CRHM Manual

March 15, 2013

Centre for Hydrology
University of Saskatchewan

Table of Contents

List of Figures

1 Introduction

This document provides the user-manual for the Lower Smoky River Model (LSRM), developed on the Cold Regions Hydrological Model (CRHM) platform. It does not cover every aspect of the platform's capabilities.

1.1 Licence

1.2 System requirements

CRHM is written in C++. The minimum available memory requirement is 32 MB; system files occupy approximately 50 MB of disk space.

CRHM is modest in its demand on the operating system. It does not require complex features of the Windows API such as access to the internet, the Registry or to multimedia. Consequently, CRHM runs well on older versions of Windows, and even on non-Windows operating systems using the Wine compatibility layer, as described below. Therefore incompatibility with future versions of Windows is unlikely.

CRHM is a research model under constant development. New versions are available through the website of the Centre for Hydrology at http://www.usask.ca/hydrology/CRHM.php. New versions are also accessible directly through a DropBox folder set up for AESRD at https://www.dropbox.com/sh/7szk9w42aoxh72x/8HRVyQ1pd8?m.

The software is intended to run on the Microsoft Windows operating system (98, ME or later): however, it will also run under OS X* and Linux, .using the program Wine (http://www.winehq.org/).

**Note – CRHM will work on an Intel Macintosh, but not a PowerPC. To confirm type, click the Apple Logo (on the top status bar), and choose 'About this Mac': there should be mention of 'Intel'.*

1.3 Installation

Instructions for the installation of CRHM on Microsoft Windows, OS X and Linux operating systems are provided in Appendix 1.

2 CRHM Concepts and Basic Terminology

This section provides a high-level overview of the program CRHM. More details are provided in the relevant sections .

CRHM is a modelling platform to construct hydrological models suitable for Canada. It provides a way to assemble algorithm (called *modules*) of the individual physical processes which control fluxes and states of water and energy throughout the hydrological cycle. Once the selected modules have been linked together by the system, they can be run as a hydrological model (*project)*. The model requires meteorological *observations* and *parameters* describing the drainage *basin*. The platform supports the input of different observations (see below) and parameters (such as slope, land-cover, geology and soil attributes) to the modules within discrete regions within a given basin These regions are termed *Hydrological Response Units* (HRUs).Regions within a basin which have more or less similar attributes have similar physical processes and generally have comparable hydrological behaviours The platform provides a method for integrating models for each distinct environment, using different sets of process modules in each. More detail is provided in section 3.2.

Observations are time-series of meteorological data such as precipitation, temperature, relative humidity, wind speed and solar radiation: These basic inputs are available to all modules. The data are supplied in one or more observations files, which also define the maximum range of run dates and the model time step: the contents and format of these files are described in section 2.2.2. In some cases, new versions of observations may be generated by modules included in a model, for its internal use: for example, incoming solar radiation can be adjusted for the slope and aspect in each HRU.

Values generated as outputs and passed between modules are known as *variables.* **T**hese may be plotted on the chart in the main GUI and may also be saved to file.

The key strength of CRHM thus lies in its provision of a flexible modelling infrastructure, which allows the basin to be split into distinct HRUs, and also supports the representation of the physical processes acting in different environments by appropriate modules.

The Lower Smoky River Snowmelt Runoff Model (SRSRM) has been implemented by selecting modules which represent the physical processes at work within the Lower Smoky basin, such as snow redistribution by wind, sublimation, the interactions between topography and vegetation, and the effects of frozen mineral soils during the main melt season. These have been coupled and checked by the system, and the required observations and parameters identified, to provide the ‘turnkey’ model.

2.1 The User Interface

CRHM provides a conventional Microsoft Windows Graphical User Interface (GUI), but the core functionality may also be accessed through a Command-Line Interface (CLI) as a console application. This section first describes the GUI, and then provides details of the CLI.

2.1.1 Main Window

The main CRHM GUI, shown in Figure 1, provides access to the platform's functions, and displays graphical output. This section describes the menu options and other controls.

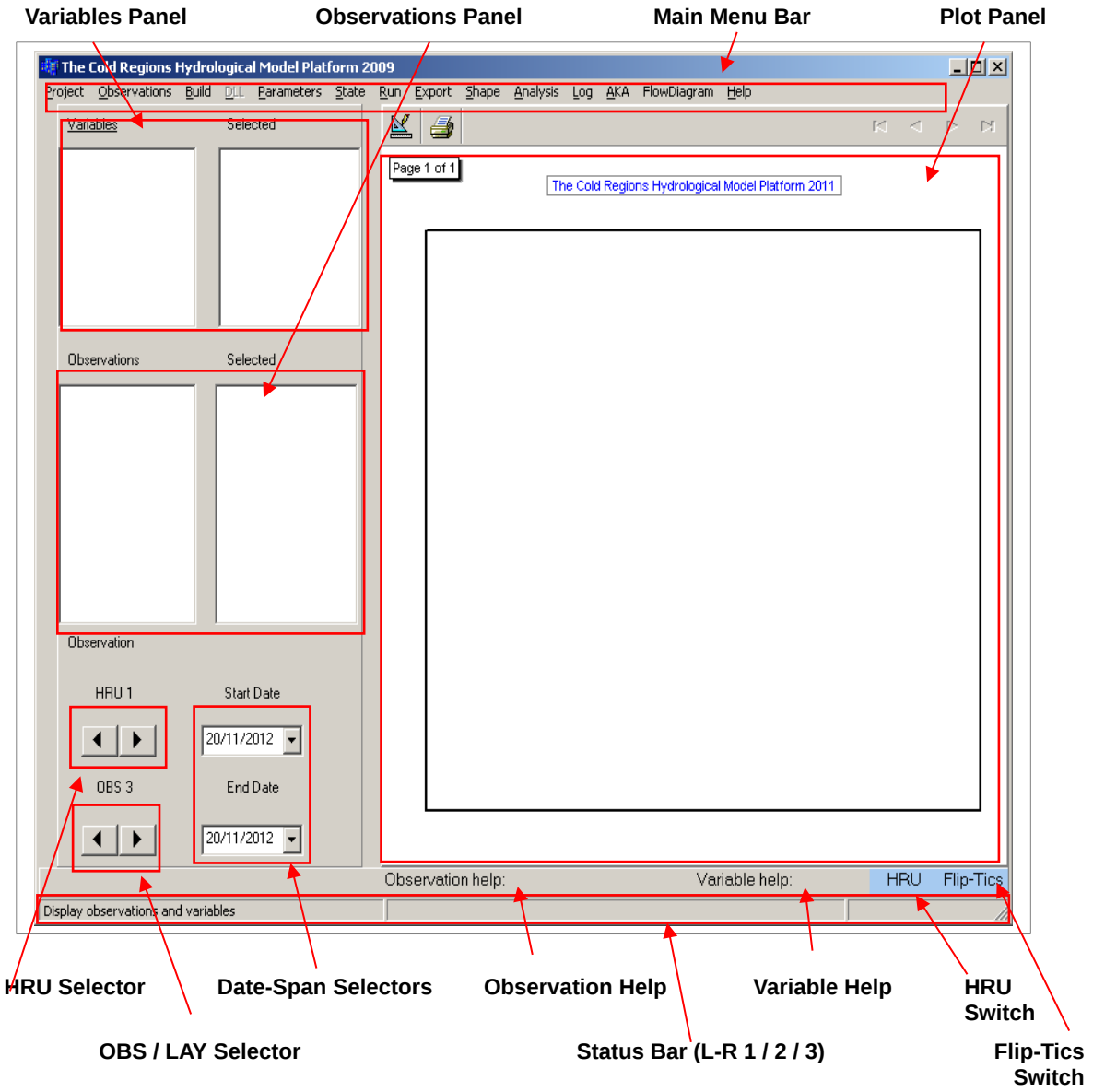


Figure 1: The main CRHM GUI.

2.1.1.1 Main menu

Project Menu

A CRHM **Project** defines individual model, including its modules, input observations, HRUs, parameters and other options set by the user.

Project > Open

Loads the standard 'Open File' dialog. On selecting a Project File (*.prj), the system will load the model's structure, inputs, and other settings into memory, and it will become the current project.

*Note - if the **File > AutoRun** flag was set when the file was saved, CRHM will run the model immediately. If **File > AutoExit** was saved to the file, CRHM will close on completion of the run.*

Project > Save

Saves the current project to file (using the current project name, if already set). The folder or directory into which the file is saved will be used as the project's Project Directory.

*Note - if **File > AutoRun** or **File > AutoExit** is currently set, then these settings will be written to the file, and executed when it is next loaded.*

Project > Save As

Saves the current project to new file.

Project > Close

Closes the current project and erases it from memory: also unloads any open observation files.

Project > Report

Generates reports on the state of the current model, through the Project Report Form.

Project > AutoRun

If this option is checked when a project is saved, the project will run as soon as its file is loaded into CRHM.

Project > AutoExit

If this option is checked when a project is saved, CRHM will close down on

completion of the model run.

Project > Log

Select options for saving selected model outputs (as displayed in the right list-box in the Variables Panel) to a CRHM Output File:

- Last: Save values from the last time-step of the model run only

- All: Save values from every time-step of the model run

Important:

If neither option is checked, a CRHM Output File will not be written.

Project > Save Chart Template

If checked, the chart display settings (layout, colours) will be saved to the Project File.

Project > Plot Refresh Rate

Sets the frequency with which the graphical plot is refreshed during a model run. The refresh frequencies are:

- **daily**
- **bi-weekly**
- **weekly**
- **monthly**
- **yearly**
- **at end**

Note – these options are also available during a model run, by left-clicking inside the chart area and selecting from a pop-up menu.

Project > Freq_Default

Used with a CRHM model requiring no external observations: this option sets the time step.

Note - it is only possible to set this value when no observation files are open.

Project > Exit

Exits CRHM.

Observations Menu

The meteorological observations which drive the model are stored in observation files, which have the extension .obs. This menu manages these .obs files.

Observations > Open

Opens / loads an observation file. Once loaded, its name is appended to this menu and the names of the observations which it contains are listed in the Observations Panel.

Observations > Close All

Closes all open observation files.

Observations > [Obs Filenames]

Every loaded observation file is listed as a menu-item. Choosing one of them (on confirmation by the user) will close the corresponding file.

Build Menu

These options support the selection of the individual modulesof processes which form the model.

Build > Construct

Loads the Model Construction Form.

When building a new model from scratch, this supports the assembly of the individual process modules into a project file.

When a previously-built model has been loaded into the system (i.e. by opening a project file), this menu displays the details of its modules and allows the list of modules to be edited.

Build > Clear Modules

Removes all modules from the current model.

Build > Macro

Loads the Macro Programming Form: this supports the development of scripts, which may be used to test algorithms, to diagnose model output, to add new process modules and to define complete models using groups. More information is provided in section 2.1.3 and in Appendix 4 (Macro.chm).

Build > [Pre-Built Models]

The standard installation includes a number of pre-built models, which are listed on this menu. These are not relevant to the Smoky River model.

DLL Menu

This option is not relevant to the Lower Smoky River Model.

Parameters Menu

Parameters describe the static physical and spatial characteristics of the basin and the HRUs within it, such as area, elevation, soil attributes and land-cover. This menu loads the Parameter Definition Form used to edit the parameters.

State Menu Used to save the state variables (declared within each module by their developers) at the completion of a model-run for a specified time interval to create an Initial State File. This menu provides the following options;

State > OpenInitState: open a previously-saved Initial State File, to be used as the starting point for a new run.

State > SaveState: save the final state of the next model-run to a file.

State > SaveStateAs: save the final state of the next model-run to a new file.

Run Menu

Executes the model in its current configuration. Note that in order to run, at least one output variable must have been selected into the list-box on the right of the Variables Panel (see section 2.1.1.2 for details of how to do this). When the model is run, the chart in normally updated (the data is plotted and the axes are adjusted as required) during each time step. Left-clicking on the chart area during a run will display a pop-up menu with the following options:

- **ignore**: no action
- **terminate run immediately**: stop the run
- **daily update**: update chart at daily intervals
- **bi-weekly update**: update chart at twice-weekly intervals
- **weekly update**: update chart at weekly intervals
- **monthly update**: update chart at monthly intervals
- **yearly update**: update chart at annual intervals
- **update at end of run only**: update chart only on completion of run

Export Menu

Loads the Export Results Form: supports the selection and preview of output variables for export, and provides options for saving them to files in various formats.

Shape Menu

Loads the Shape / Spatial Display Form, in which graphical representations of the basin or HRUs may be coloured according to the value of output variables or parameters.

Analysis Menu

On completion of a run, output variables may be fitted to a range of curve-types through the Output Variables Analysis Form, to which access is provided from this

menu-option.

Log Menu

This option loads the Model Log Form, which (following a run) details any errors and/or warnings for the current model, information reported by modules, and lists information about the observations supplied to HRUs.

AKA Menu

Not currently used by CRHM. Stands for “Also Known As”. This menu was used to define connections between variables and observations. This feature has been deprecated.

FlowDiagram Menu

The CRHM core displays the assembly and order of the various physical process modules in a model , and the logical flow of observations, parameters and variables between them. This option loads the Model Flow Diagram Display to provide a graphical view of the model’s overall structure.

Help Menu

This option provides access to user support, general information, and additional details about the system.

Help > CRHM Help

Displays CRHM Help, which provides standard contents / index / search options.

Help > Modules_New

Provides detailed information about the modules available within the system.

Help > About

Provides information about the version of CRHM.

2.1.1.2 Variables panel

This panel lists the variables generated by the modules within the current model. The list on the left lists all variables: the one on the right lists only those which have been selected as ‘output variables’ to be plotted on the graphical display when the model is run.

The list on the left may be organized in several ways. Left-clicking on the label above it: will cycle through the following options;

- **Variables:** full list of all variables
- **Variables by Module:** lists variables under the module which generates them
- **Diagnostic Variables:** these are variables which are of specific interest when diagnosing
- **Private Variables:** these are variables used internally in a module

Left-clicking a variable in the list will display its full name and units in the right-most field of the status bar, beneath the ‘Variable Help’ label.

The main plot displays output variables for each HRU. To select a variable for display, do the following:

- Set the HRU of interest, using the HRU Selector.
- In the left-side listbox, select (left-click) the variable. Its description and units will be displayed in the right-most field of the status bar. The same units will also be applied to the y-axis of the chart.
- Now right-click the variable name: this will display a pop-up menu with the following options;
    - **Add:** selects the variable, for the HRU currently selected using the HRU Selector: it will appear in the selected variables list, with a numeric suffix identifying this HRU.
    - **HRUs Add:** selects the variable for the current HRU and all others *with a higher index*:e.g., if HRU 2 is selected, and the current model configuration defines 4, then the variable will be selected for HRUs 2 to 4, and all will appear in the list on the right, together with their appropriate suffixes.
    - **LAYs Add:** used by custom soil-moisture modules and not relevant to the Smoky River Model.

- **HRUs LAYs Add:** used by custom soil-moisture modules and not relevant to the Smoky River Model.
    - **HRUs LAYs Add:** as above, but selects the variable for all soil layers, for the current HRU and all others with a higher index.

Note - if a variable is to be available for plotting, it must be selected for display before *the model is run.*

2.1.1.3 Observations panel

When an observation file is loaded (i.e. by **Observations > Open**), the observations in the file are added to the list-box on the left side of this panel.

Left-clicking on a listed observation displays its description and units (as provided in the obs. file), followed by the name of the obs. file in which it was supplied, in the central field of the status bar, beneath the ‘Observation Help’ label.

An observation can have a dimension greater than one. To display a particular column use the OBS Selector.

Right-clicking on a listed observation invokes a pop-up menu. It is also possible to select multiple observations, using standard Windows key combinations (shift+ left click for contiguous sets, control+left click for individual items). The following options are available:

- **Add Ins**: plots the observation in the plot panel: the observation will be listed in the list-box on the right of the panel as **nnn(X)**, where **nnn** is the observation-name, and **X** is the current value of the OBS Selector.
- **AddArray**: where an observation has a dimension greater than 1, this option adds adds every element of the observation array. So in the example of 5 HRUs with a single value in each, the right-side list will display **t(1)** through **t(5)**.
- **Function**: displays a list of functions which may be applied to observations:

Option	Displays
Observation	original data
Watts to MJ/Int	original data converted from Watts to Mega-Joules per time interval
MJ/Int to Watts	original data 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 values
Positive	sum of daily values which are greater than zero
Total	total of interval values over run period
Total/Freq	used to compare totals of daily values with interval values
First	first value of each day only
Last	last value of each day only

After selecting an option, adding the observation to the right-side list will plot its calculated value.

*Note - the chosen **Function** option will continue to be applied for any selected observation until* **Function > Observation** *(to display the raw values), or another function, is chosen.*

To hide the options box, choose an option other than that selected when it is first displayed.

2.1.1.4 HRU Selector

This control selects the output variable HRU to be displayed, as described in section 2.1.1.2.

2.1.1.5 OBS / LAY Selector

The LAY function is not used by the Smoky River modules.

2.1.1.6 Plot Panel

The selected observations and output variables are plotted during the model run.

It is possible to control which observations and elements are displayed using the check boxes

displayed for each observation / variable in the upper-right of the plot. The ‘Flip-Tics’ toggle in the lower-right toggles these boxes: note that left-clicking toggles the variables displayed, and right-clicking toggles the observations displayed.

Note:

To zoom in on the display, position the pointer over the chart, and drag down and to the right (i.e. draw an “L”) to select the rectangular area to be displayed.

To zoom out, drag up and to the left (i.e. draw a “7”).

This button loads loads the plot edit / design dialog, to change the appearance of the chart.

This button loads the standard printing dialog,

2.1.1.7 Status bar

This area is split into three regions. It is used to display additional information related to current selections / options, and to provide information during the model run.

2.1.1.8 Labelling switch

This switch toggles between displaying the HRU indexes and names defined in the 'basin' module. Not applicable to group projects.

2.1.2 Model Construction Form (loaded by Build > Construct menu option)

This form, shown in Figure 2, manages the construction of a new CRHM model , and the editing of existing models, from the available process modules.

