Purpose of CRHM

An integrated assessment of land use impacts on water balance and streamflow is necessary in order to make sound recommendations for improved land management practices. To achieve this, results from recent process-based hydrology research have been integrated into a Cold Regions Hydrological Model (CRHM). The model provides the user with advanced techniques for the calculation of water balance and streamflow.

Many of the ideas for the CRHM model came from the Modular Modeling System (MMS) developed by George H. Leavelsey, U.S. Geological Survey (USGS).

Overview

Hydrological modelling has been complicated by the increasing complexity of environmental and water-resource based problems and the broad range of scientific disciplines that must be incorporated into a system in order to adequately deal with the issues. Choosing a simulation from a wide selection of models to address the study objectives, data constraints and spatial and temporal scales of application is often very difficult.

CRHM uses modular modelling tools to develop, support and apply dynamic model routines. The integrated system of software provides the framework to develop and evaluate physically-based algorithms and effectively integrate selected algorithms into an operational model. Existing algorithms can be modified or new algorithms can be developed and added as modules to the module library. Modules from the library are coupled to create a physically-based model suitable for the specific application.

CRHM is a new strategy developed to incorporate specific and often neglected aspects of hydrology:

1. flows of snow, ground and soil water and energy between adjacent land units,
2. water movement in snow and frozen soils,
3. snow and rain interception in forest canopies,
4. effect of slope and aspect on "vertical exchange processes",
5. coupled mass and energy balance controls on process rates, and
6. coupling between soil moisture, groundwater and base flow.

The model is sensitive to land use and climate so that is can be used for assessments of impacts of changes to these conditions on the hydrological state of a watershed as indexed by soil water, streamflow, etc. The model development is a multiple year project with increasing utility as components are added to the model. A modular object-oriented structure will make the model relatively easy to update and improve as new research results become available. Ultimately CRHM will provide a scientific tool, or methodology that provides the hydrologist with well-defined techniques for calculating the water balance and generation of streamflow runoff in cold climate regions.

Components of CRHM.

CRHM has the following components:

1. Observations – time-series meteorological data at varying intervals,
2. Parameters – Spatial data (e.g. basin area, elevation, and cover type) are generated using a GIS interface tool to assist the user in basin delineation, characterization and parameterization of HRU. HRU are subdivisions of the basin characterized by the operator from an understanding of the hydrological processes, terrain and land use.
3. Modules – Algorithms implementing the hydrological/physical processes. The model data structure is specified by the declarations in the modules but is implemented globally by the CRHM platform.
4. Variables and States are created by the declarations in the modules.

CRHM Model Platform.

The CRHM Model Platform performs the following services:

Basic functions.

1. Configures the model to the number of HRU and HRU layers.
2. Builds the selected modules into a working model after checking the structure and data flow of the model.
3. Links the Observation files to the model.
4. Links the parameter data to the model.
5. Permits initial state files to be set up as input to the model or as output to receive the final state of the model.
6. Sets the duration of the model run.
7. Selects the desired state/variable values to be displayed and available for output.
8. Executes the model.
9. Provides interaction with the graphical display.

Housekeeping functions.

- Save and Load project files to allow the model (project) to be saved as an entirety which can be later loaded and run.
- Help for operating the CRHM platform and help describing the functionality of the module, variables and states.
- Exporting the model output to files for use by other applications (e.g. Microsoft Excel).
- Exporting the model output for later input to compare with other CRHM model runs with different parameter values.
- Statistical and graphical tools to analyze input data and the model performance.
- Model module flow diagrams to demonstrate data flow within the model. Driving observations or input parameters are superimposed on the flow diagram to help the user to visualize their entry into the model.
- Model output may be superimposed upon HRU outlines to aid spatial visualization of the model results.
- Observations may be displayed as a diagnostic tool to detect data problems. This is enhanced by the capability to plot the time series data as daily mean, daily maximum, daily minimum, daily sum and cumulative sum. Other functions are also available.
- Observation data may also be manipulated using filters. These filters take various forms. Examples are scaling, unit changing, time interval changing and replacing missing or faulty data with adjacent or interpolated data.
- User can synthesize input observation data using functions to generate sine/ramp/pulse/log etc. waveforms as a function of time. These simple driving inputs are indispensable for diagnostic testing as actual meteorological data can be too complex to initially comprehend and test algorithms.
- Parameters may be displayed, edited and saved or loaded from files. Two options are available. The first is from text files and the second is from database files.
- CRHM is compatible with ESRI® shapefile software. ARCGIS® data can be imported as a shapefile to set parameter values and HRU and basin perimeter coordinates.

Expandable Aspects.

1. Users can create their own modules with basic knowledge of C++. These modules are linked to make an executable dynamic linked library (DLL) which is loaded into CRHM. The user written modules are handled identically to the original modules.
2. Users can create help files describing the capabilities of their custom modules and CRHM will automatically integrate the help file into the CRHM help menu.
3. Users can replace existing CRHM modules with custom versions of a module to test enhancements, simplifications or to add diagnostic variables.

CRHM Terminology

The purpose of this section is to provide the user with a glossary of terms and definitions used in the CRHM user's guide.

Observations.

The meteorological or climate time series data required for model input (e.g. air temperature, wind speed). Observation files must follow the ASCII space or tab delimited format with an '.obs' extension. Access to these files is via the CRHM *Observations* pull down menu.

Parameters

.

The basin/HRU spatial data required as model input (e.g. area, elevation, cover type). The basin/HRU spatial data can be manually inserted into the model or imported as a text file with a '.par' extension or a database file with a '.dbx' extension. Access to parameters is by the CRHM *parameters* pull down menu. Parameters are also known as coefficients.

Hydrological Response Unit (HRU).

HRUs are subdivisions of the basin characterized by the operator from an understanding of the hydrological processes and land use.

Build.

This feature is on the CRHM pull down menu and allows the user to select modules and to create a model.

Construct.

A CRHM pull down menu option located within the Build feature that allows the user to determine more information about individual modules and to add/delete modules to/from the current model.

Run.

This feature is on the CRHM pull down menu and allows the user to execute the model.

State.

This feature on the pull down menu allows the user to save the final state of a model run. This state can then be used as the initial state of the model for a subsequent run starting from that time on. This feature can save the user a lot of time. For example the model could be run once over winter to determine snow accumulation and the final model state saved in the spring. The model could then be run many times from this time on with a range of melt parameters to check out different melt scenarios.

Project.

A feature from the menu that allows the user to save a model to a project file. The save includes modules, observation files, parameters, displayed variables etc. This project file can be loaded and run at a later time.

Module.

Each module represents a physically-based algorithm (e.g. Evaporation, Melt etc.) or a modeling procedure (e.g. basin, obsBad) and are added to a model using the Build feature in the CRHM pull down menu.

Macro.

The macro feature allows the user to create simple modules as text commands in the macro screen within CRHM instead of having to code the module off line in C++. Macros are intended for testing algorithms and diagnosing model output. There is a speed penalty and limited capability.

Check.

A Selection within the Construct option that verifies that all the supporting modules required by the modules selected for the current model are available

and attempts to load these modules or makes suggestions if there is a choice of modules.

Chart.

Refers to the graph displayed in the Graphical Output window (TeeChart®) when observing data or outputing model results.

Spreadsheet.

A term referring to the table containing data. Example are the Build and Parameters screens in the CRHM pull down menu.

Model Concepts.

A CRHM model is constructed out of Modules - the basic building blocks implementing the model algorithms. The modules depend upon the transmission of information between themselves and the outside world. The information must be processed in an orderly manner - so the order or sequence of the modules in the execution chain is important. The model information breaks down into three groups.

- Parameters - physical parameters like elevation, area, vegetative/agricultural use etc. which are used in the module algorithms. Assumed to be constants. Also used as initial values for variables.
- Observation - meteorology information providing the driving force of the model. Source is data files. Could also be generated data.
- Variables - state variables and value variables which store the states of the modules and provide the interchange of values between the modules.

Organisation.

Sequence or Order of modules.

This is determined from the ***variable*** flow through the model. When a specific module requires an input ***variable*** from another module it follows that the module generating the required ***variable*** must appear earlier in the chain of execution of modules. Within the CRHM platform there is logic that will automatically order the modules of a model according to their variable hierarchy.

Parameters must be defined in every module they are used in. The CRHM platform has the capability of grouping parameters with the same name and with identical values for each HRU together and calling them ***basin*** parameters. Changing the value of a basin parameter changes the value for every module sharing the parameter.

Observation data is assumed to be available to all modules from the beginning of a time period till its end. The module order is independent of the observations.

Problems with the simple concept.

When models are only run over a short time period of time, parameters can be assumed to be constant. However, most parameters gradually change with time and can change dramatically with the season. They in fact become variables.

Observations are the driving force of the model. However, they are not constant over the area being modelled as the observations have to be interpolated to determine actual values for the individual HRUs of the model. The observations may also have to be corrected for the elevation of the HRU.

Solution.

The operation of parameters and observations were extended to handle the shortcomings outlined above.

Parameters.

Since operational parameters vary with time and season, it was decided to make them input variables with the proviso that if they could not be satisfied by another module output they would link to a constant parameter of the same name defined within the current module. This has many advantages. Modules can inititially be tested using constant parameters when the variable name is unsatisfied. Later they can be inserted into a functional model and the parameter variable satisfied by an earlier module output of the same name as the parameter.

Observations.

Another category of Observation was created. This combines the properties of an observation with a variable. A module can create an observation which can satisfy the observation input requirement of a later module. However, if a file observation of the same name is available, it will override the module generated observation. A distinguishing feature of a module generated observation is that it determines the order of the modules in the model as if it was a normal variable.

Observation dimensions.

Some ambiquity arises with respect to observation dimensions which is made more difficult by the fact that observations generated by modules are normally dimensioned NHRU.

File observations are normally dimensioned NOBS. In practice most often there is only a single observation (not a value for each HRU). However, the module call to declreadobs returns at runtime the actual number of observations available and the programming logic can be designed to handle fewer values. In the case of macro modules the observation access is limited to the maximum number of legal values.

File observations are normally dimensioned as NOBS. The value of NOBS is not normally of much use as it returns the value for the observation with the maximum number of observations.

Module Interconnection.

Module varies between very simple and complex depending upon the amount of control the user builds into the module. Modules have three types of inputs and two types of outputs.

Observations.

Observations are traditionally measurements. Since the number and variety of field observations vary immensely, provision was built into the CRHM Platform to make them **optional**. An example of the use of this feature is precipitation. It can be daily or interval. To handle this transparently the *obs* module asks for both but makes them optional. Then when the model runs it accepts whatever is available and scales the values accordingly.

Observations may be available for every HRU or may only be collected at one point. CRHM modules can handle this. In modules the programmer has full control and in macro modules the last value of available observation data is duplicated for subsequent HRUs.

The concept of declared (module generated) observations was introduced. This allows the user to generate from the field (measured) observations declared (calculated) observations for every HRU. An example of this is the macro *Slope_Qsi*. This macro generates incoming short wave radiation on a sloping HRU from one measurement of incoming short wave radiation on the level to adjust for cloud cover using theoretically calculated clear sky short wave radiation.

Variable Inputs.

Variables are the quantities exchanged between modules. They are never **optional** as it assumed that modules always require these inputs. The fact that variable inputs are mandatory allows the CRHM Platform to automatically construct a model from individual modules.

Parameters.

Parameters are the constant spatial and physical coefficients required by the modules. They can with the same name exist independantly for every module in the model using them. However, if the values for a particular parameter are identical for two or more modules when the project is saved they will become basin parameters the next time the project is loaded.

It became evident that on occasions, parameters are not constant but need to vary over the course of a model run. An example being vegetation height. One method of handling this is to change the vegetation height from being a parameter to a variable input. This has the disadvantage of always requiring that the variable input height must always be satisfied by the output of another module. An alternative method is to leave height as a parameter and allow modules to change the parameters during the model run. This can easily be done using a macro module which can read the vegetation height from an observation file or calculate the crop height from an algorithm and change the parameter every interval.

AKA Screen.

Despite all the flexibility built into Observations and Variables all model requirements could not be met. Some examples follow.

1. Different Time Units. - Various melt models were implemented. Some calculate melt every interval and others over a day. It was very difficult to handle the output of the different melt modules without customising the modules that received their output. The AKA screen allows the normal connections between modules to be broken and custom modules to be inserted to convert outputs to match the requirements of the next module. This could be mean a change in units or interval, day to interval or vice versa.
2. Modules requiring enhanced observations.
3. User Name Preferences. - Simple observation names may be renamed. E.g. "t" can be renamed "T" etc.

CRHM Criteria for linking modules to create a model.

The order that modules appear in the flow diagram is the order that they are

executed. If a module output is used in an interval before the module is executed the variable value used will be the model initial value in the case of the first interval or the previous interval value there after. State variables are handled as a special case. Their values are continuous and there last interval value is often used to calculate their new value for the current interval. Examples are albedo and SWE.

1. Modules are arranged by variable input requirements.
2. State variables are ignored in the arrangement of the modules. This may be overridden using the setpeer command if neccessary.
3. Modules are arranged by declared observation requirements.
4. Parameters never influence the arrangement of modules.
5. The setpeer command delays the loading of a module until the declared variable input is available (i.e. calculated for current interval).

System Requirements

- An IBM®-compatible machine with a Pentium or AMD processor.
- CD-ROM or USB drive.
- A hard disk with at least 50 megabytes of available disc space.
- Microsoft Windows 98, ME, XP, NT or later.
- A graphics display compatible with Microsoft Windows.
- At least 32 megabytes of available RAM.

Introduction.

The CRHM program is installed from a downloaded installation file "CRHM_Distribution.zip". The program will run on Microsoft operating systems. However, on Apple operating systems a Windows emulator is required.

Complete the following steps to be able to run CRHM on your Windows computer.

Installing CRHM.

1. Run 'Setup.exe' on the CRHM installation file.
2. Correct Name and Company if the default on your system is incorrect.
3. Select the installation directory. The default is 'c:\Program Files\CRHM'. Another choice might be 'C:\CRHM'.
4. Select the program folder name. Default is good.
5. Application can be launched immediately.
6. Exit

Note.

When CRHM is installed in the directory 'c:\Program Files\CRHM' most installations will not allow the user to write to the sample files (i.e. edit them) as the permission is READ only. It is suggested that these sample directories be copied to a location where the user has full READ/WRITE privileges.

Module Distribution Files.

- CRHM.chm - HTML help file for all models.
- Newodules.chm - HTML help file for modules.
- Macro.chm - HTML help file for macro programming.
- CRHM.exe - Self standing non extendable model.
- Newmodules.cpp - Module C++ code. Provided for reference only.
- NewModules.h - Module C++ header file.

The first three help files are also supplied as PDF files.

Command line interpreter.

Normally CRHM is used as a window program. However, in order that it can execute from a project file or display an observation file it must have a command line interpreter (cil).

The cil is able to handle project, observation and parameter files. An example is "CRHM_new.exe Project.prj MyObs1.obs MyObs2.obs Myparam1.par Myparam2.par". No special error handling is implemented and any errors will cause the loading to stop and the program display the error. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces), e.g. "C:\Program Files\CRHM\Examples\MacroExample1.prj"

1. *.prj - only the first project file is recognized. Any other is ignored.
2. *.obs - multiple observation files can be loaded. No error handling is done. Faulty file format or duplicate variable definitions will stop the loading.
3. *.par - multiple parameter files can be loaded. The project file using all the parameters must be loaded first. If parameters are doubly defined this is not recognised as an error. However, unknown parameters will cause an error stopping loading. An error will occur if there are too few parameter values defined for a parameter, i.e. number of values not equal to the number of HRUs in the model or group.

Automation.

For optimisation it is convenient to run a model and generate output and change the value of the parameter and repeat. After multiple runs the output files are processed with the parameter changes to determine sensitivity etc. CRHM has this capability.

1. Create a project file for the desired modules.
2. Save the project file with these options set: AutoRun, AutoExit and either Log/Last or Log/All.
3. From the saved project or parameter file create a series of batch parameter files to insert in the command line to vary the desired parameter.
4. Change the basin RUN_ID parameter every CRHM execution to identify the model output file, file name is CRHM_output with RUN_ID appended.
5. CHRM just before closing sets the registry entry,

HKEY_CURRENT_USER/CRHM_output/basin RUN_ID to the integer value of RUN_ID. By checking this registry entry a calling program can check the progress of CRHM in processing command line requests.

6. In Microsoft Excel, code from http://www.cpearson.com/excel/ShellAndWait.htmt can be used to ensure that the last process is blocked until the Excel RefreshAll picks up all the new values.

Introduction.

The CRHM model has the following help files which are accessible from the Help menu within CRHM or can be run as separate applications.

1) CRHM.chm - describing the operation of the CRHM platform. *2)* New_Module.chm - describing the hydrological modules included with CRHM.
3) Macro - describing advanced features in CRHM available to write simple modules and debugging.

PDF conversions of these files are also distributed.

Run CRHM using one of the following methods.

1. Select "Start/All Programs/CRHM Programs".
2. Double click on the CRHM shortcut on the desktop.
3. Double click on a project or observation file indicated by a *.prj or *.obs extension respectively.
4. Select a project or observation file indicated by a *.prj or *.obs extension respectively and right click "SendTo" and then the version of CRHM required.

Sample Project and Observation files.

These directories are installed in the same directory as CRHM.exe by the installation software.

