewer must be made when MMS is installed. Consequently, the appearance and functionality of the help system will depend on the selected viewer.

All on-line documentation is either written in or converted to HTML. *Help* menu items and *Help* buttons in MMS are referenced by hypertext links to the appropriate sections of these documents. After a *Help* selection is made, the *Help* system loads the appropriate section of the on-line documentation. As with any hypertext document, there may be links to other related documents.

A model specific HTML file is written by *xmbuild* to the user's *models* directory. This contains information about model construction and links to the model specific modules. This file is used anytime help is requested for model or modules.

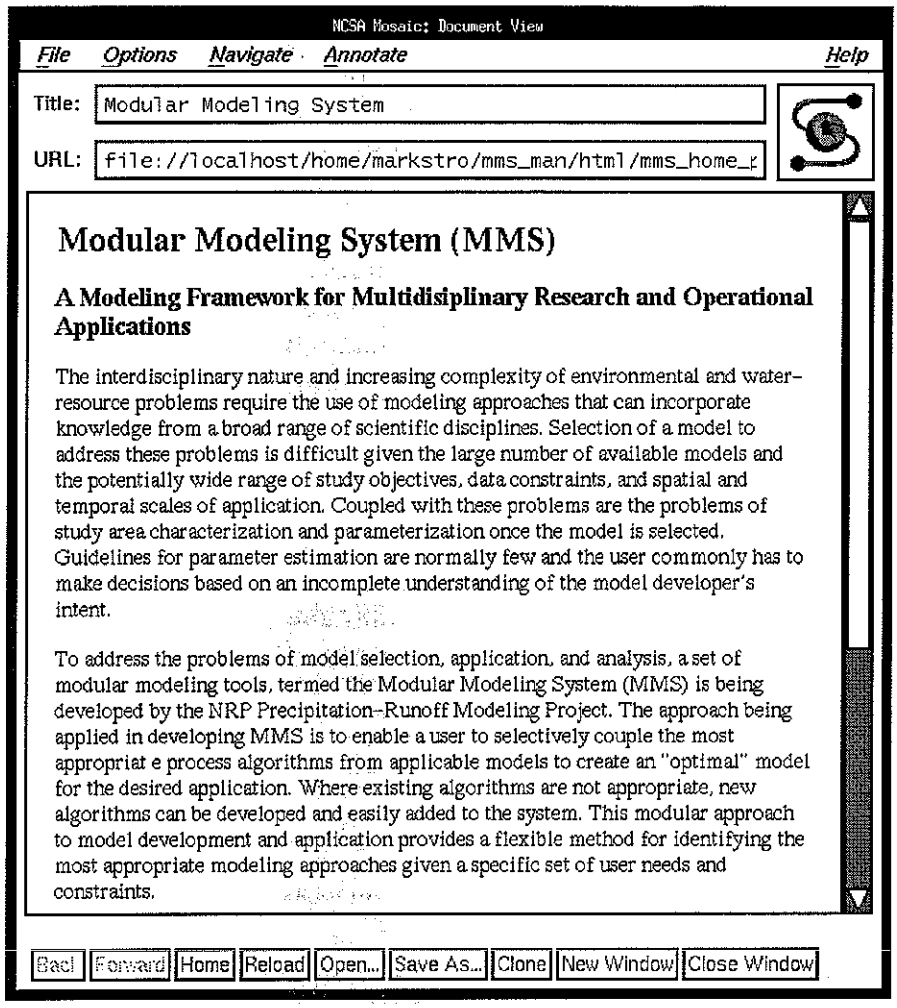

Figure 4.38 MMS system documentation in HTML viewer.

5. MODEL BUILDING: XMBUILD

xmbuild provides a GUI (fig. 5.1) to assist users in building executable models by linking together MMS modules selected from the module library. The conventions used in the *xmbuild* GUI for selecting and manipulating information are the same as those described in section 4.1 of Chapter 4, *MMS Graphical User Interface.*

5.1 Module Library

The module library is composed of modules located in a number of directories. By default, these directories are all the subdirectories in the *mms/modules* directory of the MMS master directory and all subdirectories in the *<user_workspace>/modules/src* and *<user_workspace>/modules/hier* directories of the user's workspace. *xmbuild* will consider any file with a ".c" or ".f" extension in these directories to be module source code.

The list of directories that compose the library is maintained in the file *<user_workspace>/control/workspace.path.* Additional directories, located on the user's workstation or another workstation, may be added to the library by adding their full path name to this file. Care should be taken to correctly specify the paths, especially when working on systems with auto-mounting file systems.

It is also possible to remove module directories from the modular library by editing the *<user_workspace>/control/workspace.path* file and removing the desired directory paths from the list. However, removing directories may cause problems if the user attempts to rebuild an existing model that uses a deleted directory.

5.2 Representations of Models Within xmbuild

There are three methods that *xmbuild* uses to store models. These are: (1) schematic files, (2) HTML documentation files, and (3) executable files. A single model will have all representations once it has been built. These files are stored in the *<user_workspace>/models* directory.

5.2.1 Schematic Model File

As the user selects the desired modules from the module libraries, a graphical representation of the model and the links between the modules appears on the *Current Model* field of the *xmbuild* GUI (fig. 5.1). The schematic file is used to store this graphical representation of the model. This file may be saved and reloaded later. Schematic model files are distinguishable by their ".schem" extension. A schematic file may be loaded, edited, renamed, and then saved to create a variation of the original model. Once this graphic corresponds to the desired model, the source code of the associated modules is then compiled into an executable version.

5.2.2 HTML Model Documentation File

When the user builds a model, an HTML documentation file is written. The file contains information about the compilation process, links to the module documentation, and any text associated with the model. HTML model files are distinguishable by their ".html" extension. This file is displayed in a web browser any time the user requests information about this model.

5.2.3 Executable Model File

After the executable model has been built, it may be run by the user. This file has the same name as the schematic file, but it has no extension on the file name. *xmbuild* creates this by compiling the selected modules and linking them with the MMS libraries. These libraries provide the graphical user interface and utilities which are common to all models built in MMS.

5.3 Main xmbuild Interface Window

Executing *xmbuild* produces the main xmbuild window (fig. 5.1). This window is divided into three fields; the *Module Locations*, *Available Modules*, and *Current Model* fields.

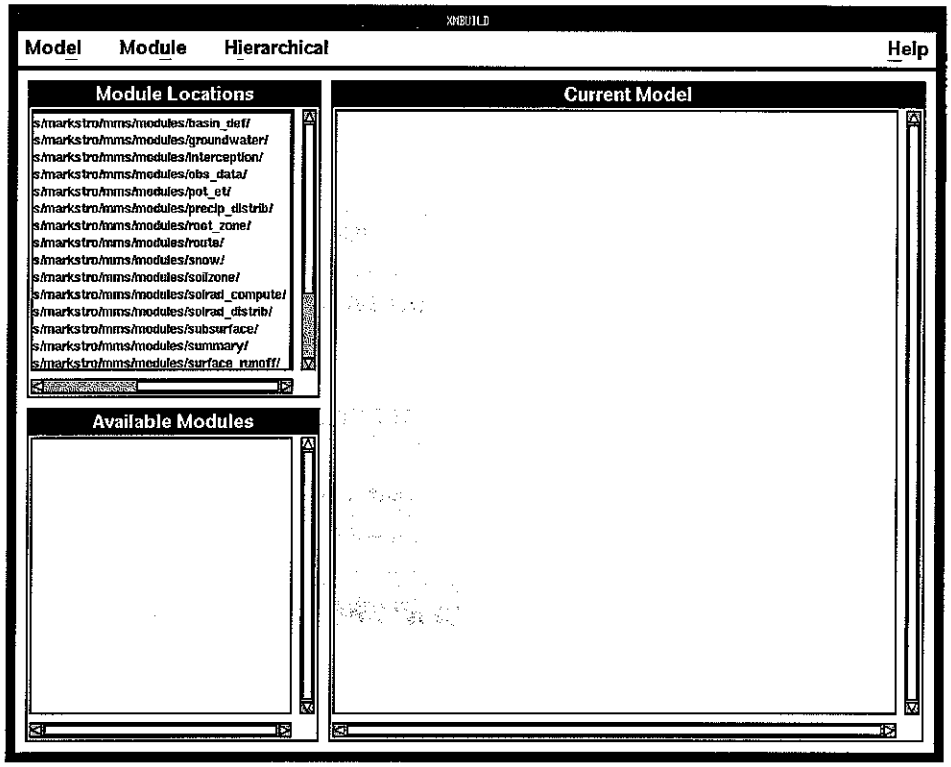

Figure 5.1 xmbuild Graphical User Interface window.

5.3.1 Module Locations Field

The *Module Locations* field is in the upper left of the main window and lists the directories comprising the module library. Each directory typically corresponds to a class of module available for simulation. For example, one module directory may be "evapotranspiration." All evapotranspiration modules can be located in this directory. Users are then able to select the evapotranspiration module best suited to their needs. To see the modules located in a directory, select the directory name from the list by double clicking on the name with the ***left*** mouse button. The available modules will then appear in the *Available Modules* field.

5.3.2 Available Modules Field

The *Available Modules* field is located in the lower left corner of the main window and contains icons which represent the modules available from the selected module directory. To select a particular module, move the cursor on top of the desired module icon and click

on it with the *left* mouse button. The module icon will then appear on the *Current Model* field. *xmbuild* will not allow selection of a specific module more than once.

5.3.3 Current Model Field

This field displays the status of the model being constructed. Specifically, the window shows the selected module icons and the data connections between them. The input variables, (or input "slots") in a receiving module need to be connected to the output variables, (or output "slots") in another module. Connections between input and output slots are performed automatically by *xmbuild* when a new module is added to *Current Model* field. Red lines indicate the unidirectional flow of information from one module to another module, yellow lines indicate bidirectional data flow between two modules.

If there is more than one possible link, the first one found will be made. However, it is possible to manually override any of these automatic links through standard model editing functions. If the default link is not the link desired, it may be modified using one of three methods.

One method is to double click the *left* mouse button on a module icon. This opens a window displaying the module *Input and Output Slots* (fig. 5.2). Clicking on one of the *Input Slot* buttons with the *left* mouse button opens a *Links* window displaying a list of modules that provide the selected variable as an output. The names of modules which are not on the *Current Model* field but are in the modular library appear with an asterisk in front of their names. Clicking with the *left* mouse button on a module name that has no asterisk assigns the link for the selected variable to the selected module. If the desired module is not on the *Current Model* field, then the user must first select it by choosing the appropriate module directory in the *Module Locations* field and then clicking on the appropriate icon in the *Available Modules* field.

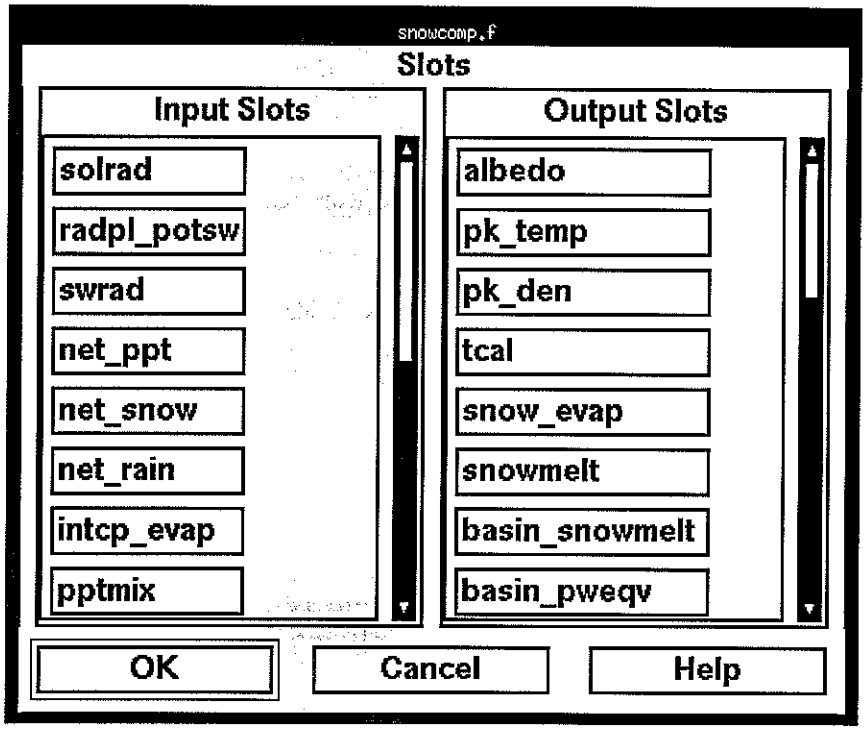

Figure 5.2 Slot window.

A second method of modifying the links between two modules is to place the cursor on a module, click and hold the ***right*** mouse button, move the cursor to the other module and release the mouse button. This will open a window displaying the possible links between these two modules. Selecting the desired variables to link is done by clicking on each one with the ***left*** mouse button. After all desired variables are selected, pressing the *OK* button will make the selected links.

The third method is accomplished by clicking on a specific link between modules with the ***left*** mouse button. When a link is selected, a window opens displaying all the possible variable connections between the two modules. The variable names that are highlighted are currently linked. This list can be modified to change the status of selected variables by clicking on the variable name using the ***left*** mouse button. Pressing the *OK* button will link only those variables whose names have been highlighted. Highlighted variables that were previously linked to other modules will have their links changed to the selected module pair.