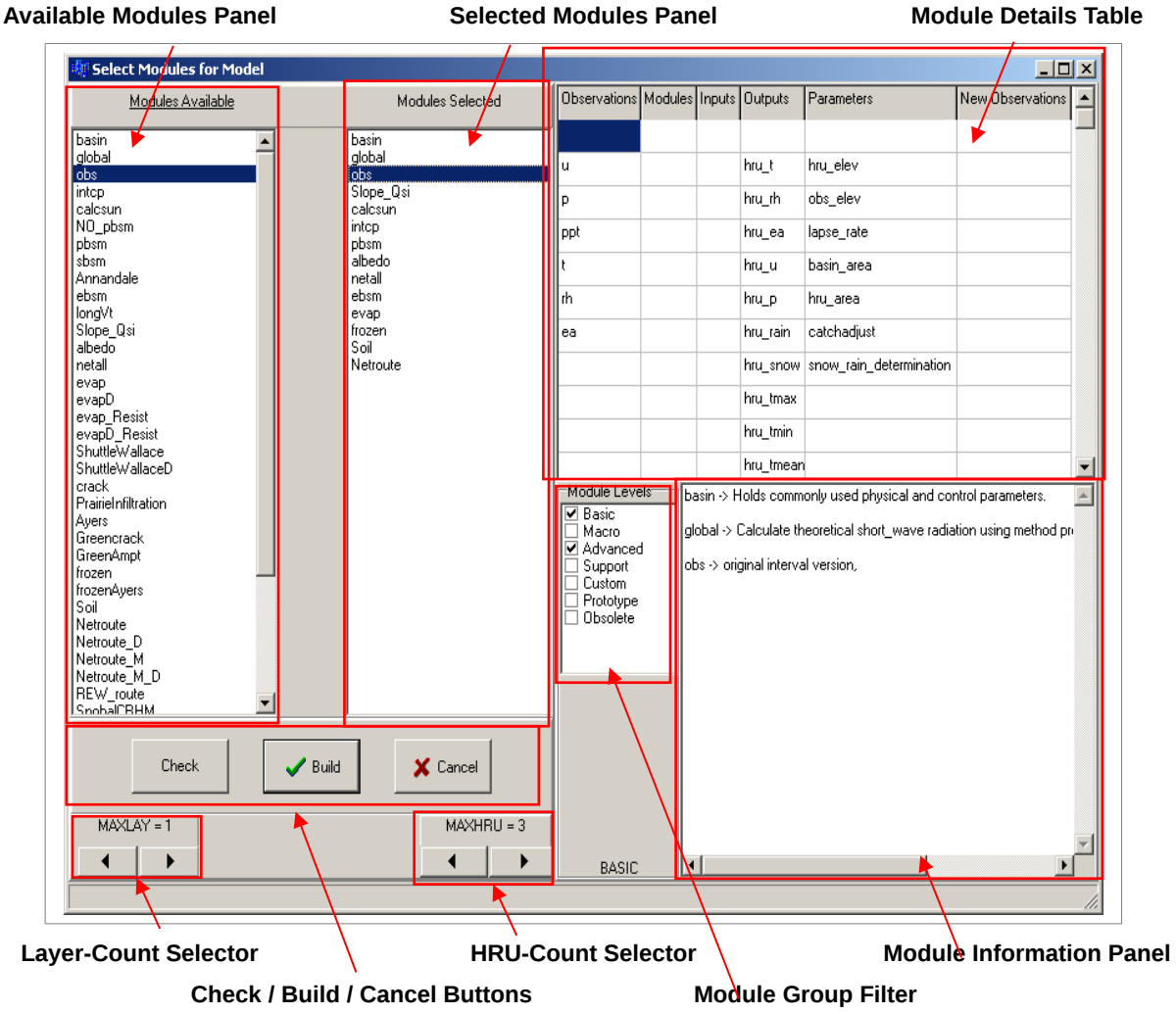


Figure 2: The CRHM Model Construction Form

The Available Modules Panel lists all modules currently available, including 'native' CRHM modules, loaded macros and groups. Use the Module Group Filter check-boxes to add or remove module categories. Left-clicking the 'Modules Available' label above the list toggles between a sorted and unsorted order.

Left-clicking an item on the Available Modules list will display details of the external observations, parameters, and input / output variables associated with that module in the Module Details Table, and additional information in the Module Information Panel. If the module

generates any new observations (i.e., output observation variables, these are listed in the last column. The ‘Modules’ column displays the source of each input-variable: an asterisk denotes that it may be sourced from any suitable module.

The Module Information Panel will also detail any existing variants of a chosen module. For example, module longVt exists in three ‘flavours’

- longVt uses the raw observation of incoming shortwave radiation
- longVt#1 uses a daily average incoming shortwave radiation computed externally
- longVt#2 uses a daily average incoming shortwave radiation adjusted for slope

Left-clicking on a macro in the modules list displays the script in the Module Information Panel.

Right-clicking on a selected item on the Available Modules list displays a pop-up menu with the Add option, allowing the module to be added to the model: it will then be listed in the right-side panel.

The number of HRUs and layers should be set using the count-selector controls.

Note - if either dimension is subsequently reduced, the model will continue to load the reduced parameters appropriately. However, if the count increases, then the model the parameters of the last HRU will be duplicated for any additional HRU(s) / layer(s).

Having assembled the modules of interest, left-clicking the Check button will analyze the input- and output-variables associated with each module, and will determine the logical order in which to run the modules. The system will identify any unsatisfied variable inputs and suggest modules capable of satisfying them. The user must add them manually.

On successful completion of this stage, left-clicking the **Build** button will load the model into memory. This process may be stopped using the **Cancel** button.

Note that a new model will not be automatically saved to a project file: this can be done using the **Project > Save** or **Project > SaveAs** menu-options.

If, however, the model already exists, and has been altered, then the user will be prompted to save its HRU / layer parameters to a parameter file, which it is possible to load into the Parameter Definition Form.

2.1.3 Macro Programming Form (loaded by Build > Macro menu option)

This form enables users to incorporate a relatively simple, but powerful, set of programming instructions into a model.

Macros were originally intended to provide a way of retrieving (and setting) values of observations and variables at specific stages during processing, to assist with debugging new modules, but their use has been expanded substantially. They are also used to link modules which would not otherwise connect by adjusting units, scaling and/or timing. If separate observations are not available for all HRUs within a basin, macros may be used to provide values by interpolation. The model may then be instructed to use these 'synthesized' observations for each HRU rather than the raw values supplied by the observation file.

Macros support the development and testing of new algorithms to represent specific hydrological processes, the definition of model-specific process modules, and even entire models. They may also be used to create sub-models from a set of modules, termed Groups: more information is provided in the CRHM Help Files (search for **declgroup**). Full details of the macro programming language and associated functionality are available in Appendix 4, Macro.chm.

The programming form is a simple text editor, and supports standard cut / paste key combinations. New lines are added by **Ctrl + Enter**.

The **File > Save** and **File > SaveAs** menu-options allow individual macros to be saved to macro files (with extension ***.mcr**).

Note – if any of the text within the programming panel is selected (highlighted) at the point of choosing either of these options, then only this text will be saved to file: to save the entire contents of the panel, ensure that no text is highlighted.

However, whilst the ***.mcr** files are useful for external storage of macro code, they are not used internally by the platform: to save a macro to CRHM, so that it is available for modelling use, use the **Save Changes** button. The **Cancel Changes** button undoes any unsaved changes to the current macro, and returns to the last saved version. Macros are saved along with all other details of a model whenever its project file is saved.

Macros saved to a file may be loaded into the programming form at the current cursor location using the **File > Open** menu-option. Although macro(s) loaded in this way will be included in the project file when it is saved, in order for them to be 'visible' within the model, the user must first exit and then re-start CRHM, and load this project file.

Note – if any text within the programming panel is selected (highlighted) at the point of opening a macro file, the code loaded from the file will overwrite that selection.

The form also provides the means of introducing an existing model (saved to a project file) as a new 'macro group'. This structure would generally represent a commonly-associated set of modules, for which the processing order and input / output have been checked and remain fixed, for a set number of HRUs, as a single logical entity, which may then be incorporated as a sub-system within a larger model. The **File > CreateGroup** menu option prompts for the project file: pressing **Save Changes** adds the group to the system. It will then be visible in the Model Construction Form, from where it may be added to the current model.

2.1.4 Project Report Form (loaded by Project > Report)

Displays summary information about the current project, including its dimensions (counts of observations, HRUs), observations, date-range, modules, parameters and initial and final state. The form provides the following menu options:

File: options for saving the report (to a Report File (*.rpt)) or printing it

Execution: No longer in use, so greyed out.

Chart: No longer in use, so greyed out.

Bld: No longer in use, so greyed out.

Lists: No longer in use, so greyed out.

Global: No longer in use, so greyed out.

AKA: No longer in use, so greyed out.

Hierarchy: lists the modules (and any macros and/or groups) which form the model, together with their input variables, and the modules which provide them.

ExtractGroup: if the model includes any 'macro groups' (models included as sub-systems within a larger project), these are listed individually: selecting any of these will display the same set of details as those provided for the full project by the main report.

2.1.5 Parameter Definition Form (loaded by Parameters menu)

Parameters are static values which represent physical attributes of the basin or its HRUs. These may include area, elevation, slope, aspect, details of vegetative cover, surface texture, sub-surface characteristics, and other relevant details. The parameters of any model depend on those of its modules.

To set the model parameters:

1. Build the model in the Model Construction Form. The number of HRUs are set at this time. Save your project.
2. Open the Parameter Definition Form.
3. Select each module from the list on the left. This displays a row for each parameter, and a column for each HRU, in the table on the right, with default values for each (where provided by the corresponding module).

   Left-clicking on the label above the table cycles through the following options, which control the list of parameters displayed:

   - All parameters
   - Basic parameters
   - Advanced parameters

- Private parameters

The range allowed for each parameter is shown in the extreme right-hand columns of the table. Left-clicking inside a cell will display information about that parameter in the panels towards the bottom of the form. To edit a parameter for a specific HRU, double-click in the corresponding cell.

4. Save the parameters. The parameter values are save into the .prj file when you save the project.

Parameters used by multiple modules are grouped under the 'Shared' category: these are identified by the system when the model is built. When viewing the parameters for a particular module, these shared parameters will be identifiable by a 'Shared' flag.

To over-ride this, and make a parameter value apply only to a specific module:

1. Select the required module
2. Click in a cell in the row for the parameter to be altere
3. The parameters form will be unloaded: reload it, and click the target module:
4. You will see that the parameter now has separate (default) values for each HRU
5. These must now be altered to the required module-specific values
6. Press **Enter** to commit these changes; you will be prompted to confirm.

Note – if a local parameter is subsequently set to the same value as the corresponding shared parameter, and the project is saved and then re-loaded, then the system will again identify it as a shared parameter (i.e., it will apply the same value to all modules in which it is used).

Whenever the model is re-built (i.e., the **Build** button is clicked in the Model Construction Form), the parameter values stored within the project revert to the default values.

2.1.6 Export Results Form (loaded by Export menu header)

This form provides a list of the model's output variables, and options for saving these values to a file in various formats.

The available output variables are listed on the left side of the form: left-clicking any of these adds it to the 'Selected' list. Left-clicking an item in the 'Selected' list will remove it from the list (i.e., de-select it).

When the **Preview/More** button is clicked, the values for each selected output variable, for every modelled time setp, are written to the panel on the right of the form: 1000 timesteps are displayed for each successive click.

The button below the **Preview/More** button, when clicked, cycles through a series of date formats:

- Observation (separate columns for year, month, day, hour, minute)
- MM/DD/YY hh:mm
- Excel (date as single-precision real, time as short integer)
- Excel + Julian + Date

The **Observation** and **Excel** options generate output which is readable by CRHM as an observation file (*.obs).

To save the output to file, using the currently selected format, use the **File > Save** or **File > SaveAs** menu options.

2.1.7 Shape / Spatial Display Form (loaded by Shape menu header)

This form provides options for displaying spatial representation of a model's HRUs as polygons, which may be coloured to indicate variations in a chosen parameter or output variable.

The HRU polygons may be loaded into the system from a proprietary perimeter file (*.per), or a standard ESRI ShapeFile, using the **File > Open** menu option on this form. The system will display a warning if the number of polygons in the file does not match the count of HRUs in the current model.

Note – the polygons must be ordered in the ShapeFile in the same way as the corresponding HRUs: i.e., the first polygon should represent HRU 1, the second HRU 2, and so on.

If detailed polygons of HRU perimeters are not available or required, there is also an option to display a simple grid, in which each cell represents an HRU: to use this, choose **File > Grid**.

To list the output variables available for mapping following a model run, choose the **Display > Variables** option, highlight the variable of interest (which must have been generated by the model for all HRUs), then set the date and time for which the value is to be displayed from the controls below the list.

To see parameters, choose **Display > Parameters**, and select the item of interest.

The display's appearance (colours, shades, scaling, etc.) are controlled by the TeeChart Editor, which is loaded from the 'Edit' icon.

2.1.8 Output Variables Analysis Form (loaded by Analysis menu)

This form supports the fitting of curves to output variables generated by a model run, and for plotting one variable against another. Options for the type of curve are provided by a set of 'radio buttons' in the lower left of the form: if **Polynomial** or **Fourier** is chosen, a selector is displayed to the right of this panel, through which the function's order may be set.

The date-range within which to fit the data to a curve may be set using the selectors in the central-lower area of the form.

On clicking either an output variable listed in the upper-left panel of the form, or the 'Preview' button, the selected variable will be fitted to the chosen curve type, and numerical and graphical displays generated. The numerical output may be saved to file using the form's **File > Save** or **File > SaveAs** options.

The format of the numerical display may be controlled using the button below the 'Preview' button, which will cycle through the following options on successive clicks:

- Obs (separate columns for year, month, day, hour, minute)
- Excel (date as single-precision real, time as short integer)
- MM/DD/YY hh:mm
- X/Y (shows plot co-ordinates)

The appearance of the chart is editable through the form loaded by clicking the 'Edit' icon.

2.1.9 Model Log Form (loaded by Log menu)

This form has two text panels, which show information about a completed model run.

The upper panel lists any issues identified by CRHM relating to the model:

- Errors prevent the model from running
- Warnings potential problems (but possibly set intentionally by user)

The lower panel displays the project and observation filename(s), the date and time of the run, and information provided by each module, as set by the designer of each module during its development: this may include values of internal variables, or warnings of possible problems.

The **Flags** menu header lists special parameters supplied to each module to control their behaviour: for example, module **evap** has an **evap_type** flag to control the method used to estimate evaporation (Granger, Priestley Taylor, or Penman Monteith) within each HRU.

The **Coefficients** menu header lists parameters supplied to each module with continuous values for each HRU.

The output may be saved to file or printed using options on the **File** menu.

The contents of either panel may be removed, using the **Clear > Errors** or **Clear > Debug** menu options.

2.1.10 Model Flow Diagram Display (loaded by FlowDiagram menu)

This form depicts the overall logic of the current model as a graphical representation of the individual modules,the output variables which they generate, and either the observations or parameters supplied to them.

There are three menu headings:

- File: supports printing of the diagram.
- Copy: supports copying of the diagram to the clipboard, as either a bitmap or metafile.
- Selection: on clicking the heading, the diagram toggles between showing observations and parameters as the raw inputs.

Some modules generate new versions of observations, which are then available to all modules which run subsequently: these are donated by a '#' suffix. For example, module **Slope_Qsi** adjusts solar radiation for slope and/or aspect, generating **QsiS#** and **QsiD_Obs#**.

Modules sometimes retrieve variables from modules other than those which run immediately previously. If this is a 'read-only' action it is termed 'Gets': if 'read-write' (i.e., it changes the value), 'Puts'. These are indicated by **G*** and **P*** within the flow diagram.

2.2 File types

CRHM reads input from and writes output to sequential ASCII text files. Note that if you create input-files on a Macintosh or Linux/Unix system will result in an error message unless you save them explicitly to the MS-DOS/Windows format. If your text editor or spreadsheet does not provide this option, use a program like ToFroDOS (which is a free program) to convert the files to the MS-DOS/Windows format. ToFroDOS is available (as of March 2013) from http://www.thefreecountry.com/tofrodos.

2.2.1 Project Files (*.prj)

Project files are generated by CRHM to store the full defininition of a model: they include details of the Observations, HRUs, Parameters, Variables, Modules and various run-time and display options set by the user. They should not be edited manually, except by advanced users who are aware of the risks!

2.2.2 Observation Files (*.obs)

The meteorological or climate time-series data required as the fundamental inputs to any model implemented within CRHM are supplied in one or more Observation Files, with one file for each station or instrumentation array. Observation Files have a filename extension of *.obs*, and comprise Header and Data Sections. Specific sets of observations may be associated with a particular HRU (see section *2.2.2.2* for more details).

The most basic CRHM models use a single obs file. The simplest possible set of required observations is temperature and precipitation. Most CRHM models also require relative humidity and wind-speed. Solar radiation is required by many modules, although it can be estimated from air temperatures using the Annandale method. If a module or macro requires observations which cannot be calculated, then these values may also be specified in the .obs file.

Note that the system depends on the following labels / identifiers for the standard observations:

Obs	Description	Units
T	air temperature	ºC
Rh	relative humidity	%
U	wind speed	m/s
P	precipitation, as actual data for every interval	mm/interval
Ppt	precipitation, as daily data, repeated for every interval	mm/day
Qsi	incident short wave	W/m^2
Qso	reflected short wave	W/m^2
Qn	net radiation	W/m^2
Note: all observations are assumed to apply to a point just above the ground surface or canopy		

The interval between observations may be of any duration, but must be consistent throughout the dataset, with a value supplied for every time step.

Note: it is possible to configure a CRHM model comprising modules which require no external observations: in this event, the modelling timestep is set using the ***Project > Freq_Default*** *menu option.*

Header Section

The file header *must* begin with a single label (or comment / description) line, which describes the file.

The label line is followed by the *Data Definition*, which comprises a line for each observation variable. Each line contains the following, which are delimited by one or more spaces or a tab:

1. The variable's unique label or identifier

2. The variable's dimension: this is the number of values provided for this observation for each time step. For example, for 5 HRUs, each observation would have dimension = 5. See note below on HRU / Observation Indexing.
3. Its units in parentheses, using the standard CRHM format. Note: CRHM checks the observation units with those expected by the modules in the model: if found to be different, details are reported to the Log Screen.
4. Optionally, a comment (e.g. instrument type and height)

After the *Data Definition*, any ***Filter Definitions*** are listed. These provide functions for a variable before it is used. These must be prefixed with a **$** as the first character in the line (i.e., column one).

The end of the Header is marked by four (or more) pound / hash (#) symbols. This may be followed, on the same line, by a series of column-headers providing the observation identifiers.

Comment lines, prefixed by **$$**, may be included in the header section. No extra lines are allowed (including empty ones), and there must be no empty spaces at the start or end of any line.

This is an example of the header of an .obs file (Badlake73_76.obs). Comments are in blue.

```
made from Bad4.mms ..........Label/comment line
t 1 (°C) ..........Variable identifier
rh 1 (%) ..........Variable identifier
u 1 (m/s) ..........Variable identifier
SunAct 1 (h) ..........Variable identifier
ppt 1 (mm/d) ..........Variable identifier
Qsi 1 (W/m^2) ..........Variable identifier
Qso 1 (W/m^2)..........Variable identifier
Qn 1 (W/m^2) ..........Variable identifier
$Qsi mul(Qsi, 277.8) ..........Filter identifier
$Qso mul(Qso, 277.8)..........Filter identifier
$Qn mul(Qn, 277.8) ..........Filter identifier
$ea ea(t, rh) (kPa) vapour pressure ..........Filter identifier
########################################## ..........End of header
1973 1 1 1 0 -15.50 81.50 3.10 0.00 0.00 0.00 0.00 -0.09
1973 1 1 2 0 -15.60 81.30 1.40 0.00 0.00 0.00 0.00 -0.09
1973 1 1 3 0 -15.50 81.30 3.10 0.00 0.00 0.00 0.00 -0.08
```

2.2.2.1 Data section

The data section requires a line for all observations made at a particular time, comprising;

- A date / time value
    - Formatted either as **yyyy mm dd hh mm** or as standard Microsoft decimal time

(days since Dec 31, 1899) to at least 5 decimal places.

- Gaps in the data must be filled with a consistent and identifiable numeric flag (e.g. **9999**), and handled appropriately using a filter.
- The chosen date/time format must be used consistently within a file (i.e. – they cannot be mixed).
- The dataset in the file must always begin one interval into the day, and end at midnight of the last day. The usual modelling interval is 30 or 60 minutes, so each set of daily data would begin at 00:30 (30min) or 01:00 (60min), and finish at 24:00 or 00:00.