1) CRHM_project_examples - provides comprehensive example projects for diverse climatic regions in Canada.
2) Examples - simple examples including some macro module examples.
3) Examples2 - Macro examples.

NOTE: *Always click on the left side of the mouse when asked to select an option or feature in CRHM unless otherwise instructed. Then a right click will display any choices of action available.*

Main Window.

The Cold Regions Hydrological Model Platform program is run from this window. Options not available in the program version being used are greyed out.

Project.

The **Project** menu provides selections that apply to the project. It allows a complete model to be saved in a file (*.prj). A previously saved model may be loaded and run. Every detail of the model is saved, including model modules, observation files, Dll files, parameters, states, times and variables and observations displayed.

The usual file options are available; ***Open***, ***Save***, ***Save As***, ***Close*** and ***Exit***.

Report.

This choice generates a report giving all the particulars of the the current model. It may be written to a file or be printed. There are also other selections for program maintenance. The normal choices are: .

Hierarchy

This option is used to check that the modules are used in the correct order in the project and that variables are never used until the latest value for the current time step is calculated..

Extract Group

This option allows a single group of a complex model to be extracted and run as a simple project for debugging purposes or inclusion in another project.

Extras

NaN_check - used to check variable data for out of range values. Resets to OFF after the next model run.

LogVarLoad - normally the output variables in the Variables Selected listbox are loaded from the project file. When this option is selected the ouput is filled from every occurrence in the current project of the variables listed in the file: 'LogVarLoad.lvl'. This file contains only one line. This line uses spaces or commas as the delimiter between the desired variables.

For example if the first line of the file is "soil_moist, soil_rechr SWE", for a simple project the Variables Selected listbox would be filled with - soil_moist(1) ... soilmoist(nHRU) + soil_rechr(1) ... soilrechr(nHRU) + SWE(1) ... SWE(nHRU).

For a group project the Variables Selected listbox would be filled with - soil_moist@A(1) ... soilmoist@A(nHRU) + soil_rechr@A(1) ... soilrechr@A(nHRU) + SWE@A(1) ... SWE@A(nHRU), where nhru is the number of HRUs in group A.

+ soil_moist@B(1) ... soilmoist@B(nHRU) + soil_rechr@B(1) ... soilrechr@B(nHRU) + SWE@B(1) ... SWE@B(nHRU). where nhru is the number of HRUs in group B.

+ repeated for all other groups in the project.

Log Time Format

The default time format is the Microsoft decimal number of days from the year 1900. This can be changed to MM/DD/YYYY or YYYY-MM-DD from the default MS format.

File, Execution thru RenameGroup

These choices are only used for program diagnostics

AutoRun.

This choice determines if a model is automatically run when a project is loaded.

AutoExit

This choices causes CRHM to exit after completing the project run. Used for batch runs.

Log.

When a model is run Log provides three options for saving the model ouput and mass balance to a file, "CRHM_output[_nn].txt/.log" file suitable for importing into another application.

1. Last - The values of the selected variables are saved for the last time step of the model run to the log file with the extension 'txt'.
2. All - The values of the selected variables are saved every time step of the model run to the log file with the extension 'txt'.
3. Debug_Screen - Writes the debug screen to a file with extension '.log'.
4. Summary - Writes the variables requested by the user to be summarized are written to a file with the extension '.sum".

The output file name is "CRHM_output" except when the basin RUN_ID parameter is greater than zero, then the ID value is appended using an underscore, e.g. "CRHM_output_123" for an ID value of 123. The file is always saved in the project directory.

The format of the data text file is as follows. The first line of the file consists of the text "time" followed by the selected variable names. The next line consists of the text "units" followed by the variable units. In the case of the 'Summary' screen the user will have to modify the units listed according to the fuction used and the timebase to generate the output. The subsequent lines have the model time in the first column followed by the variable values for the time step. This format is easily imported into Excel and other applications. An alternative method of saving the model output is to use the Export menu which supplies more flexibility but cannot be automated.

The debug screen is copied as displayed to the file.

SaveChartTemplate.

Using the TChart editor the graphical screen appearance can be changed. This option allows the chart theme to be saved to the project file.

Plot refresh rate.

Updating the graphics screen significantly slows down CRHM. This option allows the update rate to be set. Not refreshing the graphics screen until the

model run is finished is fastest.

Freq_Default.

When a CRHM project without an observation is executed this option sets the timestep. An example is a project only using the module "global".

Observations.

Observation files are selected using the Observations feature from the CRHM pull down menu. Multiple files may be selected but the first file specified determines the maximum modeling period and the time step used by the model. Later files will be skipped over to the start time of the first file and may end earlier than the first file. Later files need not have the same frequency as the first file. The data in the later files may also be discontinuous (sparse). When the model is saved as a project all the observation file names are saved with the model in the project file.

NOTE: *At any point during the operation of the model, the user can check the lower left hand corner box of the CRHM window to automatically view the status of the program. This box will also identify and describe the feature on the CRHM window that the cursor is pointing to.*

Selecting Observations from the main menu gives the following choices,

Open.

Opens an Observation File. When an observation file is opened all the observation data names defined in the file are displayed in the Observation listbox. When any of the names are selected, the description of the data if given in the data file is displayed in the status box followed by the file name. When an observation file is opened, its name is added to the observation menu list. The file may be closed by selecting the file name on the menu list.

CloseAll.

Closes all open observation files.

Displaying Observations.

If an observation is selected and right clicked a menu pops up allowing the observation to be added to the Selected listbox. Observations listed in the Selected listbox are immediately displayed on the chart.

Formats for displaying Data.

Observation - interval data.

VP_saturated - Saturated vapour pressure for temperature

W_to_MJ - the data is converted from Watts to Mega Joules per time interval.

MJ_W - the data is converted from mega Joules per time interval to Watts.

Average - daily average value.

Minimum - daily minimum value.

Maximum - daily maximum value.

Daily Sum - sum of daily interval values.

Positive - sum of daily interval values that are greater than zero.

Total - total of interval values over run period

First - displays data once at the beginning of the day.

Last - displays data once at the end of the day.

Peak - peak value over period.

Count - number of intervals when the value is non zero.

Count0 - number of intervals when the value is zero.

Delta - change in the variable over the period. For the first period, the first value is used as the datum. Thereafter, the last value of the last period is used.

Build menu.

The Build feature of the CRHM pull down menu allows the user to choose pre-defined models, build their own model from the available modules and to delete all modules from the current model.

pre-defineded models.

The pre-defineded models are created by the creator of modules to demonstrate the module application. Current examples are Evap and PBSM. To change the number of HRUs or layers from their current values, the user must use *Clear Modules* if an existing model is loaded and then change the number of HRUs or layers.

Clear Modules.

This menu option removes all currently selected modules. It should be used before defining a new model. Unless *Clear Modules* is executed it is not possible to change the number of HRUs or layers from their current values.

Construct.

The Construct screen allows the user to add or delete modules from the current model and to retrieve more information about a module interface. Models can be checked and built.

Defining Models.

On the lower right hand corner of the Construct window there are two *UpDown* controls called **MAXLAY** and **MAXHRU**. These allow the user to set the number of HRUs or layers to be used in the model. These should be set before any modules are added. To change the dimensions for an existing model it must be rebuilt from scratch after changing the dimensions. A parameter file from the original model will reload all the parameters correctly if the number of dimensions is the same or reduced. However, if the number of dimensions is

increased the extra dimension of parameters will be set to the values of the last HRU defined. Any new modules added will have the module default parameter values.

The Construct window displays on the left hand side a list of all the available modules. When a module is selected a spreadsheet on the right hand side of the screen displays the module interface. This description includes required inputs, parameters, observations and variables and also the output variables generated by the module.

By **right clicking** on a module in the Modules Available box the module will be added or deleted from the model. When assembling a model the last module should be selected and then the Check option selected. CRHM will then determine the supporting modules required and add them to the model by moving them to the Selected listbox. When a module has wildcard inputs the program can only make suggestions to which modules will satisfy the input requirements. The user can then add the desired module from the list of suggestions.

Selecting the **Build** option box exits the screen making the new model the current model. If changes are made to an existing model the user will be asked if they wish to save their existing parameters to a *.par or *.dbx file for reloading by the user after the the modified model is built. N.B. It is preferable to use *.par files as *.dbx files are limited to 10 character parameter names.

The **Cancel** option box exits the screen without making any changes to the model.

Creating a Model from scratch.

Run CRHM and do not load any project or modules.

1. Set # of HRUs to the desired value.
2. Load module *basin* if it is desired to give HRUs text names.
3. Load other modules.
4. Build model - no need to save parameters.
5. Set parameters to desired values.

6. Save model to a project file.

Having observation files open does not affect the build procedures but the file names will be saved with the project.

Parameters or Coefficients.

The Parameters feature from the CRHM pull down menu allows the user to incorporate the physical and spatial data requirements of the modules . The basin/HRU spatial data can be manually inserted into the model or imported as a file with a (.par) or (.dbf) extension. When the model is saved as a project all the parameters are saved with the model in the project file.

The **Parameters** window displays on the left hand side a list of all available modules. When a module is selected from the **Modules** box, the spreadsheet displays the number of **HRU**s, **parameter type** and **parameter value**. The **Maximum** and **Minimum** value of each parameter is also displayed. It is in this spreadsheet that the parameter value can be highlighted and altered. When the parameter value is selected, the extended description of each parameter is given along with its units. This is visible on the Parameters window on the lower left hand corner to the right of **Help** and **Units**.

Basin Parameters are parameters that are shared between modules using identical values. The program when originally loading modules into the model determines which parameters are shared and have the same values for all HRUs and converts them into basin parameters. To make parameters local to a module go to the parameter window and select the module, then click on 'basin' opposite the parameter it is desired to make local. If these local parameters are ever set equal to the basin parameter of the same name, they will resort back to the basin parameter but only after the model is saved and the project is reloaded. Basin parameters are displayed at the bottom of the spreadsheet flagged as 'basin'. Basin parameters are edited by clicking on the module 'basin'. Changes apply to all modules using the basin parameters.

Once established, the parameters for the model can be saved with the **File** pull down menu. The File pull down **Open** option allows parameter (**.par**) files and (*.dbf) files to be imported.

Handling Parameters when Adding/Removing Modules.

Every time a model is 'Built' the parameters in **all** modules revert back to the default values programmed into the individual modules. To overcome this CRHM queries the user if they want to save the current parameters into a

parameter file (.par). After the model is 'Built' immediately re-load using the *Parameter* screen the original parameters saved in the *.par file. The original model parameters will be loaded if the number of dimension remains the same or is reduced. However, if the number of dimensions is increased the extra dimension of parameters will be set to the values of the last HRU defined in the parameter file. Any new modules added will have the module default parameter values

Parameter Setting Precedence.

When a project is loaded the parameter values are assigned values using the following sequence.

1. Module parameters are given the default parameter values defined in the original module code.
2. The 'basin' parameters values defined in the project file are given to all parameters of that name (e.g. "hru_area"). Applies to every module or group using the specified parameter.
3. Parameters specified by module and group are set to the values given in the project file.

Converting Module parameters to Basin parameters.

CRHM was written to allow individual module parameter values to be different from the global, i.e. *basin* values for test purposes. To return a module parameters to global parameters, e.g. 'evap - Ht' to 'basin - HT', all the heights must be made identical and the project saved and then the model reloaded from the project file.

Creating a Model from a DataBase File (.dbf).

A new model can be created using parameters created by a GIS. The ArcView® database file (*.dbf) contains the parameters (fields) for all the HRUs (records) in the model. The database file is loaded from the *Parameter* screen using the file menu before any modules are loaded. The # of HRUs is automatically set to the number of records in the database file and a basin module is created. The user can then proceed to the *Build* screen and select the desired modules for the model. When the new model is being built, CRHM asks

if the current parameters should be saved. Reply *YES* and supply a file name (.par). After the model is built return to the *Parameter* screen and load the parameters from the temporary file (.par) that you saved them in.

Incompatible Parameter Names.

CRHM parameter names use upper and lower characters and it appears that the ArcView® database file (*.dbf) must be uppercase only. To manage this problem every parameter name when raised to all uppercase must be unique. For example, hru_GSL and hru_gsl are not unique and represent the same parameter. The actual parameter name appearing in the model is that declared in the first module loaded in which the parameter is used. The following names in the first column are the ArcView® database file (*.dbf) names and the second column gives the CRHM model names as determined from the CRHM module 'basin'.

BASIN_AREA basin_area

HRU_AREA hru_area

HRU_ELEV hru_elev

HRU_LAT hru_lat

HRU_ASL hru_GSL

HRU_ASL hru_ASL

State.

The **State** feature from the CRHM pull down menu allows the final state of a model run to be saved. This final state can then be used as the initial state of a sequential model run.

A good example of the value of this feature is if snowmelt is being examined using different parameters. Instead of always having to run the model from the previous fall every time the model is executed, save the final State of the fall to March run. Then the model need only be Run from April to the end of melt using the final State of the earlier Run as the initial State for the April Run.

Notes.

When an intial state file is used the state values are loaded into the CHRM model variables after the Module::init routines have been executed but before the Module::run rountines are executed for the first time in the model run. This ensures that variables set by decisions made at earlier times are maintained and are not dependent upon the Module::init routines handling past events correctly.

The State of the model is assumed to be described fully by the model state variables.

Saving the final state of a model run to a file happens only once, i.e. after the final state is saved to the specified file, the file name must be re-entered before it will be written to again. When the file to save the final state is specified, it is not displayed anywhere. On the other hand the initial state file name is remembered and displayed in the State pull-down menu and used every model run until cancelled by clicking on the file name.

Select Output Variables.

This is done by **left clicking** on the desired output in the Variables ListBox and then **right click**ing and choosing from the drop down menu from **Add/ HRUsAdd/ LaysAdd/ HRUsAddLaysAdd** . The selection will appear in the **Selected** box, located immediately to the right of the Variables ListBox. The number in brackets is the HRU/LAY selected. If the requires the same output for a different HRU use the up/down arrows on the **HRU** box located immediately below the Variables box. If the HRUsADD or LAYsADD selection is made then the HRUs or LAYs are added from the selected HRU/LAY to the maximum possible value.

The variables ListBox has several ways of listing variables. The default one is **Variables by Module**. By clicking on the title the display may be cycled through **Variables by Module/ Diagnostic Variables/ Private Variables/ All Varables**.

Modifying selected variables.

First select a variable from the selected variables ListBox. Then right click and choose from the following choices **Delete/ Negate/ Abs/ AddObsFunct**. The first three choices are straightforward. The last one applies the previously selected observation function to the selected variable and displays the compound name in the Observation selected ListBox. The trace will not appear until values for the variable are available.

Run

CRHM is executed by selecting **Run** from the CRHM pull-down menu. Run will not execute the model unless at least one model output has been selected from the **Variables** box on the upper left hand side of the CRHM window.

You can now watch the **Graphical Output** window as CRHM executes the model and updates the output. The status display box on the lower left hand corner of the CRHM window will show you the progress of CRHM outputting the data.

The model run may be interrupted by right clicking on the plot area. A menu will appear allowing the user to select an action from,

1) continue,

2) terminate the run immediately,

3) daily update,

4) bi-weekly update,

5) weekly update,

6) monthl updatey,

7) yearly update and

8) update at end of run.

When 'update at end of run' is selected the user must wait until the run is finished to regain control. The date is not updated.

The project execution time is consderably reduced by limiting the chart update frequency.

Modifying Graphical Display.

The data displayed on the chart is controlled by the contents of the Variable and Observation Selected ListBoxes. The observation traces will appear immediately after their selection. The variable and variable function traces will be drawn as soon as their data is available.

Flip-Tics.

This a convenient way of changing the traces displayed on the chart. Left clicking on the Flip-Tics button will compliment the variable traces and right clicking will compliment the observation traces. Where compliment means turning the traces which were ON to OFF and those that were OFF to ON. Using the flip-Tics button while the Ctrl key is depressed will toggle all the variable or observation traces between ON and OFF. Flip-Tics actions apply to all traces even if they do not show in the legend.

Number of Traces Exceed the Legend Size.

There is no ScrollBar on the legend so traces not displayed in the legend cannot be individually turned ON and OFF. However, there is a comprehensive editing feature in the TChart editor. This feature may be entered by clicking on the ICON displaying a pencil/square/rule in the top left hand corner of the chart.

Export.

The Export feature from the CRHM pull down menu allows you to examine your output as a list in a window and to write it to a file in a variety of formats. The **Excel** choice can be easily imported to Microsoft Excel. The Observation (YY/MM/DD hh:mm) and Excel formats allow the file to be used as input to CRHM. The other options display in various formats of MM/DD/YY hh:mm and Julian day.

When previewing, only 500 lines are displayed at a time. To preview the next 500 lines click the preview button again.

Methods of Displaying Data.

Description	extension	frequency
Observations	none	interval
Watts to MJ/int	_WtoMJ	interval
MJ/int to Watts	_MJtoW	interval
Average	_Avg	daily
Minimum	_Min	daily
Maximum	_Max	daily
Daily sum	_Sum	daily
Positive	_Pos	daily
Total	_Tot	daily
Total/Freq	_Tot/Freq	daily
First	_First	daily
Last	_Last	daily

Notes.

- The fundamental output time step is given in the table above.
- The extension indicates the method used to output the data. The basic observation has no extension.
- The first method used determines the interval used in the output.