Module icons can be moved to any user-desired location on the *Current Model* field. To move an icon, place the cursor on top of the icon, push and hold the ***left*** mouse button,

drag the icon to a new position, and release the mouse button. All module connections will be redisplayed at the new location.

The *Current Model* field also shows the status of model completion. Modules represented with red icon buttons require additional links to input variables, while module icons that are displayed in green have all inputs satisfied. When all icons are green the displayed schematic is ready for saving and the model is ready to be built. The *save* and *build* options are discussed in detail in the *Model Menu* section below.

5.3.4 Menu Bar

Across the top of the main interface window (fig. 5.1) is a menu bar that provides a number of pull-down menu options to load, build, and save models and select and execute various system options. The menu bar options are:

- *Model*
- *Module*
- *Hierarchical*

These options are discussed in detail in the next sections.

5.4 Model Menu

The functions available in the *Model* menu allow the user to load, build, and save models and to exit MMS. All options under the *Model* menu are selected using the *left* mouse button unless otherwise noted in the text description of the function. The file pull-down menu contains the following options.

5.4.1 Load

Clicking on this option produces a *Load a File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *<user_workspace>/models* and the file names are filtered to display only those ending with an extension of ".schem". These are the schematic figures that were created and stored during a previous execution of *xmbuild.* A schematic file can be selected from this directory or another directory of the user's choice. A schematic file can be selected by double clicking on the name in the *Files* list or by entering the full path name in the *Selection* window and clicking on the *OK* button. Selecting a schematic file will load it into the *Current Model* field of the main window (fig. 5.1).

5.4.2 Save

Clicking on this option produces a *Save a File* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is

<user_workspace>/models and the file names are filtered to display only those ending with an extension of ".schem". Double clicking on a name in the *Files* list of this window will save the model schematic currently displayed in the *Current Model* field of the main window (fig. 5.1) using the selected file name. A schematic file can also be saved using another name in this or in another directory of the user's choice by entering the full path name in the *Selection* window and clicking on the *OK* button.

5.4.3 Build

This option is used to build the model displayed in the *Current Model* field of the main window. Clicking on this option opens a *XMBUILD Model Information* window (fig. 5.3). The desired name for this model is entered in the *Model Name* field. When compilation is complete, the name entered will have the prefix "x" added to it so that a model named "mymodel' would become "xmymodel" and would be executed by using this name.

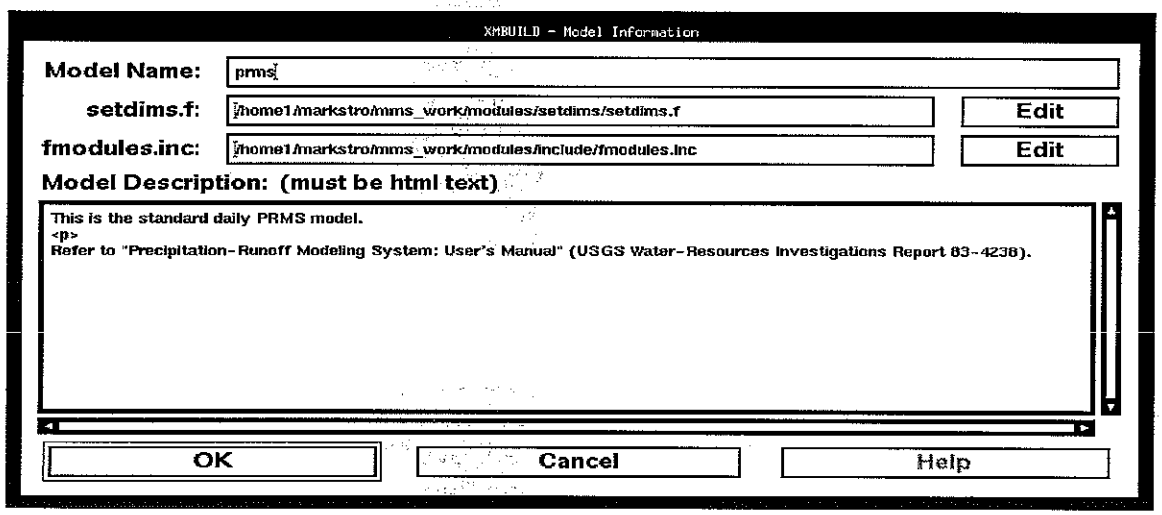

Figure 5.3 XMBUILD - Model Information window.

The *setdims.f* and *fmodules.inc* fields are system specific files that are used with FORTRAN coded modules. MMS requires that all dimension names be declared in a special-purpose module *setdims.f* that is located in the *<user_workspace>/modules/setdims* directory. Its primary function is to declare the dimension names and bind them to the sizes of the array declarations. The *setdims.f* module is discussed in more detail in Chapter 6.

The maximum sizes of the dimensions are specified in an include file which is *fmodules.inc* for FORTRAN or *cmodules.h* for C. This file should be referenced in all module

source files to insure a common declaration for all arrays. The maximum size declarations are required for FORTRAN modules because dynamic storage allocation is not available in FORTRAN 77. The *fmodules.inc* file is also discussed in more detail in Chapter 6.

The paths to the *setdims.f* and *fmodules.inc* files are entered in their respective entry fields in the window. Selecting the *Edit* button at the end of each entry field will open the file in the given path and present it in the default system editor.

Selecting the *OK* button in the *XMBUILD Model Information* window initiates the compilation and linking stage of model building. If the name selected for this model already exists, a dialog window will open to inquire if the user wants to *overwrite* the existing file or *rename* this model. Selecting the *rename* option returns the user to the *XMBUILD Model Information* window where a new name can be entered.

With the correct name selected, the system begins to link the modules in an order of operation that is dependent on the input and output variable dependencies of the selected modules. If a situation occurs where there is no clear dependency order the system will open a *Loop Resolver* window (fig. 5.4) which presents the names of the two modules whose order of operation cannot be determined with the current algorithms. The module that should run first is identified by clicking on the module name and then *OK* using the ***left*** mouse button. The system may require user intervention on more than one pair of modules and a *Loop Resolver* window will open for each decision pair.

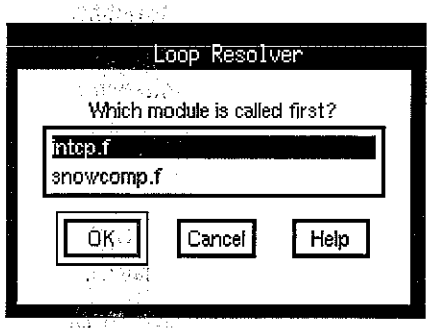

Figure 5.4 Loop Resolver window.

After any *Loop Resolver* intervention and prior to the actual compilation a *Check Module Order* window (fig. 5.5) will open and present the list of modules comprising the selected model in the order of computation that has been determined. At this point the user can accept the module order by pressing the *Done* button or the order can be adjusted. The instruction at the top of the window says "Select the module to move." Select the desired module by clicking on it with the ***left*** mouse button which will highlight

the name. The instruction then changes to "Select the insertion point." The insertion point is the nth point in the current series. For example, if one desires to move the first module to become the fifth module, select the first module and then select the fifth module in the current series as the insertion point. This will move the first module to position five and the modules that were previously two through five will move up to one through four.

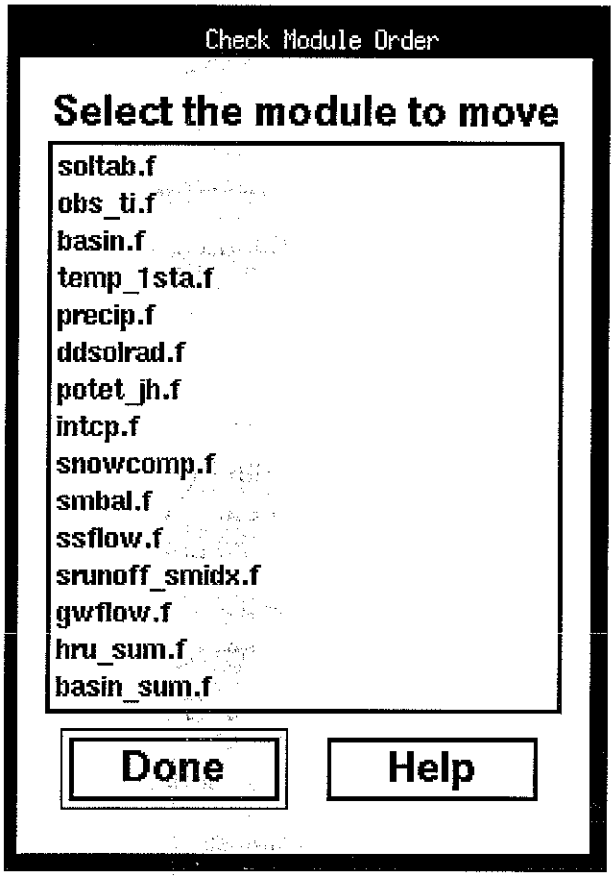

Figure 5.5 Check Module Order window.

When the modules are in the desired order, pressing the *Done* button initiates the model building process. When compilation and linking are complete, a window with the message "Compilation Complete" is displayed. This window is removed by pressing the *Cancel* button.

5.4.4 Exit

Clicking on the *Exit* option of the *Model* menu (fig. 5.1) will terminate *xmbuild*. A window will appear to verify the *Exit* command.

5.5 Module Menu

The functions available in the *Module* menu allow the user to remove one or more modules from the *Current Model* field of the main window (fig. 5.1). All options under the *Module* menu are selected using the ***left*** mouse button unless otherwise noted in the text description of the function. The file pull-down menu contains the following options.

- *Clear*
- *Remove Module*

5.5.1 Clear

This option will clear the *Current Model* field. It will remove all modules and connections from the screen. It cannot be undone. A window will appear to verify the *Clear* command.

5.5.2 Remove Module

This option will remove user-selected modules from the *Current Model* window. It will also delete the connections to the removed modules. It cannot be undone. To select a module, place the cursor on the module and click the ***left*** mouse button. This module will then be shaded in either a highlighted green or red color to indicate it has been selected. To select more than one module, hold the ***shift*** key down when clicking the ***left*** mouse button on additional modules to be selected. To un-select all modules, click on a blank canvas area of the *Current Model* field with the ***right*** mouse button.

5.6 Hierarchical Menu

The functions available in the *Hierarchical* menu allow the user to group two or more modules to form an aggregate or higher-level module in the *Current Model* field of the main window (fig. 5.1) and to disaggregate these groups. These groups of modules are called "Hierarchical Modules." In contrast with a "model," a hierarchical module does not have all its input slots fully connected. In a hierarchical module, the non-connected input slots of any of its component modules become the input slots of the hierarchical module. All the output slots from the component modules become output slots of the hierarchical module. Hierarchical modules can be used in the construction of more complex models.

5.6.1 Save Hierarchical

To make a hierarchical module, select all the modules that will comprise the single hierarchical module. To select more than one module, press and hold the ***Shift*** key while clicking on the module with the *left* mouse button. Once all the modules are selected, choose the *Save Hierarchical* option.

Clicking on this option produces a *Make Hier* window that is presented as a standard Motif file selection dialog (fig. 4.1). The default directory that appears in the dialog is *<user_workspace>/modules/hier*. Double clicking on a name in the *Files* list of this window will save the hierarchical module using the selected file name. A hierarchical module can also be saved using another name in this or in another directory of the user's choice by entering the full path name in the *Selection* window and clicking on the *OK* button.

5.6.2 Expand Hierarchical

This option enables a user to expand a hierarchical module into its component modules. To use this option select the hierarchical module by clicking on the module with the *left* mouse button. Then select the *Expand Hierarchical* option from the menu.

5.7 Building the PRMS Example

An example application of *xmbuild* can be run to build an executable version of the daily mode components of the USGS Precipitation Runoff Modeling System (PRMS) (Leavesley and others, 1983). The modules for this model are located in the Master Directory region of MMS. To build this model, the user must have created a user workspace as described in Chapter 2. Once the user workspace is created, change directory to the *<user_workspace>/control* directory and execute the command *source setmms*.

Next execute the command *xmbuild* (*xmbuild* is located in the *mms/bin* directory). Select *Load* from the *File* menu. A file selector window (fig. 5.6) will appear. Select the file *xprms.schem* from the right hand list and press the *OK* button. A schematic of the PRMS model will be loaded in the *Current Model* window (fig. 5.7). Next, select *Build* from the *File* menu. The *Model Information* window (fig. 5.3) will appear. It will have default values filled in for the *Model Name*, the *setdims.f* file, and the *fmodules.inc* file.

Use the default values by selecting the *OK* button. If *xmbuild* is unable to determine the calling order of the modules (as with the *PRMS* model), it will display the *Loop Resolver* window (fig. 5.6). Select the module which should be called first in the run loop. In the case of PRMS, *intcp.f* comes before *snowcomp.f*, and *srunoff_smidx.f* comes before *smbal.f*. A window will open to display the module sequence and the user can change the order at this point or accept the order as shown. When the module order has been resolved, *xmbuild* will write the makefile necessary to compile the modules into a model and start

compiling. This makefile is named *Makefile* and is written to the *<user_workspace>/make* directory. Upon completion, a window will open informing the user that the compilation process is finished. The name of the executable model is *xprms* and it is located in the *<user_workspace>/models* directory.