- The observation values
  - In the same column-order as the row-order in which they are listed in the definition section, separated by spaces or tabs.
  - Observations with multiple dimensions should use a new column for each element: e.g., a profile / transect with 5 values will have 5 columns for each observation in each time step (in consistent order).
- No comments or empty lines are permitted, and there must be no empty space at the start or end of any line.

2.2.2.2 HRU-Observation indexing

Observations must be associated with the HRU to which they apply. You do this by using the Parameter Definition Form, shown in Figure 3.

The steps are:

1. Left-click the **Parameters** menu header
2. In the list of modules, left-click **obs**
3. Left-click the label above the table on the right until it reads ‘Advance parameters’
4. In the first (grey) column of the table on the right, there will be rows for;

   HRU_OBS[1]: temperature, relative humidity and vapour pressure

   HRU_OBS[2]: precipitation

   HRU_OBS[3]: wind-speed

   HRU_OBS[4]: incoming short-wave radiation

   HRU_OBS[5]: other observations
5. For each row, each column corresponds to an HRU
6. Specify the number of the observation to be applied to each HRU, being sure to press **Enter** each time

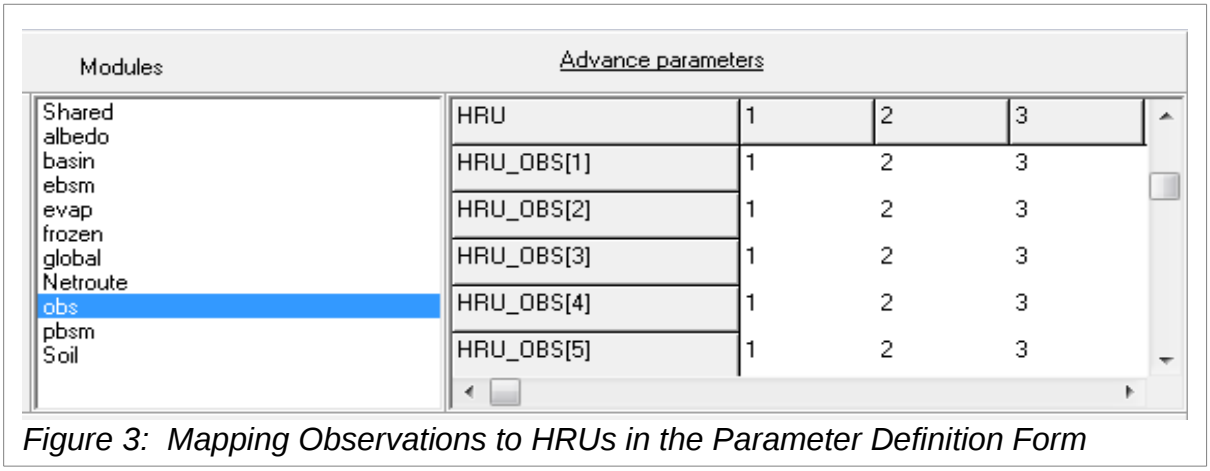


Figure 3: Mapping Observations to HRUs in the Parameter Definition Form

Note – There may be fewer sets of observations available than there are HRUs: the system is able to apply a given observation to any HRU. e.g., if there is only one set of observations available for these three HRUs, each column would be set to '1': if there are two sets, they would read '1' or '2', depending on which was more appropriate.

2.2.2.3 Observation File Types

CRHM accepts observations in a single file or in multiple files: however, all observations of the same type must be in the same file (i.e. all columns for temperature must be in a single file, they may not be split over two or more files).

The system obtains details from Observation Files depending on the order in which they are added to a Project, and their contents.

First File

The first Observation File loaded for a Project: the system determines the time step of the model from the interval between individual data values in this file, and also determines the total period over which the model runs.

The First File must have no missing values: if these exist, they should be filled either

- By interpolation between the available values
- With a consistent and identifiable numeric flag (e.g. **9999**), and handled using a filter.

Interval Files

These are subsequent continuous Observation Files, matching the start and end dates/times of the First File, with an observation interval less than or equal to one day. Interval Files must have no missing values.

First files and Interval Files are the easiest to use for modelling, as they represent complete, consistent datasets, so no status-checking is required to handle missing data or date mis-matches.

Short Interval Files

These are subsequent Observation Files which partially overlap the First File. They must have no missing values within their date / time-ranges. The model will run only for the period of overlap between the Short File and First File.

Sparse Files

These are subsequent Observation Files with an interval different from that in the first file: a filter or macro must be used to infill the gaps.

2.2.3 Macro Files (*.mcr)

These files store code written using the platform's macro-development functionality (see section 2.1.3).

2.2.4 Initial State Files (*.int)

These files store details of the outputs at the completion of a model-run for a specified time interval. They are written and read by the system, and it is dangerous to edit them manually

2.2.5 HRU Polygon Definition Files (*.per)

These files provide definitions of HRU polygons to the system, for use in the spatial display loaded by the **Shape** menu-option.

The first line of the file is not read, so may be used for a label, comment or description.

It is followed by repeated sections of the same format, one for each HRU, comrpising;

- Separator: one or more # symbols (only one separator line between HRUs).

2.2.6 Output Files

- Separator (one or more # symbols.)
- HRU name (as will be displayed on the plot)
- The X, Y values defining the polygon vertices should then be added, separated by spaces: these may span several lines

If either of the **Project > Log > Last** or **Project > Log > All** menu options are checked at run-time, the outputs will be saved to a file. The first option saves only the values at the end of the run, the second saves those for every time-step through the run.

The output file is written to the directory containing the model's project file with a base filename of *CRHM_output.txt*. If the basin ***RUN_ID*** parameter (NB – not the basin variable ***run_ID***) is positive, then this is appended: e.g. ***RUN_ID*** = 123 would save to *CRHM_output_123.txt*.

The first line of the file contains the label for the model time, and the selected variable names. Subsequent lines have the model time in the first column, followed by the variable values for that time-step. This tab-delimited format is easily imported into most spreadsheets and other applications. Model outputs may also be saved using the **Export** menu, which is more flexible, but cannot be automated.

2.2.7 Report Files (*.rpt)

The system supports the saving to file of thefull set of details of a model run, as viewed through the Project Report Form. Sections include;

- Model description
- Date / time of report
- CRHM Version
- Name of project file
- Model dimensions
- Details of AKAs
- Observations supplied
- Model date-range
- Modules included
- Parameters
- Initial state
- Final state

2.3 Command Line

CRHM may be run from the command line, by specifying the executable, followed by the appropriate project, observation and parameter files: e.g.,

```
CRHM.exe Proj1.prj Obs1.obs Obs2.obs Params1.par Params2.par
```

This enables a model to be run without using the GUI. In order for output to be saved, the **Log_All** or **Log_Last** flag must be included in the project file.

Notes:

- *Long paths / filenames should be enclosed in double quotes.*
- *The CLI implements no error handling: any errors will stop the load / run, with the error being displayed.*
- *Only one project file is permitted: any file after the first with a .prj suffix is ignored*

3 Building a model

This section outlines the steps involved in building a model on the CRHM platform.

3.1 Constructing a project

The project is the basic management unit for a model implemented in CRHM: the system writes all its details (describing the process modules and macros, observations, parameters and output variables, and other settings applied by the user) to a project file. This file may be created at any point using the **Project > Save** or **Project > SaveAs** menu options: this must be repeated after every step of the construction process.

The process has the following stages;

1. Inspect the basin: identify the physical processes thought to be acting within it. Identify HRUs, and potentially Groups of HRUs with distinct characteristics and therefore physical processes.
2. Set the number of HRUs to be modelled (Model Construction Form).
3. Select modules to represent the physical processes (Model Construction Form): check that CRHM is able to build a complete model from the selected modules: if not, add required additional modules and repeat. Build and save the model.
4. If required, write macro scripts to connect modules, provide diagnostic outputs, or add new processes: add them to the model, re-build and save it.
5. Prepare observations file(s).
6. Set HRU parameters, and associate observations with HRUs (Parameter Definition Form).
7. If required, set observation / variable substitutions (AKA / Module Customization Form).
8. Run the model.

3.2 HRU delineation and biophysical parameter selection

Basic implementations of CRHM apply a set of modules to a basin, in which the physical processes at work are broadly comparable. Sub-areas of the basin, termed Hydrological Response Units (HRUs), may have different biophysical or hydrological parameters, and / or different meteorological observations. The number of HRUs to be modelled must be set *before any modules are selected*, using the HRU-Count Selector on the Model Construction Form.

Parameters are generally static values, such as slope, land-cover, geology and soil attributes: these are set for each HRU through the Parameter Definition Form (section 2.1.5). However, they may sometimes be dynamic – for example, crop height increases through the growing season. Dynamic values may either be supplied as observations (under the fifth ('miscellaneous') category), or adjusted programatically within the model using a macro. More details are provided in the CRHM Help Files in Appendix 4.

Where biophysical characteristics vary more widely across a basin, distinct landscape units would be expected to host similar physical processes and have comparable hydrological

behaviour: an example would be a basin which transitions from alpine to foothill-forest to prairie. The platform provides a method for integrating models for each distinct environment.

3.2.1 Principles

In an individual model, the same modules are run for all HRUs, with the outputs depending on the observations and parameters specified for each. Some modules provide additional options, which allow fine tuning – for example, the **evap** module supports modelling of interval evaporation using one of three different techniques, which may be set separately for each HRU.

HRUs need not be spatially contiguous, but must form a logically consistent component of the basin's overall structure. The outputs from each modelled time step are routed in a cascade through the entire basin, so there must be a clear order from 'higher' to 'lower' HRUs. The system provides methods for associating each HRU with its relevant biophysical / hydrological parameters and meteorological observations.

HRUs should therefore have consistent

- Biophysical structures: soils, vegetation, slope, elevation, aspect and area
- Hydrological states: snow water equivalent, snow internal energy, intercepted snow load, soil moisture and water table
- Hydrological fluxes: snow transport, sublimation, evaporation, melt discharge, infiltration, drainage and runoff

3.2.2 CRHM-tools

An open source Python Graphical User Interface (GUI) application, CRHM-tools, was developed to allow for the automated and systematic creation of basin relationships for input to CRHM. The open source Python language was chosen for its cross-platform compatibility, its readable novice friendly syntax, and ability to allow for rapid application development. Because of the strengths in the Python architecture, a modular tool system, akin to what exists in ArcGIS, was implemented allowing for a user to easily and seamlessly create their own functions. These functions allow for terrain-based calculations, such as slope, aspect, and fetch, or statistical classification such histogram binning. Manual classification is also possible, allowing for a user to fine-tune the results. Due to the reliance upon the respected open source GDAL library, any single-band raster is compatible (such as ArcGIS rasters), including masked 'no-data' regions. HRUs are generated by combining various classified 'primary landclass' data into HRUs, and parameterizing the HRUs using 'secondary landclass' data.

3.3 Module selection

The core of a model implemented on the CRHM platform is the suite of modules representing the physical processes at work in a basin. These are chosen in the Model Construction Form, as described in section 2.1.2. The **Help > Modules_New** menu-option on the CRHM GUI loads a comprehensive library of descriptive details for every module, including its purpose, the input parameters, observations and variables it expects, and the variables it generates.

The modules included in a model will depend primarily on the basin's environmental characteristics: for example, in alpine basins slope, aspect and shading are likely to influence snow accumulation, ablation and melt: in a forest setting, canopy interception / ablation / infiltration and long-wave re-radiation would be important: on the prairie, the effects of wind re-distribution of snow, frozen soil infiltration and crop growth would be more relevant.

CRHM also supports the assembly of groups of modules into sub-models: *Structures* are 'parallel' module collections, only one of which is executed for any given HRU, while *Groups* are 'series' module collections, run in sequence for all HRUs with which they are associated. The CRHM Help Files, which are listed in Appendix 4, provide additional information under 'Groups and Structures'.

3.4 Structures

When the HRUs in a basin differ significantly, it may be necessary to model the processes using different modules. It is also sometimes useful to be able to apply different sets of modules to the same physical setting, to compare the performance of competing representations of the same process, or to force alternative logical paths depending on the inputs supplied (e.g., a basin may be a wetland in wet conditions, or grassland in a dry year). These requirements are supported in CRHM by Structures .

Structures are built by writing a script detailing each sub-set of modules, in the Model Construction Form (section 2.1.2): this script then becomes part of the larger model, being incorporated in the same way as any other basic module.

The application of each structure to each HRU is controlled using the **HRU_struct** parameter .

More details are provided in the CRHM Help Files, in Appendix 4, in under **declstruct**.

3.5 Groups

In larger and/or more complex basins comprising several distinct hydroclimatic settings, a model may be built of a series of sub-models, one for each environment, using Groups. Groups may also be used to execute the same set of modules using different parameters and/or observations.

The modules and number of HRUs in each group are detailed through a script written in the Macro Programming Form (section 2.1.3): this is incorporated into the larger model in the same way as for Groups.

It is also possible to load an existing project as a new group, through the same form (**File > CreateGroup**): thus, the sub-model for each distinct setting may be developed and tested separately, and then integrated into the larger model.

Appendix 1 CRHM installation instructions

1.1 Install CRHM on Windows

If the CRHM installation software was downloaded as a zip-file, unzip it to a folder on your computer, and run **`setup.exe`**. If the software was supplied on digital media, run **`setup.exe`** from the disk. You will be prompted to accept the licensing conditions, and to confirm the installation folder.

CRHM will then be ready to run from the 'Start' menu. It is also possible to run CRHM by double-clicking the name of a project file (*.prj) or observation file (*.obs) in a folder window.

Note: when installed on some configurations of Windows XP, the following error-message may appear when CRHM is run:

Appendix1Figure 1: Error which may occur when running CRHM with some configurations of Windows XP

If this happens, running the WindowsXP-KB935448-x86-ENU.exe patch should solve the problem. As of November 2012, this is available from the Microsoft web site at www.microsoft.com/en-us/download/details.aspx?id=24529.

1.2 Install CRHM on Apple OSX

Using an account with 'Administrator' privileges,

- First, confirm that you have **Java** installed: open Terminal (search in Spotlight) and type **java -version**. If Java is installed, you will be shown a build and version number. Otherwise, will say **command not found** will be displayed.

  If you need to install Java:

  - On OSX 10.7 or 10.8 run the Java Preference application, at

    http://helpx.adobe.com/x-productkb/global/install-java-jre-mac-os.html

  - On OSX 10.6 install from Apple;

    https://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=20719

  - On OSX 10.5 install from Apple;

https://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=20720

- Install **XCode**:
  - On OSX 10.6 or above, install from the Mac App Store;
    https://itunes.apple.com/us/app/xcode/id497799835
  - On OSX 10.5 or below, install from Apple Developer Website (free - sign in with an Apple ID);
    https://developer.apple.com/downloads
- Install **XQuartz**, from http://xquartz.macosforge.org/landing/
- Install **MacPorts**, from http://www.macports.org/install.php
- Open Terminal and type the following:

```
echo export PATH=/opt/local/bin:/opt/local/sbin:\$PATH$'\n'export
MANPATH=/opt/local/man:\$MANPATH | sudo tee -a /etc/profile
```

then

```
if [ `sysctl -n hw.cpu64bit_capable` -eq 1 ] ; then echo
"+universal" | sudo tee -a /opt/local/etc/macports/variants.conf;
else echo "not 64bit capable"; fi
```

finally, if XCode was installed successfully

```
sudo xcodebuild -license
```

- Install **Wine**: this may take over an hour to compile

```
sudo port install wine
```

- Install **CRHM**

```
wine setup.exe
```

- CRHM will now be installed at

```
cd ~/.wine/drive_c/Program\ Files/CRHM
```

To run the system, use

```
wine CRHM.exe
```

- To add a CRHM icon to the dock, launch the AppleScript editor (**Script Editor** on 10.4 or 10.5, **AppleScript Editor** on 10.6 or higher), and create a new script;

```
tell application "Terminal"
 do script "/opt/local/bin/wine ~/.wine/drive_c/Program\\
Files/CRHM/CRHM.exe"
end tell
```

Press **compile**, and save the file: drag the saved file to the dock.

Credits: Steps from http://www.davidbaumgold.com/tutorials/wine-mac/#part-1

1.3 Install CRHM on Linux

CRHM (and other Windows programs) can be run under Linux using Wine (http://www.winehq.org). Note that Wine only works on X86 systems.

The steps are:

- Install Wine using your package manager

  On Debian-based systems (i.e. Debian, Ubuntu, Mint) you can use the command

  `sudo apt-get install wine`

  On Ubuntu you can also use either the Synaptic Package Manager or the Ubuntu Software Centre to install Wine.

- Use Wine to execute CRHM

  From the command line or a script, the command to open the CRHM GUI is

  `wine /path/to/CRHM.exe`

  where **`/path/to/`** is the appropriate path on your system.

  Note that **`CRHM`** is capitalized.

  CRHM may also be executed specifying the project file, i.e.

```
wine /path/to/CRHM.exe /path/to/projectfile.prj
```

If the project file is in the same directory as **CRHM.exe**, and this is the default directory, then the paths to **CRHM.exe** and the project file can be omitted.

It is also possible to execute CRHM from your file manager. In Ubuntu, when Wine is installed, the file manager (Nautilus) will recognize .exe files as Windows executables. Just right-click the file and select the option **Open With Wine Windows Program Loader**.

Appendix 2 Creative Commons Public Licence

Creative Commons Public Licence

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

1 "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

2 "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

3 "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

4 "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

5 "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

6 "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

7 "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

8 "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

9 "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,
- to Distribute and Publicly Perform the Work including as incorporated in Collections.
- For the avoidance of doubt:
  - Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
  - Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
  - Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested.

If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(b) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

- If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

- No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

- This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

- The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at **http://creativecommons.org/**.

Appendix 3 GNU Public Licence (GPL)

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same

freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2)

tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source

includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under

the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to

produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source

may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because

modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately

under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on

those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered

work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is

in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all. For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work. The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the

combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY

OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE

PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF

SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with

the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

More information about the GPL, is available at http://www.gnu.org/licenses/gpl.html.

Appendix 4 CRHM Help Files

CRHM.chm

Modules_new.chm

Macro.chm

Table of Contents

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:

<u>Basic functions.</u>

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.**

<u>Housekeeping functions.</u>

1. **Save and Load project files to allow the model (project) to be saved as an entirety which can be later loaded and run.**
2. **Help for operating the CRHM platform and help describing the functionality of the module, variables and states.**
3. **Exporting the model output to files for use by other applications (e.g. Microsoft Excel).**
4. **Exporting the model output for later input to compare with other CRHM model runs with different parameter values.**
5. **Statistical and graphical tools to analyze input data and the model performance.**

6. **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.**
7. **Model output may be superimposed upon HRU outlines to aid spatial visualization of the model results.**
8. **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.**
9. **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.**
10. **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.**
11. **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.**
12. **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 or NT.**
- **A graphics display compatible with Microsoft Windows.**
- **At least 32 megabytes of available RAM.**

Introduction.

The CRHM program is installed from an installation CD-ROM. There are many versions of Windows, 98, NT, Me etc.. The program will run on most of these operating systems without any trouble.

There are two basic versions of the CRHM program. The first CRHM.exe is distributed with all available modules built-in. The second is CRHM_DLL.exe which allows the end user to create their own modules as a DLL using a CodeGear C++ Builder with limited knowledge of C++ and combine them with the current module library. Both of the programs require supporting library DLLs in the same directory.

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

Installing CRHM.

1. **Run 'Setup.exe' on the CRHM program CD-ROM.**
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**

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.