- If the first output is interval then any following daily values are repeated every interval of the day.
- If the first output is daily then any following interval values output only the first interval value of the day.
- When the output is displayed on the screen the heading gives the number of lines of data that is available to display.

Analysis

The **Analysis** feature from the CRHM pull down menu allows the user to perform statistical analysis on the output data. There are several Analysis options that will not be examined in this exercise but will be covered in detail in Chapter 7. Upon selecting Analysis for the CRHM pull down menu you will see **evap(1)** and **evap(2)** in the upper left box of the **Analysis Form** window.

DO: select

1. Analysis
2. Right toggle arrow in lower centre of Analysis Form window until Order 5 is displayed
3. Left click evap(1)

You should see the largest box in the centre of the Analysis Form window scroll through the evap(1) output data. CRHM will create a polynomial at the top of the same window and create a graph in the box to the right showing the data in red complete with regression curve displayed in green.

You will see several more options on the Analysis Form window that will be discussed further in Chapter 7.

DO: select

1. File
2. Exit

Log

The **L**og feature from the CRHM pull down menu displays the log screen consisting of two scrolling windows. The top window displays information about the model build. Entries consist of *ERRORS* and *WARNINGS*. Any errors are fatal and the model run will terminate. Before proceeding any further the user will have to rectify the problems. An example of an *ERROR* is when a required observation is missing from the observation file(s). An example of a warning is when the model informs the user that the model is using *ppt* (the daily precipitation divided by the number of intervals/day) and not the actual interval precipitation *p* .

The lower window is used to display information from the module. Its use is determined by the module designer. Possible applications might be to inform the user if a variable has a value outside its normal range or the model is proceeding without meeting desired criteria. Another use is for debug output during module development.

AKA Screen.

The AKA function was used in the development of CRHM to provide features before they could be implemented properly. It is not supported and should never be used.

The **AKA** menu provides an overview to all the inputs and outputs of a CRHM model. It has two major views,

1. Variables - shows all module output and input variables.
2. Observations - displays all observations pertaining to the model. Observations are denoted as *simple* or *declared* (terminated with a '#' SYMBOL). The former are from original meteorological data and the latter are module generated observations, usually derived from the former, i.e. from meteorological data.

The main purpose of this screen is to control the flow of observations within a model. An important secondary use is to allow the user to rename simple observations.

Screen Components.

The centre of the screen is a spreadsheet. The two columns on the right hand side lists the outputs and their sources. These are available for inputs to other modules. The source can be a module or 'observation' if a simple observation. Along the top of the spreadsheet the modules and their inputs are listed and underneath the input is listed again opposite its source.

Radio buttons.

By selecting one of the two mutually exclusive buttons, the user can choose the Variables or Observations display.

Script.

This text pad displays the script instructions to CRHM that control the model. It is largely intended to be used as an informational screen. The file menu

allows the script to be saved and restored to alter another model without having to re-type the instructions. The scripts for the Variable/Observations are handled separately and are compiled into the current model using the 'save to model' button.

ListBox.

Lists all the modules in the current model.

File menu.

It allows the script to be saved in a file (*.aka). A previously saved script may be loaded and applied to the current screen.

The usual file options are available; ***Open***, ***Save***, ***Save As*** and ***Exit***.

'Save to Model'

Saves the changes to the current screen script to the model.

'Void Changes'

Will remove any current changes up to the last 'Save to Model' for this screen. To remove all changes use 'Escape'. However this will remove all changes in all screens since the AKA screen was loaded.

'Remove Unused'

Removes all unused observations or variables.

'Escape'

Exits AKA screen restoring all the model scripts to what they were before the AKA screen was executed.

N.B. The normal way of exiting the AKA screen is via 'Exit' in the top right hand corner or in the file menu. This will save any changes made.

Editing Screens.

Rename.

Allows simple observations to be renamed to what is available in the open observation file(s). It does not check availability till run time. Display cell "u", will change to e.g. "[u -> UUU]" if the observation "u" has been renamed "UUU". The observation "UUU" must exist when the model is run.

Connect and Re-connect.

This operation allows the user to change module inputs. This feature is executed by first selecting the cell beside the desired new name (column 2, it will be empty if not a renamed simple observation). Next, right click on the module input to be changed. The cell will now contain the new name. Left clicking returns to the original value. Display cell "QdroDext" will change to "[QdroDext -> QdroFlatD]" if the input "QdroDext" is redirected to "QdroFlatD". When the model is run the normal input "QdroDext" will use the "QdroFlatD" values. An observation example is "Qsi" becoming "[Qsi -> QsiA#]". Instead of the simple observation "Qsi" being used the calculated "QsiA#" from the module "Annandale" will be used. In the examples "--> QsiA#" and "--> QdroFlatD" will appear opposite the source of the variables to indicate to the user it is being used.

FlowDiagram.

The **FlowDiagram** feature of the CRHM pull down menu generates a flow diagram for the current model. The primary function of the display is to show the progression of variables through the model modules. An additional feature of the flow diagram is the display of either module input Observations or module input Parameters to illustrate the driving inputs of the module or the coefficients controlling the module.

Module generated Observations.

File Observations names are always shown on the left hand side of the screen. Module observations outputs are displayed on the right hand side of the module generating them followed by a # symbol, e.g. **Qli#**. Note that file observations take precedence over module generated observations.

Puts.

The flow diagram is very useful to reveal the effect of *Put*s. Normally the value of a variable is only set/changed by the module that declares it. However, sometimes other modules have to update the variable value during the timestep. Understanding the sequence of events during the timestep is important. Put inputs are flagged by the letter 'P' beside the input.

File menu.

This menu allows the flow diagram to be saved to a file using Save and SaveAs options. The print selection prints the flow diagram. The Printer Setup configures the printer. The Portrait/Landscape selection would be the most frequently used.

Copy menu.

This saves the flow diagram to the clipboard. There are two choices Bitmap and Metafile. User should test each format with their import application. Results are dependent on applications used.

Selection menu.

This is a toggle control which switches the secondary input from Observations to Parameters. The variable flow pattern is always displayed.

Notes.

The flow diagram is scaled to fit one printer page.

Functions and Summary.

The CRHM program has an extensive capability for examining functions of model input observations and output variables.

Timebases.

Select the operating time period for the function. Note that "Interval" is not applicable to functions.

1) Interval - CRHM execution time step determined by the first observation file loaded. Varies from 15 minutes to 24 hours.

2) Daily - the value is calculated for the day.

3) Monthly - the value is calculated for the month.

4) Calendar Year - the value is calculated for the calendar year.

5) Water Year - the output is calculated over the water year defined by the month.

6) All - the value is calculated for the entire run length.

The timebase is set by the user using the "SELECTION BOX and DISPLAY" situated above the "Start Date" label on the main screen by LEFT clicking on the box.

If Water Year is choosen the starting month of the year is selected by RIGHT clicking on the box to cycle through the months of the year.

The functions available are;

1) Average - mean value of the interval values over the timebase period.

2) Minimum - minimum value of the interval values over the timebase period.

3) Maximum - maximum value of the interval values over the timebase

period.

4) Total - sum of the interval values over the specified timebase period.

5) Positive - sum of positive interval values over the timebase period.

6) First - first interval value of the timebase period.

7) Last - last interval value of the timebase period.

8) Count - the time (days) that the interval values are greater than zero over the specified timebase period.

9) Count0 - the time (days) that the interval values are equal to zero over the specified timebase period.

10) Delta - change over the timebase period. Calculated as (Last interval of the current timebase period - Last interval of the preceding timebase period). For the very first time period of a run, the First value of the first time period is used instead of the Last interval of the preceding timebase period as this value may not always be available.

Use "Close" to return to the main screen when finished.

Miscellaneous Interval Functions;

These interval functions are always available and are independent of the selected timebase period.

1) Observation - interval values as read from the observation file.

2) VP_saturated - interval saturated vapour pressure calculated from the interval temperature value.

3) Watts to MJ/Int - unit conversion Watts to MJ/interval.

4) MJ/Int to Watts - unit conversion MJ/interval. to Watts.

Use "Close" to return to the main screen when finished.

Application to Observations.

1) To display an observation, select an observation and RIGHT CLICK on it to bring up the **Add/AddArray/Function** menu and select Add. The observation will be displayed on the graphic screen using the function displayed under the Observation listbox. By default this is "Observation" initially.

2) To change the function type, select any observation and RIGHT CLICK on it to bring up the Add/AddArray/Function menu and select "Function". Select the desired function from the pop-up menu. Alternatively, CLICK on the current function selected to sequentially step through the functions.

3) If an observation has multiple dimensions they may all be displayed at once by selecting AddArray. N.B. if OBS or LAY display is greater than 1, only the values from that array element to the last available will be displayed.

Application to Variables Selected ListBox.

The functions may also be applied to the CRHM model output Variables. The process is similar to that outlined above.

1) To display a Variable, select a variable and RIGHT CLICK on it to bring up the **Delete/Negate/Abs/AddObsFunct** menu and select AddObsFunct. If the project has already been run, the selected function of the Variable will be displayed immediately otherwise it will only be displayed after the project is run. Every time the model is run the variable functions will be refreshed.

2) Changing the function type is the same as for observations, Select any observation and RIGHT CLICK on it to bring up the Add/AddArray/Function menu and select Function. Select the desired function from the pop-up menu.

Display Mode.

The output displayed on the chart is set as follows,

a) "Display off" is used when the user is sending the output to the "Summary" file at the end of the run and does not wish the function values to be displayed on the screen during the run. This reduces reduces execution time and screen clutter. The tracees can be enabled at the end of the model run. The values will be written to the "Summary" file if selected (parameter "basin RUN_ID" > zero).

b) "Display Final Value" is used to display the end value of a timebase period over the entire timebase period. The values will be written to the "Summary" file if selected (parameter "basin RUN_ID" > zero).

c) "Display "Trend Value" is used to display the function values over a timebase period as they are processed.The values will be written to the "Summary" file if selected (parameter "basin RUN_ID" > zero).

Use "Close" to return to the main screen when finished.

Writing Summary to a file.

Functions derived from variables and observations are written to the summary file. However, basic variables and basic observations can be written to a file using the export capability.

The name of the summary file by default is "CRHM_summary.sum". However, if the project uses the module "basin" and the parameter "RUN_ID" is greater than zero, its value will be appended to the file name, e.g. "CRHM_summary_1.sum".

The summary file is only written at the end of running a project. However, the graphics screen is updated immediately when a function variable is added if the input data used is already available.

Different Displays of Daily Totals.

When displaying daily values they can be displayed on the actual day and some times with a 24 hour delay. This is normal as observation like t, RH, u, and Qsi are applied on that day. However, when CRHM is using daily simulations the variables such as runoff, soil moisture and melt are used as input to the next process on the following day.

Help

Microsoft HTML Help is used in the program. The help Topics are stored in html files with the extension (.htm). The HTML Help Project Editor is used to compile a *.CHM help file. The help system may be extended by the end user using compatible programs. The help information is accessible by;

- selecting a Help topic from the CRHM Help menu. Then using the table of contents to navigate through the various topics.
- using *Context* help by holding the cursor over a program window component and pressing F1.

About

- Gives the version of the program.

Observations

CRHM requires as input the climate or meteorological data (which the model refers to as observations) for the watershed being modeled. A program module (e.g. **obs**) reads the climate or meteorological data from the data file into the model. The model is capable of making use of data from one or more stations. If more than one source of climate or meteorological data is available for an area (watershed) the model is capable of using a specified station set of observations (t, rh, u, p, ppt, Qsi) for each HRU.

The usual modeling time step is a **half hour** or **one hour**. Daily data starts at 00:15, 00:30 or 01:00 and finishes at 24:00 or 00:00 of the next day. Gaps in the data must be filled with synthesized data.

Observation data is read from **ASCII, space or tab delimited file**s. An example file is shown in Example 1. Other data formats can be converted by the user using Excel and/or a text editor. Data sets do not have to be merged into one file as the model will access data from multiple observation files. The file extension must be **.obs**.

Example 1.

Observation data for Bad Lake, Saskatchewan January 01 1973 to December 31 1973

t 1 (°C) 2 meter

rh 1 (%)

u 1 (m/s) 10 meter

SunAct 1 (h) sunshine hours

ppt 1 (mm/d) precipitation mm

form_data 1 ()

Qsi 1 (W/m^2)

Qso 1 (W/m^2)

Qn 1 (W/m^2)

##

1973 1 1 0 0 0 -15.10 82.00 5.30 0.00 0.00 0.00 0.00 0.00 -0.11

1973 1 1 1 0 0 -15.50 81.50 3.10 0.00 0.00 0.00 0.00 0.00 -0.09

1973 1 1 2 0 0 -15.60 81.30 1.40 0.00 0.00 0.00 0.00 0.00 -0.09

1973 1 1 3 0 0 -15.50 81.30 3.10 0.00 0.00 0.00 0.00 0.00 -0.08

1973 1 1 4 0 0 -15.30 81.30 1.40 0.00 0.00 0.00 4.50 0.24 0.00 0.00 -0.07

Line 1 is a file description, usually defining where the data is from and any other pertinent information. Starting with line 2, the observations are listed in separate lines. Each entry consists of a variable name, followed by the dimension of the observation. If the observation is an array or profile the number will be greater than one. Next the units of the observation is given enclosed in parentheses, Other information about the observation, such as instrument height, can also be entered as a comment at the end of the line. This comment and will appear in the CRHM observation help box. If the units are not enclosed in parentheses they become part of the comment.

The line containing the string of pound signs #### (minimum of 4 required) defines the end of the header and the beginning of the the data observations. The data is space or tab delimited and starts with year, month, day, hour, minute and then the observations data is listed in the same order as specified in the header. The field delimiter is one or more spaces or a tab.

An alternative to "year, month, day, hour, minute" is decimal time and date used by Microsoft in Excel and other applications. It is much easier to generate observation files using this format. For example, "1998 5 1 10" would be replaced by "35936.875". The two time formats cannot be mixed within an observation file.

It is very important to have no "hanging" tabs at the end of lines or extra record terminators or tabs at the end of observation files.

Notes.

CRHM checks the units of the observation with what is required by the model module and if different reports it to the "log" screen.

HRU/Observation indexing.

The simplest CRHM model has one set of observations. That is one value of each of the following observations; t, rh, u, and p (or ppt) for every timestep of the model run This is very limiting when the area being modelled is large and diverse in elevation and exposure. CRHM is capable of indexing each HRU to a different observation element using the parameter HRU_OBS. This allows every HRU to use a different set of observations. To further expand the flexibility, the observations are divided into the five groups described below.

The simplest would be if the HRU and the observation were aligned in sequence.

HRU	1	2	3	4, ...
Observation	1	2	3	4

The following illustrates a random arrangement.

HRU	1	2	3	4, ...
Observation	1	7	5	2

If the indexing table requests a value that does not exist, the highest value defined is used. In the above, if there are only 4 elements in the temperature dataset, the HRUs requesting the 5th and 7th array would be truncated to 4. The highest possible element index.

Five HRU/Observation arrays.

The observations are grouped into five categories according to how the field data is collected or the probable database source.

1. t, rh, ea.
2. u.
3. p and ppt.
4. Q or other radiation components.

5. miscellaneous category to be used as required.

HRU	1	2	3	4, ...
Observations t, rh, ea.[1]	1	7	5	2
Observation u. [2]	1	2	2	1
Observation p, ppt. [3]	1	1	1	1
Observation Q. [4]	2	2	1	1
Observation misc. [5]	1	1	1	1

It should be noted that for this system to be most useful, the observations for the five categories should be in five separate files. This is optional as long as the observations are addressed accordingly.

Observation File Preparation.

Data observation files are sequential ASCII text files with a time field as the first field on every line followed by the observation data fields. There is a line (record) in the file for every data interval. CRHM files are similar with the addition of a data header which defines the observation names, the order in which the observations occur on every line and finally giving help information describing the file and the observations.

Observation file layout.

Header.

1. The first line of the file can be used to describe the file. It must be present and cannot be more than one line.
2. Data definition. Each line consists of a variable name, dimension of the variable, variable units in the CRHM format and lastly an optional comment. The delimiter is one or more spaces.
3. Filter definitions. These always have a '$' in column one and must follow all data definitions.
4. "#####" - flag indicating the end of the header. Only 4 '#' symbols are required. The remainder of the line is ignored.

No extra lines are allowed, including null lines. All lines must start in column one.

Comment lines are allowed in the header after the first line. Use "//" or "$$" in column 1 and 2. Comment lines are only allowed in the header portion of the file.

Data.

- The data must always begin one interval into the day and end at midnight of the last day.
- The time format can either be "yyyy mm dd hh mm" or MS decimal time with at least 5 decimal places, e.g. "27181.04167".
- The delimiter can be space or tab. The tab is preferable for data inspection

since the columns can be made to line up.
- No comments or null lines are permitted.
- Any missing data must be filled with a numeric flag, e.g. "9999" which can be massaged using a filter.

UNIX files.

UNIX observation files cause CRHM to fault. These files should be copied and pasted into a new text file created by TextPad or other PC text editor.

Assembling Data.

Excel® is a convenient program for assembling the field data files into one file covering the desired period. It has the capability of importing and merging data files and cutting and pasting individual columns simplifying data editing. The observations should be assembled in columns. The first column should be time. Beginning from the first interval of the first day and should end at midnight of the last day of the period.