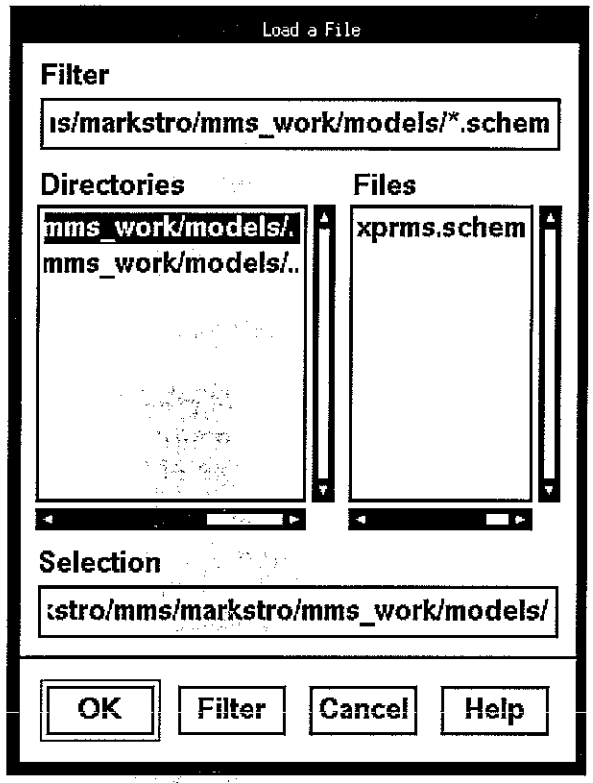

Figure 5.6 Selecting the PRMS Model.

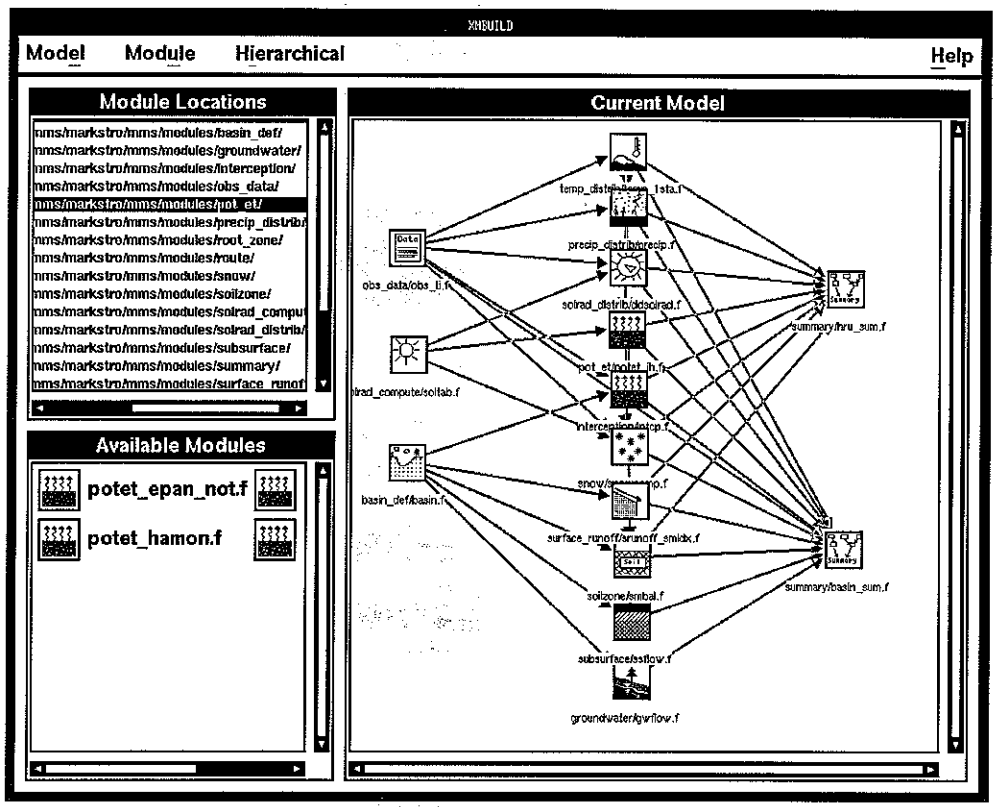

Figure 5.7 PRMS Model in the Current Model window.

6. MODULE DEVELOPMENT

6.1 Introduction

A module is the basic building block of MMS. To insure that modules interface properly in MMS, a module structure and a few associated coding rules have been defined. Each module is self-contained, both conceptually (it may call no other module directly) and physically (it must be contained in a single file). Communication between modules is accomplished by using the MMS internal databases and a library of MMS standard functions that can be included in module source code. Modules can be written in either FORTRAN or C.

This sections describes the structure of a module and the coding conventions and system functions used in writing a module. Discussions include details of internal module structure, FORTRAN and C module examples, and a description of the MMS programming functions. The information in this section is provided for users who wish to write their own modules. While not necessary for other users, a review of the material in this chapter may provide additional insight to system operations and capabilities.

6.2 Module Structure

A module is composed of up to five basic submodules that are written as functions. These functions are *main, declare, initialize, run*, and *cleanup*. Each of the five functions must return an integer error code. A return value of zero indicates no errors were detected. The relations among these functions are shown in figure 6.1 and discussed in detail below.

6.2.1 Main Function

The *main* function is the interface from the MMS run controller to the module and simply directs all system calls to the appropriate module function. All calls from MMS to a module go to the *main* function with a single string argument of "*declare*", "*initialize*", "*run*", or "*cleanup*". *Main* checks for these strings and then calls the appropriate function. All modules must have a *main* function.

6.2.2 Declare Function

The *declare* function sets up the internal parameter and public variable databases. All parameters and public variables must be declared to MMS in the *declare* function. The MMS library functions *declparam* and *declvar*, discussed below, are used to declare these parameters and variables respectively. The *declare* function is called when MMS is started and is called only once during an MMS session. All modules must have a *declare* function.

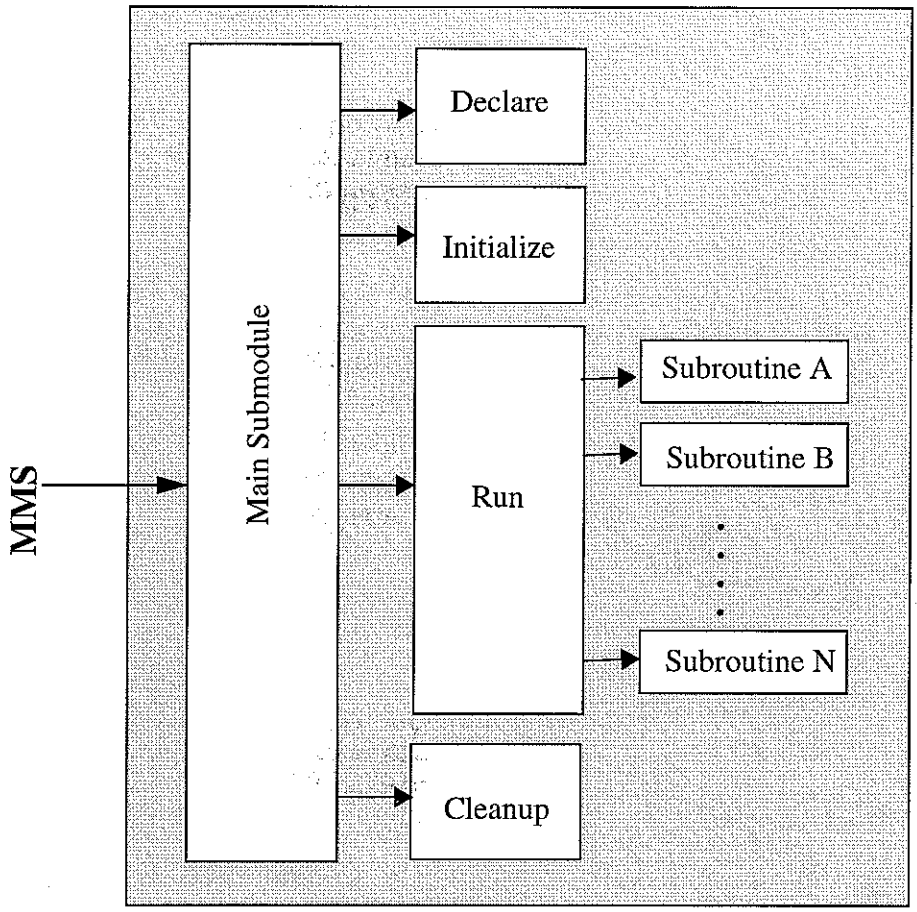

Figure 6.1. Relations among MMS module functions.

6.2.3 Initialize Function

The *initialize* function is used to initialize user-specified variables at run time and to read the current parameter values from the parameter database. Because the user may edit the parameter database between runs, the *initialize* function is used to update the parameter values in the module before the *run* function is called. C modules may also use this function to allocate memory space for variable arrays. The *initialize* function is executed only once at the beginning of each model run and before the simulation starts. The *initialize* function may not be required in all modules.

6.2.4 Run Function

The *run* function contains the algorithms used to simulate the process(es) represented by the module and it is called at every timestep in the simulation. The *run* function may be composed of one or more functions for C code or subroutines for FORTRAN code (for example, fig. 6.1). Additional functions or subroutines are typically used to

further modularize the code within a module. The *run* function may not be required in all modules. Some modules may be used only to initialize selected physical system characteristics.

6.2.5 Cleanup Function

The *cleanup* function is issued at the conclusion of a simulation run, typically to free dynamically allocated storage. C modules may use *cleanup* to free allocated memory, while FORTRAN modules may not need a *cleanup* submodule.

6.3 Converting Existing Code Into MMS Modules

MMS modules may be written as new code or may be created from existing programs. There are several issues which determine the ease with which existing programs can be converted to MMS modules. One issue is the structure of the original code. If the existing model is written in a modular structure with subroutines performing well defined tasks, the job will be much easier. A second issue is use of global variables. Heavy use of global variables and common blocks complicates the conversion procedure because in MMS all parameters, variables, and data must be passed between modules using the MMS library functions. A third issue is the degree to which control and simulation are intertwined in the model. Module conversion will be much more difficult if time loops, space loops, and other conditions are interspersed within the simulation algorithms.

In general, programs that are well structured using functions or subroutines to break the code into component processes or functionality are typically easier to convert to modules. Also, code that uses few global variables is also easier to port than code that makes extensive use of global variables.

The "modularizing" of an existing model should begin with a breakdown of the processes simulated by the model. A clear picture of what these processes are and how they affect each other should be ascertained before any recoding begins. Once the process algorithms have been identified, the corresponding pieces of source code should be isolated. All references to global variables should be removed, resulting in subroutines with well defined input and output parameters variables. The resulting functions or subroutines can then be used as the basis for the *run* function in each module. Once the *run* function is coded, the *declare, initialize, main*, and *cleanup* functions can be added.

A number of MMS library functions are provided for use in module code development to facilitate access to system capabilities and communication among modules. The library functions can be grouped into the three general categories. These are (1) database access, (2) system information, and (3) file I/O. Database access functions allow the system to build the internal databases and the modules to access these databases and to transfer information between modules. System information functions provide the modules with MMS status information. File I/O functions are for access to input and

output system tools. These functions are summarized in figure 6.2 and are described in

Type	Function	
Database Access Functions	**decldim**	declare resizable array dimension
	declfix	declare fixed size array dimension
	declparam	declare parameter
	declvar	declare public varaible
	dimstr	get a dimension as a text string
	getdatainfo	get data file description string
	getdim	get a dimension size as an integer
	getdimdesc	get a dimensions description text
	getdimname	get the name of a dimension index text
	getoutname	get full path of output file as a string
	getparam	get a copy of a parameter array
	getvar	get a copy of a public variable array
	putvar	update a public variable array
	readvar	update a public variable array from data file
	unitparam	get parameter unit string
	unitvar	get public variable unit string
System Information Functions	**dattim**	get the date and time information
	deltim	get the delta time for the current timestep
	djulian	get julian day with fractional part
	getstep	get the current timestep count
	julian	get the julian day
	units	get a unit conversion factor
File I/O Functions	**cpdble**	print a double precision array to standard output
	cpint4	print an integer array to standard output
	cpreal	print a real array to standard output
	cpstr	print a string to standard output
	dpdble	print a debug double precision array
	dpfloat	print a debug float array
	dpint4	print a debug integer array
	dplong	print a debug long integer array
	dpstr	print a debug string
	dpreal	print a debug real array
	opdble	print a double precision array to the output file
	opfloat	print a float array to the output file
	opint4	print an integer array to the output file
	oplong	print a long integer array to the output file
	opstr	print a string to the output file
	opreal	print a real array to the output file
	updble	print a double precision array to a file
	upfloat	print a float array to a file
	upint4	print an integer array to a file
	uplong	print a long integer array to a file
	upstr	print a string to a file
	upreal	print a real array to a file

Figure 6.2. MMS library functions.

more detail in section 6.8.

The current time-space structure supported by MMS can be visualized as nested loops. The outer loop is time, the next loop is modules, and the inner loop is space. The length of a timestep in the time loop is computed from the input data. For each timestep, one loop is made through all the modules, executing the *run* function. Each module then loops through the applicable spaces. Thus, the space loop is inside the module.

The following FORTRAN and C modules are presented as examples of the structure and coding conventions used in the development of a module.

6.4 FORTRAN Module Example

The module in this example uses a simplified algorithm to compute an evapotranspiration loss value and then subtract this loss value from measured rainfall at all rainfall gauges to obtain an estimate of a public variable called *netprecip*.