If you are new to CRHM and need help to get started, this section describes the menus used to operate the CRHM model. The observation file (Winter1974.obs) and project file (PBSM74.prj) allow the user to explore the operation of the model before becoming involved with their own model and data.

The instal CD-ROM contains four versions of the model.

1. **Basic - CRHM.exe This is the generic version and should normally be used. It is not extendable and only contains the standard modules.**
2. **Extendable - CRHM_DLL.exe. This model is similar to the basic version but allows the user to add their own modules.**
3. **Customized - CRHM_Quinton etc.. These versions are specially created using only selected modules and may or may not be extendable.**
4. **Extendable - CRHM_DLL_NONE.exe. This version is the same as the Extendable version but has no built in library of modules.**

All versions of the CRHM model use the same help file *CRHM.chm* **to describe the operation of the CRHM platform. CRHM will automatically load the** *Module.chm* **help file to provide assistance with the basic modules. DLLs will automatically load** *module name.chm* **help files when available from the writers of modules.**

Run CRHM using one of the following methods.

1. **Select "Start/All Programs/CRHM Programs" and then select the version of CRHM required.**
2. **Double click on the desired CRHM name or icon 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.**

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.

AutoRun.

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

Log.

When a model is run Log provides two options for saving the model ouput to a "*.txt" log file suitable for importing into another application.

1. **Last - The values of the selected variables for the last time step of the model run are saved into the log file.**
2. **All - The values of the selected variables are saved to the log file for every time step of the model run.**

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. Note that the basin parameter RUN_ID is used and not the basin variable run_ID.

The format of the text file is as follows. The first line of the file consists of the text "time" followed by the selected variable names. 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.

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.

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.

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.

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.

DLL menu.

The CRHM module library can be extended by loading DLLs containing other modules.

Clicking on *Open* opens a standard dialogue box. After a Dll file is opened its file name will be added to the end of the DLL pull down menu. The DLL may be unloaded by clicking on its name or clicking *Close All.*

The modules loaded from a DLL will appear in the Build/Construct modules available listbox. If a module loaded from a DLL has the same name as an existing module a warning messagebox will appear before it is replaced. When a replaced module is removed the original module is not restored.

If a model is saved as a project file uses a manually loaded DLL, the DLL will be automatically loaded the next time the project is opened.

If the author of a DLL module has compiled a help file named "module name.chm" and it is in the same directory as the DLL containing the module, the help file will automatically be added to the Help menu.

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 HRUs, 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.

Run

CRHM is executed by selecting the Run feature from the CRHM pull-down menu. The Run feature 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.

This is done by left clicking on the required output from the Variables box. The selected output will now appear in the Selected box, located immediately to the right of the Variables box, followed by a number in brackets. The number in brackets is the HRU selected. If the user requires the same output for a different HRU, the toggle arrow on the HRU box, located immediately below the Variables box, is changed and the output is reselected from the Variables box. There will now be two of the same variables selected in the Selected box followed by different numbers in brackets.

DO: select

1. **Left click evap (from Selected box)**
2. **Right click evap (from Selected box)**
3. **Left click the right toggle arrow of the HRU box (should change to 2)**
4. **Left click evap (from Selected box)**
5. **Right click evap (from Selected box)**
6. **Run**

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

The Graphical Output window should display the evap output for HRUs 1 and 2 in green and red. There are several options on the tool bar directly below the CRHM pull down menu that allows the user to alter the graphical output. It is typically at this point that the user would use the Edit icon (identified below) to ready the chart for printing.

The model run may be aborted by right clicking on the plot border. Select OK from the dialogue box to terminate run or cancel to continue.

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.

Shape.

The Shape screen allows model parameters and output to be overlayed on a map. The coordinates of the polygons defining the HRUs are input to the program and then the values of the Model parameters or output for the individual HRUs determines the colour of the HRU polygons. There are two possible sources of 'shape files'. The first is a text formatted file with extension '*.per', which can easily be constructed by the user. Alternative shape files generated by a G.I.S. in ArcView format (*.shp and *.shx) can be read.

File Menu.

- This dialogue allows the user to choose the file containing the HRU polygon information. The default extension is '*.per' or '*.shp' . There should be a polygon defined for each HRU in the shape file. Warnings will be reported for too few or too many shapes.

Variable Display.

Only one multi-dimensional variable can be displayed at a time. Avalue must have been calculated for every HRU in the model run or an error will be reported.

1. On first entering the "Shape" screen select "File". "Open" allows the user to select a "Shape" file or a user constructed "*.per" file. The other choice is a grid of boxes.
2. Next, select either a parameter or variable to be displayed using "Display"..
3. If a variable has been selected, now select the desired date and time to be plotted. Remember that the model has to be run first so that data is available.
4. It will now be necessary to use the TeeChart Editor to achieve the desired chart appearance.

TeeChart Edit Chart.

Select Edit icon (triangle etc.).

Select "Series/series1".

Select "Grid 3D/Palette".

Select "Step" to select the step size.

Select "Steps" to select the total # of steps. The selection to the right is a step size multiplier.

The style can be selected from; Pale, Strong, Gray scale, Inverted Gray and Rainbow.

The scaling can be extended down to zero by using "Use palette minimum". This also starts the first step at 0.0 instead of the value of the first point.

An alternative colour range selection can be used by selecting "Range" and then selectiong "Start" and "End" colours for the range.

Time Selection.

- The date and time are selected using the date and time controls on the lower left of the screen. These are constrained to the time period of the model run.

Parameter Display.

- Allows parameter values to be displayed on the basin outline.

ArcView Shape Files.

- The two files with extensions, *.shp and *.shx must be available.

File Format for "*.per".

1. The first line is a comment line not used by the program.
2. Followed by a line of '####'. Note that only one pound sign is necessary.
3. First HRU name for plotting purposes
4. The polygon X-Y pairs follow on successive lines using one or more spaces as separators between X, Y data points and pairs. As many points as desired can be put on a line and pairs of X-Y points may be split over lines.
5. Followed by a line of '####' to indicate the end of the polygon data for this HRU.
6. items 3, 5 and 6 are repeated for every HRU in the model simulation.

Example File (*.per).

Comment

######

HRU 1

0 0 1 0 1 1 0 1 0 0

######

HRU 2

0 1 1 1 1 2 0 2 0 1

######

HRU 3

1 0 2 0 2 1 1 1 1 0

######

HRU 4

1 1 2 1 2 2 1 2 1 1

######

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 Log 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 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.

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.

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 files. 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 *FILLITREND* 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	Comments
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						unknown/snow/rain
albedo						
discharge rate	m3/s					
soil heat flux	W/m2					
Parameters Basin/HRU						
area	hectares					
slope	degrees					
aspect	degrees					
vegetation cover type						plus area
soil type						plus area
elevation	meters					
latitude	degrees					
vegetation cover height	meters					
vegetation cover density	percent					summer/winter
interception storage capacity	mm					
leaf area index	m2/m2					
fetch distance	Meters`					
soil properties						texture/storage/conductivity/rooting depth

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 for t = temperature and rh (%) = the relative humidity.

$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 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 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 sine(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. Alocal 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].

decllocal, 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.

decllocal, 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. Atypical 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. Amacro 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.**

Introduction

The modules written for the CRHM Platform are programmed as members of a C++ class called ClassModule. There are many advantages derived from making the modules class members but the most obvious to the casual modeler is the isolation from the main program coding. As a result of this isolation, less knowledge of the complexities of programming in C++ is required to write a CRHM model module.

The user's modules communicate with the CRHM platform through a well defined interface that is straightforward and easily understood. There are some storage concepts of *pointers* **and** *storage allocation* **which are used but these parts are handled by following the** *rules* **demonstrated in the examples.**

To make the CRHM platform recognize the custom user modules, calls to a class Administer are made by the user. The user first identifies the new modules written. Next, defines any user defined models as a list of existing modules and the newly defined modules. Finally, a call is made to a routine which executes when the CRHM model loads the DLL at run time.

Module Description

A model module uses some or all of the following CRHM input/output variables;

- **Observation Variables – the climatic input data stored in data files at the time interval used for model simulation. Their values cannot be changed in a module.**
- **Observation Function Variables – calculate mean/minimum/maximum/daily total/total/ or access to the first and last values of the day. Their values cannot be changed in a module.**
- **Parameter Variables – the physical constants used by the model like elevation, latitude, instrument heights and algorithm constants. Their values cannot normally be changed in a module.**
- **Variable Inputs – the input data to a module from other module outputs. These may refer to a specific module or be wild '*'. The variables can be accessed by two methods. When accessed using *declputvar,* their values can be changed in the module. When accessed using *declgetvar,* their values cannot be changed in the module.**
- **Declared Variables – the output of a module, which may be the final output of the model or the intermediary data to be used by the next module in the model flow. Their values can be changed in the module. These variables can be set to "Local" by setting option. The only change with this option is that the variable is displayed in the "Local Variable" list.**
- **Local Variables (Storage allocation handled by *decllocal* (...) - this type of variable is similar to a Declared Variable but cannot be used as input to other modules. It is also useful for debugging as *decllocal* (...) can be changed to *declvar* (...) to make it visible to CRHM. Their values can be changed in the module.**
- **Local Variables (Storage allocation handled by user) – modules temporary storage used at the module programmer's discretion and exist throughout the model run for local use within the module only. Invisible to the CRHM program.**

Three other simple variables are important to the model modules;

- **nhru - the number of HRUs to be implemented in the model. This is a *dynamic* variable as its value is only known to the model when it is run.**
- **nlay - the number of soil of soil layers to be implemented in the model. This is a *dynamic* variable as its value is only known to the model when it is run.**
- **nobs – the number of areal observations available of a particular type. At present it is set to the dimension of the highest dimensioned observation variable when the observation files are read.**

The above information is defined in the ClassExample.h file as illustrated in the following example:

```
class ClassExample : public ClassModule {

  public:

  class ClassExample(String Name = Example, String Version = "undefined") : ClassModule(Name, Version) {};

  long nhru;

// declared observation variables

  const float *t;

// declared variables

  float *Rn;

  float *Qg;

  float *Qs;

// declared parameters

  const float *F_Qg;

  constfloat *F_Qs;

// variable inputs

  float *net;

  void decl(void);
```

```
    void init(void);

    void run(void);

    void finish(bool good); // optional for this module as there is no local storage used

};
```

The above coding is taken from a module used in calculating evaporation. The module inputs a variable from an observation data file and one other from another module. The module has three outputs and the module algorithm uses two parameters to describe actual physical constants. Note that all the user chosen variable names are superceded by float * indicating to the compiler that these are pointers to single precision numbers. In FORTRAN terminology these are arrays. Text case is significant in the C language, *t is not the same as *T. In FORTRAN these names would identify the same variable, but not in C or C++.

The other addition is the list of routines used by the module. These routines always have the same name. These routines are written specially for each module and implement the variables and algorithms for the module. The functions of the routines are as follows:

- **decl – declares all the variables/parameters used by the module. The different types; Observation, Declared, etc., are described above. This routine provides the link between your module and the CRHM Platform.**
- **init – implements local storage allocation, initialization of variables and parameters and user checks, e.g. << total HRU area equals the basin area>>. The values set by init may be changed by the initial state file if used. Routine initialisation at the beginning of a model run is best done in the function run testing for the first entry using function getstep() == 1.**
- **run – executes the module code every time step. This is the guts of the module - the part that does the real work.**
- **finish - handles any output at the end of a model run and the de-allocation of any local storage used by the module. This module is not always required.**

There is no need for a destructor (~ClassExample} as finish handles all deallocation of user variables and the other variables used, observation, parameter, .etc. are handled automatically by the CHRM platform.

Module Implementation

In the previous section the broad specifications of the module are defined in the *header* or ClassExample.h portion of the source code. This section describes the detailed portion which must be added to ClassExample.cpp implementing our module. To relate to a practical example we will describe the module ClassExample 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.

The module ClassExample::decl(void) declares all the variables used by ClassExample making them visible to the CRHM Platform. The program code follows:

```
void ClassExample::decl(void) { // Line 1

   declreadtobs("t", OBS, "temperature", "(°C)", &t); // Line 2

   declgetvar("netall", "net", "(MJ/m^2*int)", &net); // Line 3

   declparam("F_Qg", NHRU, "3*0.2", "0.0", "1.0", "Qg=F_Qg*Rn", "()", &F_Qg); // Line 4

   declparam("F_Qs", NHRU, "0.0", "0.0", "1.0", "Qs=F_Qg*Rn", "()", &F_Qs); // Line 5

   declvar("Rn", NHRU, "net", "(mm/Int)", &Rn); // Line 6

   declvar("Qg", NHRU, "ground flux", "(mm/Int)", &Qg); // Line 7

   declvar("Qs", NHRU, "storage flux", "(mm/Int)", &Qs); // Line 8

} // Line 9
```

Comment:

1. **The routine consists of lines 1-9. Line 1 gives information about routine and line-9 marks the end. Note that '//' indicates that the remainder of the line is a comment.**
2. **Line 2 declares that the modules uses the observation called t in the observation data file and that it will be stored within the module at address &t. When the variable is accessed the '&' is ignored and t is referenced using t[ii]. In this case the same variable name (t) was used in the observation file and also within the module but this is not necessary. The NOBS indicates that the dimension of t is *nobs*. Another frequently used value is *ONE*. The next two strings give a description of the variable and its units for use within CRHM.**
3. **Line 3 declares that the module uses the variable net from module netall. Within the module it will be stored at address &net. It will have the dimensions of the number of HRUs, i.e.*nhru* and may be referenced as *net[0]* through *net[nhru-1]*. Again the variable name (net) need not be the same within this module as the in the source module.**
4. **Lines 4 and 5 declare parameters used by the module. These follow the earlier conventions. The parameter. F_Qg is dimensioned *nhru*, is referenced as &F_Qg. The last two text strings give a description of the variable and its units for use within CRHM. The first three text strings define the default value(s) for the parameter and its minimum and maximum values. The default value can be entered for each HRU by a comma sequence consisting of single numbers or using the repeat operator '*'. If there are insufficient values the last defined value is used to satisfy the remainder of the array.**
5. **Lines 6 through 8 define the module output. As before the fields consist of name, dimension, description, units and address. Again the variable name within the module can be different to that used externally.**

   The routine ClassExample::init(void) is trivial in ClassExample as its only function it to set up the number of HRU's. The program code follows:

```
void ClassExample::init(void) {

   nhru = getdim(NHRU); // transfers current #HRU's to module

}
```

The algorithm code to be executed every time interval is written into the routine ClassExample::run(void). The program code follows:

```
void ClassExample::run(void) {

   for (int hh = 0; hh < nhru; hh++) {

      Rn[hh] = net[hh]/(2.501-0.002361*t[hh]); // MJ/(m2 Int) to mm/(m2 Int)

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

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

   }
```

```
}


void ClassExample::finish(bool good) {

// any code for output or clean up

return true  // or false

}
```

Since the main code is used for every HRU, it is put inside a *for* loop. As C++ array references are in the range of 0 to maximum-1, the integer ii is initialized to 0 and incremented every iteration by one to a maximum of nhru-1. 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.

Module Installation

The user module must be made visible to the CRHM Platform by creating an instance of class Administration and identifying the components of our new class . This is done by the defining the routine MoveModulesToGlobal in ClassModule.h and CPP files by add the following lines:

The header file there should contain the lines;

```
using namespace std;

extern "C" void __declspec(dllexport) MoveModulesToGlobal(String DllName = "Example");
```

In the CPP file, instantiate an instance of class Administer and then call its routines to identify new user modules and optional models. After the additions are defined load them into the CRHM platform by calling LoadCRHM.

```
Administer DLLModules("Ver: 12/19/01", "HelpFileName");

void MoveModulesToGlobal(String DLLName){

   DLLModules.AddModule(new ClassExample("Example", "Current date")); // date will appear in CRHM reports

   DLLModules.LoadCRHM(DLLName);

}
```

This completes the registration process.

Support Library Routines.

AddModule (member of class Administer).

- **void AddModule(ClassModule *Module)**

AddModel (member of class Administer).

- **void AddModel(String ModelName, String ModelModules)**

LoadCRHM (member of class Administer).

- **void LoadCRHM(String DllName)**

Declare Variables.

N.B. declare *value* **as 'float *value;' and** *layvalue* **as 'float **layvalue;' or** *value* **as 'llong *value;' and** *layvalue* **as ' long **layvalue;'. If a variable has a default value, it is optional. However, if a later optional value is useded, any earlier default values must also be given a value.**

- **void declvar(string variable, TDim dimen, string help, string units, float **value, float ***layvalue = NULL, bool PointPlot = false, bool StatVar = false)**
- **void declvar(string variable, TDim dimen, string help, string units, long **value, float ***layvalue = NULL, bool PointPlot = false, bool StatVar = false)**
- **void decllocal(string variable, TDim dimen, string help, string units, float **value, float ***layvalue = NULL, bool share = false)**
- **void decllocal(string variable, TDim dimen, string help, string units, long **value, float ***layvalue = NULL, bool share = false)**

  For more information about TDim see enumerations.

Declare State Variables.

State variables are identical to normal variables except that their value is saved to file when the model state is saved.

- **void declstatvar(string variable, TDim dimen, string help, string units, float **value, float ***layvalue = NULL, bool PointPlot = false, bool StatVar = true)**
- **void declstatvar(string variable, TDim dimen, string help, string units, long **value, float ***layvalue = NULL, bool PointPlot = false, bool StatVar = true)**

Declare Parameters.

N.B. declare *value* **as 'const float *value;' and** *layvalue* **as 'const float **layvalue;' or** *value* **as 'const long *value;' and** *layvalue* **as 'const long **layvalue;'.**

- **void declparam(string param, TDim dimen, string valstr, string minstr, string maxstr, string help, string units, const float **value, const float ***layvalue = NULL)**
- **void declparam(string param, TDim dimen, string valstr, string minstr, string maxstr, string help, string units, const long **value, const float ***layvalue = NULL)**
- **TStringlist* declparam(string param, TDim dimen, string data, string help, TStringlist *StringList)**

Declare Get Variables.

- **long declgetvar(string source, string name, string units, const float **value, const float ***layvalue = NULL)**
- **long declgetvar(string source, string name, string units, const long **value, const long ***layvalue = NULL)**

  Returns either 1 for interval data or the Global::Freq (# of intervals in the day) for daily data. The nterval type is determined by examining the units string of the variable for "int" or "day". The value returned allows the module to scale the variable correctly.

N.B. declare *value* **as 'const float *value;' and** *layvalue* **as 'const float **layvalue;' or** *value* **as 'const long *value;' and** *layvalue* **as 'const long **layvalue;' as these values can never be changed.**

Declare Put Variables.

- **long declputvar(string source, string name, float **value, float ***layvalue = NULL) Returns the same as declgetvar.**
- **long declputvar(string source, string name, long **value, long ***layvalue = NULL) Returns the same as declgetvar.**

N.B. declare *value* **as 'float *value;' and** *layvalue* **as 'float **layvalue;' or** *value* **as 'long *value;' and** *layvalue* **as 'long **layvalue;' to allow values to be changed.**

'Put Variables' have to be used with care as the variable has been 'declared' by another module and if that module assigns a new value to the variable after the module doing the 'put', the change to the variable will be lost. It is primarily intended to be used for variables which are used to accumulative quantities, e.g. 'SWE', 'cumPpt' etc. which use SWE = SWE - melt (SWE -= melt).

However, it can be used for variables which are assigned values (SWE = newSWE) if the order of the modules is determined by other variables definitions which ensure the modules doing assignments always execute before modules doing 'puts'.