The time field can be created using *=DATEVALUE("02/17/03") + TIMEVALUE("0:30")* to assign a time to a cell. If two cells are defined in succession then the *FILL*/*TREND* function can be used to fill a 'blocked' column with successive times for the desired period. The month/day/year order entering dates is determined by the regional settings of the PC. A #VALUE! error or an unexpected date will occur when the incorrect order is used.

It is recommended that at this stage all cells outside the required area of the spreadsheet be deleted otherwise they will also be exported from the spreadsheet together with the desired data causing needless problems.

The time and date column must be formatted to display as decimal time. Select the date and time column which normally would be *column A*. Select *Format cells...* and then select *Number*. Set the number of decimal places to 5. The spreadsheet should display the date and time as a fractional number in the 37,000 range.

Exporting data from Excel.

After the data is assembled, save the file as an Excel file for future use. Select

File/SaveAs.. then select the desired folder. Click in the *Save As Type:* box and select *Text(Tab delimited)*. Enter the desired file name and select *Save*. At this point it is best to close the file after saving the text file as the Excel prompts can be misleading.

Handling time in Excel.

Quite often the format of time downloaded from data loggers is ugly and difficult to put into the proper format. Often it is easier to generate the time column from scratch using an Excel functions. The method described above is best. Another method is as follows.

1. Insert the date function in a cell, i.e. date(year, month, day). Say in cell K4.
2. Insert in the first row and the first column of the datalogger data "K4 + 1.0/F" where K4 is the location of the cell holding the date function and F is the daily frequency of the data. Say this is cell K5
3. Repeat for the remainder of the column "$K5 + 1.0/F" in cell K6, "$K6 + 1.0/F" in cell K7, etc.
4. Format the column as "Number" with 5 "Decimal places".
5. This column should be exported as the first column in the CRHM observation file.

Creating CRHM obs file.

The text editor, TextPad is recommended for this step. NotePad is limited as it is unable to display formatting characters like tabs. TextPad is able to do column cuts and pastes.

Open the *.txt file saved by Excel. Turn on *visible formatting* by clicking the ¶ icon. Check that there are no extra fields at the end of lines or at the end of the file. If the columns are ragged increase the tab size in the *Configure/Preferences.../Tab/Tab size* menu.

At the beginning of the file insert a comment line giving a the file description. Next insert a line for each data field consisting of the name to be used by CRHM followed by the number of data items in the field. This is usually one. The remainder of the line can be used as a descriptive comment. An example line is "t 1 air temperature at 1m.". Field names cannot have embedded spaces. One or

more spaces separates the fields. The comment can have embedded spaces.

Every column of data after the time field has to be accounted for by the data definitions in the header section. The end of the header is marked by a line containing four or more pound signs, e.g. "#####".

If observation filters are used they are inserted between the last data definition and the line containing the pound signs. Observation filters are modifiers which allow the input observation values to be modified before being used by CRHM. Examples are changing units either using a dedicated function ("$Tc FtoC(Tf)") or using one or more arithmetical filters ("$a add(var, c)", "$a sub(var, c)", "$a mul(var, c)" and "$a add(div, c)"). Other uses are changing the reference height of measurements ("$u2 refwind(u, Z2, Zm, Ht)"), replacing error flagged data with *good* values ("$d missing(var, c1, c2)", "$d missinginter(var, c1, c2)" and "$d missing0(var, c1, c2)") and for distributing daily precipitation over every interval of the day ("$p smear(p, 0, 0)" or "$var expand(var, c_freq)").

It is best to test the observation files separately in CRHM before loading them into a project. A common problem is missing intervals, i.e. if a buffer overflow has occurred with a data logger a block of data will be missing. This will be indicated by CRHM detecting a *sparse* file.

Time Simulation (Can only be used in a separate *.obs file containing no real data).

Sometimes it is convenient to generate synthetic data instead of field obervations. A special filter makes this possible.

$Sim(StartTime, EndTime, Interval) where

StartTime as mm/dd/yy or mm/dd/yyyy

EndTime as mm/dd/yy or mm/dd/yyyy

Interval in hours. Minimum 0.5

When this filter is put in an observation file, no interval data is ever read but time periods from 'StartTime' to 'EndTime, are generated at the interval

specified. The time simulation filter requires the addition 0f filters which generate actual data values e.g. "const(C)".

Wave Synthesis (Can only be used in a separate *.obs file with $Sim(...) and not in a regular observation file).

$Fract sine(period phase start end)

$Fract square(period phase start end)

$Fract ramp(period phase start end)

$Value exp(start end B) Value = $e^{B(t - t0)}$

$Value log(start end B) Value = ln(B*(t - t0))

$Value pow(start end B) Value = $(t - t0)^B$

$Value poly(start end a0 a1 a2 a3 a4) where X = (t - t0), Value = a0 + a1*X + $a2*X^2 + a3*X^3 + a4*X^4$

$Fract pulse(start end), where

period is in days. For less than one day the time format can be used, e.g. 12 hours can be '0.5' or '12:00:00'.

phase is in days. For less than one day the time format can be used, e.g. 12 hours can be '0.5' or '12:00:00'.

start is start date (mm/dd/yy) or (mm/dd/yy_hh:mm:00)

end is end date (mm/dd/yy or (mm/dd/yy_hh:mm:00))

Fract will be a timeseries varying between -1.0 and 1.0 determined by the function.

Value is the timeseries calculated by the function.

Example of Wave Synthesis.

```
Simulation test (06/10/02)

$$ comment line - Test pulse generation

$Sim(01/01/01, 01/05/2001, 1) simulation time period

$Fract pulse(01/02/01, 01/03/01) one day pulse

$P pulse(01/01/01_1:0:0, 01/01/01_12:0:0)

$Sine sine(12:0:0, 0, 01/01/01, 01/02/01)

$Ramp ramp(1, 0, 01/01/01, 01/02/01)

$Square square(1, 0, 01/01/01, 01/02/01)

$$ comment line - delayed functions

$P1 pulse(01/03/01, 01/04/01)

$S1 sine(1, 0, 01/03/01, 01/04/01)

$R1 ramp(1, 0, 01/03/01, 01/04/01)

$Q1 square(1, 0, 01/03/01, 01/04/01)

##########################################
```

Using TextPad to create observation files without using Excel.

Since TextPad is able to cut and paste columns the only problem is adding the time field for CRHM into the file. Fortunately, the time field can be generated by using the filter "`$Sim(01/01/01, 01/05/2001, 1)`" and merging the time field column into the data file.

4.1.2 Parameters

The model requires a parameter file (**.par**) to specify the spatial parameters for each HRU and the basin. For complex watersheds the easiest way to generate the .par file is with a **GIS interface**. A digital elevation model (DEM) and layers of ancillary data (soils, vegetation, etc.) are used to create a file like the one shown in Example 2. However, at the current stage of the model's development the actual spatial representation of the HRUs was considered of lesser importantance, therefore the number of HRUs has been intentionally limited. As well, the GIS interface is not fully operational and spatial parameters are presently input into the model manually.

Parameters are required for the basin and the individual HRUs. It may be that GIS layers are available for some parameters or there may only be a single value available for the entire basin. This should be clearly defined and discussed between the user and modeller.

Sample Parameter file - Example 2.

Description - to be added

######

basin basin_area 5

basin basin_name 'CRHM Basin Model'

basin hru_area 1 1 1 1 1

basin hru_ASL 0 0 0 0 0

basin hru_elev 637 637 637 637 637

basin hru_GSL 0 0 0 0 0

basin hru_lat 51.32 51.32 51.32 51.32 51.32

basin hru_names 'HRU' 'HRU2' 'HRU3' 'HRU4' 'HRU5'

crack fallstat 50 50 50 50 50

crack Major 5 5 5 5 5

ebsm delay_melt 0 0 0 0 0

ebsm nfactor 0 0 0 0 0

ebsm tfactor 2 2 2 2 2

evap evap_type 0 0 0 0 0

evap Ht 0.1 0.2 0.3 0.4 1

evap Zref 1.5 1.5 1.5 1.5 1.5

evap Zwind 10 10 10 10 10

global Time_Offset 0 0 0 0 0

netall hru_alb 0.17 0.17 0.17 0.17 0.17

net_rn F_Qg 0.2 0.2 0.2 0.2 0.2

net_rn F_Qs 0 0 0 0 0

obs catchadjust 0 0 0 0 0

obs tmax_allrain 0 0 0 0 0

obs tmax_allsnow 0 0 0 0 0

pbsm distrib 1 1 1 1 1

pbsm fetch 1000 1000 1000 1000 1000

pbsm Ht 0.1 0.2 0.3 0.4 1

route Kstorage 0 0 0 0 0

route Lag 0 0 0 0 0

route order 1 2 3 4 5

route whereto 0 0 0 0 0

smbal cov_type 3 3 3 3 3

smbal soil2gw_max 0 0 0 0 0

smbal soil_moist_init 187 187 187 187 187

smbal soil_moist_max 375 375 375 375 375

smbal soil_rechr_init 30 30 30 30 30

smbal soil_rechr_max 60 60 60 60 60

smbal soil_type 2 2 2 2 2

######

Sample data preparation table.

Table 1 shows a list of climate observations and spatial parameter requirements. The data provider should provide as much of this information as possible and submit a completed copy of the table to the user. If the data is available in units other than those stipulated in Table 1, the model can do the necessary conversions.

Table 1

DATA	**Units**	**Measurement ht.**	**Time Step**	**Time Period**	**Data Type/Source**	**Com**
Observations						
temperature	degrees					
relative humidity	percent					
wind speed	m/s					
wind direction	degrees					
actual sunshine	hours					
incoming radiation flux	W/m2					
reflected radiation flux	W/m2					

net radiation flux	W/m2					
depth of precipitation	mm					
precipitation form						unkn
albedo						
discharge rate	m3/s					
soil heat flux	W/m2					
Parameters Basin/HRU						
area	hectares					
slope	degrees					
aspect	degrees					
vegetation cover type						plus
soil type						plus
elevation	meters					
latitude	degrees					
vegetation cover height	meters					
vegetation cover density	percent					sumr
interception storage capacity	mm					
leaf area index	m2/m2					
fetch distance	Meters`					
soil properties						textu

Observation Filter

Observation filters allow the observation data to be preprocessed before being used in the model. Filters are declared in the observation heading after the declaration of variables. Examples of their use follows:

- relative humidity is often the meteorological observation, whereas vapour pressure is the common input to model modules. One solution is to make the individual modules handle the change of variable but this makes the modules more complex and slower. The conversion has also to be done in every module requiring vapour pressure. A more efficient solution is to make the data conversion once when the data is initially read into the model directly from the input data file.
- If wind measurements are made at 10 metres and the model modules require a wind referenced at 2 metres. One solution is again to make the modules handle the conversion of measurement height but this necessitates defining the relevant translation parameters in every module.

Filters

The filters are included after the data variables in the file are defined but before the line of '#' symbols separating the data header from the actual observation data. Filter lines begin with a '$' followed by the variable name for the generated data, i.e. $ea.

The data filtering filters; "missing", "missinginter", "missingrepl" and "missingFlagAfter" are best used separately and the results saved and used as a new observation file. This limitation arises because all the data must be read to determine the last "good value" and other filters may used data values before missing values are corrected.

The filter name follows with the required parameters enclosed in brackets. The parameter list consists of variables included in the observation file or variables generated by earlier filters and the numerical constants used by the filter. An example of a filter to modify wind reference height is:

$u2 refwind(u, 10, 2, 1) "this is a comment", where parameters for refwind are refwind(u, Zm, Z2, Ht)

where:

u2 = u1*log((Z2 - d)/Z)/log((Zm - d)/Z)

u2 (m/s) is the name of the new variable,

Zm (m) - the actual measurement height = 10,

Z2 (m) - desired reference height = 2,

Ht (m) the vegetation height = 1, and it assumed that

d = 2/3*Ht and

Z = 0.123*Ht.

Both spaces or commas can be used to delimit parameters. Any input after the closing bracket is a comment and will appear in the program variable help box.

Similarly the filter for vapour pressure is:

$ea ea(t, rh) where t is the temperature observation name and rh (%) is the relative humidity observation name.

where:

ea = sat _ea(t)*rh/100.0,

t is the temperature measurement and

rh (%) is the relative humidity.

Defined filters

$$ comment line.

$u2 refwind(u, Z2, Zm, Ht) u = wind at reference height Zm, u2 = desired wind at reference height Z2 and Ht = vegetation height.

$ea ea(t, rh) ea = desired vapour pressure (kPa) for t = temperature and rh (%) = the relative humidity.

$rh rh(t, ea) rh = desired relative humidity (%) for t = temperature and ea (kPa) = the vapour pressure.

$RHi RH_WtoI(t, rh) RHi = RH w.r.t. ice of moist air at ambient temperature t, rh = RH w.r.t. water.

$Tc FtoC(Tf) Tc = conversion of Tf(°F) to Tc(°C).

$Tc KtoC(Tk) Tc = conversion of Tk(°K) to Tc(°C).

$Tk CtoK(Tk) Tk = conversion of Tc(°C) to Tk(°K).

$d missingC(var, c1, c2, c3), where **var** is the data being checked. Values less than or equal c1 or greater than or equal c2 are replaced with c3.

$d missing0(var, c1, c2), where **var** is the data being checked. Values less than or equal c1 or greater than or equal c2 are replaced with 0.0.

$d missing(var, c1, c2), where **var** is the data being checked. Values less than or equal c1 or greater than or equal c2 are replaced with the most recent 'good' value. If first line of data is not 'good' data, warning is issued. Data replacement does not begin until after an interval with 'good' data.

$d missinginter(var, c1, c2), where **var** is the data being checked. Values less than or equal c1 or greater than or equal c2 are replaced with a linearly interpolated value calculated from the 'good' values before and after the missing values. If first line of data is not 'good' data, warning is issued. Data replacement does not begin until the first 'good' data after the 'bad' data. Missing data at the end of the file is left unchanged.

$d missingrepl(var, c1, c2, var2), where **var** is the data being checked. Values less than or equal c1 or greater than or equal c2 are replaced with the value from var2.

$d missingFlag(var, c1, c2), where **var** is the data being checked. Output is 0.0 except when the value is less than or equal c1 or greater than or equal c2, when the output is 1.0. Value is checked ***before*** any replacement filters are executed.

$d missingFlagAfter(var, c1, c2), where **var** is the data being checked. Output is 0.0 except when the value is less than or equal c1 or greater than or equal c2, when the output is 1.0. Value is checked ***after*** any replacement filters are executed.

$p smear(p, Time1, Time2) p = daily precipitation as first value of day, Time1 = start time or <= 0 to indicate only negative values of precipitation have to be processed. Time2 is stop time or <= 0 for end of file.

$a abs(var) a = absolute value of variable

$a add(var, c) a = variable var plus constant c.

$s sub(var, c) s = variable var minus constant c.

$m mul(var, c) m = variable var multiplied by constant c.

$d div(var, c) d = variable var divided by constant c.

$pow powV(var, A, B) pow = $A*var^B$.

$exp expV(var, A, B) exp = $A*e^{B*var}$.

$log logV(var A B) log = A*ln(var*B).

$C const(c) C = constant c.

$a addV(var, var1) a = variable var plus variable var1.

$s subV(var, var1) s = variable var minus variable var1.

$m mulV(var, var1) m = variable var multiplied by variable var1.

$d divV(var, var1) d = variable var divided by variable var1.

$R random(seed) series of random numbers initialised by seed of last call. Random number generator is shared between all calls.

$TimeShift(Ts), where Ts is the time shift in days. Negative values move the file time backwards. The fractional portion should an integral number of time

intervals, i.e. n*1/24, n*1/48 etc..

Using ForceInterval to Change Observation File Interval.

This filter changes the time step interval of an entire observation file. Depending upon the initial interval the interval length can increase or decrease. Calling,

$ForceInterval(48)

will create 30 minute interval data. Calling,

$ForceInterval(6)

will create 4 hourly interval data.

The range of the frequency parameter is 1 to 288. Special consideration has to be given to "rate" inputs with units of mm/int, MJ/int etc., where the daily sum has to be kept constant despite the change in time interval. The inputs have to be flagged with a negative column count. N.B. observations like daily precipitation (ppt), mm/d are not affected and are always positive.

sample header

p -1 (mm/int) N.B. negative sign. If it had been daily precipitation it would be be "ppt 1 (mm/d)".

Qsi 1 (W/m^2)

QnD 1 (MJ)

rh 1 ()

SWE 1 (mm)

t 1 (°C)

u 1 (m/s)

$ForceInterval(24) change interval to hourly from what ever it was originally.

########

Time Simulation (Can only be used in a separate *.obs file containing no real data).

Sometimes it would be convenient to generate synthetic data instead using field observations. A special filter makes this possible.

$Sim(StartTime, EndTime, Interval) where

StartTime as mm/dd/yy or mm/dd/yyyy

EndTime as mm/dd/yy or mm/dd/yyyy

Interval in hours. Minimum 0.5

When this filter is put in an observation file, no interval data is ever read but time periods from 'StartTime' to 'EndTime - 1 Interval' are generated at the interval specified. The time simulation filter does not generate time fields unless additional filters are used to define the data to be generated, e.g. 'const(C)' etc..

Wave Synthesis (Can only be used in a separate *.obs file with $Sim(...) and not in a regular observation file).