6.4.1 Main Function

The *main* function of this module is named *etrans* and is declared as an integer*4 type function. Following the function name is an include statement for the file *fmodules.inc*. This file contains the declarations of the maximum sizes for all array dimensions used in the model. The dimension size MAXRAIN is located in this file. See section 6.6.2 for a detailed discussion of *fmodules.inc*.

The next code segment contains the standard FORTRAN declarations for arrays or variables. These will be used as storage for public variables or as local copies of dimensions or parameters values obtained from the databases. Those variable and parameter values that must always occupy a fixed area in memory, and therefore hold their values between successive calls to the module, must be declared as `save` in FORTRAN code.

The variable `retval` is the return variable and its value is passed back to the calling MMS function. `retval` is initially set equal to zero for a no error status.

The string argument `arg`, passed to *etrans*, is tested to see if its value is "declare", "initialize", or "run". The appropriate function is then called. A return a value of zero from the function indicates there was no error. If the function terminates abnormally, a nonzero error code is returned. At the end of *main*, the value of the function *etrans* is set equal to `retval` and returned to the run controller in MMS. If the returned value is nonzero, an error condition is indicated and the MMS session will be terminated.

```
c***********************************************************
c
c main etrans routine
c***********************************************************

      integer*4 function etrans(arg)

      include 'fmodules.inc'
```

```
character*(*) arg
integer*4 retval, etrun, etinit, etdecl
integer*4 nrain, ntemp
real loss(MAXRAIN), netprecip(MAXRAIN)
real tzero, tall

save loss, netprecip, tzero, tall, nrain, ntemp

retval = 0

if (arg.eq.'declare') then
   retval = etdecl(loss, netprecip)

else if (arg.eq.'initialize') then
   retval = etinit(tzero, tall, nrain, ntemp)

else if (arg.eq.'run') then
   retval = etrun(loss, netprecip, tzero, tall, nrain, ntemp)

end if

etrans = retval

return
end
```

6.4.2 Declare Function

The function *etdecl* is an integer*4 type function and is called from *etrans*, the module's *main* function. The addresses of the module's declared variables are passed into *etdecl* via the argument list of the function call. As in *main*, the file *fmodules.inc* must also be included and all variables are declared by type. The value of *etdecl* is set to "1", indicating an abnormal termination if it is returned to main. If there are no problems, the value of *etdecl* is reset to "0" just prior to executing the return statement for this function.

The library function *declvar* is used to declare all the public variables and the library function *declparam* is used to declare all the parameters used in the module. When using *declvar*, the address of the public variable is registered with the internal variables database. However, with *declparam*, the parameter database allocates memory for the values and keeps track of these memory locations. Thus, no reference to parameter addresses are made.

Both functions are applied in an "if statement" format. A nonzero return value indicates an error condition and will cause the return statement to be executed. Since the value of *etdecl* is still "1" this will cause the MMS session to terminate. The system functions *declparam* and *declvar* are discussed in detail in sections 6.8.3 and 6.8.4 respectively.

```
c ******************************************************
c etdecl - declare  parameters and public variables for
c          the etrans module
```

```
c*******************************************************
      integer*4 function etdecl(loss, netprecip)

      include 'fmodules.inc'

      real loss(MAXRAIN)
      real netprecip(MAXRAIN)
      integer*4 retval

      etdecl = 1

      if(declvar('etrans','loss', 'nrain', MAXRAIN,
     + 'real', 'inches', loss).ne.0) return

      if(declvar('etrans','netprecip', 'nrain', MAXRAIN,
     + 'real', 'inches', netprecip).ne.0) return

      if(declparam('etrans', 'tzero', 'one', 'real', '10.0',
     +'-10.0', '20.0', 'Temp for zero ET',
     +'The temperature below which the evapotranspiration is ' //
     +'assumed to be zero.', 'degrees').ne.0) return

      if(retval.ne.0) return

      if(declparam('etrans', 'tall', 'one', 'real', '120.0',
     +'90.0', '140.0', 'Temp for complete ET',
     +'The temperature above which the evapotranspiration is ' //
     +'assumed to be equal to precipitation.', 'degrees')
     +.ne.0) return

      etdecl = 0

      return
      end
```

6.4.3 Initialize Function

The function *etinit* is an integer*4 type function and is called from the *main* function *etrans*. The addresses of the variables and parameters being initialized by this function are passed in the argument list of the call. Variables and parameters initialized by this function must be declared in the *main* function using the `save` statement.

As in *main*, the file *fmodules.inc* must also be included and all variables and parameters are declared by type. The value of *etinit* is set to "1", indicating an abnormal termination if it is returned to *main*. If there are no problems, the value of *etinit* is reset to "0" just prior to executing the return statement for this function.

The MMS library function *getparam* is used to get the value of the parameters `tzero` and `tall`. Note that the *getparam* function makes a copy of the values in the parameter database and stores the copy in the module's local memory.

The MMS library function *getdim* is used to obtain the size of the dimensions `nrain` and `ntemp`. Since parameter and dimension values will not change during the run, the queries to the database need to be made only once here in the *initialize* function.

```
c ********************************************************
c etinit - initializes the etrans module
c
c********************************************************

      integer*4 function etinit(tzero, tall, nrain, ntemp)

      include 'fmodules.inc'

      real tzero, tall
      integer*4 nrain, ntemp

      etinit = 1

      if(getparam('etrans', 'tzero', 1, 'real', tzero).ne.0) return
      if(getparam('etrans', 'tall', 1, 'real', tall).ne.0) return

      nrain = getdim('nrain')
      if(nrain.eq.-1) return

      ntemp = getdim('ntemp')
      if(ntemp.eq.-1) return

      etinit = 0

      return
      end
```

6.4.4 Run Function

The function *etrun* is an integer*4 type and is called from the module's *main* function *etrans*. The addresses of the variables and parameters used in this function are passed in the argument list of the call. As in *main*, the file *fmodules.inc* must also be included and all variables and parameters used are declared by type. The *run* function may also declare its own local variables.

The value of *etdecl* is set to "1", indicating an abnormal termination if it is returned to *main*. If there are no problems, the value of *etdecl* is reset to "0" just prior to executing the return statement for this function.

The MMS library function *getvar* is used to obtain the current values of the public variables `rainfall, tminf`, and `tmaxf` which were declared and assigned values by another module. Never use *getvar* to reference a public variable declared in the current module.The reference to the memory occupied by that variable is already available to the module.

If one or more of the variables needed were in the input data file then the *readvar* library function could be used to read the variables from the time series data file buffer and put them in the public variable database. In this example all the input variables were read in a separate observations module. This approach is typically used when a variety of pre-processing steps are needed to adjust or distribute the raw input data for use by more than one module. These pre-processing steps can be done in a single module or in separate modules for each input variable.

The next code segment computes the current timestep values for the variables `loss` and `netprecip` which are dimensioned by `nrain`. Because these variables are declared in this module, they are available to all other modules using the *getvar* library function.

In the last code segment, the *dpreal* library functions print the values of `tmean`, `rainfall`, `loss`, and `netprecip` to the system output file when the debug level is set to a value of 2. The debug level is set in the command line with the *-debug* option.

```
c ************************************************************
c
c etrun -  etrans run module
c
c************************************************************

      integer*4 function etrun(loss, netprecip, tzero, tall,
     + nrain, ntemp)

      include 'fmodules.inc'

      real loss(MAXRAIN), netprecip(MAXRAIN)
      real tzero, tall

      real rainfall(MAXRAIN)
      real tminf(MAXTEMP), tmaxf(MAXTEMP)
      real tmean, ttot, stationtmean, lossfraction
      integer*4 nrain, ntemp, i

      etrun = 1
c
c read in the rain and temp arrays
c

      if(getvar('obs','rainfall', MAXRAIN,
     + 'real', rainfall).ne.0) return

      if(getvar('obs','tminf', MAXTEMP, 'real', tminf).ne.0) return
      if(getvar('obs','tmaxf', MAXTEMP, 'real', tmaxf).ne.0) return

c
c compute the mean temps
c
      ttot = 0.0
```

```
      do 100 i = 1, ntemp
        stationtmean = 0.5 * (tmaxf(i) + tminf(i))
        ttot = ttot + stationtmean
  100 continue

      tmean = ttot / ntemp
c
c estimate the evapotranspiration loss fraction, assuming no loss
c at tzero, total loss at tall
c
      lossfraction = (tmean - tzero) / (tall - tzero)
      if (lossfraction.gt.1.0) lossfraction = 1.0
      if (lossfraction.lt.0.0) lossfraction = 0.0
c
c compute the loss and netprecip
c
      do 200 i = 1, nrain
        loss(i) = rainfall(i) * lossfraction
        netprecip(i) = rainfall(i) - loss(i)
  200 continue

c
c print av temp, loss and net precip
c
      call dpreal('Mean temp :', tmean, 1,2)
      call dpreal('Total precip :', rainfall, nrain, 2)
      call dpreal('Loss :', loss, nrain, 2)
      call dpreal('Net precip :', netprecip, nrain, 2)

      etrun = 0

      return
      end
```

6.4.5 Cleanup Function

There is no cleanup function needed for this module because there was no dynamically allocated storage.

6.5 C Module Example

This is the C version of the same evapotranspiration module used in the FORTRAN example.

6.5.1 Main Function

The *main* function of this module is named *etrans_c* and is declared as a long type. Preceding the function name is an include statement for the file *cmodules.h*. This header file contains the declarations of the maximum sizes for all array dimensions used in the model that are modifiable prior to a model run and are also used in one or more

FORTRAN modules in the model. Those dimensions that are modifiable but are used only in C modules for arrays that are dynamically allocated do not have a maximum size limit constraint. The dimension size MAXRAIN is located in this file.

The next code segment contains the standard C declarations for arrays or variables. These will be used as storage for public variables or as local copies of dimensions or parameters values obtained from the databases. Those variable and parameter values that must always occupy a fixed area in memory, and therefore hold their values between successive calls to the module, must be declared as `static` in C code.

The variable `retval` is the return variable and its value is passed back to the calling MMS function. `retval` is initially set equal to zero for a no error status.

The string argument `arg`, passed to *etrans_c*, is tested to see if its value is "declare", "initialize", "run", or "cleanup". The appropriate function is then called. A return value of zero from the function indicates there was no error. If the function terminates abnormally, a nonzero error code is returned. At the end of *main*, the value of `retval` returned to the run controller in MMS. If the returned value is nonzero, an error condition is indicated and the MMS session will be terminated.

```
/*********************************************************
 *
 * main etrans routine
 *
*********************************************************/
#include cmodules.h

long etrans_c(arg)
  char *arg;

{
  long retval = 0;
   static float loss[MAXRAIN], netprecip[MAXRAIN];
   static float tzero, tall;
   static long nrain, ntemp;
   static float *rainfall, *tminf, *tmaxf;

   if (!strcmp(arg,"declare"))
                    retval = etdecl(loss, netprecip);

   else if (!strcmp(arg, "initialize"))
                    retval = etinit(&tzero, &tall, &nrain, &ntemp,
                              &rainfall, &tminf, &tmaxf);

   else if (!strcmp(arg, "run"))
                    retval = etrun(loss, netprecip, &tzero, &tall,
                              &nrain,&ntemp, rainfall, tminf, tmaxf);

   else if (!strcmp(arg, "cleanup"))
                    retval = etclean(rainfall, tminf, tmaxf);
```

```
   return retval;
}
```

6.5.2 Declare Function

The function *etdecl* is a long type and is called from *etrans_c*, the module's *main* function. The addresses of the module's declared variables are passed into *etdecl* via the argument list of the function call.

The library function *declvar* is used to declare all the public variables, and the library function *declparam* is used to declare all the parameters used in the module. When using *declvar*, the address of the public variable is registered with the internal variables database. However, with *declparam*, the parameter database allocates memory for the values and keeps track of these memory locations. Thus, no reference to parameter addresses are made.

Both functions are applied in an "if statement" format. A nonzero return value indicates an error condition and will cause the return statement to be executed. This returns a value of 1 to *main* which when passed back to the calling MMS control function will cause the MMS session to terminate. If there are no errors, a value of 0 is returned.

```
c ************************************************************
 * etdecl - declare parameters and public variables for the
 *          etrans module
 *
c************************************************************/

long etdecl(loss, netprecip);
 float *loss, *netprecip;

{
 if(declvar("etrans","loss", "nrain", MAXRAIN, "float", "inches", loss))
            return(1);

 if(declvar("etrans","netprecip", "nrain", MAXRAIN, "float", "inches",
            netprecip)) return(1);


 if(declparam("etrans", "tzero", "one", "float", "10.0", "-10.0",
            "20.0", "Temp for zero ET", "The temperature below
             which the evapotranspiration is assumed to be zero.",
             "degrees")) return(1);

 if(declparam("etrans", "tall", "one", "float", "120.0", "90.0",
               "140.0", "Temp for complete ET", "The temperature above
               which the evapotranspiration is assumed to be equal to
               precipitation.", "degrees" )) return(1);
```

```
 return(0);

}
```

6.5.3 Initialize Function

The function *etinit* is a long type function and is called from the *main* function *etrans_c*. The addresses of the variables and parameters being initialized by this function are passed in the argument list of the call. Variables and parameters initialized by this function must be declared in the *main* function using the `static` statement.

The MMS library function *getparam* is used to get the value of the parameters `tzero` and `tall`. Note that the *getparam* function makes a copy of the values in the parameter database and stores the copy in the module's local memory.