Returns either 1 for interval data or the Global::Freq (# of intervals in the day) for daily data. The nterval type is determined by examining the units string of the variable for "int" or "day". The value returned allows the module to scale the variable correctly.

Declare Grp Variables.

- long declgrpvar(string variable, string queryvar, string help, string units, const float **value, const float ***layvalue = NULL, bool PointPlot = false)
- long declgrpvar(string variable, string queryvar, string help, string units, const long **value, const long ***layvalue = NULL, bool PointPlot = false)

'queryvar' is the name of the variable to be searched for in other groups. It will not find the 'queryvar' unless it is contained in another group, i.e. ignores 'queryvar' in modules outside groups.

The 'layvalue' variable must be dimensioned large enough to hold the information from all groups containing the 'queryvar'.

The 'layvalue' variable

The 'value' variable holds the dimension of

Returns either 1 for interval data or the Global::Freq (# of intervals in the day) for daily data. The nterval type is determined by examining the units string of the variable for "int" or "day". The value returned allows the module to scale the variable correctly.

N.B. declare *value* as 'const float *value;' and *layvalue* as 'const float **layvalue;' or *value* as 'const long *value;' and *layvalue* as 'const long **layvalue;' as these values can never be changed.

Declare Read Observation.

N.B. declare *value* as 'const float *value;' or *value* as 'const long *value;'.

- long declreadobs(string variable, TDim dimen, string help, string units, const float **value, bool optional = false, const float ***layvalue = NULL)
- long declreadobs(string variable, TDim dimen, string help, string units, const long **value, bool optional = false, const long ***layvalue = NULL)

"dimen" can be NHRU or NOBS.

When NHRU is used the observations are referenced to CRHM variables using the table HRU_OBS. If there are too few observations, the last one is used multiple times. If there are too many values the extra values are ignored. If values in the table exceed the number of observations they are set to the maximum number of observations. This function returns the number of HRUs in the project or group.

When NOBS is used the number of observations is read. This function returns the number of observations read. This is independent of the number of HRUs in the project or group.

Returns the dimension of the observation, usually one.

Declare Read Observation Function.

N.B. declare *value* as 'const float *value;' or *value* as 'const long *value;'.

- long declobsfunc(string obs, string variable, const float **value, TFun typeFun, bool optional = false)

Declare Observation.

N.B. declare *value* as ' float *value;' or *value* as ' long *value;'.

- long declobs(string variable, TDim dimen, string help, string units, const float **value)
- long declobs(string variable, TDim dimen, string help, string units, const long **value)

Get Number of HRUs.

- long getdim(TDim dimen)

Get Number of observations for an Observation Variable or Observation Function Variable

- long getdimObs(string variable) e.g. long tcnt = getdimObs("t")

Get Interval Number.

- long getstep(void)

Last Model Run Loop.

- bool laststep(void)

Get Date and Time.

- long julian(char *when) // when is "start", "now" or "end".
- void dattim(char *type, long *itime) // type is "start", "end" or "now"

Read Observations from other Interval.

- bool ReadAheadObs(long Offset = 0)

This routine allows the user to access other interval observations relative to the current interval observations. For example, a value of '1' changes observations values to the next interval. A value of '-1' changes observations values to the previous interval. On return ReadAheadObs restores access pointers to point to the current interval but does not re-read the observation values. After using ReadAheadObs a module should call ReadAheadObs() to restore observation values ready for the next module. The module returns false if an access to an interval outside the model run range is attempted.

Debug Output Variables.

- void __fastcall LogMessage(const char *S, float V, TExtra Opt = BLANK)
- void __fastcall LogMessage(const char *S, float V, float V1, TExtra Opt = BLANK)
- void __fastcall LogMessage(const char *S, float V, float V1, float V2, TExtra Opt = BLANK)
- void __fastcall LogMessage(const char *S, long V, TExtra Opt = BLANK)
- void __fastcall LogMessage(const char *S, TExtra Opt = BLANK)

- void __fastcall LogMessage(long hh, char *S, float V, TExtra Opt = BLANK)
- void __fastcall LogMessage(long hh, char *S, long V, TExtra Opt = BLANK)
- void __fastcall LogMessage(long hh, char *S, TExtra Opt = BLANK)

- void __fastcall LogMessageA(long hh, const char *S, float V1, const float HRU_area, const float Basin_area, TExtra Opt = BLANK)

  outputs hh, V1, V1*HRU_area and V1*HRU_area/Basin_area.

- void __fastcall LogMessageA(long hh, const char *S, float V1, const float Basin_area, TExtra Opt = BLANK)

  outputs hh, V1 and V1/Basin_area.

  where Opt = BLANK, DD, TT and DT for "date only", "time only" and "date and time".

Debug Output String.

- void __fastcall LogDebug(long h, char * S, float v)
- void __fastcall LogDebugD(char *S)
- void __fastcall LogDebugT(char *S)
- void __fastcall LogDebug(char *S)

Log Errors to CRHM.

1. void __fastcall LogError(CRHMException Except)
2. void __fastcall LogError(String S, TExcept Kind)

Where

```
class CRHMException {

    public:

    string Message;

    TExcept Kind;

    CRHMException() : Message(""), Kind(NONE) {};
```

```
    CRHMException(string Message, TExcept Kind) :

    Message(Message), Kind(Kind) {};

};
```

Useful Program Variables

These variables are in class Global and are referenced by including an extern statement and preceding the variable name with 'Global::' or 'global::' as shown in the following examples.

```
extern Global global;

long currentFreq = global::Freq; // or Global::Freq
```

Variables available:

- double Interval - as fraction of day, i.e. range of 1/48 to 1.0.
- double DTstart - model start time as equivalent to TDateTime.
- double DTend - model end time as equivalent to TDateTime.
- TdateTime DTnow - current model time.
- long Freq - intervals/day
- long DTindx - current loop index.
- long DTmin - current run minimum index.
- long DTmax - current run maximum index.
- long IndxMax - last model loop index based upon length of first observation file loaded.

Eliminating data points from point plots.

Somtimes it is not desired to plot points when they are out of range. A mechanism has been provided to do this. If a floating point number is set equal to the external value xLimit or a long value is set to the external value iLimit TeeChart will not add it to the plot series. See following segments of code taken from a module. For line plots a line will be drawn from the last 'good' point to the subsequent next 'good' data point.

```
#include "Common.h"

extern double xLimit;

extern long lLimit;

...

declvar("d", NHRU, "maximum frost depth", "(m)", &d, NULL, true);

declvar("id", NHRU, "maximum frost depth", "(m)", &id, NULL, true);

...

d[hh] = xlimit; // not plotted any other legal value is plotted.

id[hh] = ilimit; // not plotted any other legal value is plotted.
```

Hiding Variables from CRHM after debugging modules.

Notice how declvar and decllocal have similar parameters.

```
    void declvar(string variable, TDim dimen, string help, string units, float **value, float ***layvalue, bool
    PointPlot = false)

    void decllocal(string variable, TDim dimen, string help, string units, float **value, float ***layvalue, bool
    share = false)
```

As a result a variable can be initially made visible by declaring it using 'declvar'. After the module is finished, simply changing the declaration to 'decllocal' will reduce its visibility to within the module. At this time the data is not being plotted so 'PointPlot' is not necessary.

The share option allows cooperating modules to share local variables without them being visible to the CHRM platform.

Module Skeleton.

Header

```
#include "ClassModule.h"
using namespace std;
extern "C" void __declspec(dllexport) MoveModulesToGlobal(String DllName = "Example");
class ClassExample : public ClassModule {
  public:
  class ClassExample(string Name = "your name", Version = "undefined") : ClassModule(Name, Version) {};
  long obsCnt;
// declared observation variables
  const float *obs;
// declared variables
  float *var;
// declared parameters
  const float *par;
// variable inputs
  const float *t;
  void decl(void);
  void init(void);
  void run(void);
  void finish(bool good); // optional for this module as there is no local storage used
  class ClassExample* klone(string name) const;
};
```

Body

```
using namespace std;
using namespace CRHM;
Administer DLLModules("Ver: 12/19/01"); // this parameter will appear in CRHM reports
void MoveModulesToGlobal(String DLLName){
  DLLModules.AddModule(new ClassExample("Example", "current date or revision"));
  DLLModules.LoadCRHM(DLLName);
}
ClassExample* ClassExample::klone(string name) const{
    return new ClassExample(name);
```

```
}
void ClassExample::decl(void) {
    declvar("OutVar", NHRU, "net", "(units)", &var);
    declparam("par", NHRU, "3*0.2", "0.0", "1.0", "description", "()", &par);
    obsCnt = declreadobs("t", NOBS, "temperature", "(°C)", &t); //
}

void ClassExample::init(void) {
    nhru = getdim(NHRU); // transfers current # of HRU's to module
}

void ClassExample::run(void) { // executed every interval
    for (int hh = 0; hh < chkStruct(); ++hh) {
        var[hh] = t[min<long> (hh, obsCnt]*par[hh];
    }
}

void ClassExample::finish(bool good) { // only required for local storage and final output
// any code for output or clean up
} */
```

Changes to Modules for use in Groups and Structures.

Two changes are necessary to allow CRHM to handle modules in Groups and Stuctures.

Addition to the header (*.h) file of the module,

```
ClassOurClass* klone(string name) const; // where ClassOurClass is your class name.
```

and the addition to the source code (*.cpp) file of the module,

```
ClassOurClass* ClassOurClass::klone(string name) const{
return new ClassOurClass(name, "10/26/06"); // date is your date.
}
```

This code allows the CRHM plarform to create multiple copies of the module.

Structures modification.

Modules are designed to execute the same module code for every HRU. Provision had to be made to control the HRUs that a module executes for.

In the typical module there is a loop similar to;

```
for(int hh = 0; hh < nhru; ++hh){
  // code to be executed
}
```

The simplest approach is to modify it to;

```
for(hh = 0; chkStruct(); ++hh){
  // code to be executed
}
```

The function bool chkStruct(void) and the variable hh are declared in ClassModule. Now control is transferred to CRHM by chkStruct() and CRHM is able to skip the execution of any HRU by incrementing hh or by returning false, terminate execution of the entire loop.

In the *init* function of a module it is normal to use the former loop since everything must be initialised and in the *run* and *finish* functions use the latter loop to limit change of variable output to that determined by the allowed values of HRU. Care must be taken since the module variables are shared with other modules and nothing that the other modules set can be overwritten.

To handle unusual situations these alternatives are also provided.

```
for(int hh = 0; chkStruct(hh); ++hh){ // bool chkStruct(long &hh)
  // code to be executed
}
```

and

```
for(int hh = 0; chkStruct(hh, newmax); ++hh){ // bool chkStruct(long &hh, long newmax) where newmax <= nhru.
  // code to be executed
}
```

The former is to handle multiple loops and the latter is to handle a loop that handles a reduced range. These extensions were required for *pbsm*.

Module Parameters.

The fields in the parameter creation call are as follows:

void declparam(string param, TDim dimen, string valstr, string minstr, string maxstr, string help, string units, float **value, float ***layvalue = NULL, int defcnt = 1)

Defined as:

- The parameter name used in the CRHM model.
- The parameter dimension from 'ONE' to 'SIX, 'NHRU', 'NOBS', 'NLAY' and NDEF. Note that 'NLAY' is actually dimensioned as [NLAY][NHRU] and 'NDEF' is dimensioned [defcnt][1].
- NFREQ, NREB, NLAY0 and NDEFN are dimensioned NFREQ [Global::Freq][nhru], NDEFN[defcnt][nhru]. NREB is an array [nhru] of pointers to variables for internal use only.
- Default parameter value string. This is described below.
- Minimum value the parameter can be set to.
- Maximum value the parameter can be set to.
- Help string.
- String specifying the parameter units.
- Storage location of parameter as float**. This variable name in the module code does not have to be the same as used in the CRHM model, i.e. parameter 1.
- Storage location for multiple layers or defines as float***. Optional.
- Number of definitions when using NDEF. Optional.

Parameter Initialization.

- Each individual values can be specified, e.g. "1,2,3".
- If too few values are specified the last values will be set to zero until the number of values equals the value of NHRU in the running model or *defcnt* in the current module call.
- Compare with "[1, 2, 3]", when the last given is duplicated until the number of values equals the value of NHRU in the running model or *defcnt* in the current module call.
- Repeat factors can be used as required, e.g. "2*4.1, 1*3.8, 3.9, 8*9.81". Giving the following values 4.1, 4.1, 3.8, 3.8, 3.9...
- Repeat factors must be simple constants, e.g. 1,5,6 etc. Expressions are not acceptable, e.g. 5*6, (5*6), etc.
- Values may be calculated if enclosed in braces, e.g. "(3^0.5), 4*(1+(2+3.7)/9.0, ... Operators are +, -, *,/, %and ^
- Square brackets differentiate layers, e.g. "[1,2,3][4,5,6][7,8,9]". The values defined for the last layer will be repeated as required.
- Repeat factors may be used with layers, e.g. "2*[4,5,6]2*[7,8,9]".
- Repeat factors may be used with defines, e.g. "2*[4]2*[7]". Note that only a single value is enclosed by the square parenthesis.
- Sequences increasing by one may be generated using "1, 2, 3!" or "[1, 2, 3!]". If five values are required these will be generated, e.g. 1, 2, 3, 4, 5.

Notes.

Example from module:

```
declparam("Ta", NHRU, "10", "-10.0", "50.0", "annual air temperature", "(°C)", &Ta);

declparam("soil_type", NLAY, "2", "1", "4","HRU soil type: 1= sand, 2= loam, 3= clay, 4 = organic", "()",
&soil_type, &soil_type_lay);

declparam("por", NLAY, "[0.5", "0.0", "1.0]", "0.0", "1.0", "porosity", "(m3/m3)", &por, &por_lay);

declparam("por", NDEF, "0.52, 0.51, [0.5]", "0.0", "1.0", "porosity", "(m3/m3)", &por, &por_def, 5);
```

where the parameter variables are declared as follows:

```
const float *Ta;

const long *soil_type; const long **soil_type_lay;

const float *por; const float **por_lay;

const float *por; const float **por_def; // variable por is not used but must be declared.
Defined values are por_def[0][0], por_def[1][0] ... por_def[7][0].
```

Concatenation of strings.

The length of strings should be limited for easiest user comprehension. However, a string can be split across lines using "String" // " string continuation".

Array dimensions.

The NHRU and NLAY dimensions are set in the CHRM Parameter screen when a model is setup. The NDEF dimension is determined by the value of *defcnt* in the user written module by the call to the routine declparam("por", NDEF, ...defcnt). The NDEF values assigned are common for all HRU.

Module Distribution Files.

- CRHM.chm - HTML help file for all models.
- Modules.chm - HTML help file for basic modules.
- Macro.chm - HTML help file for macro programming.

- CRHM.exe - Self standing non extendable model.
- shapelib.dll required by CRHM.exe and be in the same directory as the executable file.

- CRHM_DLL.exe - Extendable model. Requires the following files to run:
  - borlndmm.dll,
  - cc3280mt.dll,
  - CRHMGlobal.dll,
  - shapelib.dll.
- To create DLL User Modules for CRHM_DLL.exe requires in addition to the files for CRHM_DLL.exe, the following files:
  - User written module (*.h and *.cpp),
  - ClassModule.h,
  - ClassCRHM.h,
  - DefCRHMGlobal.h,
  - common.cpp/h,
  - SnobalDefines.h,
  - CRHMGlobal.lib, and as for CRHM_DLL
  - borlndmm.dll,
  - cc3280mt.dll,
  - CRHMGlobal.dll,
  - shapelib.dll.

cc3280mt.dll and borlndmm.dll are distributed with C++Builder and re-distriduted with the CRHM DLL models.

Project Files.

CRHM was developed using "2007 C++Builder, Professional Edition" which is available at the academic price if used for Educational Non-commercial use. However, CodeGear also distribute "Turbo C++ Explorer" which is free.

The application installation disk contains sample project files for the different C++Builder versions. These project files are not interchangeable. There is an example file for a simple "Example DLL module and a project to generate all of the modules in CHRM from NewModules.cpp/h.

The directory structure is as follows:

...\CRHM_DLL\CRHM_2008\bin contains executables

\lib contains "CRHMglobal.lib

\prj contains project files for "newmodules" and "example_DLL" for "2007 C++Builder, Professional Edition". "CRHM_dll_module.cbproj" and Example_dll.cbproj".

\prj_turbo contains "Example_dll.bdsproj" and "Example_dll.bpf"

\src\CRHM DLL module

src\Example_dll

src\shared contains "ClassModule.h", "ClassCRHM.h", "GlobalCommon.h", "common" and "SnobalDefines"

Creating a project in C++Builder.

All development is being undertaken using "2007 C++Builder, Professional Edition" but "Turbo C++ Explorer" C++Builder may also be used.

There are two examples supplied with the installation and can be used as prototypes.

Executing the DLL module.

1. Select either the RUN icon or the Run menu - "Run/Run". CRHM_DLL will execute.
2. Select the DLL you just built using "DLL|Open". In this case open file \Program Files\CRHM_DLL\Development\ExampleModuleDLL.DLL". A dialogue box asking if you wish to load the module "Example" will pop-up. Select "Yes" or "Yes to All"
3. Build a CRHM model using "Build|Construct". Right click on the module "Example" in the lefthand listbox. It should now appear in the righthand listbox. Click on "Build". CRHM will return to the main screen with the model built.
4. "OutVar" should show in the top lefthand listbox. Right click it and select "Add". "OutVar" is now a CRHM model output and will appear in the to righthand listbox.
5. A model cannot run without driving climatic variables, called observations. Select "Observation|Open" and select "\Program Files\CRHM_DLL\winter1974.obs". All the names of the climatic data available in this observation file will appear in the lower lefthand listbox.
6. To run our CRHM model consisting of the module "Example", click "Run" on the main menu.
7. To save the model for future use. use "Project|Save" etc.

Using other Compilers.

CRHM DLLs can only be compiled with "2007 C++Builder, Professional Edition" and "Turbo C++ Explorer". The reason for this is that classes are inherited from CRHM and then the new class modules created are returned to CRHM as pointers.

Module Local Memory Allocation.

One Dimensional Case.

In the simplest case an array dimensioned equal to the number of HRUs is required in a module to store data between time intervals. To do this requires the following steps:

1. Add to the *.h file the line:

   "float *OurLocalVar;"

2. Add to the module Init() routine in the *.cpp the following:

   "nhru = getdim(NHRU);" Only required once.

   "OurLocalVar = new float [nhru];"

3. In the module Run() routine use the array as desired not forgetting it is referenced as 0 through nhru-1:

   "float X = OurLocalVar[n]"

4. Finally in the module finish() de-allocate the local array using:

   "delete [] OurLocalVar;"

5. Repeat for as many arrays as needed.

Two Dimensional Case.

Assume an array dimensioned (FIVE by NHRU) is required

1. Add to the *.h file the line:

   #define FIVE 5;

   "float **OurLocalVar;"

2. Add to the module Init() routine in the *.cpp the following:

   "nhru = getdim(NHRU);" Only required once.

   "OurLocalVar = new float *[FIVE];"

   "for(int ii = 0; ii < FIVE; ++ii) OurLocalVar[ii] = new float [NHRU];

3. In the module Run() routine use the array as desired not forgetting it is referenced as 0 through FIVE-1 and 0 through nhru-1.

   "float X = OurLocalVar[j][n]" where 0 <= j < FIVE and 0 <= n < NHRU.