$Fract sine(period phase start end)

$Fract sin(period phase start end)

$Fract cos(period phase start end)

$Fract square(period phase start end)

$Fract ramp(period phase start end)

$Fract pulse(start end), where

period is in days. For less than one day the time format can be used, e.g. 12 hours can be '0.5' or '12:00:00'.

phase is in days. For less than one day the time format can be used, e.g. 12

hours can be '0.5' or '12:00:00'.

start is start date (mm/dd/yy) or (mm/dd/yy_hh:mm:00)

end is end date (mm/dd/yy or (mm/dd/yy_hh:mm:00))

Fract will vary between -1.0 and 1.0 depending on the function.

Value is the value calculated by the function.

$Value exp(start end A B) Value = $A*e^{B(t - t0)}$

$Value log(start end A B) Value = A*ln(B*(t - t0))

$Value pow(start end A B) Value = $A*(t - t0)^B$

$Value poly(start end a0 a1 a2 a3 a4) where X = (t - t0), Value = $a0 + a1*X + a2*X^2 + a3*X^3 + a4*X^4$

Example of Wave Synthesis.

```
Simulation test (06/10/02)

$$ comment line - Test pulse generation

$Sim(01/01/01, 01/05/2001, 1) simulation time period

$Fract pulse(01/02/01, 01/03/01) one day pulse

$P pulse(01/01/01_1:0:0, 01/01/01_12:0:0)

$S sin(12:0:0, 0, 01/01/01, 01/02/01) // legacy sine is also
acceptable

$S cos(12:0:0, 0, 01/01/01, 01/02/01)

$R ramp(1, 0, 01/01/01, 01/02/01)

$Q square(1, 0, 01/01/01, 01/02/01)
```

```
$$ coment line - delayed functions

$P1 pulse(01/03/01, 01/04/01)

$S1 sine(1, 0, 01/03/01, 01/04/01)

$R1 ramp(1, 0, 01/03/01, 01/04/01)

$Q1 square(1, 0, 01/03/01, 01/04/01)

##############################################
```

Observation File Types.

1. First file - the first observation file loaded in the project. It determines the model operating time step interval and also the maximum model run time period.
2. Interval file - any subsequent continuous observation file with a time interval less or equal to one day.
3. Short interval time - any subsequent interval time which begins earlier or ends later than the first interval file.

First files , interval files and short interval files cannot have any missing data and must be contiguous. First files and interval files are the easiest to use for modeling as the data set is complete and no status checking is required to handle missing data. Note that in the preparation of these files the editor may have used interpolation or some other method to generate missing values to make them complete over the desired time period.

4. Sparse file - an observation file with an interval data of greater than daily or non continuous data.

Sparse files of any time step when accessed by a model module require status checking and programming to handle the periods when no data is available. The following example demonstrates how to handle intervals with missing data. Note that the special values for double xLimit and for long lLimit declared in common.h, are used to indicate undefined values and these values are not plotted by CRHM.

```
class ClassSparse : public ClassModule {

public:

   ClassSparse(string Name = "Sparse", String Version = "undefined") :
   ClassModule(Name, Version){};

   long nhru;

// declared variables
```

```
    float *Ourt; // °C
    float *Ourtshort; // °C
    float *Ourt2; // °C
    float *Ourt2mean;// °C
    float *OurD1; //
    float *OurD1s; //
    float *OurD2; //
    float *OurS1; //
    float *OurS2; //
// declared parameters
// declared observations
    const float *t;
    const float *tshort;
    const float *t2;
    const float *S1;
    const float *S2;
    const float *D1;
    const float *D1s;
    const float *D2;
// declared observation function.
    const float *t2mean;
```

```
// Handles
    ClassVar *tHand;
    ClassVar *tshortHand;
    ClassVar *t2Hand;
    ClassVar *D1Hand;
    ClassVar *D1sHand;
    ClassVar *D2Hand;
    ClassVar *S1Hand;
    ClassVar *S2Hand;
// Routines
    void decl(void);
    void init(void);
    void run(void);
};
void ClassSparse::decl(void) {
    declvar("Ourt", NHRU, "t", "()", &Ourt);
    declvar("Ourtshort", NHRU, "tshort", "()", &Ourtshort);
    declvar("Ourt2", NHRU, "t2", "()", &Ourt2);
    declvar("Ourt2mean", NHRU, "t2mean", "()", &Ourt2mean);
    declvar("OurD1", NHRU, "D1", "()", &OurD1);
    declvar("OurD1s", NHRU, "D1s", "()", &OurD1s);
```

```
    declvar("OurD2", NHRU, "D2", "()", &OurD2);
    declvar("OurS1", NHRU, "S1", "()", &OurS1);
    declvar("OurS2", NHRU, "S2", "()", &OurS2);

    tHand = declreadobs("t", NOBS, "t", "()", &t);
    tshortHand = declreadobs("tshort", NOBS, "tshort", "()", &tshort);
    t2Hand = declreadobs("t2", NOBS, "t2", "()", &t2);
    D1Hand = declreadobs("D1", NOBS, "?", "(?)", &D1);
    D1sHand = declreadobs("D1s", NOBS, "?", "(?)", &D1s);
    D2Hand = declreadobs("D2", NOBS, "?", "(?)", &D2);
    S1Hand = declreadobs("S1", NOBS, "?", "(?)", &S1);
    S2Hand = declreadobs("S2", NOBS, "?", "(?)", &S2);

    declobsfunc("t2", "t2mean", &t2mean, AVG);
}
void ClassSparse::init(void) {
    nhru = getdim(NHRU);
}
void ClassSparse::run(void) {
    for(int hh = 0; hh < nhru; ++hh) {
        Ourt[hh] = t[0];
```

```
if(tshortHand->FileData->GoodInterval)
   Ourtshort[hh] = tshort[0];
else
   Ourtshort[hh] = xLimit;

if(t2Hand->FileData->GoodInterval)
   Ourt2[hh] = t2[0];
else
   Ourt2[hh] = xLimit;


Ourt2mean[hh] = t2mean[0];

if(D1Hand->FileData->GoodInterval)
   OurD1[hh] = D1[0];
else
   OurD1[hh] = xLimit;

if(D1sHand->FileData->GoodInterval)
   OurD1s[hh] = D1s[0];
```

```
        else
            OurD1s[hh] = xLimit;

        if(D2Hand->FileData->GoodInterval)
            OurD2[hh] = D2[0];
        else
            OurD2[hh] = xLimit;

        if(S1Hand->FileData->GoodInterval)
            OurS1[hh] = S1[0];
        else
            OurS1[hh] = xLimit;

        if(S2Hand->FileData->GoodInterval)
            OurS2[hh] = S2[0];
        else
            OurS2[hh] = xLimit;
        }
}
```

Sample program report.

CURRENT TIME: 5/6/03 10:30

CRHM Version: NON-DLL 5.04

PROJECT FILE NAME:

TestFiles.prj dated 4/29/03 15:06

DIMENSIONS:

nhru 1

nlay 1

nobs 1

OBSERVATIONS:

C:\CRHM\TestFirstFile.obs (4/11/98 01:00 - 6/1/98 00:00 Interval = 01:00)

C:\CRHM\TestDaily.obs (4/12/98 00:00 - 6/1/98 00:00 Interval = daily)

C:\CRHM\TestDaily2.obs (4/13/98 00:00 - 6/1/98 00:00 Interval = sparse data file)

C:\CRHM\TestSparse.obs (4/28/98 12:00 - 5/30/98 12:00 Interval = sparse data file)

C:\CRHM\TestDailyShort.obs (4/15/98 00:00 - 4/30/98 00:00 Interval = daily)

C:\CRHM\TestFirst2.obs (4/11/98 02:00 - 6/1/98 00:00 Interval = 02:00)

C:\CRHM\TestFirstFileShort.obs (4/12/98 17:00 - 4/29/98 07:00 Interval = 01:00)

MODULES:

Sparse CRHM basic 05/02/03

DATES:

1998 4 11

1998 6 1

PARAMETERS:

INITIAL STATE:

Description of observation files to run sample program.

- TestFirstfile.obs - hourly time step supplying t.
- TestFirst2.obs - every second temperature from TestFirstfile.obs.
- TestFirstFileShort.obs - middle time period of TestFirstfile.obs with temperature variable called tshort.
- TestDaily.obs - daily observations D1.
- TestDailyShort.obs - same as TestDaily.obs
- TestDaily2.obs - every second daily observations from TestDaily.obs and called D2
- TestSparse.obs -sparse observations S1 and S2.

Observation Step Values.

The biggest problem in CRHM has been the mix of DAILY alogorithms and INTERVAL alogorithms in the same model. DAILY alogorithms are normally processed at the end of every day since only then are daily means or totals of climate or meteorological data available. INTERVAL alogorithms can be handled every interval.

STEP Value.

In CRHM there is a variable called STEP which is incremented every interval. The value for the first interval of a model run is 1. This is always the first interval of the first day of the model run, e.g. 00:30AM or 1:00AM. If Step is divided by the number of measurements taken per day (FREQ), i.e. 24 or 48, the remainder will be 1 for the first interval of the day or 0 for the last interval of the day. This relationship is used in CRHM modules to determine when to calculate DAILY alogorithms.

Example1.

```
if(STEP%FREQ == 0)

    {calculate an expression for the last interval of the day} or
```

Example2.

```
if(STEP%FREQ == 1)

    QsiD = Qsi

else

    QsiD = Qsid + Qsi
```

The latter code will calculate the total incoming short-wave radiation for the day. If it is combined in a module with the former code, the DAILY alogorithm will

access the daily total incoming short-wave radiation.

Regular Expression.

The text editor TextPad© has this very useful feature. It provides a handy method for cleaning up observation files. The following examples show how to handle spaces and tabs.

In all examples the *regular expression* option show be selected in the replace dialogue box. Replacements can apply to the entire file or selected text.

Remove leading line spaces.

Find what: ^[*space*\t]+ Replace with: *nothing*

where *space* is the space character and *nothing* means literally nothing.

Remove extra spaces and tabs at end of line.

Find what: [*space*\t]+$ Replace with: *nothing*

Replace spaces with tabs in the observation file data.

Select all of the observation file after the pound symbols.

Find what: [*space*\t]+ Replace with: \t

Add extra dummy column to simplify copying/pasting.

Select all of the observation file after the pound symbols.

Find what: $ Replace with: \t or \t9999 to make it more visible.

Creating Observation Files using TextPad.

Climatic data for driving CRHM comes from many sources and in many different formats. However, it is usually possible to put the data into columns and then the following procedure using CRHM and TextPad can be used to generate a CRHM observation file.

Steps.

1. Create dates - Use the project file "Make_Dates.prj". The first part of this project is listed below.

Description - Make_Dates.prj to create obs file

Version: CRHM 3.04 Creation: 05/14/09 14:09

Dimensions:

######

nhru 1

nlay 1

nobs 1

######

Macros:

######

######

Observations:

######

######

Dates:

######

1979 1 1 1

1985 10 5

######

Modules:
... etc.

Under field "Dates: "

Edit the start date and end date to the required range. The format is YYYY MM DD and for the start date there is the additional field FREQ, where FREQ is the number of intervals in the day, e.g. Daily - 1, Hourly - 24, etc.

- Run this project in CHRM. Then export RUN_ID as either an "Observation" or "Excel" file. The former is more convenient as the dates are in a readable format.

Observation format.

run_ID(1) 1

######

1979 1 1 0 0 1

1979 1 2 0 0 1

1979 1 3 0 0 1

Excel format.

run_ID(1) 1

######

29856 1

29857 1

29858 1

Adding Data Header and Columns.

Data can be prepared using Excel and exported as a text tab deliminated file or in TextPad. TextPad is a text editor with the additional feature that its "Block Select Mode" allows for column editing. For column editing, tabs instead of spaces makes it possible to align the columns.

Fortunately, TextPad has the "Regular expression" feature option when searching and replacing which allows the user to do "intelligent" find/replaces. If a file has used spaces between columns they can be changed to tabs by enabling "Regular expression" and applying:- "Find what: [*space*\t]+ Replace with: \t". Now it is only necessary to change the tab size in order to accommodate wider columns.

It is best to handle the data columns first and add the header portion of the file last. When pasting data into the Dates.obs file transfer columns containing dates and times and use them to compare with with the CRHM dates to detect any errors, missing data, incorrect intervals etc. After thorough checking, delete these extra columns. There should be no extra spaces or tabs at the end of lines or at the end of the file.

The file dates.prj has a variable run_ID(1). Its header and data column should be deleted.

The header should be added with TextPad with the "Block Select Mode" deselected. The "####" symbols separate the header and data portions of the file.

The first line must be present and is not used by CHRM. It can be used by the user to comment the file.

Every column of data requires a line specifying the name to appear in CRHM,

# of columns (normally 1), units enclosed in brackets, followed by an optional comment. One or more spaces separates the fields.

After these lines filters may be added.

Testing.

The file is tested by loading into CRHM. CRHM will find most errors and suggest solutions. Depending upon the error, it may be necessary to close CRHM before testing the corrections.

Notes.

1. Use only full days. A day starts ONE interval after midnight and ends at midnight. E.g. "1979 1 1 1 30" to "1979 1 2 0 0" (or "1979 1 1 24 0") where format is "YYYY MM DD hh mm".
2. Dates must be sequential with NO gaps.
3. Missing data fields must have a place holder, e.g. -9999 etc.
4.

Macro.

This capability of the CRHM program allows users to create simple modules suitable for testing algorithms and for diagnosing CRHM model output.

Local Variables.

Local variables are defined using the the keyword "var". For example "var i", "var i var j" or "var i, j".

CRHM variables.

CRHM variables as those defined in the "declreadobs", "declgetvar", "declparam", "declvar" and "declobs" declarations. Note that the latter three types are defined in the current macro module and the first two types are derived from other CRHM modules in the model. The macro commands are enclosed in a for loop which is executed NHRU times. A local variable "hh" is defined so that values for every iteration may be saved in the CRHM macro module variable output. Note that local variables are not accessible outside the macro module except by saving their values into CRHM variables.

Arithmetical Operators.

1. +, - addition/subtraction
2. *. / multiplication/division
3. ^ exponentiation
4. % modulus
5. (...) brackets enclosing an arithmetical expression.
6. [n] array element index. Order for 2-D is [hh][ll], i.e. hru first. Elements are referenced 1, 2, 3, 4 ... Cannot be an expression. Use var i; i = J+k; array[i], not array[j+k].

Logical Operators.

1. || OR.
2. && AND.
3. != Not equal.
4. == Equal.
5. <= Less Than or Equal.
6. < Less Than.
7. >= Greater Than or Equal.
8. > Greater Than.
9. ! Logical Not. (Faulty)

Control Statements.

if (condition) ... else ... endif

- multiple statements or none are permitted in the TRUE and FALSE fields.
- The "if" statement must always be followed by a closing "endif" statement.
- "else" is optional if there are no FALSE statements to execute.
- Multiple "if" statements are permitted.
- "if" statements can appear within other "if" statements.
- Lowercase must be used for "if', "else" and "endif".
- "else" "if" must always be entered as two separate words.
- Example :- if ... else if ...endif endif.

while(condition) ... endwhile.

- while condition is true the code in the body of the while is executed.

for(initialization; condition; increment) ... endfor.

- no field may be left empty.
- initialization sets initial value of optional loop counter.
- condition when FALSE terminates the loop.
- condition can be a compound logical statement, e.g. "for (X = 0; lastX - X > 0.01 && max < 1000; max = max +1)".
- initialization and increment fields can have multiple statements separated by commas, e.g. "for (i = 0, j = 0; i < 10; i = i+1, j = j+2)".

Subroutine Library.

1. sin deg, where deg(°)
2. cos deg, where deg (°)
3. exp
4. log
5. log10
6. min
7. max
8. estar t
9. patmos Ht, in kPa, where Ht (m) is the height.
10. rhoa t, ea, Pa, in (kg/m^3), where t (°C), ea (kPa) and Pa (kPa) is the atmospheric pressure.
11. spec_humid ea, Pa, in (kg/kg) where ea (kPa) and Pa (kPa) is the atmospheric pressure.

12. PI
13. DAY - current day
14. MONTH - current month
15. YEAR - current year
16. JULIAN - Julian day of the year.
17. FREQ - number of time intervals in a day.
18. STEP - current interval starting at 1.
19. GROUP - Current group index. 1 to maximum number of groups.
20. STRUCT - Current struct index. 1 to maximum number of structs.
21. FIRSTINT - True for the first interval of the day. When STEP % FREQ equals 1.
22. LASTINT - True for the last interval of the day. When STEP % FREQ equals 0.
23. NO_DISPLAY - If variable is set to this value it will not display. When exported creates a sparse file.
24. RAND - random numbers between 0.0 and 1.0.
25. ReadAheadObs - write to this function to read observations before and after the current interval. Writing -2 will cause all observations referenced by a "declreadobs" declaration in this module, to refer to the interval two periods earlier, +2 to the period two intervals later and 0 will return module read observations to the current interval. Reading from ReadAheadObs returns the status, 1 - error (outside available observation range). HRU_OBS is not used to access the observation. Observations are read in sequence as stored in the file.
26. WriteAheadObs - use this function to write the values of the current interval observations to permanent storage. Useage is to read from the desired

interval using ReadAheadObs function. Then changing the value of the desired observation and then writing the new values to observation storage using the function WriteAheadObs with the same interval offset..