The MMS library function *getdim* is used to obtain the size of the dimensions `nrain` and `ntemp`. Since parameter and dimension values will not change during the run, the queries to the database need to be made only once here in the *initialize* function. The values of `nrain` and `ntemp` are then used to dynamically allocate storage for the variables `rainfall, tminf`, and `tmaxf`.

All library functions are applied in an "if statement" format. A nonzero return value indicates an error condition and will cause the associated return statement to be executed. This returns a value of 1 to *main* which when passed back to the calling MMS control function will cause the MMS session to terminate. If there are no errors, a value of 0 is returned.

```
/*********************************************************
 *
 * etinit - initializes the etrans module
 *
*********************************************************/

long etinit(tzero, tall, nrain, ntemp, rainfall, tminf, tmaxf)
 float *tzero, *tall;
 long *nrain, *ntemp;
 float **rainfall, **tminf, **tmaxf;

{

 if(getparam("etrans", "tzero", 1, "float", tzero))
 return(1);

 if(getparam("etrans", "tall", 1, "float", tall))
 return(1);
```

```
 if((*nrain = getdim("nrain")) == -1)
 return(1);

 if((*ntemp = getdim("ntemp")) == -1)
 return(1);

/*
**  Allocate the local arrays.
*/
 (*rainfall) = (float *)malloc (*nrain * sizeof (float));
 (*tminf) = (float *)malloc (*ntemp * sizeof (float));
 (*tmaxf) = (float *)malloc (*ntemp * sizeof (float));

 return(0);

}
```

6.5.4 Run Function

The function *etrun* is a long type and is called from the module's *main* function *etrans_c*. The addresses of the variables and parameters used in this function are passed in the argument list of the call. The *run* function may also declare its own local variables.

The MMS library function *getvar* is used to obtain the current values of the public variables `rainfall, tminf`, and *tmaxf* which were declared and assigned values by another module. Never use *getvar* to reference a public variable declared in the current module.The reference to the memory occupied by that variable is already available to the module.

If one or more of the variables needed were in the input data file, then the *readvar* library function could be used to read the variables from the time series data file buffer and put them in the public variable database. In this example all the input variables were read in a separate observations module. This approach is typically used when a variety of pre-processing steps are needed to adjust or distribute the raw input data for use by more that one module. These pre-processing steps can be done in a single module or in separate modules for each input variable.

The next code segment computes the current timestep values for the variables `loss` and `netprecip` which are dimensioned by `nrain`. Because these variables are declared in this module, they are available to all other modules using the *getvar* library function.

In the last code segment, the *dpfloat* library functions print the values of `tmean, rainfall, loss,` and `netprecip` to the system output file when the debug level is set to a value of 2. The debug level is set in the command line with the *-debug* option.

```
/*********************************************************
 *
 * etrun -  etrans run function
 *
*********************************************************/

long etrun(loss, netprecip, tzero, tall, nrain, ntemp, rainfall,
    tminf, tmaxf)
 float *loss, *netprecip;
 float *tzero, *tall;
 long *nrain, *ntemp;
 float *rainfall, *tminf, *tmaxf;

{
  float tmean, ttot, stationtmean, lossfraction;
 long i;

 /*
 * read in the rain and temp arrays
 */

 if(getvar("obs","rainfall", MAXRAIN, "float", rainfall))
 return(1);

 if(getvar("obs","tminf", MAXTEMP, "float", tminf))
 return(1);

 if(getvar("obs","tmaxf", MAXTEMP, "float", tmaxf))
 return(1);

 /*
 * compute the mean temps
 */

 ttot = 0.0;

 for (i = 0; i < *ntemp; i++) {
 stationtmean = 0.5 * (tminf[i] + tmaxf[i]);
 ttot = ttot + stationtmean;
 }
 tmean = ttot / (*ntemp);

 /*
 * estimate the evapotranspiration loss fraction, assuming no loss
 * at tzero, total loss at tall
 */

 lossfraction = (tmean - *tzero) / (*tall - *tzero) ;
 if (lossfraction > 1.0) lossfraction = 1.0;
 if (lossfraction < 0.0) lossfraction = 0.0;

 /*
 * compute the loss and netprecip
```

```
 */

 for (i = 0; i < *nrain; i++) {
 loss[i] = rainfall[i] * lossfraction;
 netprecip[i] = rainfall[i] - loss[i];
 }

 /*
 * debug print - av temp, loss and net precip
 */

 dpfloat("Mean temp :", &tmean, 1, 2);
 dpfloat("Total precip :", rainfall, *nrain, 2);
 dpfloat("Loss :", loss, *nrain, 2);
 dpfloat("Net Precip :", netprecip, *nrain, 2);

 return(0);

}
```

6.5.5 Cleanup Function

The function *etclean* is a long type and is called from the module's *main* function *etrans_c*. The addresses of the variables whose storage is to be freed are passed in the argument list of the call.

```
/*******************************************************
 * etclean - frees the local arrays in the etrans module
 *******************************************************/

long etclean(rainfall, tminf, tmaxf)
 float *rainfall, *tminf, *tmaxf;

{

 free (rainfall);
 free (tminf);
 free (tmaxf);

 return(0);

}
```

6.6 The Setdims Special Function and Include Files

6.6.1 Setdims

MMS requires that all dimensions be declared in a special-purpose function. This function is called *setdims* and is located in the *<user_workspace>/modules/setdims* directory. Its primary function is to declare the dimensions and bind them to the sizes of

the array declarations. This is accomplished using the *decldim* library function. An example of a FORTRAN version of *setdims* function is:

```
c**********************************************************
c setdims.f: sets the dimensions
c**********************************************************
      integer*4 function setdims()

      include 'fmodules.inc'

      setdims = 1

      if (decldim('nhru', 1, MAXHRU, 'Number of HRUs').ne.0) return
      if (decldim('nobs', 1, MAXOBS, 'Number of stations').ne.0) return
      if (decldim('ngw', 1, MAXGWR, 'Number of GWs').ne.0) return
      if (decldim('nrain', 1, MAXRAIN, 'Rain Gauges').ne.0) return
      .
      .
      .
      setdims = 0
      return
      end
```

The function *setdims* is an integer*4 type function and is called from MMS prior to calling any of the modules with the "declare" argument. The file *fmodules.inc* is included to define the maximum size of each dimension, which is one of the arguments in the *decldim* function. The value of *setdims* is set to "1", indicating an abnormal termination if it is returned to the MMS calling function. If there are no problems, the value of *setdims* is reset to "0" just prior to executing the return statement for this function.

A C version of *setdims* is shown below. The function is a long type and performs as described in the FORTRAN version above. The file *cmodules.h* is the C code equivalent of the FORTRAN *fmodules.inc*.

```
/**********************************************************
** setdims.c: sets the dimensions
**********************************************************/

#include "cmodules.h"

long setdims ()
{
      if (decldim ("nhru", 1, MAXHRU, "Number of HRUs"))
                  return (1);
      if (decldim ("ngw", 1, MAXGWR, "Number of GW reservoirs"))
                  return (1);
      if (decldim ("nssr", 1, MAXSSR, "Number of SS reservoirs"))
                  return (1);
      }
```

6.6.2 Include Files

The maximum sizes of the dimensions in the *setdims* function are specified in an include file. This is *fmodules.inc* for FORTRAN or *cmodules.h* for C. This file is referenced in all module source files to insure a common declaration for all arrays. An example of *fmodules.inc* is:

```
include 'fsystem.inc'

 integer MAXHRU
 integer MAXGWR
 integer MAXSSR

 parameter (MAXHRU = 125)
 parameter (MAXGWR = 20)
 parameter (MAXSSR = 50)
```

The corresponding C version of *cmodules.h* is:

```
#include "mms.h"

#define MAXHRU 125
#define MAXGWR 20
#define MAXSSR 50
```

The include files *fsystem.inc* in *fmodules.inc* and *mms.h* in *cmodules.h* are system files that specify the type for selected system functions. These two include files are provided to the user and are installed with the system.

6.7 Module Documentation

All modules should be documented to provide all users with the information needed to assure proper use of the module as well as providing users with a full understanding of the functions performed and the assumptions made in the module. This documentation should accompany the module in any distribution of the module.

The module documentation should be installed to make it available to users via the MMS Help function. For inclusion in MMS, the module documetation must be written in, or converted to, Hypertext Markup Language (HTML). A HTML viewer, such as Mosaic or Netscape, can be used to view all MMS help files. To be availabe to the Help funciton, the documentation must be installed in the *mms/doc/modules* directory of the MMS master directory or the *<user_workspace>/modules/doc* directory.

A module document format has been designed to standardize module documentation in MMS. An example of the documentation format is given in Attachment 2. The component parts of the module document format are:

- **Name** - Name of the module.
- **Module Process** (or Function) - The type of process or function being simulated.
- **Definition** - Brief definition of module function
- **Creation Date** - Date module was created.
- **Parameters Declared** - List of parameters declared and their definitions.
- **Variables Declared** - List of variables declared and their definitions.
- **External Variables Used** - List of variables used that are declared in other modules.
- **Description** - A text description of the module function(s), the equations used, assumptions made, suggested methods for parameter estimation, and any other information that may be useful to the user.
- **References** - List of references that were cited in the Description section.
- **Developer(s) Name and Address** - The name and address where a user can obtain additional information about the module. This may be a mail address or an e-mail address. Inclusion of telephone or fax numbers is left to the discretion of the developer(s).

6.8 MMS Function Library Reference

As noted above, MMS library functions are provided for use in module code development to facilitate access to system capabilities and communication among modules. These functions are summarized in figure 6.2. A detailed description of each library function and a source code example in both FORTRAN and C are provided below.

6.8.1 decldim - declare a dimension

```
decldim (dname, dval, dmax, ddescr)
```

`dname` - dimension name.
`dval` - array size.
`dmax` - maximum array size.
`ddescr` - a string description of the dimension.

This function allocates entries in the dimension database. The function should only be called from the special *setdim* function. The dimension name, `dname`, is the lookup key for this dimension in the dimension database. The default array size `dval` is the initial

allocation size of parameter arrays of this dimension. The constant maximum array size `dmax` is defined in the include file (*fmodules.inc* for FORTRAN or *cmodules.h* for C).

The sizes of the arrays used for storage of parameters and public variables are referenced to these declared dimensions. Users may edit the size of a dimension during run time, thus automatically updating all arrays of this dimension. The system may also change the dimension sizes when loading a new parameter file, using the dimensions specification in the new file. However, an array cannot be resized to something larger than `dmax` from within the model.

An example for a *setdims* source file for FORTRAN is:

```
if (decldim ('nhru', 1, MAXHRU, 'Number of HRUs'))
          return
```

An example for a *setdims* source file for C is:

```
if (decldim ("nhru", 1, MAXHRU, "Number of HRUs"))
          return (1);
```

Here the dimension `nhru` is declared with a maximum dimension size of `MAXHRU`.

FORTRAN Binding

```
decldim (dname, dval, dmax, ddescr)
          char*(*) dname
          integer dval
          integer dmax
          char*(*) ddescr
```

C Binding

```
decldim (dname, dval, dmax, ddescr);
          char *dname;
          long dval;
          long dmax;
          char *ddescr;
```

Returns

An error code: 0 = OK; 1 = error.

6.8.2 **declfix** - declare a fixed dimension

```
declfix (dname, dval, dmax, ddescr)
```

`dname` - dimension name.
`dval` - array size.
`dmax` - maximum array size.
`ddescr` - a string description of the dimension.

This function will declare a dimension whose size should remain fixed. It provides the exact same functionality as *decldim* except that dimensions declared with this function cannot be edited. Dimensions that should not be modified, such as number of months in the year, should be declared with *declfix*.

This function allocates entries in the dimension database. The function should only be called from the special *setdim* function. The dimension name string `dname` is used as the lookup key for the dimension in the dimension database. The default array size `dval` is the allocation size of parameter arrays of this dimension. The constant maximum array size `dmax` is defined in the module include file (*fmodules.inc* for FORTRAN or *cmodules.h* for C).

An example for a *setdims* source file for FORTRAN is:

```
if (declfix ('nmonths', 1, MAXMON, 'Number of months'))
            return
```

An example for a *setdims* source file for C is:

```
if (declfix ("nmonths", 1, MAXMON, "Number of months"))
            return (1);
```

Here the dimension `nmonths` is declared with a maximum dimension size of `MAXMON`.

FORTRAN Binding

```
declfix (dname, dval, dmax, ddescr)
            char*(*) dname
            integer dval
            integer dmax
            char*(*) ddescr
```

C Binding

```
declfix (dname, dval, dmax, ddescr);
```

```
char *dname;
long dval;
long dmax;
char *ddescr;
```

Returns

An error code: 0 = OK; 1 = error.

6.8.3 **declparam** - declare a parameter

```
declparam (module, param, dimen, type, valstr, minstr, max-
 str, desc, help, unit)
```

`module` - module name.
`param` - parameter name.
`dimen` - dimension name. The string "one" signifies that the parameter is a scalar.
`type` - parameter type. This should be 'integer', 'real' or 'double' in FORTRAN, or "long", "float" or "double" in C.
`valstr` - parameter default value(s).
`minstr` - minimum parameter value.
`maxstr` - maximum parameter value.
`desc` - A short (up to 80 characters) description for display via the interface.
`help` - A detailed (up to 1024 characters) description to help the user set the parameter.
`unit` - units in which the parameter is expressed.