4. Finally in the module finish() de-allocate the local array using:

   "for(int ii = 0; ii < FIVE; ++ii) delete [] OurLocalVar[ii];"

   "delete [] OurLocalVar;"

5. Repeat for as many arrays as needed.

Two Dimensional Case with One Dynamic Dimension.

Sometimes an array size is determined by a physical phenomenon, for example the number of melt waves in a snowpack. In FORTRAN this is handled by making the array as large as the worst case scenario. The following illustrates a method that allows the array to expand as needed.

1. Add to the *.h file the line:

   #include <vector>

   "vector<float> *OurLocalVar;"

2. Add to the module Init() routine in the *.cpp the following:

   "nhru = getdim(NHRU);" Only required once.

   "OurLocalVar = new vector<float> [NHRU];"

3. In the module Run() routine use the array as desired not forgetting it is referenced as 0 through nhru-1.
   a. To increase the dimension by an element;

"OurLocalVar[thisHRU].push_back(Xvalue1);" the array for this HRU will now have a length of 1

"OurLocalVar[thisHRU].push_back(Xvalue2);" the array for this HRU will now have a length of 2

OurLocalVar[thisHRU].push_back(XvalueN);" the array for this HRU will now have a length of n

b. The values are kept in order and can be accessed as follows

"Xvalue1 = OurLocalVar[thisHRU][0];"

"Xvalue2 = OurLocalVar[thisHRU][1];"

"XvalueN = OurLocalVar[thisHRU][N-1];"

c. At any time the current size of an array can be determined using;

"int SizeHRU = OurLocalVar[thisHRU].size();"

d. The array size can be reduced but this is not usually of value unless memory size is critical. Values can be re-written using:

OurLocalVar[thisHRU][N-1] = NewValue;"

4. Finally in the module finish() de-allocate the vector array using:

"delete [] OurLocalVar;"

5. Repeat for as many arrays as needed.

This method is very useful for making an array of a user class containing multiple variables. However, you may have to write default, copy and assignment constructors.

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;
}
```

Order of Execution.

When CRHM is run the sequence of events is complex. First the modules are initialised. Followed by;

1. Clear ReadListN and FunctListN. These lists determine the observations read every time step before the execution of the modules.
2. Set BuildFlag to CRHM::INIT.
3. Step through project modules calling the module "decl" and "init" entry points. The former transfers the addresses of variable and parameters and sets up the observation read lists. The latter executes the init routine for the modules.
4. Set BuildFlag to CRHM::RUN to actually run the project. The followig sequence of events occur for every time step.
    1. call routine 'ReadObs'
        1. call module obs pre_run once. Assumes module 'obs' first module to use observations with a 'declreadobs ' call.
        2. calls 'ReadVar' for the 'declreadobs ' calls.
        3. at the beginning of each day, calls 'UserFunct' for the 'declreadobs ' function calls.
        4. at the beginning of each day, calls 'CustomFunct' for the temperature and vapour pressure observations. For temperature applies to interval t and daily t_min and t_max. For vapour pressure to interval ea. ? daily.

decl routine.

- CRHM::BUILD - This mode sets up the lists for the build screen.
- CRHM::DECL - When a project is loaded this mode constucts the observation, variable and parameter objects.
- CRHM::INIT - Before a module is executed this mode transfer the observation, variable and parameter addresses of objects to the module.
- CRHM::RUN - not used.

init routines.

Does not use the BuildFlag ever.

run routines.

Does not use the BuildFlag ever.

Global control variables.

1. Global::CRHMStatus = 0; // = 1 module control, = 2 main control. Both inhibit output.
2. Global::CRHMControlSaveCnt

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))".

Write CRHM help to a File.

CRHM help consists of a series of topic files which can be printed by subtopic from within the CRHM help. Sometimes it would be more convenient if an entire book, (heading and all subtopics) could be extracted and saved into a file for printing or importing into another application. The following work around may work for you.

You can get a combined HTML file that contains all the text in a particular contents book as follows.

1. **Right-click the book and choose "Print".**
2. **Choose "Print the selected heading and all subtopics".**
3. **When the print prompt appears, ALT+TAB to Windows explorer.**
4. **Browse to your TEMP folder, usually "C:/windows/temp", and find the latest file of the pattern "~hh*.htm". This is a concatenation of all the HTML files in the book you printed.**
5. **Copy this file elsewhere before cancelling (or continuing) the print job, as the file will then be deleted.**
6. **This file can be imported into MS Word or Frontpage.**

N.B.

- **Some operating systems put their temporary files in other locations. Windows XP uses c:\Documents and Settings\Owner\Local Settings\Temp.**
- **Windows XP appears to hash the name as a random combination of letters and numbers with the extension "htm".**
- **There is no need to copy the file in Windows XP as it uses a unique name for each temporary file and deletes them at the end of the session.**

Table of Contents

Introduction.

This help file describes the enhanced modules written for application in CRHM_new.

Using Demonstration Models.

CRHM has examples of models using different combinations of modules built into the Build menu screen. To use them use the following steps.

1. **Run CRHM.**
2. **Goto the Build/Construct screen to select the desired number of HRUs and exit this screen using Build to register the selected number of HRUs..**
3. **In the Build screen select the example model desired from the list.**
4. **In the Observation screen select a suitable observation file. The file "C:\Program Files\CRHM\Examples\Winter1974.obs" can be used for demonstration purposes only.**
5. **Select some variables to display and click on Run to execute..**

This is a training exercise. The usual warnings apply, i.e. the selection of modules, parameters and set of observations must be relevant to the physical application to achieve meaningful results.

Notes.

- **It is best to select the number of HRUs before selecting a sample model, then the default parameters are optimum. When the number of HRUs is increased after selecting the model, the last HRU parameter value in the earlier project is duplicated for the new HRUs.**
- **If a project is already loaded when a sample project is selected, the current modules loaded will be replaced by the model selected. The observation files and number of HRUs will not be affected.**
- **When a sample model is selected,. its description is added to the first line of the prospective project file and also inserted into the TChart title.**
- **The user must customise all parameters in the model to define their application.**

Module Variations.

The original CRHM module is very limited in its loading ability and could not be made to load in different modes without a lot of difficulty and unused portions of the module still visible. The input variable/observations, output/observations variables and required parameters were all fixed and always visible. An attempt was made to work around this by making observations optional and modules generating declared observations and the use of the AKA screen. This allowed modules to perform in different ways as dictated by the available driving data. However, this was not very easy for the user to use. There were also numerous unused parameters and variables as these had to be declared for all possible uses of the module and the user had to know which ones applied to the current module application.

Variations were introduced to overcome the limitations outlined above. Module names have been extended as follows. The original name say "longVt" will run as always. However, variations of the original module will be possible, denoted as "longVt#1", "longVt#2", etc. These variations are enhanced versions of the original module, with different inputs requirements, say synthetic short-wave "QsiD_Var" (Annandale) instead of the measured observation Qsi. The parameters and input/output variables can also be customised. The module variations can also have different algorithms. Descriptions of current variation modules follow.

Annandale - short-wave estimation.

1. Annandale - outputs QsiA#, QsiD# and QsiS# short-wave observations (interval, daily and interval on slope) and hru_SunAct variable.
2. Annandale#1 - outputs QsiA_Var, QsiD_Var, QsiS_Var (interval, daily and interval on slope) variables with hru_SunAct variable.

calcsun - sunshine hours.

1. calcsun - input SunAct Obs or Qsi Obs to calculate Qsitot if sunshine hours not available.
2. calcsun #1 - Qsi observation to calculate Qsitot.
3. calcsun#2 - QsiDobs (W/m^2) observation.
4. calcsun#3 - QsiD_Var (W/m^2) from Annandale.

Canopy, CanopyClearing and CanopyClearingGap - calculate short, long and all-wave radiation components at the snow surface.

1. CanopyClearing - input Qsi and Qli observations.
2. CanopyClearing#1 - input Qsi observation and QliVt_Var from longVtX etc..
3. CanopyClearing#2 - input Qli observation and Qsi_Var from longVtX etc..
4. CanopyClearing#3 - input Qsi_Var and QliVt_Var from longVtX etc..

ebsm - energy-budget snowmelt model for the Canadian Prairies.

1. ebsm - no inputs for Brunt. Optional temperature and radiation index. QnD can be used with Use_QnD for legacy projects.
2. ebsm#1 - input interval variable Qnsn_Var (W/m^2) from CanopyClearing etc..
3. ebsm#2 - input interval observation Qnsn (W/m^2).
4. ebsm#3 - input daily observation QnD (MJ/m^2*d).

longVt - long-wave using terrain view factor.

1. longVt module - input Qsi observation.
2. longVt#1 module - input QsiDObs observation from module Slope_Qsi.
3. longVt#2 module - input QsiD_Var observation from module AnnandaleX#1 etc..

Needleleaf - calculate short, long and all-wave radiation components at the snow surface.

1. Needleleaf - input Qsi and Qli observations.
2. Needleleaf#1 - input Qsi observation and QliVt_Var from longVtX etc..
3. Needleleaf#2 - input Qli observation and Qsi_Var from longVtX etc..
4. Needleleaf#3 - input Qsi_Var and QliVt_Var from longVtX etc..

obs - handle driving observations.

1. obs - standard version to handle interval observations.
2. obs#1 - inputs daily t_max and t_min to generate hru_tmax and hru_tmin for Annandale.

pbsm - blowing snow transport.

1. pbsm - input hru_u.
2. pbsm#1 - input hru_Uadjust from walmsley_wind etc.

pbsmSnobal - blowing snow transport.

1. pbsmSnobal - input hru_u.
2. pbsmSnobal#1 - input hru_Uadjust from walmsley_wind etc.

Soil - soil moisture throughout the year.

1. Soil - standard version.
2. Soil#1 - runoff is limited by culvert flow.

Slope_Qsi - modified short-wave on a slope.

1. Slope_Qsi - output declared observations QsiS# and QsiDObs#.
2. Slope_Qsi#1 - output variables QsiS_Var and QsiD_Var.

SnobalCRHM - energy balance snowmelt model.

1. SnobalCRHM - input Qsi and Qli observations.
2. SnobalCRHM#1 - input variables Qsisn_Var and Qlisn_Var from module CanopyClearing.
3. SnobalCRHM#2 - input observation Qsi and QliVt_Var from module longVtX etc..
4. SnobalCRHM#3 - input variables QsiS_Var from Annandale and QliVt_Var from module longVtXetc..

REW_route - handles the routing of surface and subsurface runoff from RBs (representative basins).

1. REW_route - uses Muskingum for routing.
2. REW_route#1 - uses Clark for routing.

How to use Module Variations.

The variation modules behave like any other module. The only difference is that in the construct screen there are additional steps to selecting them. In the construct screen the modules available are displayed in the listbox to the left of the screen. The modules displayed are determined by the level(s) selected in the "Module Level" list. When a module is selected in the modules available listbox the description of the module is displayed in the information area in the bottom left hand side of the screen. When the module has variations additional lines are displayed giving a list of the possible variations. Add the module to your project as normal. Now, when the module is right clicked in the Modules Selected listbox two additional options are available, Next variation and Last variation. Use these selections to select the variation you want. The requirements and outputs of the module will automatically be displayed in the adjacent module description area. At this point proceed as normal using the Check/Build/Cancel options to develop your project.

obs

Defined in Classobs. This module reads the climatic data from the observation data file(s) into the model. The time step is determined by the observation file interval. For normal use, the interval will be half or one hour. At present the module copies the observations to the HRU's. This module will be expanded in the future to interpolate observations from two or more observation sites to the distributed HRUs. One of the precipitation observations "p" and "ppt" must be present. If both are used (e.g. "p" for rainfall and "ppt" for snowfall) they are merged to generate "hru_p". When both "p" and "ppt" are used together, both must be present in every interval and set to zero when not used. "ppt" can also be in a separate "daily" file containing the daily rainfall. When "ppt" is combined with the interval data only the first interval value of every day is read and used as the total precipitation for the day. All other values are ignored.

Observations

- u (m/s) - average wind speed over time step.
- p (mm/int) - interval precipitation. Optional if ppt available. p and ppt values are merged.
- ppt (mm/d) - daily precipitation. Optional if p available. p and ppt values are merged.
- t_max (°C) - variation #1 daily maximum temperature. Only used for daily 24 hour time step.
- t_min (°C) - variation #1 daily minimum temperature. Only used for daily 24 hour time step.
- t (°C) - air temperature using the INTVL declobs function to read the entire days temperature data. Then modified in buffer by parameters lapse rate and ClimChng_t.
- rh (%) - relative humidity using the INTVL declobs function to read the entire days relative humidity data.
- ea (kPa) - vapour pressure using the INTVL declobs function to read the entire days vapour pressure data.

N.B. Either rh or ea are used. If both are available only rh is used. The other is calculated from the array of rh or ea values, using the adjusted temperature (lapse rate and climate change) and the parameter ElevChng_flag (maintain rh or ea with temperatue change).

Variables

The following are from interval observations.

- tday_intvls (°C) - temperature array holding interval values. Values have Dimensioned [interval][dim].
- rhday_intvls (%) -RH array holding interval values. Dimensioned [interval][dim].
- eaday_intvls (kPa) - ea array holding interval values. Values are calculated from above t and RH. Dimensioned [interval][dim].
- t_obs (°C) - temperature array holding the original observation interval values before adjustment for lapse rate and climate warming. Note that HRU_OBS redirection have been applied. Access t-obs [period in day, dim].

The following variables are derived for each HRU from the observations.

- hru_t (°C) - temperature
- hru_rh (%) - relative humidity.
- hru_ea (kPa) - vapour pressure
- hru_u (m/s) - wind speed.
- hru_p (mm/int) - precipitation. Calculated from the interval precipitation (p) and/or (daily precipitation (ppt) /frequency) or all the daily precipitation (ppt) during the first daily interval of the day as determined by the parameter "ppt_daily_distrib".
- hru_rain (mm/int) - rain.
- cumhru_rain (mm) - cumulative rain
- hru_snow (mm/int) - snow.
- cumhru_snow (mm) - cumulative snow.
- hru_newsnow () - 0/1 for no/yes in the last interval.
- hru_tmean (°C) - daily mean temperature.
- hru_tmax (°C) - daily maximum temperature.
- hru_tmin (°C) - daily minimum temperature.
- hru_umean (m/s) - daily mean wind speed.
- hru_rhmean (%) - daily mean RH.
- hru_eamean (kPa) - daily mean vapour pressure.
- Pa(kPa) - atmospheric pressure.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- hru_elev (m) - altitude.
- obs_elev (m) - altitude.
- lapse_rate (°C/100m) - lapse rate correction.
- ElevChng_flag () - Elevation change control; 0 - maintain RH, 1 - keep Vp within Vsat maximum.
- precip_elev_adj () - precipitation height adjustment {adjusted p(or ppt) = p(or ppt)*(1.0 + precip_elev_adj*elev_difference/100)}.
- HRU_OBS () - array[5][dim] of values indexing observations to HRUs. The order is 1) t, rh and ea; 2) p and ppt; 3) u; 4) Q; and 5) for special use.
- snow_rain_determination() - snow/rain determination: 0 - air temperature, 1 - ice bulb temperature.

- tmax_allrain (°C) - precipitation is all rain when the temperature is greater or equal to this value.
- tmax_allsnow (°C) - precipitation is all snow when the temperature is less or equal to this value.
- catchadjust () - 0/1/3 for none/Goodison/Geonor.
- ppt_daily_distrib () - 0/1 - daily precip all in first interval /equally divided over the day.
- ClimChng_t (°C) - Climate change additive temperature change.
- ClimChng_precip (fraction) - Climate change multiplative p/ppt change.

Variable Inputs

- none

Notes

The parameter HRO_OBS applies to all observations. E.g t, rh, u, ea, Qsi, Qli, Tg etc..

The parameter HRO_OBS is normally defined in the module "obs".

The parameter HRO_OBS has a default value of 1 for every HRU. This implies that there is only one set of observations and that all HRUs use this set of values. However, if the observation file had six sets of temperature data, the observation has "t 6 (°C)" in its header then the following HRU_OBS settings are possible.

HRU_OBS = 1, 2, 3, 4, 5, 6 then the HRU1, HRU2 ... use temperatures T1, T2, T3 ... as driving inputs.

HRU_OBS = 4, 5, 6,1, 2, 3 then the HRU1, HRU2 ... use temperatures T4, T5, T6, T1, T2, T3 as driving inputs.

HRU_OBS = 4, 4, 4, 4, 4, 4 then the HRU1, HRU2 ... use only temperatures T4 as the driving input.

However, if the observation file has only three sets of temperature data, i.e. "t 3 (°C)" in its header, the last defined observation is duplicated for the higher values.

HRU_OBS = 4, 5, 6,1, 2, 3 then the HRU1, HRU2 ... use temperatures T3, T3, T3, T1, T2, T3 as driving inputs.

When the observation variables are displayed in CRHM they are displayed in the same order as in the observation file. When the observations are displayed from the HRU, e.g. hru_t, the order will be determined by the HRU_OBS values.

Basin

Defined in Classbasin. This module contains no active code. It declares general parameters for the model. Examples of the parameters declared are: basin area, HRU area, latitude, elevation, ground slope (GSL) and aspect angle (ASL).

Observations

- none

Variables

- run_ID () - run identification.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- hru_lat (°) - latitude.
- hru_elev (m) - altitude.
- hru_GSL (°) - ground slope.
- hru_ASL (°) - aspect.
- basin_name - text string.
- hru_names - text strings.
- RUN_ID - integer number. . Used to uniquely identify "CRHM_output" log files when RUN_ID is positive.
- RUN_START - run start time (Automation).
- RUN_END - run end time (Automation).
- INIT_STATE - initial state file (Automation).
- RapidAdvance_to - element[1] - date to advance to mm/dd/yyyy.
- Loop_to - element[1] - date to loop to mm/dd/yyyy and element[2] - number of loops before continuing.
- StatVars_to_Handle - state variables to allow to change.

Variable Inputs

- none

Notes

1. If the sum of the HRU areas is not within 0.01 km^2 of the basin area, the basin_area is set to the sum of the HRU areas.
2. The parameter values specified in basin override any values specified in the model modules. The values in the modules are the default values for testing.
3. The basin parameters are overriden when the parameters are edited and saved in a project or parameter file.
4. If basin_name or hru_names contain embedded spaces the individual names nust be enclosed in single quotes.
5. The first field defined in the parameter 'StateVars_to_Handle' has special significance. If it is explicit, i.e. 'soil_moist@A' then all other fields are matched exactly. However, if the first parameter is a root CRHM variable name, e.g. 'soil_moist' it is treated as a wildcard and will match any group having the 'soil_moist' variable, i.e. 'soil_moist@A', 'soil_moist@B', 'soil_moist@C' etc..

NO_pbsm.

Defined in ClasspNO_bsm. This module calculates snow water equivalent from snow fall. Sublimation will be added later. It is intended to be used instead of the Prairie Blowing Snow Model when there is no blowing snow. An example would be in a forest.

Observations

- **none**

Variables

- **SWE (mm) - snow water equivalent. State variable.**
- **cumSno (mm) - snow (net_snow) accumulation from beginning of winter.**

Parameters

- **basin_area (km2) - basin area.**
- **hru_area (km2) - hru_area.**
- **inhibit_evap (flag) - an output parameter set true when the SWE is greater than zero. It is used to inhibit evaporation from the evaporation modules.**

Variable Inputs

- **newsnow () - 0/1 for no/yes from module obs.**
- **net_snow (mm Δt) - snow fall from wild module - intcp, brushintcp etc.**

Notes

pbsm (Pomeroy and Li, 1999)

Defined in Classpbsm. This module calculates snow transport and sublimation. The snowfall observation can be a daily total or the actual interval snowfall. The former case is handled by assuming that the snowfall is uniformly distributed over every interval of the day. The transport and sublimation of blowing snow are calculated every interval using the interval wind speed, air temperature and relative humidity. The model has been extended to handle the transport of snow between HRUs. At the end of a day when snow transport has occurred, snow transported from HRUs with low roughness is distributed over the HRUs with greater roughness according to the fractions specified in the distribution parameter. No transported snow enters the HRU with the lowest roughness. When HRUs with lower vegetative height fill to their maximum, the excess is distributed over the remaining unfilled HRUs.

Observations

- none

Variables

- SWE (mm) - snow water equivalent. State variable.
- Subl (mm Δt) - the mass of snow lost from an HRU by sublimation in time step, Δt – expressed as an equivalent average depth of water over an HRU.
- cumSubl (mm) - cumulative sublimation.
- Drift (mm Δt) - the mass of snow lost from an HRU by snow transport in time step, Δt – expressed as an equivalent average depth of water over an HRU.
- cumDrift (mm) - cumulative transport
- cumSno (mm) - snow (net_snow) accumulation from beginning of winter.
- Prob () - interval probability of blowing snow.
- BasinSnowLoss (mm Δt) - transport out of basin.
- cumBasinSnowLoss (mm) - cumulative transport out of basin.
- cumBasinSnowGain (mm) - cumulative transport into basin.
- snowdepth (m) - calculated snow depth from SWE (Gray/Pomeroy).

Parameters

- fetch (m) - fetch distance.
- Ht (m) - crop height.
- distrib () - distribution fractions. Value for HRU 1 controls snow transport into the basin.
- N_S (1/m^2) - vegetation number density.
- A_S (m) - stalk diameter or silhouette.
- basin_area (km2) - basin area.
- hru_area (km2) - hru_area.
- inhibit_evap (flag) - an output parameter set true when the SWE is greater than zero. It is used to inhibit evaporation from the evaporation modules.
- inhibit_bs (flag) - an input inhibiting blowing snow when set equal to 1. Inhibited HRU is still able to receive drift from other HRUs.

Variable Inputs

- hru_t (°C) - air temperature from module obs.
- hru_rh (%) - relative humidity from module obs.
- hru_u (m/s) - wind speed from module obs.
- newsnow () - 0/1 for no/yes from module obs.
- net_snow (mm Δt) - snow fall from wild module - intcp, brushintcp etc.

Notes

1. The parameter *fetch* cannot be less than 300m.
2. The parameter *distrib* for the first HRU is used differently from all the other HRUs. It determines the drift into the model basin based upon the drift out of the first HRU. BasinSnowGain is equal to distrib[HRU1]*Drift[HRU1].
3. The first HRU, i.e. the lowest vegetation height. When the snow is redistributed, the *distrib* values for the HRU's not filled to their vegetation height are summed and each HRU receives its share. The sum of distributions need not add to 1.0.

E.g. if the transport D is redistributed over HRUs: A, B and C with their *distrib* parameter having values of a, b and c respectively. The snow transport would be distributed as:

a*D/(a + b +c), b*D/(a + b +c) and c*D/(a + b +c). Note that any excess from a HRU filling to its vegetation height is deposited in the last HRU, i.e. the one having the tallest vegetation.

3. If the value of *distrib* is less than 0 for an HRU, all snow transport to its capacity is assumed to be caught by this HRU. The next HRU is the beginning of a new blowing snow regime.

4. Cumulative drift from the last HRU is not removed from the HRU and is redistributed within the HRU.

pbsmSnobal (Pomeroy and Li, 1999)

Defined in Classpbsm. This module calculates snow transport and sublimation. The snowfall observation can be a daily total or the actual interval snowfall. The former case is handled by assuming that the snowfall is uniformly distributed over every interval of the day. The transport and sublimation of blowing snow are calculated every interval using the interval wind speed, air temperature and relative humidity. The model has been extended to handle the transport of snow between HRUs. At the end of a day when snow transport has occurred, snow transported from HRUs with low roughness is distributed over the HRUs with greater roughness according to the fractions specified in the distribution parameter. No transported snow enters the HRU with the lowest roughness. When HRUs with lower vegetative height fill to their maximum, the excess is distributed over the remaining unfilled HRUs. This is a modification of the original PBSM. SWE is now a state variable of Snobal accessed by pbsmSnobal using a PUT.

Observations

- **none**

Variables

- **hru_subl (mm Δt) - the mass of snow lost from an HRU by sublimation in time step, Δt – expressed as an equivalent average depth of water over an HRU.**
- **cumSubl (mm) - cumulative sublimation.**
- **hru_drift (mm Δt) - the mass of snow lost from an HRU by snow transport in time step, Δt – expressed as an equivalent average depth of water over an HRU.**
- **Drift_out (mm Δt) - the mass of snow lost from an HRU by snow transport in time step, Δt – expressed as an equivalent**
- **Drift_in (mm Δt) - the mass of snow gained from an HRU by snow transport in time step, Δt – expressed as an equivalent**
- **cumDrift (mm) - cumulative transport.**
- **cumSno (mm) - snow (net_snow) accumulation from beginning of winter.**
- **Prob () - interval probability of blowing snow.**
- **BasinSnowLoss (mm Δt) - transport out of basin.**
- **cumBasinSnowLoss (mm) - cumulative transport out of basin.**
- **cumBasinSnowGain (mm) - cumulative transport into basin.**
- **snowdepth (m) - calculated snow depth from SWE (Gray/Pomeroy).**

Parameters

- **fetch (m) - fetch distance.**
- **Ht (m) - crop height.**
- **distrib () - distribution fractions. Value for HRU 1 controls snow transport into the basin.**
- **N_S (1/m^2) - vegetation number density.**
- **A_S (m) - stalk diameter or silhouette.**
- **basin_area (km2) - basin area.**
- **hru_area (km2) - hru_area.**
- **inhibit_evap (flag) - an output parameter set true when the SWE is greater than zero. It is used to inhibit evaporation from the evaporation modules.**
- **inhibit_bs (flag) - an input inhibiting blowing snow when set equal to 1. Inhibited HRU is still able to receive drift from other HRUs.**

Variable Inputs

- **hru_t (°C) - air temperature from module obs.**
- **hru_ea (kPa) - vapour pressure from module obs.**
- **hru_u (m/s) - wind speed from module obs.**
- **net_snow (mm Δt) - snow fall from wild module - intcp, brushintcp etc.**
- **SWE (mm) - snow water equivalent. Put.**
- **z_s (m) - snowcover depth. Put.**
- **rho (kg/m^3) - snow density. Put.**

Notes

1. **The parameter *fetch* cannot be less than 300m.**
2. **The parameter *distrib* for the first HRU is used differently from all the other HRUs. It determines the drift into the model basin based upon the drift out of the first HRU. BasinSnowGain is equal to distrib[HRU1]*Drift[HRU1].**
3. **The first HRU, i.e. the lowest vegetation height. When the snow is redistributed, the *distrib* values for the HRU's not filled to their vegetation height are summed and each HRU receives its share. The sum of distributions need not add to 1.0.**

E.g. if the transport D is redistributed over HRUs: A, B and C with their *distrib* parameter having values of a, b and c respectively. The snow transport would be distributed as:

a*D/(a + b +c), b*D/(a + b +c) and c*D/(a + b +c). Note that any excess from a HRU filling to its vegetation height is

deposited in the last HRU, i.e. the one having the tallest vegetation.

3. If the value of *distrib* is less than 0 for an HRU, all snow transport to its capacity is assumed to be caught by this HRU. The next HRU is the beginning of a new blowing snow regime.

4. Cumulative drift from the last HRU is not removed from the HRU and is redistributed within the HRU.

sbsm (Richard Essery, Long Li and John Pomeroy, 1999)

Defined in Classsbsm. This module is a simplified blowing snow model which can reproduce PBSM results very closely with much less computational effort. The snowfall observation can be a daily total or the actual interval snowfall. The former case is handled by assuming that the snowfall is uniformly distributed over every interval of the day. The transport and sublimation of blowing snow are calculated every interval using the interval wind speed, air temperature and relative humidity. The model has been extended to handle the transport of snow between HRUs. At the end of a day when snow transport has occurred, snow transported from HRUs with low roughness is distributed over the HRUs with greater roughness according to the fractions specified in the distribution parameter. No transported snow enters the HRU with the lowest roughness. When HRUs with lower vegetative height fill to their maximum, the excess is distributed over the remaining unfilled HRUs.

Observations

- none

Variables

- SWE (mm) - snow water equivalent. State variable.
- wet_snow (mm) - wet snow mass
- Subl (mm Δt) - the mass of snow lost from an HRU by sublimation in time step, Δt – expressed as an equivalent average depth of water over an HRU.
- cumSubl (mm/day) - daily sublimation.
- Drift (mm Δt) - the mass of snow lost from an HRU by snow transport in time step, Δt – expressed as an equivalent average depth of water over an HRU.
- cumDrift (mm/day) - daily transport.
- cumDriftIn (mm/day) - cumulative transport to HRU. (local variable).
- cumSno (mm) - snow (net_snow) accumulation from beginning of winter.
- Prob () - interval probability of blowing snow.
- snow_age (hr) - snow age. (local variable).
- BasinSnowLoss (mm Δt) - transport out of basin.
- cumBasinSnowLoss (mm) - cumulative transport out of basin.
- cumBasinSnowGain (mm) - cumulative transport into basin.

Parameters

- fetch (m) - fetch distance.
- Ht (m) - crop height.
- Zr (m) - Ratio of aerodynamic roughness length to vegetation height. See below for typical values.
- distrib () - distribution fractions. Value for HRU 1 controls snow transport into the basin.
- basin_area (km2) - basin area.
- hru_area (km2) - hru_area.
- inhibit_evap (flag) - an output parameter set true when the SWE is greater than zero. It is used to inhibit evaporation from the evaporation modules.

Variable Inputs

- hru_t (°C) - air temperature from module obs.
- hru_rh (%) - relative humidity from module obs.
- hru_u (m/s) - wind speed from module obs.
- newsnow () - 0/1 for no/yes from module obs.
- net_snow (mm Δt) - snow fall from wild module - intcp, brushintcp etc.

Notes

1. The parameter *fetch* cannot be less than 300m.
2. The parameter *distrib* need not be set for the first HRU, i.e. the lowest vegetation height. When the snow is redistributed, the *distrib* values for the HRU's not filled to their vegetation height are summed and each HRU receives its share. The sum of distributions need not add to 1.0.

E.g. if the transport D is redistributed over HRUs: A, B and C with their *distrib* parameter having values of a, b and c respectively. The snow transport would be distributed as:

a*D/(a + b +c), b*D/(a + b +c) and c*D/(a + b +c). Note that any excess from a HRU filling to its vegetation height is deposited in the last HRU, i.e. the one having the tallest vegetation.

3. If the value of *distrib* is less than 0 for an HRU, all snow transport to its capacity is assumed to be caught by this HRU. The next HRU is the beginning of a new blowing snow regime.

4. This module can have a none zero value for transport and sublimation even if the vegetation height is greater than the snow cover depth.

5. Cumulative drift from the last HRU is not removed from the HRU and is redistributed within the HRU.

Typical values fo Zr.

landcover	ice	soil	open tundra	sparse shrub tundra	shrub tundra	dense shrub tundra	sparse forest
vegetation height (ht)	0.01	0.01	0.08	0.08	1.0	1.0	3.0
Zr	0.1	0.1	0.05	0.05	0.05	0.05	0.08

Code.