Macro Declarations.

N.B spaces may be included in text fields if the entire field is enclosed in double quotes.

To create a parameter in the module,

declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", "(my units)"[,Int].

Parameter macro variables are by default floating point. If is necessary to use an existing CRHM integer parameter this can be done by adding "Int" to the end of the normall call;

declparam, inhibit_evap, NHRU, [0], 0, 1, "inhibit evapatation, 1 -> inhibit", "()", "Int".

To manage 2-D parameters the dimension NDEFN is implemented.

declparam, Distrib, NDEFN, [1.0], 0.0, 100.0, "Test 2D parametert", "()"

The order of element access to Distrib [HRU][LAY].

To change the value of a CRHM parameter declared in another module,

declputparam, module_name, variable_name, (units).

To use a CRHM observation within the module,

declreadobs, t, NOBS, description, (units). N.B. access is limited to the available observations. Last available observation is used to satisfy any remaining requests.

To use a CRHM observation function within the module,

declobsfunc, t, tfunc, FUNC. N.B. access is limited to primitive observations. FUNC from "AVG, MIN, MAX, DTOT, POS, TOT, FIRST, LAST, MJ_W and

W_MJ".

To create a new CRHM variable for the module,

declvar, OutVar, NHRU, description, (units) [,Int].

decldiag, OutVar, NHRU, description, (units)[,Int].

declocal, OutVar, NHRU, description, (units)[,Int].

To manage 2-D parameters the dimension NDEFN is implemented.

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ().

The order of element access to Test_NDEFN [HRU][LAY].

To create a new **state** CRHM variable for the module,

declstatvar, OutVar, NHRU, description, (units).

To create a new CRHM local variable for the module. N.B. this a variable local to this module. Not to be confused with a parser local variable.

declocal, OutVar, NHRU, description, (units).

To use a CRHM variable from another module,

declgetvar, module_name, variable_name, (units).

To use a CRHM variable declared in another module and alter its value,

declputvar, module_name, variable_name, (units).

To use a CRHM parameter declared in another module and alter its value,

declputparam, module_name, variable_name, (units).

To create a CRHM observation from existing observations, parameters and variables,

declobs, t2, NHRU, description, (units). N.B. if observation is already defined

by an observation file - does nothing.

To force modules into a desired loading order,

setpeer, PeerVar, PeerRank, where PeerVar is a CRHM variable that the current module must be loaded after and the PeerRank is the offset at this level.

This command is required when a module has no input variables to allow CRHM to determine the position of the module in the model order. A typical case is a module whose inputs consist of observations. Automatically it will be loaded early in the model even if it uses declared observations from other modules because all types of observations have the same priority.

To force the module to load after a declared observation has been calculated set PeerVar to 'ObsName#'. The # symbol differentiates between a variable named 'ObsName' and a declared observation named 'ObsName'.

The PeerVar cannot be a variable that is accessed using a declputvar as these variables have no rank value. Examples of these variables are "SWE", Sd, soil_moist, soil_rechr, hru_actet and hru_cum_actet.

Macro Structure.

1. The first line of a Macro is its name. This is the name that it is identified by in the model. Macro and Module names must be unique. Text after the module name is handled as the module description.

2. Next follows the declaration section. Each declaration is on a new line.

3. The "command" line ends the declaration section and begins the code to be executed.

4. The execution code is free format and may be indented and commented.

5. The end of the macro definition is indicated by the "end" statement on a new line.

Comments.

Code may be documented line using "//". Any text after the "//" is ignored and

handled as a comment.

To use spaces in declaration descriptive(text) fields enclose the desired text in double quotes, e.g. declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", " (my units)"

Array references are in the range of 1 to the maximum number of HRUs.

Element[0] is illegal. When using a standard observation variable the element access is [1], e.g. T[1], u[1] etc. If the array element is not specified it will default to [1]. Not recommended.

When accessing observations, the element is limited to the maximum defined element for the observation.

Macro Edit Screen.

This screen is a simple text editor. At present no "smarts" are built in. The screen has the capability to cut and paste to and from itself and to and from other applications. Macro modules can be saved from the screen using the File menu. The default file extension is "*mcr". These macro files are never used by CRHM and are for the use of the user only. The two buttons allow the user to save the screen changes to CRHM or cancel current changes and return to the last saved CRHM screen in the model.

To create a new line use CTRL + Enter.

When loading a macro file (*mcr), it will by default insert the text into the edit screen at the position of the cursor. However, if the edit screen has a selection, it will be replaced by the contents of the file.

When saving a macro, the entire edit screen is saved to the file unless there is a selection and in that case only the selected text will be saved.

Saving Macros.

Macros are automatically saved to the CRHM project file when the model is saved as a project in the main screen file menu. A macro may also be saved as a

file in the Macro Edit Screen for import into another project. The file extension used is ".mcr". Since CRHM loads executable Macros from the project file, to utilise code in a "*.mcr" file the file must be loaded into CRHM using the Macro Edit Screen and then the project saved. Exit from CRHM and then re-run CRHM and load the project file.

Flow Screen.

Since macro modules used for debugging may not be required to satisfy inputs to the current CRHM model, CRHM will detect them as unused. To keep the macro modules, always select "NO" in the "Remove module" dialogue box. Macro declared observations are labelled with a trailing "#". For example the Macro declared observation "MyObs" will be displayed as "MyObs#". This notation differentiates declared Macro Observations from declared Macro Variables.

Declared Observation Linking Priority.

When a model is run and an Observation is available from a file (field observation) and also from a Macro, CRHM by default will use the observation from the file.

When a Macro defines an observation that should have a higher priority than the file observation, its name should have a trailing "#' sign, e.g. "MyObs#" which will display in the flow screen as "MyObs#". When this convention is used, "hard code" Modules have to contain extra code to handle the special name.

Macro Example.

The following macro definitions demonstrate the following features.

1. Macros are named by the user.

2. Multiple macros may be defined at once.

3. Standard CRHM parameters allow macro physical outputs to be easily set and modified like normal CRHM modules.

4. Any Observations from the CRHM model may be accessed.
5. Any CRHM module/macro output variable may be accessed.
6. CRHM variable outputs may be generated to be used by other macro or standard CRHM modules.
7. Local variable values are preserved from time interval to time interval.
8. Writer should provide a description and units for the variables and parameters used to permit CRHM to supply help information to the user.

MyMacro1 optional module description

declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", (my_units)

declreadobs, t, NOBS, description, (units)

declvar, OutVar, NHRU, description, (units)

declvar, XOutVar, NHRU, description, (units)

declgetvar, obs, hru_t, "(°C)"

command // code to be executed

OutVar[1]=param[1]*t[1] OutVar[2]=param[2]*t[1] OutVar[3]=param[3]*t[1] // array element access by numeric value (range 1 - # HRUs)

var i i=i+1 XOutVar= sin(i) var j j=i+180 XOutVar[2] = sin(j) XOutVar[3] = cos(PI/36*i)

end // end of code and end of module definition

MyMacro2 // beginning of next module definition

declparam, param2, NHRU, 0.2, 0.0, 1.0, description, (units)

declreadobs, t, NOBS, description, (units)

declvar, Z, NHRU, description, (units)

declvar, Y, NHRU, description, (units)

declgetvar, Macro1, OutVar, (units)

command

Z[hh]=param2[hh]*t[1]

Y[hh] = param2[hh]*OutVar[hh]

end

Evaporation Example.

Evaporation // module name

declparam, A, NHRU, 0.023, 0.0, 1.0, "description", (mm/day) // declarations

declparam, B, NHRU,17.8, 0.0, 100.0, "description", (°C)

declparam, Zref, NHRU,1.5, 0.001, 100.0, Zref, (m)

declparam, Zwind, NHRU,10, 0.001, 100.0, Zwinf, (m)

declparam, Z0, NHRU,0.001, 0.001, 100.0, Zo, (m)

declvar, EvapAlg, NHRU, "evaporation_algorithm", (MJ/(m2/day))

declvar,cum, NHRU, "cum_evaporation_algorithm", (mm)

declvar,Ra, NHRU,Ra, (s/m)

declgetvar, obs, hru_tmean, "(°C)" // mean air temperature

declgetvar, obs, hru_tmin, "(°C)" // minimum air temperature

declgetvar, obs, hru_tmax, "(°C)" // maximum air temperature

declgetvar, obs, u, "(m/s)" // wind velocity

command // module code

var U U=max(u[0], 0.2) // assume minimum wind velocity to prevent divide by zero errors

Ra[hh] = log(Zref[hh]/Z0[hh])* log(Zwind[hh]/Z0[hh])/0.4^2*U

EvapAlg[hh] =-A[hh]*(hru_tmean[1] - B[hh])*Ra[hh]*(hru_tmax[1] - hru_tmin[1])^0.5*1/(245*2.501)

cum[hh] = cum[hh] + EvapAlg[hh]

end // end of module

Macro Implementation of C++ module.

To relate to a practical example we will design a macro to simulate the module ClassExample described earlier which converts interval net radiation in MJ/(m2-Int) calculated by an earlier module to mm/(m2-Int) of water, i.e. kg/(m2-Int) of water. The air temperature from an observation is required to carry out the conversion. Two other outputs are calculated as a fraction of the module output. These fractions are specified by the parameters F_Qg and F_Qs.

Example // name of micro module

declreadtobs(t, OBS, Temperature, (°C))

declgetvar(netall, net, "(MJ/m^2*int)")

declparam(F_Qg, NHRU, 3*0.2, 0.0, 1.0, Qg=F_Qg*Rn, ())

declparam(F_Qs, NHRU, [0.0], 0.0, 1.0, Qs=F_Qg*Rn, ())

declvar(Rn, NHRU, net, (mm/Int))

declvar(Qg, NHRU, ground_flux, (mm/Int))

declvar(Qs, NHRU, storage_flux, (mm/Int))

// The algorithm code to be executed every time interval and for every NHRU is written into the command area**.** The program code follows:

command // code is executed for number of HRUs with hh varying between 1 and # HRUs.

Rn[hh] = net[hh]/(2.501-0.002361*t[1])

Qg[hh] = Rn[hh]*F_Qg[hh]

Qs[hh] = Rn[hh]*F_Qs[hh]

end

Since the command code applies to every HRU, it is executed inside a for loop. The output variable Rn, is calculated from the observation temperature and an output variable net calculated in another module. Outputs Qg and Qs from this module are the product of the output Rn and the parameters F_Qg and F_Qs.

Example of "for" and 2-D arrays.

Test_declvar

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ()

declvar, Test_NDEFN2, NDEFN, "Test 2D variable", ()

declparam, Test_par_NDEFN, NDEFN, [1.0], 0.0, 100.0, "Test 2D parametert", "()"

command

var Fred [NHRU][7]

var ll

ll = 1

for(ll = 1; ll <= NHRU; ll = ll +1)

Fred[hh] [ll] = Test_par_NDEFN[hh][ll]*5

```
  Test_NDEFN[hh][ll] = Test_par_NDEFN[hh][ll]

  Test_NDEFN2[hh][ll] = Fred[hh][ll]

endfor

end
```

Example of accessing variables and parameters from another module, in this case pbsm_M.

```
Test_getvar

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ()

declvar, Test_NDEFN_P, NDEFN, "Test 2D variable", ()

declgetvar, pbsm_M, Results, ()

declparam, distrib, NDEFN, 1.0, 0.0, 100.0, "Test 2D parametert", "()"

command

var ll

for(ll = 1; ll <= NHRU; ll = ll +1)

  Test_NDEFN[hh][ll] = Results[hh][ll]

  Test_NDEFN_P[hh][ll] = distrib[hh][ll]

endfor

end
```

As always when sharing a parameter between modules, all values of the parameter should be made the same in every module, then the project saved and reloaded when the shared parameter should appear only in the "basin" module.

Known problems.

If a major change is made to a macro, i.e. insertion or deletion of a declaration, the user should exit from the macro entry screen and immediately save the project and then exit from CRHM. When CRHM is restarted it will execute properly. At present CRHM is not handling some aspects of allocation/deallocation of variables correctly.

Groups and Sructures.

Group. A collection of modules executed in sequence for all HRUs.

After using CRHM for a while, it was found to be inconvenient to always handle the individual modules. To overcome this groups were introduced. A group module, is a collection of modules which can be used in place of specifying the individual modules. When the group is defined the modules must be specified in the correct execution order. Use of groups and a relevant naming convention allow models to be easier implemented and understood. Unecessary detail can be hidden from the flow diagrams and documentation. The larger building blocks simplify the implementation of larger models.

Since different groups can have the same variable outputs in a model it is necessary to enhance the output variable names so that they do not conflict with one another. Variable names can already be long to be meaningful, a short suffix seemed to be the best way to differentiate the repeated names and the root name still to be recognizable. Suffix @A, @B, @C... are used where @A is used for outputs of the first group, @B is used for the next group etc.

Group application:

1. If groups are defined with the same modules, it is possible to execute the models in parallel using different parameters or driving observations.
2. If groups are defined as different models, it is possible to execute the models in parallel using identical parameters and driving observations to check different responses.

Structure. A parallel collection of modules. Only a selected one of them is executed for any HRU in a time step.

Again after using CRHM it was found that it was not always desired to execute the same module for every HRU. A structure handles this situation. The selection of the module for every HRU can be programmed statically, e.g. example 1. below or dynamically by using a preceding module to select the

module to be used for the current time step, e.g. example 2. below.

Since structures can have the same variable outputs in a model it is necessary to enhance the output variable names so that they do not conflict with one another. Variable names can already be long to be meaningful, a short suffix seemed to be the best way to differentiate the repeated names and the root name still to be recognizable. Suffix @a, @b, @c... are used where @a is used for outputs of the first structure, @b is used for the next structure etc. Groups use an upper case suffix while structures use a lower case suffix.

Structure application:

1. Comparison of algorithms; it is possible to specify a different module, say from evap, evapD, ShuttleWaite and ShuttleWaiteD for every HRU and track the different responses. Some of the modules, say evap, may be used more than once with parameters selecting Granger, Priestley-Taylor and Penman-Monteith giving more combinations.
2. Sometimes HRUs require diverse modules to be representative of the unit. An example would be forested and open farmland. By using the structure capability a general model can be customised to handle individual HRUs differently.
3. Sometimes HRUs change their characteristics due to excess water or lack of it. Using a structure, the module selection can be dynamically changed. If an HRU can experience dry spells, moderate rainfall and very wet conditions with flooding then it might be desireable to treat it using a grasslands module, wetlands module or a slough module respectively. The decision about which module to use would be made by a preceding module based upon the availability of moisture.

AKA Interaction with Groups.

In the normal useage of AKA in non-group models, the variables and observations are addressed uniquely by specifying the varable or observation and the module. However, with groups, variables and observations all are referenced by the group name. This lack of resolution means that the source and destination of variables and observations cannot be defined precisely. For example, if the user attempts to change say Qsi to the declared observation Qsi#, all occurrences of Qsi would be changed even the input to the module generating Qsi#, causing a loop. To prevent this from happening, if an attempt is made to

change an input of a module to the same name as one of its outputs, it will be ignored.

Declared observations, e.g. Qsi# do not have future data available to be able to generate any daily function, i.e. mean, max etc. CRHM detects these calls and leaves the observation as a simple observation, i.e. Qsi. In most situations this is the most desireable selection anyway.

Macro declgroup.

The Macro Group feature allows a number of modules to be grouped under one name and treated as one module.

The first line of the macro is the module or group name. In this case MyGroupA, MyGroupB and MyGroupC.

The following lines up to the *command* token, list the modules making up the group. They must be arranged in the correct execution order.

The tokens *command* and *end* complete the definition of the group.There should not be any lines between *command* and *end*.

Multiple macro groups and macros can be defined together.

MyGroupA

declgroup // defaults to number of HRUs defined in the model.

obs

calcsun

command

end

MyGroupB

declgroup 5 // five HRUs.

intcp

pbsm

albedo

netall

```
    ebsm
    evap
  command
  end
  MyGroupC
  declgroup 0  // defaults to  number of HRUs defined in the model.
    crack
    smbal
    route
  command
  end
```

Module Naming Convention.

Group variable names cannot use the original variable name otherwise their would be a naming conflict. To avoid this problem, the first macro group defined has the suffix "@A" the next "@B", "@C" etc.Because of this the search order defined in the a following section has been adopted.

Group parameters.

The group will have all the parameters of every module in the group. If a parameter is used by more than one module in the group these modules all share the same parameter values.

Module Linking Order.

Modules can use "*" or explicitly specify the source module name, e.g. *.hru_t and Obs.hru_t for the linking to work correctly. Obs.hru_t can only link to the module"Obs" and no other module or group.

The module linking search order is as follows.

1. Specific module, e.g. Obs.hru_t. Will only match Obs.hru_t.
2. Wild root module or group, e.g. *.hru_t and *.hru_t@A. *.hru_t would match any module with an output hru_t. *.hru_t@A will only match the hru_t output of the first group defined i.e. with suffix @A
3. Wild group module stripped of its group suffix. *.hru_t@A is reduced to *.hru_t before searching. *.hru_t@A will match any module with an output hru_t.
4. Wild group module stripped of its group suffix and a search is made of any groups after their group suffix has been removed. *.hru_t@A is reduced to *.hru_t before searching all groups for *.hru_t, i.e. after their suffix has been removed.