The default (`valstr`), minimum (`minstr`), and maximum (`maxstr`) parameter values can be entered for scalar parameters and for multi-dimensional parameters. Two examples are:

```
"100"
"1*10.0, 2*23.1, 2*23.3"
```

In the first example, all values are initialized to 100. In the second example, the first value is initialized to 10.0, the second and third to 23.1, the fourth and fifth to 23.3, and the remaining values also to 23.3. As illustrated by this example, values may be repeated any number of times by using a "*" in the default value string. This repeat count is optional and if not specified, defaults to 1. Delimiters between value groups are spaces or commas or both. If there are too few entries to fill the array to the size indicated by the

dimension, the last entry will be repeated. If there are too many entries, the routine will print an error message and exit abnormally.

Modules must declare all parameters used within them. Multiple modules may declare the same parameter. MMS will recognize that these multiple declarations are for the same parameter and create a common entry in the parameter database. Inconsistent parameter declarations between modules will result in a warning message, but not necessarily an error.

The parameter database is built and initialized when MMS is started. *declparam* is used in the declare function of the module source code. It sets up the memory and fills in default values. These defaults may be overwritten by the values which come from the parameter input file. The values may also be modified by the user via the spreadsheet editor.

An example for FORTRAN is:

```
 if (declparam('snow', 'snowinfil_max', 'nhru', 'real',
+ '2.', '0.', '20.',
+ 'Maximum snow infiltration per day in inches',
+ 'Maximum snow infiltration per day in inches',
+ 'inches').ne.0) return
```

An example for C is:

```
if (declparam ("snow", "snowinfil_max", "nhru", "float",
          "2.0", "0.0", "20.0",
          "Maximum snow infiltration per day in inches",
          "Maximum snow infiltration per day in inches",
          "inches"))
          return (1);
```

FORTRAN Binding

```
declparam (module, param, dimen, type, valstr, minstr, max-
 str, desc, help, unit)
          char*(*) module
          char*(*) param
          char*(*) dimen
          char*(*) type
          char*(*) valstr
```

```
char*(*) minstr
char*(*) maxstr
char*(*) desc
char*(*) help
char*(*) unit
```

C Binding

```
declparam (module, param, dimen, type, valstr, minstr, max-
 str, desc, help, unit);
          char *module;
          char *param;
          char *dimen;
          char *type;
          char *varstr;
          char *minstr;
          char *maxstr;
          char *desc;
          char *help;
          char *unit;
```

Returns

An error code: 0 = OK; 1 = error.

6.8.4 **declvar** - declare a public variable

```
declvar (module, extname, dimen, maxsize, type, help, units,
 intname)
```

`module` - module name.

`extname` - external variable name used by the system.

`dimen` - dimension. The string "one" signifies that the variable is a scalar.

`maxsize` - maximum array size.

`type` - type. This should be 'integer', 'real' or 'double' in FORTRAN, or "long", "float" or "double" in C.

`help` - A detailed (up to 1024 characters) description to help the user set the parameter.

`units` - units.

`intname` - internal name used in the module source code.

This function is used to declare a varaible for inclusion in the public variables database. All variables do not need to be declared. Rather, only those FORTRAN or C variables that are to be made available to other modules and/or selected system functions, such as plotting and statistical analysis, are usually declared. *declvar* is used in the declare function of the module source code. Declaration of the same variable by multiple modules is considered an error.

The variables database is built when MMS is started. Values for declared variables are updated as the model runs. The variable database only contains the information for the current time step.

Variable values are available to any module at any time during the run, but generally only the declaring module may modify the values. The values may be overwritten by another module with the *putvar* function.

An example for FORTRAN is:

```
 if(declvar('precip', 'hru_ppt', 'nhru', MAXHRU, 'real',
+ 'Adjusted precip on each HRU',
+ 'inches',
+ hru_ppt).ne.0) return
```

An example for C is:

```
if (declvar ("precip", "hru_ppt", "hru", MAXHRU, "float"
          "Adjusted precip on each HRU",
          "inches",
          hru_ppt)) return (1);
```

FORTRAN Binding

```
declvar (module, name, dimen, maxsize, type, help, units,
 value)
          char*(*) module
          char*(*) extname
          char*(*) dimen
          integer maxsize
          char*(*) type
          char*(*) help
          char*(*) units
          char*(*) value
```

C Binding

```
declvar (module, name, dimen, maxsize, type, help, units,
 value);
          char *module;
          char *extname;
          char *dimen;
          long maxsize;
          char *type;
          char *help;
          char *units;
          char *value;
```

Returns

An error code: 0 = OK; 1 = error.

6.8.5 dimstr - return dimension size as a string

`dimstr (dimension)` C
`dimstr (dimension, dim_size)` FORTRAN

`dimension` - dimension name.
`dim_size` - variable name to receive returned string.

This function returns a string representing the size of the specified dimension. The FORTRAN and C calls are slightly different.

For FORTRAN, both the dimension specifier string and the result string are passed as arguments:

```
character*10 dim_size

call dimstr ('nhru', dim_size)
```

For C, the dimension name is passed and the string is returned:

```
char *dim_size;

dim_size = dimstr ("nhru");
```

FORTRAN Binding

```
dimstr (dimension, dim_size)
          char*(*) dimension
          char*(*) dim_size
```

C Binding

```
dimstr (dimension);
          char *dimension;
```

Returns

The size of arrays of this dimension as a character string.

6.8.6 **getdatainfo -** get data file description string

```
getdatainfo (desc)
```

`desc` - description string.

This function returns the description string from the data file. This function is typically used when modules are writing report files.

An example for FORTRAN is:

```
character *80 desc

call getdatainfo(desc)
call cpstr(desc)
```

An example for C is:

```
#define STRLEN 80
char desc[STRLEN];

getdatainfo (desc, STRLEN);
printf ("Data description is %s\n", desc);
```

FORTRAN Binding

```
getdatainfo (desc)
          char*(*) desc
```

C Binding

```
getdatainfo (desc, len);
            char *name;
            int len;
```

Returns

Error code: 0 = OK; -1 = error condition.

6.8.7 **getdim** - get dimension size

```
getdim (name)
```

`name` - dimension name.

This function returns the size of the dimension of the named array. Since array dimension sizes remain constant during a single run, this function is typically used in the *initialize* function of the module and the values are passed back to *main* for use as an argument in the *run* function call.

An example for FORTRAN is:

```
integer*4 nhru

nhru = getdim('nhru')
if(nhru.eq.-1) return
```

An example for C is:

```
long nhru;

nhru = getdim ("nhru");
if(nhru == -1) return;
```

FORTRAN Binding

```
getdim (name)
            char*(*) name
```

C Binding

```
getdim (name);
```

```
char *name;
```

Returns

Error code: 0 = OK; -1 = error condition.

6.8.8 **getdimdesc -** get a dimensions description text

getdimdesc_ (char *name, long *i, char *desc, long namelen, long desclen)

```
getdimname (name, i, desc)
```

`name` - dimension name.
`i` - "ith" index specifier.
`desc` - returned description of the "ith" index.

This function is used for dimensions that have had their numeric indices changed to alphabetic names with descriptions using the *dimension index names* option in the *edit* menu of the main MMS GUI window. This function returns the description of the "ith" index of the selected dimension.

An example for FORTRAN is:

```
integer*4 nhru, i
character*30 desc

nhru = getdim('nhru')

do 10 i = 1, nhru
  call getdimdesc ('nhru', i, desc)
  call upstr('nnode', i, '                              '//desc)
.
.
.
10 continue
```

An example for C is:

```
long nhru, i;
char desc[30];

for (i = 0; i < getdim ("nhru"); i++) {
```

```
getdimname ("nhru" , i, desc)
.
.
.
}
```

FORTRAN Binding

```
getdimname (name, i, desc)
           char*(*) name
           integer i
           char*(*) desc
```

C Binding

```
getdimname (name, i, desc);
           char *name;
           int i;
           char *desc;
```

Returns

None.

6.8.9 **getdimname** - get dimension index name

```
getdimname (name, i, index_name)
```

`name` - dimension name.
`i` - "ith" index specifier.
`index_name` - returned name of the "ith" index.

This function is used for dimensions that have had their numeric indices changed to alphabetic names using the *dimension index names* option in the *edit* menu of the main MMS GUI window. This function returns the name of the "ith" index of the selected dimension.

An example for FORTRAN is:

```
integer*4 nhru, i
```

```
character*30 index_name

nhru = getdim('nhru')

do 10 i = 1, nhru
  call getdimname ('nhru', i, index_name)
.
.
.
10 continue
```

An example for C is:

```
long nhru, i;
char index_name[30];

for (i = 0; i < getdim ("nhru"); i++) {
getdimname ("nhru" , i, index_name)
.
.
.
}
```

FORTRAN Binding

```
getdimname (name, i, index_name)
           char*(*) name
           integer i
           char*(*) index_name
```

C Binding

```
getdim (name, i, index_name);
           char *name;
           int i;
           char *index_name;
```

Returns

None.

6.8.10 getoutname - get full path of output file as a string

```
getoutname (output_path, extension)
```

`output_path` - full path to output file.
`extension` - extension to put on end of output_path.

This function is used to obtain the name of an output file. It returns the full path to the current output report file with the extension string on the end.

An example for FORTRAN is:

```
integer ret
character*135 output_path

ret = getoutname (output_path, '.hru')
open (unit=81, file=output_path)
```

An example for C is:

```
char output_path[135];
FILE *fp;

if (!getoutname(output_path, ".hru")){
  fp = open (output_path, "w");
}
```

FORTRAN Binding

```
getoutname (output_path, extension)
           char*(*) output_path
           char*(*) extension
```

C Binding

```
getoutname (output_path, extension);
           char *output_path;
           char *extension;
```

Returns

Error code: 0 = OK; 1 = error condition.

6.8.11 getparam - get a copy of a parameter

```
getparam (module, parameter, maxsize, type, val)
```

`module` - module name.
`parameter` - external parameter name.
`maxsize` - maximum array size.
`type` - parameter type. This should be 'integer', 'real' or 'double' in FORTRAN, or "long", "float" or "double" in C. This must match the type when the parameter was declared.
`val` - internal parameter name.

This function is used to obtain a copy of the values found in the parameter database for the specified parameter. The function passes a pointer to an array or to a scalar storage location in the module where the copy of the parameter values are to be stored. In both C and FORTRAN, the array name passes the pointer to the array. The values in the parameter database are then written to this local parameter storage. In C one must explicitly indicate a pointer to a scalar.

An example for FORTRAN is:

```
real snowinfil_max(MAXHRU)

if(getparam('snow', 'snowinfil_max', MAXHRU, 'real',
+ snowinfil_max).ne.0) return
```

An example for C is:

```
float *snowinfil_max;

snowinfil_max = (float *)malloc (MAXHRU * sizeof (float));
if (getparam ("snow", "snowinfil_max", MAXHRU, "float",
          snowinfil_max)) return (1);
```

FORTRAN Binding

```
getparam (module, parameter, maxsize, type, val)
          char*(*) module
          char*(*) parameter
          integer maxsize
          char*(*) type
```

```
various val(maxsize)
```

C Binding

```
getparam (module, parameter, maxsize, type, val);
          char *module;
          char *parameter;
          integer maxsize;
          char *type;
          char *val;
```

Returns

Error code: 0 = OK; 1 = error condition.

6.8.12 getvar - get a copy of a public variable

```
getvar (module, variable, maxsize, type, val)
```

`module` - module name.
`variable` - external variable name.
`maxsize` - maximum array size.
`type` - variable type. This should be 'integer', 'real' or 'double' in FORTRAN, or "long", "float" or "double" in C. This must match the type when the variable was declared.
`val` - internal variable name.

This function returns a copy of the values found in the variable database for the specified variable. The function passes the pointer to an array or to a scalar storage location in the module where the variable values are to be stored. In both C and FORTRAN, the array name passes the pointer to the array. The values in the variable database are then written to this local variable storage. In C, one must explicitly indicate a pointer to a scalar.

An example for FORTRAN is:

```
real rainfall(MAXRAIN)

if (getvar('obs','rainfall', MAXRAIN, 'real',
+ rainfall).ne.0) return
```

An example for C is:

```
float *rainfall

if (getvar("obs","rainfall", MAXRAIN, "float",
          rainfall)) return(1);
```

FORTRAN Binding

```
getvar (module, variable, maxsize, type, val)
          char*(*) module
          char*(*) variable
          integer maxsize
          char*(*) type
          various val(maxsize)
```

C Binding

```
getvar (module, variable, maxsize, type, val);
          char *module;
          char *variable;
          integer maxsize;
          char *type;
          char *val;
```

Returns

Error code: 0 = OK; 1 = error condition.

6.8.13 putvar - update a public variable from a module that did not declare it

```
putvar (module, variable, maxsize, type, val)
```

`module` - module name.
`variable` - public variable name.
`maxsize` - maximum array size.
`type` - variable type. This should be 'integer', 'real' or 'double' in FORTRAN, or "long", "float" or "double" in C. This must match the type when the variable was declared.
`val` - local variable with values to use for update.

This function enables a module to update the public variables declared by another module. The module must pass a pointer to a local copy of the new variable values. In

both C and FORTRAN, the array name (`val` in the call above) indicates the pointer to this local array. The values referenced by the variable database are overwritten with the values in this local variable storage. In C, *putvar* expects the address of scalar values, so always pass a pointer.