```
float Classsbsm::sublimation(void){
        return 137.6*pow(hru_u[hh]/25.0, 5)/1000.0;
}

float Classsbsm::scale(void){
float
cond, // Thermal conductivity of air (W/m/K)
diff, // Diffusivity of water vapour in air (m2/s)
rsat, // Saturation density of water vapour (kg/m3)
tk; // Temperature (K)
float const ls = 2.838e6; // Latent heat of sublimation (J/kg)
float const m = 18.01; // Molecular weight of water (kg/kmole)
float const r = 8313.0; // Universal gas constant (J/kmole/K)
        tk = hru_t[hh] + 273.0;
        diff = 2.06e-5*pow(tk/273.0, 1.75);
        rsat = m*611.15*exp(22.45*hru_t[hh]/tk)/(r*tk);
        cond = 0.00063*tk + 0.0673;
        return ((ls*m/(r*tk)) - 1.0)/(cond*(hru_t[hh]+273.0)) + 1.0/(ls*diff*rsat);
}

void Classsbsm::prob(void){
float
mean // Mean of cummulative normal distribution
,var // Standard deviation
,rho // Snow density (kg/m3)
,sd // Snow depth (m)
,us;
bool dry_snow = hru_t[hh] < 0.0;
        mean = 0.365*hru_t[hh] + 0.00706*sqr(hru_t[hh]) + 0.91*log(snow_age[hh]) + 11.0;
        var = 0.145*hru_t[hh] + 0.00196*sqr(hru_t[hh]) + 4.23;
        if (!dry_snow) {
                mean = 21.0;
                var = 7.0;
```

```
        }
        rho = 240.;
        if (SWE[hh] > 145.45)
                rho = 69.856*log(SWE[hh]) - 74.732;
        sd = SWE[hh]/rho;
        us = hru_u[hh];
        if (sd < Ht[hh])
                us = us / sqrt(1. + 170.0*2*zr[hh]*(Ht[hh] - sd));
        Prob[hh] = 1.0/(1.0 + exp(1.7*(mean - us)/var));

        if (SWE[hh] <= wet_snow[hh]) {
                Prob[hh] = 1 / ( 1. + exp(1.7*(21.0 - us)/7.0));
                if (us <= 7.0) Prob[hh] = 0.0;
        }
        if ( sd <= 0.01 ) Prob[hh] = 0.0;
        if ( dry_snow ){
                if (us <= 3.0)
                Prob[hh] = 0.0;
        }
        else{
                if (us <= 7.0)
                Prob[hh] = 0.0;
        }
}
void Classsbsm::run(void) {
float SumDrift, total, SWE_Max, trans;
        for (hh = 0; hh < nhru; hh++) {
           if(net_snow[hh] > 0.0) {
           SWE[hh] = SWE[hh] + net_snow[hh];
           cumSno[hh] = cumSno[hh] + net_snow[hh];
           snow_age[hh] = 1.0;
        }
        else
           snow_age[hh] += dt/3600;
        if(hru_t[hh] >= 0.0)
           wet_snow[hh] = SWE[hh];
        else
           wet_snow[hh] = min<float> (SWE[hh], wet_snow[hh]);
        Drift[hh] = 0.0;
        Subl[hh] = 0.0;
```

```
if(hru_u[hh] > 3.0 && SWE[hh] > 0.0) {
   prob();
   if(Prob[hh] > 0.0) {
      Drift[hh] = Prob[hh]*transport()*dt/fetch[hh];
      Subl[hh] = Prob[hh]*((1.0 - hru_rh[hh])/scale())* sublimation()*dt;
// handle insufficient snow pack
      if(Drift[hh] + Subl[hh] > SWE[hh]) { Subl[hh] = SWE[hh] * Subl[hh]/(Subl[hh] + Drift[hh]); Drift[hh] =
SWE[hh] - Subl[hh]; } // end if
      cumDrift[hh] += Drift[hh];
      cumSubl[hh] += Subl[hh];
   SWE[hh] = SWE[hh] - Subl[hh] - Drift[hh]; } }
} // for
// distribute drift
long LastN = 0;
for (int nn = 0; nn < nhru; ++nn) {
if(distrib[nn] >= 0.0 && nn+1 < nhru) continue;
SumDrift = 0.0;
for (int hh = LastN; hh <= nn; hh++)
       if(distrib[nn] != 0.0)
          SumDrift += Drift[hh]*hru_basin[hh];
if(SumDrift > 0.0){
   for (int hh = LastN + 1; hh <= nn; hh++) {
      if(hh == nn){ // last HRU or last HRU of group
         SWE[hh] += SumDrift/hru_basin[hh];
         cumDriftIn[hh] += SumDrift/hru_basin[hh]; }
      else {
         SWE_Max = SWEfromDepth(Ht[hh]);
if(SWE_Max > SWE[hh] && distrib[hh] > 0.0) {
   total = 0.0;
   for (int jj = hh; jj <= nn; jj++)
      total = total + fabs(distrib[jj]);
   trans = SumDrift*fabs(distrib[hh])/total/hru_basin[hh];
   if(SWE_Max > SWE[hh] + trans)
      SWE[hh] += trans;
   else {
      trans = SWE_Max - SWE[hh];
      SWE[hh] = SWE_Max;
}
SumDrift -= trans*hru_basin[hh]; cumDriftIn[hh] += trans;
} // end if } // end if
} // end for (hh) LastN = nn; } // end if
```