CreateGroup in the Macro edit menu.

This command allows an existing project to be convertede to a group. The new group has the name of the original project with "_A" appended. If the project is added multiple times the other groups will have "_B", "_C", etc. appended.

Multiple different projects may be added in any order. In every case the suffix "_A", "_B" etc. will be added.

The final model is build in the usual way by going to the menu "Build/Construct" and selecting the groups and building as normal. The number of HRUs in each group is determined from the project that the group was created from originally. At this time all parameters are CRHM default values.

After the model is built the original project parameter values may be moved into the new model by proceeding to the Parameter menu and loading the parameter file - "CreateGroup.par". Care should be taken to use the current directory as the same file name is always used. If this step is not taken parameter values will default to the module parameter default values.

During the preceding steps the number of HRUs should not be changed as the number of HRUs in the original project and the generated groups must be identical or they will not updated to the values in "CreateGroup.par".

The new project should be saved at this time and re-loaded before any further editing is carried out.

The number of HRUs in any group can be changed by inserting/changing the value on the "declgroup" instruction in the Macro defining the group. Note that if the value is missing or equal to 0, the number of HRUs in the group will be the global value.

When the number of HRUs is reduced the parameters are truncated. If the number of HRUs is increased the last value is duplicated as often as necessary. An exception is a serial parameter, "1, 2, 3!" for example, then the series is expanded.

It is important the names of the groups are not changed or the link between the original parameter values saved in the file "CreateGroup.par" and the groups will be broken causing the error "Unknown Parameter in parameter file" will occur for every parameter.

Macro declstruct.

The Macro Structure feature allows for a module to be chosen from a group of modules at execution time. An example of its useage is an area which is a wetland in wet years and a grassland during dry periods. The module used to represent the area can be chosen from the structure selection by another module setting the "HRU_struct" parameter for the structure module.

The first line of the macro is the module or structure name. In this case MyStructA, MyStructB and MyStructC.

The following lines up to the *command* token, list the modules making up the structure. They can be arranged in any order and are addressed as 1, 2, etc. to n.

The tokens *command* and *end* complete the definition of the group.There should not be any lines between *command* and *end*.

Multiple macro groups/structures and macros can be defined together.

```
MyStructaA
declstruct
   MyMacro1  // executed when "MyStructaA HRU_struct" = 1
   MyMacro2
command
end
MyStructaB
declstruct
   evap
   evap_Resist
```

```
command
end
MyStructaC
declstruct
   route
   netroute
command
end
```

Module Naming Convention.

Group variable names cannot use the original variable name otherwise their would be a naming conflict. To avoid this problem, the first macro structure defined has the suffix "@a" the next "@b", "@c" etc. Because of this the search order defined in a following section has been adopted. N.B. uppercase designates a group and lowercase designates a structure.

Structure parameters.

Since a structure can contain a diverse collection of modules having different parameters the structure will have all of these parameters. However, for any HRU only the parameters used by the module selected need to be defined. Any others can be left at their default values for that HRU.

Module Linking Order.

Modules can use "*" or explicitly specify the source module name, e.g. *.hru_t and Obs.hru_t for the linking to work correctly. Obs.hru_t can only link to the module"Obs" and no other module or group.

The module linking search order is as follows.

1. Specific module, e.g. Obs.hru_t. Will only match Obs.hru_t.
2. Wild root module or group, e.g. *.hru_t and *.hru_t@A. *.hru_t would match any module with an output hru_t. *.hru_t@a will only match the hru_t output of the first group defined i.e. with suffix @a
3. Wild group module stripped of its group suffix. *.hru_t@a is reduced to *.hru_t before searching. *.hru_t@a will match any module with an output hru_t.
4. Wild group module stripped of its group suffix and a search is made of any groups after their group suffix has been removed. *.hru_t@a is reduced to *.hru_t before searching all groups for *.hru_t, i.e. after their suffix has been removed.

ClassClark - Lag and Route Technique.

This class is used to delay a CRHM HRU variable in a user module using a time shift and a storage term. Clark's Method is used and is implemented using the following code.

```
ClassClark *OurDelays;

OurDelays = new ClassClark(const float* inVar, float* outVar, const
float* kstorage, const float* lag);
```

where:

inVar[HRU] is the input variable to be processed,

outVar[HRU] is the destination for the delayed output variable,

kstorage[HRU] is the desired storage constant (days) and

lag[HRU] is the desired time shift (hours). The resolution is equal to the observation interval.

To generated the delay after every module HRU so that it can be used for a downstream HRU in the same module use:

```
    OurDelays->DoClark(hru);// note hru in the range (0 to HRUmax-
1).
```

If delayed and lagged output is not used in the same module then all HRUs can be processed at once using:

```
    OurDelays->DoClark();
```

The class is destroyed using,

```
    delete OurDelays;
```

To determine the quantity still in storage before destroying the storage object use:

```
    float residual = OurDelays->Left(hru); // note hru in the range (0 to HRUmax-1).
```

Sample Application of ClassClark.

```
class Classdelay : public ClassModule {

public:

    Classdelay(string Name = "Qdelay", String Version = "undefined") : ClassModule(Name, Version){};

    long nhru;

// declared variables

    float *Tdelay; // °C

    ClassClark *OurDelays;

// declared parameters

    const float *Kstorage;

    const float *Lag;

// declared observations

    const float *t;

// procedures

    void decl(void);

    void init(void);

    void run(void);

    void finish(bool good);
```

```
};

void Classdelay::decl(void) {

    declvar("Tdelay", NHRU, "delayed temperature", "(°C)",
&Tdelay);

    declparam("Kstorage", NHRU, "[0]", "0", "20", "Air Temperature
Storage", "()", &Kstorage);

    declparam("Lag", NHRU, "[1]", "0", "96", "lag delay", "(hour)",
&Lag);

    declreadobs("t", NOBS, "air temperature", "(°C)", &t);

}

void Classdelay::init(void) {

    nhru = getdim(NHRU);

    for(int hh = 0; hh < nhru; hh+) Tdelay[hh] = 0.0; // initial
value

    OurDelays = new ClassClark(t, Tdelay, Kstorage, Lag);

}

void Classdelay::run(void) {

    OurDelays->DoClark();

}

void Classdelay::finish(bool good) {

    delete OurDelays;

}
```

Constants defined

These are universal constants defined in CRHM model.

#define

- Stefan-Boltzmann constant - #define SB 4.899e-09 // MJ/m2-d
- von Karman's constant - #define kappa 0.41

delta(t)

Calculates the Slope of the Saturation Vapour Pressure vs Temperature Curve.

Units

- kPa/°C.

Inputs

- t (°C) - air temperature.
- function estar(t) - saturated vapour pressure.

Returns

if (t >= 0.0)

return(2504.0*exp(17.27 * t/(t+237.3)) / sqr(t+237.3))

else

return(3549.0*exp(21.88 * t/(t+265.5)) / sqr(t+265.5))

DepthofSnow(SWE)

Calculates the Depth of Snow from Snow Water Equivalent (SWE).

Units

- (m)

Inputs

- SWE (mm) - Snow Water Equivalent.

Returns

```
if (SWE > 2.05)
    if(SWE <= 145.45) // SWE 145.45 mm equivalent to 60 cm
        Snow_Depth = (SWE -2.05)/2.39
else
    Snow_Depth = (SWE +128.06)/4.5608
else
    Snow_Depth = 0
return (Snow_Depth/100.0)
```

estar(t)

Calculates the Saturation Vapour Pressure.

Units

- kPa

Inputs

- t (°C) - air temperature.

Returns

if (t >= 0.0)

return (0.611 * exp(17.27*t / (t + 237.3)))

else

return (0.611 * exp(21.88*t / (t + 265.5)))

f(t, u, Pa, Ht, Zwind, Zref)

- Calculates the interval Drying Power for modules *evap* and *evap2*.

Units

- (mm/kPa/Interval)

Inputs

- t (°C) - air temperature.
- Pa (kP) - atmospheric pressure.
- u (m/s) - wind speed.
- Ht (m) - crop height.
- Zwind (m) - wind measurement height.
- Zref (m) - temperature/humidity measurement height.

Returns

- return((0.622 * kappa*kappa * rhoa(t, Pa) * u * Interval*3600*24) / (Pa * x1 * x2))

  where

  d0 = 0.67*Ht and Z0 = 0.14*Ht

  x1 = log((Zwind-d0)/(Z0))

  x2 = log((Zref-0.67*Ht)/(Z0))

  rhoa(t, Pa) = Density of air at (t, Pa).

  kappa = 0.4 von Karman constant.

  Interval = fraction of day.

Reference

Farouki_a(n)

- Calculates the value 'a' from the expression $3a^2 - 2a^2 = n$, where n is the fractional porosity.

Units

- ()

Inputs

- fractional porosity ().

Returns

- return 'a'

Reference

Farouki T. Omar, Cold Regions Science and Technology, 5 (1981) 67-75. The thermal properties of soils in cold regions

Code

```
float Farouki_a(float fract_por) {

  float a;

  float nnew = 0.0;

  while(fabs(fract_por - nnew) > 0.001) {

    a += (fract_por - nnew)*0.25;

    nnew = 3*a*a - 2*a*a*a;

  }

  return a;
```

```
}
```

fdaily(u, Ht)

- Calculates the daily Drying Power for module *evap*.

Units

- (mm/kPa/day)

Inputs

- u (m/s) - wind speed.
- Ht (m) - crop height.

Returns

- return (a + b*u)

  where

  Z0 = Ht*100.0/7.6 (cm)

  a = 8.19 + 0.22*Z0

  b = 1.16 + 0.08*Z0

Reference

gamma(Pa, t)

Calculates the Psychrometric constant.

Units

kPa/°C

Inputs

- t (°C) - air temperature.
- Pa (kP) - atmospheric pressure.

returns

0.063

Notes

- alternative is (0.00163*Pa/lambda(t)) which is commented out.

SVDens(t)

Calculates the Saturation Vapour Density of air.

Units

- kg/m^3

Inputs

- t (°C) - air temperature.

Returns

return (1.324*exp(22.452*t/(t+273.15))/(t+273.15))

Alternatives

- SVDens = E_* *M/(R*(T + 273.15)) where

  M=18.01 molecular weight of water (kg/kmole)

  R=8313 universal gas constant (J/(kmole k))

SWEfromDepth(Snow_Depth)

Calculates the Snow Water Equivalent (SWE) from the Depth of Snow.

Units

- (mm)

Inputs

- Snow_Depth (m) - Snow Water Equivalent.

Returns

```
if (Snow_Depth > 0.6)
    SWE = 4.5608*Snow_Depth*100.0-128.06
else if (Snow_Depth > 0.0205)
    SWE = 2.39*Snow_Depth*100.0+2.05
else
    SWE=0
return (SWE)
```

SWE_prob (SWEpeak, Melt, CV)

Calculates the probability of the snow water equivalent exceeding Melt, for a snowpack having an average areal snow water equivalence of SWEpeak at the beginning of melt.

Units

- ()

Inputs

- SWEpeak (m) - average areal SWE at the beginning of melt.
- Melt (mm) - amount of SWEpeak melted this interval.
- CV () - coefficient of variation of SWE.

Return P

if(Melt<= 1.0) return 1.0; // handle log(0) error

float K = (Melt-SWEpeak)/(CV*SWEpeak)

float Sy = sqrt(log(CV*CV+1.0))

float Ky = (log(K*sqrt(exp(Sy*Sy)-1.0)+1.0) + Sy*Sy/2.0)/Sy

float t = 1 /(1+little_p * Ky)

float P = (exp(-Ky*Ky/2)/sqrt(2*M_PI)) * (a1*t + a2*t*t + a3*t*t*t)

if(P > 1.0 || P < 0.001) P = 1.0 // handle discontinuity

where:

K = frequency factor

Sy = standard deviation of the transformed variable.

Ky = the frequency factor for the transformed data having an exceedence probability equal to K.

t, a1, a2, a3 coefficients for the interval-halving method using the approximation by Abramowitz and Stegun, 1965.

Notes

The Abramowitz and Stegun interval halving approximation is not continuous and emits extraneous values when the ratio of SWE to SWEpeak is low. It can generate values greater than one and less than 0.0 in this range. If either condition occurs the probability is made equal to 1.0. The five term approximation is only marginally better than the three term approximation used here.

Coefficient of Variation sample values.

Fallow	0.3	0.58
Stubble	0.33	0.33
Pasture	0.41	0.57
Brush	0.42	0.52
Yards	0.5	0.5

Purpose.

Normally CRHM is run interactively by the user. However, if a comparison of different model runs with different parameters is required, it can be very tedious. Example Excel spreadsheets with a Visual Basic macro are described here and are distributed with CRHM. This demonstration allows the user to list in an Excel workbook the parameters for each run and the macro will execute CRHM for each set of parameters and record in the workbook the final values of variables requested by the user. These variables are the same as being displayed by CRHM as specified in the project file. The process seems involved but if taken in steps it is straightforward. The current implementation only records the values of the selected variables at the end of the model run. If an intermediate value is required (e.g. peak evaporation), the user can write a CRHM macro which saves the desired variable value till the end of the run.

Preparation of input files.

Required files are,

1. CRHM_XLS.xls - This Excel workbook specifies the names of the process files to be used and controls the execution of the macro to execute CRHM.
2. Location of CRHM - where CRHM is installed, e.g. C:\Program Files\CRHM\CRHM.exe.
3. CRHM parameter file - a text parameter file saved from the CRHM project, e.g. Parameters.par.
4. List of parameter changes - an Excel file listing the parameters for each run, e.g. Changes.xls.
5. CRHM project file - project file to be run, e.g. pbsm.prj.
6. List of Observation files - an Excel file listing the observation files to be used, e.g. Changes_Obs.xls. This file is not used if the observation file is in the project file.

Three example files are distributed with CRHM in the directory "\Program Files\CRHM\CRHM_XLS".

CRHM_XLS.xls using the files; pbsm.prj, parameters.par and

Changes.xls. This the simplest example and uses a basic project (no groups). The parameters need only to be specified by name only.

CRHM_XLS_G.xls using the files; pbsmG.prj, parametersG.par and ChangesG.xls. Since this project uses groups the parameters must be specified by the group name + the parameter name, e.g. "pbsm_GrpA Ht".

CRHM_XLS_Obs.xls using the files; PBSM_NO_obs.prj, parameters.par, Changes.xls and Changes_Obs.xls. The project file does not specify any observation files since these are determined by file list in Changes_Obs.xls. N.B. that the Excel file defining the parameters is always used as its length determines the number of times CRHM is run.

Necessary CRHM options.

The following settings are required to be set so that the process can be automated.

1. AutoRun.
2. AutoExit.
3. Log/Last.

Albert, M.R. and G. Krajeski, 1998: A fast, physically-based point snow melt model for use in distributed applications", *Hydrological Processes*, 12(11), 1809-1824.

Brunt, D., 1932. Notes on the radiation in the atmosphere. *Quarterly Journal of the Royal Meteorological Society*, 58, 389-420.

Brutsaert, W., 1982. *Evaporation into the Atmosphere*. D. Reidel Publishing Co., London, UK. 299 p.

Carey, S.K., and W.L. Quinton, 2004. Evaluating summer runoff generation using hydrometric, stable isotope and hydrochemical methods in a discontinuous permafrost alpine catchment. *Hydrological Processes*, In press.

Cionco, R.M. 1978. Analysis of canopy index values for various canopy densities. Boundary Layer Meteorology. 15, 81-93.

Clark, C.O., 1945. Storage and the unit hydrograph. *Proceedings of the American Society of Civil Engineering*, 69, 1419-1447.

Elliott, J.A., B.M. Toth, R.J. Granger and J.W. Pomeroy, 1998. Soil moisture storage in mature and replanted sub-humid boreal forest stands. Canadian Journal of Soil Science, 78, 17-27.

Erickson, D.E.L., Lin, W., and Steppuhn, H.W., 1978. Indices for estimating Prairie runoff from snowmelt. Paper presented to the 7th Symposium on Applied Prairie Hydrology. Water Studies Institute, Saskatoon, Sask.Essery, R.L.H. and J.W. Pomeroy. 2004. Vegetation and topographic control of wind-blown snow distributions in distributed and aggregated simulations. *Journal of Hydrometeorology*, in press.

Essery, R., L. Li and J.W. Pomeroy. 1999. Blowing snow fluxes over complex terrain. Hydrological Processes, 13, 2423-2438.

Farouchi, O. T. 1981 The thermal properties of soils in cold regions. Cold Regions Sci. Technol. 5, 61-65.

Garnier, B.J. and A. Ohmura, 1970. The evaluation of surface variations in solar radiation income. *Solar Energy*, 13, 21-34.

Goodison, B. E., 1978. Accuracy of Canadian Snow Gauge Measurements. J. Appl. Meteorol. Vol. 17:1542-1548.

Goodison et al., 1998

Granger, R.J. and D.H. Male, 1978. Melting of a Prairie snowpack. *Journal of Applied Meteorology*, 17(2), 1833-1842.

Granger, R.J., D.M. Gray and G.E. Dyck, 1984. Snowmelt infiltration to frozen prairie soils. Can. J. Earth Sci., 21(6):669-677.

Granger, R.J. and D.M. Gray, 1989. Evaporation from natural non-saturated surfaces. *Journal of Hydrology*, 111:21-29.