An example for FORTRAN is:

```
 if (putvar ('srunoff', 'infil', MAXHRU, 'float', infil)
+ ne.0) return
```

An example for C is:

```
if (putvar ("srunoff", "infil", MAXHRU, "float", infil))
           return (1);
```

FORTRAN Binding

```
putvar (module, variable, maxsize, type, val)
           char*(*) module
           char*(*) variable
           integer maxsize
           char*(*) type
           various val(maxsize)
```

C Binding

```
putvar (module, variable, maxsize, type, val);
           char *module;
           char *variable;
           integer maxsize;
           char *type;
           char *val;
```

Returns

Error code: 0 = OK; 1 = error condition.

6.8.14 readvar - loads a public variable from a data file

```
readvar (module, variable)
```

`module` - module name.

`variable` - variable name.

At the start of each time step, the system reads the next line from the input data file and stores the values read in a buffer. The *readvar* function takes values from the buffer and overwrites the variable in the public variables database. The system checks the number of values read, which is specified by the current value of the dimension of the public variable, against the number of values available in the data buffer. If the dimension value is not equal to the number specified in the data buffer, an error is indicated by a return value of 1.

In the following example the variable `runoff` is read from the data buffer for the current time step:

An example for FORTRAN is:

```
if(readvar('obs','runoff').ne.0) return
```

An example for C is:

```
if (readvar ("obs", "runoff"))
          return (1);
```

FORTRAN Binding

```
readvar (module, variable)
          char*(*) module
          char*(*) variable
```

C Binding

```
readvar (module, variable);
          char *module;
          char *variable;
```

Returns

Error code: 0 = OK; 1 = error condition.

6.8.15 unitparam - get the units of a parameter as a string

```
unitparam (parameter) C
unitparam (parameter, unit_str) FORTRAN
```

`parameter` - parameter name.
`unit_str` - variable name to receive returned string.

This function returns a string containing the units of the specified parameter name. The FORTRAN and C calls are slightly different.

For FORTRAN, both the parameter name string and the result string are passed as arguments:

```
character*80 unit_str

call unitparam ('snowinfil_max', unit_str)
```

For C, the dimension name is passed and the string is returned:

```
char *unit_str;

unit_str = unitparam ("snowinfil_max");
```

FORTRAN Binding

```
unitparam (parameter, unit_str)
          char*(*) parameter
          char*(*) unit_str
```

C Binding

```
unitparam (parameter);
          char *parameter;
```

Returns

The units of this parameter as a character string.

6.8.16 unitvar - get the units of a variable as a string

`unitvar (variable)` C
`unitvar (variable, unit_str)` FORTRAN

`variable` - variable name.
`unit_str` - variable name to receive returned string.

This function returns a string containing the units of the specified variable. The FORTRAN and C calls are slightly different.

For FORTRAN, both the variable name string and the result string are passed as arguments:

```
character*80 unit_str

call unitvar ('runoff', unit_str)
```

For the C call, the variable name is passed and the string is returned:

```
char *unit_str;

unit_str = unitvar ("runoff");
```

FORTRAN Binding

```
unitvar (variable, unit_str)
          char*(*) variable
          char*(*) unit_str
```

C Binding

```
unitvar (variable);
          char *variable;
```

Returns

The units of this variable as a character string.

6.8.17 dattim - get date and time information

```
dattim (when, time)
```

`when` - Specifies the time string. Must be either "start", "end", or "now".

`time` - An array of 6 integers. The integer representation of year, month, day, hour, minute, and second get filled in respectively.

This function returns time information. `dattim` can be called with either "start", "end", or "now" as an argument to obtain information on the time the simulation started, the time it will end, or the current time step. The function writes six integers to an array whose

values correspond to the year, month, day, hour, minute, and second for the requested time information.

An example for FORTRAN is:

```
integer*4 starttime(6), year, month, day, hour, min, sec

call dattim('start', starttime)

year = starttime(1)
month = starttime(2)
day = starttime(3)
hour = starttime(4)
min = starttime(5)
sec = starttime(6)
```

An example for C is:

```
int starttime[6], year, month, day, hour, min, sec;

dattim ("start", starttime);

year = starttime[0];
month = starttime[1];
day = starttime[2];
hour = starttime[3];
min = starttime[4];
sec = starttime[5];
```

FORTRAN Binding

```
dattim (when, time)
          char*(*) when
          integer*4 time(6)
```

C Binding

```
dattim (when, time);
          char *when;
          long *time;
```

Returns

Six integers to an array whose values correspond to the year, month, day, hour, minute, and second for the requested time information.

6.8.18 deltim - get delta time for the current timestep

```
deltim ()
```

Returns the length of the current timestep in hours.

An example for FORTRAN is:

```
real dt

dt = deltim ()
```

An example for C is:

```
float dt;

dt = deltim ();
```

FORTRAN Binding

```
deltim ()
```

C Binding

```
deltim ();
```

Returns

The timestep as a real or double value.

6.8.19 djulian - get julian day with fractional part

```
dulian (when, type)
```

`when` - This has the value "start", "end", or "now", depending on whether one wants the start, end, or current date.

`type` - This has the value "calendar", "solar", "water", or "absolute", depending on the type of year referred to.

This function returns an integer corresponding to the julian date of the start, end or current timestep, with respect to the calendar, solar or water years. The start dates for each year type are:

- calendar - Jan 1
- solar - Dec 22
- water - Oct 1
- absolute - base date

A FORTRAN example to obtain the length of a model run in hours:

```
 double precision dt

 dt = djulian('end', 'absolute') - djulian('start',
+           'absolute') * 24.0
```

A C example:

```
double dt;

dt = djulian("end", "absolute") - djulian("start",
           "absolute") * 24.0;
```

FORTRAN Binding

```
djulian (when, type)
           char*(*) when
           char*(*) type
```

C Binding

```
djulian (when, type);
           char *when;
           char *type;
```

Returns

the julian date.

6.8.20 getstep - get timestep number

```
getstep ()
```

This function returns the current integer count of the number of timesteps the model has run, regardless of timestep increment variability.

A FORTRAN example is:

```
integer*4 nstep

nstep = getstep()
```

A C example is:

```
long nstep;

nstep = getstep ();
```

FORTRAN Binding

```
getstep ()
```

C Binding

```
getstep ();
```

Returns

The current timestep number.

6.8.21 julian - get the julian date

```
julian (when, type)
```

`when` - This has the value "start", "end" or "now", depending on whether one wants the start, end, or current date.

`type` - This has the value "calendar", "solar" or "water", depending on the type of year referred to.

This function returns an integer corresponding to the julian date of the start, end or current timestep, with respect to the calendar, solar or water years. The start dates for each year type are:

- calendar - Jan 1
- solar - Dec 22
- water - Oct 1

A FORTRAN example to obtain the water year julian date for the current timestep is:

```
integer*4 jday

jday = julian ('now', 'water')
```

A C example to obtain the calendar year julian date for the current timestep is:

```
long jday;

jday = julian ("now", "calendar");
```

FORTRAN Binding

```
julian (when, type)
          char*(*) when
          char*(*) type
```

C Binding

```
julian (when, type);
          char *when;
          char *type;
```

Returns

the julian date.

6.8.22 units - get unit conversion factor

```
units (have, want)
```

`have` - the units being used.
`want` - the units desired.

This function returns the factor for converting values of one set of units into another (for example from metric to inch pound). *Units* is based on the Unix utility of the same name. It requires a character string containing the current units and another string containing the desired units. The factor returned is multiplied times the value of the variable in the "have" units to compute the value of the variable in the "want" units.

A FORTRAN example is:

```
character*80 have
double conv_fac

conv_fact = units (have, 'ft3/sec')
```

A C example is:

```
char *have;
double conv_fact;

conv_fact = units (have, "ft3/sec");
```

6.8.23 cprint commands - print information to standard output

cpstr - print string (C or FORTRAN) to standard output

cpint4 - print integer*4 array (FORTRAN) to standard output

cpreal - print real array (FORTRAN) to standard output

cpdble - print double precision array (FORTRAN or C) to standard output

```
cpstr (message)
cpint4 (message, int4_array, array_len)
cpreal (message, real_array, array_len)
cpdble (message, doub_array, array_len)
```

`message` - user message to identify output.
`***_array` - array to be output.
`array_len` - number of elements in the array

The *cprint* family of routines enable the programmer to print messages and the values of specified parameters and variables to *stdout*, the standard output device, which may

be mapped to a file at runtime. These are essentially the same as the *oprint* routines (see below).

A FORTRAN example for the use of `cpint4` is:

```
integer*4 retval

call cpint4('End of module, retval = ', retval, 1)
```

This will print the current value of `retval`.

An example application for a C coded module is:

```
long retval[10];

count = 10;

cpint ("End of module, retval = ",retval, count);
```

FORTRAN Binding

```
cpstr (message)
cpint4 (message, int4_array, array_len)
cpreal (message, real_array, array_len)
cpdble (message, doub_array, array_len)
          char*(*) message
          integer*4 array_len
```

C Binding

```
cpstr (message);
cplong (message, long_array, array_len);
cpfloat (message, float_array, array_len);
cpdble (message, doub_array, array_len);
          char *message;
          long array_len;
          long *long_array;
          float *float_array;
          double *doub_array;
```

Returns

None.

6.8.24 **dprint** commands - print debug information

dpstr - print debug string (C or FORTRAN)

dpint4 - print debug integer*4 array (FORTRAN)

dplong - print debug long array (C)

dpreal - print debug real array (FORTRAN)

dpfloat - print debug float array (C)

dpdble - print debug double precision array (FORTRAN or C)

```
dpstr (message, dlevel)
dpint4 (message, int4_array, array_len, dlevel)
dplong (message, long_array, array_len, dlevel);
dpreal (message, real_array, array_len, dlevel)
dpfloat (message, float_array, array_len, dlevel);
dpdble (message, doub_array, array_len, dlevel)
```

`message` - user message to identify output.
`***_array` - array to be output.
`array_len` - number of elements in the array
`dlevel` - debug level.

The *dprint* family of routines enable the programmer to print debugging messages and the values of specified parameters and variables to *stderr*, the standard error device, which may be mapped to a file at runtime. These are essentially the same as the *oprint* routines (see below), with the addition of the *dlevel* argument, which sets the minimum system debug level for which the print will be activated. The system debug level is set using the *-debug* argument on the command line when MMS is started. If the `dlevel` value passed in a *dprint* routine is equal to or less than the system level, then the print will be performed. If the `dlevel` value exceeds the system debug level, no print will occur.

A FORTRAN example for the use of `dpint4` is:

```
integer*4 retval

call dpint4('End of module, retval = ', retval, 1, 2)
```

This will print the current value of `retval`, preceded by the descriptive message in quotes, when the debug level is set to 2 or more.

An example application for a C coded module is:

```
long debug_level;
long retval[10];
real real_array[10];
double double_array[10];

count = 10;

dpint ("End of module, retval = ",retval, count, 2);
```

FORTRAN Binding

```
dpstr (message, dlevel)
dpint4 (message, int4_array, array_len, dlevel)
dpreal (message, real_array, array_len, dlevel)
dpdble (message, doub_array, array_len, dlevel
          char*(*) message
          integer*4 array_len
          integer*4 dlevel
```

C Binding

```
dpstr (message, dlevel);
dplong (message, long_array, array_len, dlevel);
dpfloat (message, float_array, array_len, dlevel);
dpdble (message, doub_array, array_len, dlevel);
          char *message;
          long array_len;
          long dlevel;
          long *long_array;
          float *float_array;
          double *doub_array;
```

Returns

None.

6.8.25 **oprint** commands - print information to MMS output file

opstr - print string (C or FORTRAN)

opint4 - print integer*4 array (FORTRAN)

oplong - print long array (C)

opreal - print real array (FORTRAN)

opfloat - print float array (C)

opdble - print double precision array (FORTRAN or C)

```
opstr (message)
opint4 (message, int4_array, array_len)
oplong (message, long_array, array_len);
opreal (message, real_array, array_len)
opfloat (message, float_array, array_len);
odpdble (message, doub_array, array_len)
```

`message` - error message.
`***_array` - array to be output.
`array_len` - number of elements in the array.
`dlevel` - debug level.

The *oprint* family of routines enable the programmer to print specified parameter and variable values to the MMS output file, the name of which is defined by the user at runtime using the graphical user interface. Values of a single parameter or variable or of multiple parameters and variables can be output in a single *oprint* statement. Multiple parameters and variables are output by writing them to a buffer and then printing the buffer with an *oprint* function. The write to a buffer can be formatted or unformatted.

A FORTRAN example for writing a formatted output string using a buffer is:

```
character*80 buffer

1002 format(2x,f5.0,2x,f3.0,2x,f3.0,f11.2,1x,f11.2)

write(buffer,1002)ryear, rmo, rday, obs_runoff(1), basin_cfs
call opstr(buffer)
```

A C example for writing a formatted output string is:

```
char buffer[80];

sprintf (buffer, "%d %d %d %f %f ", ryear, rmo, rday,
 obs_runoff(1), basin_cfs);
```

```
opstr (buffer);
```

FORTRAN Binding

```
opstr (message)
opint4 (message, int4_array, array_len)
opreal (message, real_array, array_len)
opdble (message, doub_array, array_len)
          char*(*) message
          integer*4 array_len
          integer*4 dlevel
```

C Binding

```
opstr (message, dlevel);
oplong (message, long_array, array_len);
opfloat (message, float_array, array_len);
opdble (message, doub_array, array_len);
          char *message;
          long array_len;
          long dlevel;
          long *long_array;
          float *float_array;
          double *doub_array;
```

Returns

None.

6.8.26 uprint - print information to output files referenced by dimension index.

upstr - print string (C or FORTRAN)

upint4 - print integer*4 array (FORTRAN)

uplong - print long array (C)

upreal - print real array (FORTRAN)

upfloat - print float array (C)

updble - print double precision array (FORTRAN or C)

```
upstr (dim_name, index, message)
upint4 (dim_name, index, message, int4_array, array_len)
uplong (dim_name, index, message, long_array, array_len);
upreal (dim_name, index, message, real_array, array_len)
upfloat (dim_name, index, message, float_array, array_len);
updble (dim_name, index, message, doub_array, array_len)
```

`dim_name` - name of reference dimension.
`index` - index element to use.
`message` - error message
`array_len` - number of elements in the array

The *uprint* family of routines enable the programmer to print output to files which correspond to the index names of a specific dimension. *Uprint* will write an output file for each of the named indices of the dimension. These files have the names of the corresponding index and are created in the output directory. The index names are created by the user using the *dimension index names* feature of the *edit* pull down menu in the main MMS GUI window. The index elements must be named or the functions will not produce any output.

The *uprint* function is used to write a separate file for each element of an array, where each element could be, for example, a streamflow simulation node within a large watershed. The resulting files would have the name of the node and the resulting time series stored in each file would be the simulated streamflow hydrograph at each node. The first write to a file opens the file and names it using the index name. Each subsequent write is appended to the open file.

A FORTRAN example is:

```
integer nnode, i, year, month, day
character*30 str
real value

do 10 i = 1, nnode
write (str, *) year, month, day, value
call upstr ('nnode', i, str)
10 continue
```

A C example is:

```
int nnode, i, year, month, day;
char str[30];
float value;
```

```
for (i = 0; i < nnode; i++) {
printf (str, "%d %d %d %f", year, month, day, value);
upstr ("nnode", i, str);
}
```

FORTRAN Binding

```
upstr (dim_name, index, message)
upint4 (dim_name, index, message, int4_array, array_len)
upreal (dim_name, index, message, real_array, array_len)
updble (dim_name, index, message, doub_array, array_len)
          char*(*) dim_name
          integer*4 index
          char*(*) message
          integer*4 array_len
          integer*4 dlevel
```

C Binding

```
upstr (dim_name, index, message);
uplong (dim_name, index, message, long_array, array_len);
upfloat (dim_name, index, message, float_array, array_len);
updble (dim_name, index, message, doub_array, array_len);
          char *dim_name;
          long index;
          char *message;
          long array_len;
          long dlevel;
          long *long_array;
          float *float_array;
          double *doub_array;
```

Returns

None.

7. PLANNED ENHANCEMENTS

MMS is envisioned to be composed of seven basic components. These components are:

- A modular modeling support framework.
- A graphical user interface and expert system.
- A statistical analysis package.
- Optimization and sensitivity analysis tools.
- A GIS interface for observed and model data processing, analysis, and visualization.
- A generic structured query language (SQL) database interface.
- Policy and risk analysis tools.

Work on each of these components is ongoing but at variable levels of effort. The modular modeling framework and the graphical user interface components are the most fully developed, and their current state of development is described in detail in this manual. A few statistical analysis, optimization, and sensitivity analysis tools have been included in the initial release of MMS, but many more of these types of tools are needed.

GIS tools are being developed for pre-processing functions that include (1) data display and analysis and (2) the delineation and characterization of various hydrologic and ecosystem regions using topographic, soils, vegetation, and geologic databases, and (3) the estimation of a variety of model parameters in each of the delineated regions using these databases. Watershed delineation and characterization tools exist in both GRASS and ARC/INFO and can be used to pre-process spatial data. However, a GUI is being developed to support a set of tools for watershed delineation, characterization, and parameterization that require a less extensive knowledge of GIS systems. Initial development of the GUI is being completed using ARC/INFO.

Additional pre-processing tools are planned for use with time-series data. These tools include procedures to detect bad or missing data values, replace bad or missing data using selected statistical procedures, aggregate data to longer time steps, disaggregate data to shorter time steps, and apply transform functions to produce a new time-series. Methods to create simulated time series from model output or from the analysis and extrapolation of measured data to unmeasured points or grided fields are also being developed.

MMS currently uses an ASCII flat-file format for data storage and input. However, a goal is to enable the use of a variety of databases, including SQL databases, dependent on user preference and prescribed needs. Work is ongoing with the U.S. Bureau of Reclamation (USBR), Natural Resources Conservation Service, and the Rheinisch-Wesfällische Technische Hochschule in Aachen, Germany, to develop these generic tools.

The ability to couple a variety of resource-management and risk-analysis models with user-selected process models is being developed for use in evaluating alternative resource-management policies and in developing operational short- and long-term resource-management plans. These efforts are ongoing jointly with the USBR in the area or watershed and river-system management, and with the U.S. Forest Service in the area of forest-ecosystem management. Interfaces are also being developed to import and export data and model results from and to other external data-management and analysis systems.

As the number and type of applications and modules are developed, system modifications and enhancements will be identified. Initial applications at MMS beta testing locations have identified several of these needs including (1) system procedures to handle feedback loops for selected process module combinations and (2) flexible system procedures to accommodate alternative ways to configure the entities of time, space, and module process. This includes the ability to use different methods (modules) to simulate one or more processes on different spaces delineated within a watershed or ecosystem. The next version of MMS will provide procedures to accommodate these needs.

The development of MMS is seen as an evolutionary process, and so the system will continue to evolve to meet user needs and accommodate advances and improvements in modeling methodologies, types of data available, and computer software and hardware. This manual will be updated and re-released to include these changes. In addition, documentation for the other components of MMS will be released as soon as these capabilities are fully developed and tested.

8. REFERENCES

Beck, J.V. and Arnold, K.J., 1977, Parameter Estimation in Engineering and Science: New York, John Wiley, 501 p.

Davidon, W. C., 1959, Variable metric method for minimization: U.S. Atomic Energy Commission, Argonne National Laboratories, Research and Development Report ANL-5990, ---p.

Day, G.N., 1985, Extended Streamflow forecasting using NWSRFS: American Society of Civil Engineers, Journal of Water Resources Planning and Management, v. 111 no. 2, p. 157-170.

Eagleson, P.S., 1978, Climate, soil, and vegetation - 6. Dynamics of the annual water balance: Water Resources Research, v. 14 no. 5, p. 749-764.

Fletcher, R. and Powell, M.J.D., 1963, A rapidly convergent descent method for minimization: Computer Journal, v. 6, p. 163-168.

Leavesley, G.H., Lichty, R.W., Troutman, B.M., and Saindon, L.G., 1983, Precipitation-runoff modeling system--User's manual, U.S. Geological Survey Water Resources Investigation Report 83-4238, 207 p.

Mein, R.G. and Brown, B.M., 1978, Sensitivity of optimized parameters in watershed models: Water Resources Research, v.14 no. 2, p. 299-303.

Restrepo, P.J. and Bras, R.L, 1982, Automatic parameter estimation of a large conceptual rainfall-runoff model: A maximum-likelihood approach: Massachusetts Institute of Technology, Department of Civil Engineering, Ralph M. Parsons Laboratory Report No. 267, ---p.

Rosenbrock, H.H., 1960, An automatic method of finding the greatest or least value of a function: Computer Journal, v. 3, p. 175-184.

U.S. Army Corps of Engineers, 1991, Geographical Resources Analysis Support System (GRASS) version 4.0 user's reference manual: U.S. Army Corps of Engineers Research Laboratory, 513 p.

Attachment 1

Recommended System Configuration

MMS runs on a variety of Unix platforms. It has been successfully ported to the following vendor's machines and operating systems:

- Sun (both SunOS 4 and Solaris)
- Data General (DGUX)
- Hewlett-Packard (HP-UX)
- IBM (AIX)
- Silicon Graphics
- PC (Linux)

Installation will require approximately 50 megabytes of hard disk space. This can be reduced by removing the source code directories (mms/src) after compilation. The final installation size will also depend on compilation options, models built, and the size of the data files.

Required additional software and libraries:

- X11 and Xt (Intrinsic) libraries and include files
- Motif library and include files
- C compiler (such as Gnu's gcc)
- FORTRAN compiler (for PRMS and TOPMODEL models)
- an editor (vi, emacs, etc.)
- an HTML viewer (Mosaic, Netscape, etc.) for the help system

Optional additional software and libraries:

- GRASS 4.1.x

Attachment 2

Module Documentation Example

NAME: Potet_hamon.f

MODULE PROCESS (TYPE): Potential evapotranspiration

DEFINITION: Determines whether current time period is one of active transpiration and computes the potential evapotranspiration for each HRU using the Hamon formulation.

CREATION DATE: July 1992

PARAMETERS DECLARED:

basin_area Total basin area, in acres.
epan_coef Evaporation pan coefficient.
hamon_coef Monthly air temperature coefficient used in Hamon potential evapotranspiration computations.
hru_area HRU area in acres.
hru_radpl Index of radiation plane associated with each HRU.
temp_units Indicator for units for temperature data, 0= F and°1=°C.
transp_beg Month to begin summing maximum temperature for each HRU; when sum is greater than or equal to **transp_tmax**, transpiration begins.
transp_end Last month for transpiration computations. Transpiration is computed through the end of the month.
transp_tmax Temperature index to determine the specific date of the start of the transpiration period. This subroutine sums the maximum temperature for each HRU starting with the first day of month **transp_beg**. When the sum exceeds this index, transpiration begins, °F or °C, depending on units of data.

VARIABLES DECLARED:

basin_potet Weighted average potential evapotranspiration for basin.
potet Potential evapotranspiration for each HRU, in inches/day.
transp_on Indicator for whether transpiration is occurring, 0=no, 1=yes.
transp_check Indicator for whether within period to check for beginning of transpiration, 0=no, 1=yes.

EXTERNAL VARIABLES USED

tavgc Average HRU temperature in °C.
tmaxf or **tmaxc** Maximum HRU temperature, °F or °C, depending on units of data.
radpl_sunhrs The hours of daylight for each day, in units of 12 hours.

DESCRIPTION

This module was written using FORTRAN code from the U.S. Geological Survey's Precipitation-Runoff Modeling System (PRMS) (Leavesley and others, 1983). The PRMS subroutine *pets.f* is the source for most of the code.

The time of the year in which transpiration occurs is specified as a period of months, starting with **transp_beg** and ending with **transp_end.** The specific date of the start of transpiration is computed for each HRU using the temperature index parameter **transp_tmax.** For each HRU, the sum of the maximum air temperatures is accumulated, starting with the first day of the month **transp_beg**. When the sum for an HRU exceeds **transp_tmax**, transpiration is assumed to begin on that HRU. This permits accounting in part for warmer or colder than normal spring periods. Transpiration ends on the last day of **transp_end**. **Transp_on** is equal to '1' during the transpiration period.

The potential evapotranspiration for each HRU (**potet**) for each time period is computed as a function of daily mean air temperature and possible hours of sunshine (Hamon, 1961) using the equation:

$$\mathbf{potet} = \mathbf{hamon\_coef} \times \mathbf{radpl\_sunhrs}^2 \times \mathbf{vdsat} \qquad (1)$$

where

hamon_coef is the monthly air temperature coefficient used in Hamon potential evapotranspiration computations,

radpl_sunhrs is the hours of daylight for each day, in units of 12 hours, and

vdsat is the saturated water-vapor density (absolute humidity) at the daily mean air temperature (°C) in grams per cubic meter (g/m^3) computed by (Federer and Lash, 1978):

$$\mathbf{vdsat} = 216.7 \times \frac{\mathbf{vpsat}}{\mathbf{tavgc} + 273.3} \qquad (2)$$

where

tavgc is the average HRU temperature in °C, and

vpsat is the saturated vapor pressure in millibars (mb) at the daily mean air temperature (°C) and is computed as (Murray, 1967):

$$\mathbf{vpsat} = 6.108 \times \mathbf{exp}\left[17.26939 \times \frac{\mathbf{tavgc}}{\mathbf{tavgc} + 237.3}\right] \qquad (3)$$

Hamon (1961) suggests a constant value of 0.0055 for **hamon_coef.** Other investigators (Leaf and Brink, 1973; Federer and Lash, 1978) have noted that 0.0055 underestimated **potet** for some regions. Limited experience also has shown that a constant value underestimates **potet** for the winter months more than for the summer months.

The basin weighted average potential evapotranspiration, **basin_potet**, is also computed in this module.

REFERENCES

Federer, A. C., and Lash, Douglas, 1978, Brook: A hydrologic simulation model for eastern forests: Durham, New Hampshire, University of New Hampshire, Water Resources Research Center, Research Report No. 19, 84 p.

Hamon, W. R., 1961, Estimating potential evapotranspiration: Proceedings of the American Society of Civil Engineers, Journal of the Hydraulic Division, v. 87, no. HY3, p.107-120.

Leaf, C. F., and Brink, G. E., 1973, Hydrologic simulation model of Colorado subalpine forest: U.S. Department of Agriculture, Forest Service Research Paper RM-107, 23 p.

Leavesley, G. H., Lichty, R. W., Troutman, B. M., and Saindon, L. G., 1983, Precipitation-runoff modeling system--user's manual: U.S. Geological Survey Water Resources Investigations Report 83-4238, 207 p.

Murray, F. W., 1967, On the computation of saturation vapor pressure: Journal of Applied Meteorology, v. 6, p. 203-204.

DEVELOPER NAME AND ADDRESS

George H. Leavesley
U.S. Geological Survey, WRD
Box 25046, MS 412, DFC
Denver, Colorado 80225

e-mail: george@usgs.gov