Granger, R.J. and D.M. Gray, 1990. A net radiation model for calculating daily snowmelt in open environments. *Nordic Hydrology*, 21, 217-234.

Granger, R.J. and J.W. Pomeroy, 1997. Sustainability of the western Canadian boreal forest under changing hydrological conditions - 2-summer energy and water use. In, (eds. D. Rosjberg, N. Boutayeb, A. Gustard, Z. Kundzewicz and P Rasmussen) *Sustainability of Water Resources under Increasing Uncertainty*. IAHS Publ No. 240. IAHS Press, Wallingford, UK. 243-250.

Granger, R.J., Gray, D.M. and G.E. Dyck. 1984. Snowmelt infiltration to frozen Prairie soils. *Canadian Journal of Earth Sciences*, 21(6), 669-677.

Granger, R.J., Pomeroy, J.W. and J. Parviainen. 2002. Boundary layer integration approach to advection of sensible heat to a patchy snow cover. *Hydrological Processes*, 16, 3559-3569.

Gray, D.M. 1970. Handbook on the Principles of Hydrology. Water Information
Center, Inc., Port Washington, NY

Gray, D.M., Landine, P.G., and Granger, R.J. 1984, Simulating infiltration into frozen Prairie soils in streamflow models. Canadian Journal of Earth Sciences. 22, pp. 464-472.

Gray, D.M., P.G. Landine and R.J. Granger, 1985. Simulating infiltration into frozen prairie soils in streamflow models. Can. J. Earth Sci., 22:464-474.

Gray, D.M. and R.J. Granger, 1986. In situ measurements of moisture and salt movement in freezing soils. *Canadian Journal of Earth Sciences*, 23(5):696 704.

Gray, D.M., R.J. Granger and P.G. Landine, 1986. Modelling snowmelt infiltration and runoff in a prairie environment. Proceedings of the Symposium: Cold Regions Hydrology, Fairbanks, Alaska, July, 1986. Publ: American Water Resources Association. p427-438.

Gray, D.M. and P.G. Landine, 1986. Albedo model for shallow Prairie snowcovers. *Canadian Journal of Earth Sciences*, 24(9), 1760-1768.

Gray, D.M. and P. G. Landine, 1987. An energy-budget snowmelt model for the Canadian Prairies. Can. J. Earth Sci., Vol. 25, No. 8:1292-1303.

Gray, D.M. and P.G. Landine, 1988. An energy-budget snowmelt model for the Canadian Prairies. *Canadian Journal of Earth Sciences*, 25(9), 1292-1303.

Gray, D.M. and others, 1979. Snow accumulation and distribution. In, *Proceedings, Modelling Snowcover Runoff* (eds. S.C. Colbeck and M Ray). US Army Cold Regions Research and Engineering Laboratory, Hanover, NH. 3-33.

Gray, D.M., Landine, P.G. and R.J. Granger. 1984. An infiltration model for frozen Prairie soils. *Canadian Society of Agricultural Engineers, 1984 Annual Meeting*. Paper 84-313.

Gray, D.M., Landine, P.G. and R.J. Granger, 1985. Simulating infiltration into frozen Prairie soils in streamflow models. *Canadian Journal of Earth Sciences*, 22(3), 464-472.

Gray, D.M., P.G. Landine and G.A. McKay, 1985. Forecasting streamflow runoff from snowmelt in a Prairie environment. *Canadian Society for Civil Engineering Annual Conference, Saskatoon*. 213-231.

Gray, D.M., Granger, R.J. and P.G. Landine, 1986. Modelling snowmelt infiltration and runoff in a Prairie environment. In, *Cold Regions Hydrology Symposium*. American Water Resources Association. Fairbanks, Alaska. 427-438

Gray, D.M., Toth, B., Pomeroy, J.W., Zhao, L. and R.J. Granger. 2001. Estimating areal snowmelt infiltration into frozen soils. *Hydrological Processes*. 15. 3095-3111.

Hedstrom, N.R. and J.W. Pomeroy, 1998. Measurements and modelling of snow interception in the boreal forest. *Hydrological Processes*, 12, 1611-1625.

Kuchment, L.S., V.N. Demidov and Y.G. Motovilov. 1983. *River Runoff Formation (Physically Based Models)*. Nauka, Moscow (in Russian).

Kuchment, L.S., A.N. Gelfan, and V.N. Demidov, 2000. A distributed model of runoff generation in the permafrost regions. *Journal of Hydrology*. 240, 1-22.

Kustas, W. P., Rango, A., and Uijlenhoet, R. 1994. A simple energy budget algorithm for the snowmelt runoff model. *Water Resources Research*, 30(5):1515-1527.

Leavesley, G.H., Lichty, R.W., Troutman, B.M. and L.G. Saindon, 1983. Precipitation-runoff modelling system: user's manual. *US Geological Survey Water Resources Investigations Report 83-4238*. 207 p.

Leavesley, G.H., Restrepo, P.J., Markstrom, S.L., Dixon, M., and Stannard, L.G., 1996, The Modular Modeling System (MMS): User's Manual, *Open-File Report 96-151, U.S. Geological Survey*.

Motovilov, Y.G. 1978. Mathematical model of water infiltration into frozen soils. *Soviet Hydrology*, 17(2), 62-66.

Motovilov, Y.G. 1979. Simulation of meltwater losses through infiltration into soil. *Soviet Hydrology*, 18(3), 217-221.

Ogden, F.L. and B. Saghafian, 1997. Green and Ampt infiltration with redistribution. *Journal of Irrigation and Drainage Engineering*, 123(5), 386-393.

Parviainen, J. and J.W. Pomeroy, 2000. Multiple-scale modelling of forest snow sublimation: initial findings. *Hydrological Processes*, 14. 2669-2681.

Pomeroy, J.W., 1989. A process-based model of snow drifting. *Annals of Glaciology*, 13. 237-240.

Pomeroy, J.W. and R.A. Schmidt. 1993. The Use of Fractal Geometry in Modelling Intercepted Snow Accumulation and Sublimation. *Proceedings of the Eastern Snow Conference*. 50, 1-10.

Pomeroy, J.W. and D.M. Gray. 1995. *Snow Accumulation, Relocation and Management*. National Hydrology Research Institute Science Report No. 7. Environment Canada: Saskatoon. 144 pp.

Pomeroy, J.W. and Goodison, B.E., 1997. Winter and Snow, In, (eds W.G. Bailey, T.R. Oke and W.R. Rouse) The Surface Climates of Canada, Montreal: McGill-Queen's Univ Press. 68-100.

Pomeroy, J.W. and R.J. Granger, 1997. Sustainability of the western Canadian boreal forest under changing hydrological conditions - I-snow accumulation and ablation. In (eds. D. Rosjberg, N. Boutayeb, A. Gustard, Z. Kundzewicz and P Rasmussen) *Sustainability of Water Resources under Increasing Uncertainty*. IAHS Publ No. 240. IAHS Press, Wallingford, UK. 237-242.

Pomeroy, J.W. and R. Essery. 1999. Turbulent fluxes during blowing snow: field tests of model sublimation predictions. *Hydrological Processes*, 13, 2963-2975.

Pomeroy, J.W. and R.J. Granger, 1999. *Wolf Creek Research Basin: Hydrology, Ecology, Environment*. National Water Research Institute. Environment Canada: Saskatoon. 160 pp.

Pomeroy, J.W. and L. Li. 2000. Prairie and Arctic areal snow cover mass balance using a blowing snow model. *Journal of Geophysical Research*, Vol. 105, No. D21. 26619-26634.

Pomeroy, J.W., D.M. Gray and P.G. Landine. 1993. The Prairie Blowing Snow Model: Characteristics, Validation, Operation. *Journal of Hydrology*, 144, 165-192.

Pomeroy, J.W., P. Marsh and D.M. Gray, 1997. Application of a distributed blowing snow model to the Arctic. *Hydrological Processes* 11, 1451-1464.

Pomeroy, J.W., R.J. Granger, A. Pietroniro, J.E. Elliott, B. Toth and N. Hedstrom, 1997. *Hydrological Pathways in the Prince Albert Model Forest: Final Report*. NHRI Contribution Series No. CS-97007. 153 p. plus append.

Pomeroy, J.W., J. Parviainen, N. Hedstrom and D.M. Gray. 1998. Coupled modelling of forest snow interception and sublimation. *Hydrological Processes*, 12, 2317-2337.

Pomeroy, J.W., N. Hedstrom and J. Parviainen. 1999. The snow mass balance of Wolf Creek. In, (eds. J. Pomeroy and R. Granger) *Wolf Creek Research Basin: Hydrology, Ecology, Environment*. National Water Research Institute. Minister of Environment: Saskatoon. 15-30.

Pomeroy, J.W., B. Toth, R.J. Granger, N.R. Hedstrom, R.L.H Essery, 2003. Variation in surface energetics during snowmelt in complex terrain. *Journal of Hydrometeorology*, 4(4), 702-716.

Popov, E.G. 1973. Snowmelt runoff forecasts – theoretical problems. *The Role of Snow and Ice in Hydrology*. UNESCO-WMO-IAHS. Vol. 2. 829-839.

Quinton, W. L. and Marsh P., 1999. A conceptual framework for runoff generation in a permafrost environment.. Hydrological Processes, Volume 13, 2563-2581.

Quinton, W.L., and S.K. Carey, 2004. Snowmelt runoff from sub-alpine tundra hillslopes: Major processes and methods of simulation.

Hydrology and Earth System Sciences, In press.

Quinton, W.L. and D.M. Gray, 2001. Toward modelling seasonal thaw and subsurface runoff in arctic tundra environments. *Soil Vegetation, Atmosphere Transfer (SVAT) Schemes and Large Scale Hydrological Model*s. (eds. Dolman, A.J.., Hall, A.J. Kavvas, M.L., Oki, T. and J.W. Pomeroy) IAHS Publication No. 270. IAHS Press, Wallingford UK. 333-341.

Quinton, W.L., D.M. Gray and P. Marsh, 2000. Subsurface Drainage from Hummock- Covered Hillslopes in the Arctic-Tundra. *Journal of Hydrology*, 237, 113-125.

Quinton, W.L. and P. Marsh, 1999. A Conceptual Framework for Runoff Generation in a Permafrost Environment. *Hydrological Processes*, 13, 2563-2581.

Rutter, A.J., K.A. Kershaw, P.C. Robins, and A.J. Morton, 1972. A predictive model of rainfall interception in forests, I. Deriviation of the model from observations in a plantation of Corsican Pine. *Agricultural Meteorology*, 9: 367-384.

Rutter, A.J., A.J. Morton, and P.C. Robins, 1975. A predictive model of rainfall interception in forests, II. Generalization of the model and comparison with observations in some coniferous and hardwood stands. *Journal of Applied Ecology*, 12: 367-380.

Rutter, A.J. and A.J. Morton, 1977. A predictive model of ra rainfall interception in forests, III. Sensitivity of the model to stand parameters and meteorological variables. *Journal of Applied Ecology*, 14: 567-588.

Satterlund, D.R., 1979. An improved equation for estimating longwave radiation from the atmosphere. *Water Resources Research*, 15, 1643-1650.

Schmidt R.A., Troendle C.A., and Meiman J.R. 1998, Sublimation of snowpacks in subalpine conifer forests. Can. J. For. Res. 28, 501-513.

Shook, K., 1995. *Simulation of Ablation of Prairie Snowcovers*. Ph.D

Thesis, University of Saskatchewan, 189 pp.

Shook, K., D.M. Gray and J.W. Pomeroy. 1993. Geometry of patchy snowcovers *Proceedings of the Eastern Snow Conference*, 50. 89-98.

Shook, K., J.W. Pomeroy, D.M. Gray. 1993. Temporal variation in snow-covered area during melt in Prairie and Alpine environments. *Nordic Hydrology*, 24, 183-198.

Sicart, J.E., Pomeroy, J.W., Essery, R.L.H., Hardy, J.E., Link T. and D. Marks 2004. A sensitivity study of daytime net radiation during snowmelt to forest canopy and atmospheric conditions. *Journal of Hydrometeorology*, in press.

Van Wijk W. R., (1963) Physics of Plant Environment. North-Holland Publishing Company - Amsterdam, pp.166

Zhao and Gray, 1999. Estimating snowmelt infiltration into frozen soils. *Hydrological Processes*, 13(12-13), 1827-1842.

Errors.

Mathematical errors.

DOMAIN Argument was not in domain of function, such as log(-1).

SING Argument would result in a singularity, such as pow(0, -2).

OVERFLOW Argument would produce a function result greater than DBL_MAX (or LDBL_MAX), such as exp(1000).

UNDERFLOW Argument would produce a function result less than DBL_MIN (or LDBL_MIN), such as exp(-1000).

TLOSS Argument would produce function result with total loss of significant digits, such as sin(10e70).

Macro parsing errors.

Mathematical errors occur in macros as well as in any module execution. However, another source of errors in in parsing the user code. When parsing errors occur CHRM displays the block of characters that it cannot breakdown into meaningful phrases. Common problems are:

1. No matching parentheses.
2. Every 'if' or 'while' must have a closing 'endif' or 'endwhile'.
3. Variables are accessed in the range '1, 2 ...' to 'NHRU' or "NOBS".
4. In the case of observations CRHM will limit the upper range to the number of observations defined.
5. The correct type of bracket must always be used. '[]' to define element access and '()' to block conditional statements.
6. Strings containing spaces must be enclosed in double quotes, e.g. "Text with spaces".
7. Case is important.

The error could also be in the earlier code which, though interpreting correctly is missing the support for the later code found to be in error.

Macro not giving the expected results.

Check the following.

1. Variables must define there element access, e.g. '[hh]' or '[1]' will be assumed.
2. Units must be enclosed in parentheses to be recognized, e.g. "(s)".
3. Units must always be defined with only one term in the numerator, e.g. "(m/s)".
4. There are implied enclosing brackets after the '/', e.g."(MJ/m^2*d)" is equivalent to "(MJ/(m^2*d))".

CRHM help.

CRHM help consists of a series of HTML topic files which can be printed by subtopic from within CRHM from the help menu.

The help is also available as PDF files in the CRHM installation directory.

Filter writing

Filters are derived from the class called 'Classfilter'. Writing a filter is quite straightforward but does require some care to link properly to the parameter variables and constants. The code for the Classea will be used for illustration and its calling procedure and code follows. The capability to write filters is not presently available to CRHM user.

Call

$ea ea(t, rh) ea =desired vapour pressure fot t = temperature and rh (%) = the relative humidity.

C++ Header

```
class Classea : Classfilter {

   public:

   Classea(ClassData *MyObs, String ToVar, String args, String argtypes =
"VV");

   virtual void doFunc(long Line);

}
```

Header discussion

The class constructor always has the same number of arquments. The first 'MyObs' is the class describing the input data file. Next is the class describing the variable generated by the filter. The string 'args' holds the parameter string when the filter class is constructed and finally, a constant string describing the required parameters for the filter. In this case two variables consisting of 't' and 'rh'.

C++ Code

```
Classea::Classea(ClassData *MyObs, String ToVar, String args, String argtypes)
```

: Classfilter(MyObs, ToVar, args, argtypes) {

}

Constructor discussion

The only changes ever made to this section of the code is to the user selected class name.

void Classea::doFunc(long Obs, long Line){

Data[Vs-1][Obs][Line] = estar(Data[0][Obs][Line])* Data[1][Obs][Line]/100.0;

}

Function discussion

This part performs the filters work. The array 'Data' stores all the data read from the data input file plus any new variables generated by filters. The second index, 'Obs' indexes multiple observations for the interval. The third index, 'Line' is incremented every interval, i.e. for every data line read from the data file. The first index references the input filter parameter variables. Variables and constants are counted separately from zero beginning with the first parameter. For example for a filter class whose 'argtypes' is assigned 'VVCVC' the parameters would be accessed as myFilter(V0, V1, C0, V2, C1). A constant is accessed using Constants[0] etc. The filter output variable is always accessed using Data[Vs][Obs][Line]. It follows that this function implements:

$ea ea(t, rh) .

Wind height reference change

$u2 refwind(u, 2, 10, 1) this is a comment, where parameters are refwind(u, Z2, Zm, Ht)

Code

```
class Classrefwind : Classfilter {

   public:

   float Const; // result log((Z2 - d)/Z)/log((Zm - d)/Z)

   Classrefwind(ClassData *MyObs, String ToVar, String args, String argtypes =
"VCCC");

   virtual void doFunc(long Line);

};

Classrefwind::Classrefwind(ClassData *MyObs, String ToVar, String args,
String argtypes)

   : Classfilter(MyObs, ToVar, args, argtypes) {

   if(!Error) {

      float d = Constants[2]*2.0/3.0; // zero plane

      float Z = Constants[2]*0.123; // roughness

      Const = log((Constants[1] - d)/Z)/log((Constants[0] - d)/Z);

   }

}

void Classrefwind::doFunc(long Obs, long Line){

   Data[Vs][Obs][Line] = Data[0][Obs][Line]* Const;

}
```

Comment

This example follows the pattern of the first example but has some extra features. Since the term log((Z2 - d)/Z)/log((Zm - d)/Z) is a constant and not dependent on the input data it needs only to be calculated once. It is calculated

only once in the class constructor. The calculation of the constant is only made if the error checking boolean variable 'Error' is not true. This would be set if the end user had not assigned the filter parameters correctly and avoids divide by zero errors.