```
} // end for (nn)
for (int hh = 0; hh < nhru; hh++) {
if(SWE[hh] > 0.0) const_cast<long*> (
   inhibit_evap)[hh] = 1;
else
   const_cast<long*> (inhibit_evap)[hh] = 0;
} // for
}
```

walmsley_wind (Walmsley et al., 1989)

This module is defined in Classwalmsley_wind and is parametric version of windflow model of Mason and Sykes (1979), referred as MS. The MS windflow model is a computational routine for estimating windflow over three-dimensional topography and is based on Fourier transform techniques and has a division of the inner and outer flow regions. The MS windflow model is linearized and thus only applies to low hills; the model assumes neutral thermal stratification and uniform surface roughness within the simulation region. The MS model is potentially computational costly and Walmsley et al. (1989) derived a simple parametric version of the MS model for estimating wind speed variation induced by small-scale topographic features. These topographical features include 2D hills (ridges), 3D hills, 2D escarpments, 2D rolling terrain, 3D rolling terrain, and flat terrain. Users can define their simulation topographic features from coefficients 'A' and 'B'.

Observations

- **none.**

Variables

- **hru_Uadjust (m/s) - adjusted wind speed due to topography.**
- **hru_Uchange (m/s) - amount of change in wind speed due to topography.**
- **WR () - wind ratio between adjusted wind speed and reference wind speed.**

Parameters

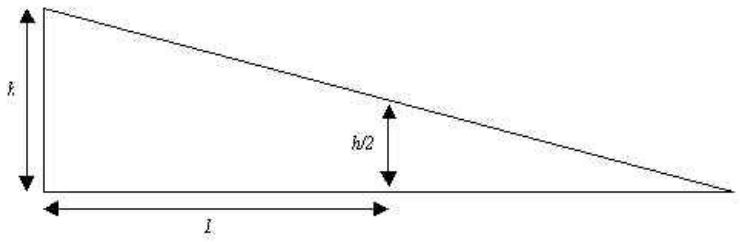
- **Zwind (m) - wind instrument height.**
- **A () - coefficient for wind speed change due to topography. Values: 0.0 – flat terrain, 2.5 – 2D escarpments, 3.0 – 2D hills, 3.5 – 2D rolling terrain, 4.0 – 3D hills, 4.4 – 3D rolling terrain.**
- **B () - coefficient for wind speed change due to topography. Values: 0.0 – flat terrain, 0.8 – 2D escarpments, 1.1 – 3D rolling terrain, 1.55 – 2D rolling terrain, 1.6 – 3D hills, 2.0 – 2D hills.**
- **L (m) - upwind half-width at half height. It is the length from the centre of a hill with height, h to the slope with half height of the hill, h/2. It is also the length from the centre of a depression with depth, h to the slope with half depth of the depression, h/2.**
- **obs_elev (m) - measurement altitude.**
- **hru_elev (m) - altitude.**

Variable Inputs

- **hru_u (m/s) (obs)**

Notes

Parameter *L* is upwind half-width at half height. It is the length from the centre of a hill with height, *h* to the slope with half height of the hill, *h/2*. It is also the length from the centre of a depression with depth, *h* to the slope with half depth of the depression, *h/2*.



Reference

Mason, P.J. and Sykes, R.I. 1979. Flow over an isolated hill of moderate slope.

Quarterly Journal of the Royal Meteorological Society **105: 383-395.**

Walmsley, J.L., Taylor, P.A. and Salmon, J.R. 1989. Simple guidelines for estimating windspeed variations due to small-scale topographic features – an update. *Climatological Bulletin* **23: 3-14.**

intcp

This module will handle summer and winter interception when the algorithms are finalized. At present it is included to handle the change of names required across the canopy because of canopy storage and evaporation. It is in effect a *null* module.

Observations

- **none**

Variables

- **net_rain (mm/int) - the amount of rainfall received by the ground surface under the canopy in time step, int. It is equal to the amount of rain received at the top of the canopy less the amount intercepted by the vegetation.**
- **cumnet_rain (mm) - cummulatve rainfall.**
- **net_snow (mm/int) - the amount of snowfall received by the ground surface under the canopy in time step, int. It is equal to the amount of snowfall received at the top of the canopy less the amount intercepted by the vegetation.**
- **net_p (mm/int) - the total amount of preciptitation (rain + snow) received by the ground surface under the canopy in time step, int. It is equal to the amount of precipitation the top of the canopy less the amount intercepted by the vegetation.**

Parameters

- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**

Variable Inputs

- **hru_rain (mm/int) - HRU rain from module obs.**
- **hru_snow (mm/int) - HRU snow from module obs.**
- **hru_p (mm/int) - HRU precipitation from module obs.**

Similar Modules.

- **brushintcp.**
- **rutter.**

Notes

CanopyClearing.

This module defined in ClassCanopyClearing, models net all-wave radiation at the snow surface under a needleleaf forest canopy from the observed incoming short-wave radiation and the Satterlund (1979) modified form of the Brunt equation. The canopy interception is modelled in summer using 'sparse' Rutter interception model (i.e. Valente 1997) and in winter a coupled forest snow interception and sublimation routine
after Hedstom & Pomeroy / Parviainen & Pomeroy/Ellis:

Observations

- **Qsi (W/m2) - incident short-wave.** ***basic module and variation 1 i.e. CanopyClearing or CanopyClearing#1**
- **Qli (W/m2) - incident long-wave.** ***basic module and variation 2 i.e. CanopyClearing or CanopyClearing#2**

Observation Variable.

- **Ts (°C) - snow surface temperature.**
- **Qnsn (W/m2) - net all-wave at snow surface.**
- **Qsisn (W/m2) - incident short-wave at snow surface.**
- **Qlisn (W/m2) - incident long-wave at snow surface.**
- **Qlisn (W/m2) - reflected long-wave at snow surface.**
- **Qlosn (W/m2) - reflected long-wave at snow surface.**

Variables

- **drip_cpy (mm/int) - canopy drip.**
- **direct_rain (mm/int) - direct rain through canopy.**
- **net_rain (mm/int) - direct rain + drip.**
- **cum_net_rain (mm) - cumulative net_rain.**
- **Subl_Cpy (mm/int) - canopy snow sublimation.**
- **cum_Subl_Cpy (mm) - cumulative canopy snow sublimation.**
- **rain_load (mm) - canopy rain load.**
- **snow_load (mm) - canopy snow load.**
- **direct_snow (mm/int) - direct snow through canopy.**
- **SUnload (mm/int) - canopy snow unloaded.**
- **net_snow (mm/int) - hru_snow less snow interception.**
- **Cum_net_snow (mm) - cumulative net_snow.**
- **u_FHt (m/s) - wind speed at canopy top.**
- **cc () - canopy coverage.**
- **intcp_evap (mm/int) - evaporation from the canopy.**
- **cum_intcp_evap (mm) - cumulative canopy evaporation.**
- **Pevap (mm/int) - Priestley/Taylor canopy evaporation when snowcover developed.**
- **k () - extinction coefficient.**
- **Tauc () - short-wave transmissivity through the canopy.**
- **ra (s/m) - resistance.**
- **Pa (kPa) - atmospheric pressure calculated from the elevation.**
- **Qnsn_var (W/m2) - net all-wave at snow surface.**
- **Qsisn_var (W/m2) - incident short-wave at snow surface.**
- **Qlisn_var (W/m2) - reflected long-wave at snow surface.**

Parameters

- **CanopyClearing () - canopy/clearing - 0/1.**
- **Ht (m) - canopy height.**
- **AlbedoCanopy () - initial average HRU snow cover albedo. Only use by canopy. However should be set to 0.0 for clearing. See note below.**
- **hru_elev (m) - HRU mean elevation.**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **Z0snow (m) - roughness length.**
- **Zref (m) - temperature measurement height.**
- **Zwind (m) - wind measurement height.**
- **LAI (m2/m2) - leaf area index.**
- **inhibit_evap (flag) - 0/1 enable/inhibit.**
- **Sbar (kg/m^2) - maximum canopy snow interception load.**
- **Zvent (m) - ventilation wind speed height.**
- **unload_t (°C) - ice-bulb temperature when canopy snow is unloaded as snow. Only use by canopy.**
- **unload_t _water(°C) - ice-bulb temperature when canopy snow is unloaded as water. Only use by canopy.**

Variable Inputs

- **hru_t (°C) - (*) air temperature.**
- **hru_rh (%) - (*) relative humidity.**
- **hru_ea (kPa) - (*) vapour pressure. Often calculated by an observation filter from t and rh.**

- hru_u (m/s) - (*) average wind speed over time step.
- SWE (mm) - (*) snow water equivalent.
- SolAng (radians) - (*) from global or similar module.
- hru_rain (mm/int) - HRU rain from module obs.
- hru_snow (mm/int) - HRU snow from module obs.
- hru_evap (*) (mm/int). - Maximum evaporation when there is no snowcover.
- QliVt_var (W/m2) - reflected long-wave at snow surface. *variation 1 and 3 i.e. CanopyClearingGap#1 or #3
- QsiS_var (W/m2) - incident short-wave at snow surface. *variation 2 and 3 i.e. CanopyClearingGap#2 or #3

Notes.

The parameter Ht is used for the canopy height (vegetation) which is consistent with CRHM..

A parameter AlbedoCanopy is used for the canopy albedo. This may be handled using an external module or macro if the value changes with the season..

There is a problem.with the application of AlbedoCanopy. It modifies Qnsn for canopy and clearing as follows:

Qnsn[hh] = Qlisn[hh] - Qlosn[hh] + Qsisn[hh]*(1.0 - AlbedoCanopy[hh]) // for both canopy and clearing

AlbedoCanopy should be set to 0.0 for clearing. Since AlbedoCanopy for canopy is low (say 0.1) the error is usually very small if this is inadvertently used for clearing..

The canopy Tauc is not corrected to handle the change of radiation extinction path length for slope and aspect effects. Refer to the module CanopyClearingGap.

CanopyClearingGap.

This module defined in ClassCanopyClearingGap, models net all-wave radiation at the snow surface under a needleleaf forest canopy from the observed incoming short-wave radiation and the Satterlund (1979) modified form of the Brunt equation. The canopy interception is modelled in summer using 'sparse' Rutter interception model (i.e. Valente 1997) and in winter a coupled forest snow interception and sublimation routine after Hedstom & Pomeroy / Parviainen & Pomeroy/Ellis:

Observations

- **Qsi (W/m2) - incident short-wave. *basic module and variation 1 i.e. CanopyClearingGap or CanopyClearingGap#1**
- **Qli (W/m2) - incident long-wave. *basic module and variation 2 i.e. CanopyClearingGap or CanopyClearingGap#2**

Observation Variable.

- **Ts (°C) - snow surface temperature.**
- **Qnsn (W/m2) - net all-wave at snow surface.**
- **Qsisn (W/m2) - incident short-wave at snow surface.**
- **Qlisn (W/m2) - incident long-wave at snow surface.**
- **Qlisn (W/m2) - reflected long-wave at snow surface.**
- **Qlosn (W/m2) - reflected long-wave at snow surface.**

Variables

- **drip_cpy (mm/int) - canopy drip.**
- **direct_rain (mm/int) - direct rain through canopy.**
- **net_rain (mm/int) - direct rain + drip.**
- **cum_net_rain (mm) - cumulative net_rain.**
- **Subl_Cpy (mm/int) - canopy snow sublimation.**
- **cum_Subl_Cpy (mm) - cumulative canopy snow sublimation.**
- **rain_load (mm) - canopy rain load.**
- **snow_load (mm) - canopy snow load.**
- **direct_snow (mm/int) - direct snow through canopy.**
- **SUnload (mm/int) - canopy snow unloaded.**
- **net_snow (mm/int) - hru_snow less snow interception.**
- **Cum_net_snow (mm) - cumulative net_snow.**
- **u_FHt (m/s) - wind speed at canopy top.**
- **cc () - canopy coverage.**
- **intcp_evap (mm/int) - evaporation from the canopy.**
- **cum_intcp_evap (mm) - cumulative canopy evaporation.**
- **Pevap (mm/int) - Priestley/Taylor canopy evaporation when snowcover developed.**
- **k () - extinction coefficient.**
- **Tauc () - short-wave transmissivity through the canopy.**
- **ra (s/m) - resistance.**
- **Pa (kPa) - atmospheric pressure calculated from the elevation.**
- **Qnsn_var (W/m2) - net all-wave at snow surface.**
- **Qsisn_var (W/m2) - incident short-wave at snow surface.**
- **Qlisn_var (W/m2) - reflected long-wave at snow surface.**

Parameters

- **CanopyClearing () - canopy/clearing/gap - 0/1/2.**
- **Ht (m) - canopy height.**
- **AlbedoCanopy () - initial average HRU snow cover albedo. Only use by canopy.**
- **hru_elev (m) - HRU mean elevation.**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **Z0snow (m) - roughness length.**
- **Zref (m) - temperature measurement height.**
- **Zwind (m) - wind measurement height.**
- **LAI (m2/m2) - leaf area index.**
- **inhibit_evap (flag) - 0/1 enable/inhibit.**
- **Sbar (kg/m^2) - maximum canopy snow interception load.**
- **Zvent (m) - ventilation wind speed height.**
- **unload_t (°C) - ice-bulb temperature when canopy snow is unloaded as snow. Only use by canopy selection.**
- **unload_t _water(°C) - ice-bulb temperature when canopy snow is unloaded as water. Only use by canopy selection.**
- **Gap_diameter (m) - diameter of gap. Only use by gap selection.**
- **Surrounding_Ht (m) - height of canopy around gap. Only use by gap selection.**

Variable Inputs

- hru_t (°C) - (*) air temperature.
- hru_rh (%) - (*) relative humidity.
- hru_ea (kPa) - (*) vapour pressure. Often calculated by an observation filter from t and rh.
- hru_u (m/s) - (*) average wind speed over time step.
- SWE (mm) - (*) snow water equivalent.
- SolAng (radians) - (*) from global or similar module.
- cosxs (radians) - (*) from global or similar module.
- cosxsflat (radians) - (*) from global or similar module.
- hru_rain (mm/int) - HRU rain from module obs.
- hru_snow (mm/int) - HRU snow from module obs.
- hru_evap (*) (mm/int). - Maximum evaporation when there is no snowcover.
- QliVt_var (W/m2) - reflected long-wave at snow surface. *variation 1 and 3 i.e. CanopyClearingGap#1 or #3
- QsiS_var (W/m2) - incident short-wave at snow surface. *variation 2 and 3 i.e. CanopyClearingGap#2 or #3

Notes.

The parameter Ht is used for the canopy height (vegetation) which is consistent with CRHM..

A parameter AlbedoCanopy is used for the canopy albedo. This may be handled using an external module or macro if the value changes with the season..

The canopy Tauc has been corrected to handle the change of radiation extinction path length for slope and aspect effects.

Originally:

Tauc = exp(-k*LAI_)

Now:

Tauc = exp(-k*LAI_*1/cosxs)

albedo(Gray and Landine, 1987)

This module is defined in Classalbedo and estimates the snow albedo throughout the winter and into the melt period. It also estimates the beginning of melt which initiates the snowmelt in the module ebsm. The algorithm developed uses daily values of net radiation, maximum and minimum daily temperatures, snowfall and snow-cover SWE.

Observations

- **none**

Variables

- **albedo () - snowcover albedo. State variable.**
- **meltflag () - indicates start of melt. 0/1 for false/true. State variable.**
- **winter () - indicates summer and winter events. 0/1. State variable.**

Parameters

- **Albedo_bare () - albedo for bare soil. Advance parameter.**
- **Albedo_snow () - albedo for fresh snow. Advance parameter.**
- **hru_lat (°) - latitude. Adjusts empirical Julian constants for Northern/Southern Hemisphere.**

Variable Inputs

- **hru_tmax (obs)**
- **hru_tmin (obs)**
- **hru_newsnow (obs)()**
- **QdroD (global) (MJ/m^2*d)**
- **SWE (pbsm)(mm)**
- **net_snow (intcp)(mm/int)**

Notes

winter

The logical flag *winter* differentiates the major spring melt from fall and late spring snowfalls and associated melt events. *Winter* is set true between Julian days 300 and 2 when the SWE of the last HRU is greater than 5 mm. *Winter* is also set true when the daily maximum temperature is less than -6°C and the diagnostic variable Qnc is less than 1.0. *Winter* is set false when any HRU has no snowcover, i.e. SWE = 0.0.

meltflag

The logical flag *meltflag* indicates the days when melt is possible. It is always true when there is snowcover and *winter* is false. Otherwise it is only set when one of the melt criteria is met,

1. **the daily minimum is greater than -4.0°C**
2. **Qnc > 1.0 AND the daily maximum > 0.0°C**
3. **Qnc > -0.5 AND the daily maximum > MT**

where Qnc = -0.371 + 5.22*QdroD[hh]*(1 - A[hh]) and is a measure of incoming short-wave radiation,

and MT = -0.064*jday + 6.69 and is a probabiliy index which increases as the radiative components increase as the year progresses.

newsnowcount

This counter is set to 0 after every snowfall and is incremented every day afterwards until amother snowfall occurs.

Albedo relationships.

At the onset of winter the albedo is set to 0.8. After a snowfall of greater than 5 cm depth (equivalent to a change in SWE of 0.25 mm, assuming a snow density of 0.005 kg/m3)

albedo_param.

This module is defined in Classalbedoparam. It sets the variable Albedo to the value of the parameter Albedo_value.

Observations

- **none**

Variables

- **Albedo () - Tthis variable is set to value of Albedo_value.**

Parameters

- **Albedo_value () - parameter value.**

Variable Inputs

- **none.**

Notes.

The module *albedo_obs* is more flexible as it does the same as this module but can also read albedo observations.

albedo_obs.

This module is defined in Classalbedoobs. It read a series of albedo observations into the variable Albedo.

Observations

- **Albedo_obs () - average albedo over the time interval. Optional.**

Variables

- **Albedo () - Observation values.**

Parameters

- **Albedo_value () - default value. Used if observation are not available and initially until first available observation.**

Variable Inputs

- **none.**

Notes

If the Observation is not available Albedo takes the value of Albedo_value.

global

Defined in ClassGlobal. This module calculates the theoretical interval short-wave direct and diffuse solar radiation. The routine handles GSL and ASL. There is an adjustment for elevation and transmissivity applicable to the region adjacent to the triangle, Edmonton, Winnipeg and Bad Lake. The interval and daily values are calculated by accumulating the 10 minute calculated values. The maximum number of daily sunshine hours is calculated. The theoretical direct-beam component of solar radiation is developed from an expression proposed by Garnier and Ohmura (1970). The solar angle is output for use in other modules. Extra variables have been added to help in determining if the module is malfunctioning

Observations

- **none**

Variables

- **Qdro (W/m^2) - clear-sky direct.**
- **Qdfo (W/m^2) - clear-sky diffuse.**
- **Qdflat (W/m^2) - clear-sky direct + diffuse on a horizontal surface. Always independent of slope and aspect.**
- **QdflatE (W/m^2) - extraterrestrial radiation, i.e. no atmosheric attenuation. Always independent of slope and aspect.**
- **QdroD (MJ/m^2) - daily clear-sky direct.**
- **QdfoD (MJ/m^2) - daily clear-sky diffuse.**
- **QdflatD (MJ/m^2) - daily clear-sky direct on a horizontal surface. Now includes the diffuse.**
- **QdroDext (W/m^2) - daily extraterrestial direct. Always independent of slope and aspect.**
- **SolAng (r) - solar angle in radians.**
- **SunMax (h) - maximum sunshine hours.**
- **pQdro (MJ/m^2*int) - clear-sky direct.**
- **pQdfo (MJ/m^2*int) - clear-sky diffuse.**
- **pSol(r) - solar angle in radians.**

Parameters

- **hru_lat (°) - latitude. Positive value indicates northern hemisphere, negative value southern hemisphere.**
- **hru_elev (m) - altitude.**
- **hru_GSL (°) - ground slope - increasing the slope positively, tilts the plane to the north with ASL = 0.**
- **hru_ASL (°) - azimuth. 0/90/180/270 - north/east/south/west facing for positive GSL.**
- **Time_Offset (h) - solar time offset from local time.**

Variable Inputs

- **none**

Notes

Calculation of Diffuse.

A simple means of calculating the dffuse clear-sky radiation is given by List (1968) as:

diffuse = 0.5((1-aw-ac)Qa - Id) where

aw = radiation absorbed by water vapour (7%)

ac = radiation absorbed by ozone (2%)

Qa = extra-terrestrial radiation on a horizontal surface at the outer limit of the earth's atmosphere.

Id = direct clear-sky radiation reaching the earth's surface on a horizontal surface.

Optical airmass.

The relative path length of the direct solar beam radiance through the atmosphere. When the sun is directly above a sea-level location the path length is defined as airmass 1 (AM 1.0). AM 1.0 is not synonymous with solar noon because the sun is usually not directly overhead at solar noon in most seasons and locations. When the angle of the sun from zenith (directly overhead) increases, the airmass increases approximately by the secant of the zenith angle. A better calculation *(Kasten, F. and A. T. Young (1989). Revised optical air mass tables and approximation formula.* ***Applied Optics 28*** *(22), 4735-4738)* **follows:**

$$m = 1.0 / [\cos(Z) + 0.50572 * (96.07995 - Z)^{-1.6364}]$$

where Z is the solar zenith angle.

Annandale

Defined in ClassAnnandale. The purpose of this module is to estimate incoming short wave radiation from daily minimum and maximum temperatures when no other radiation observations are available. Annandale et al. (2001) developed a simple emipirical method for estimating transmittance from the daily range of air temperature.

Observations

- **none**

Variables

- **TauA () - transmittance calculated from the Annandale method.**
- **hru_SunAct (h) - calculated sunshine hours.**

Parameters

- **krs () - location index, interior = 0.16 and coastal = 0.19.**
- **hru_elev (m) - HRU elevation.**

Variable Inputs

- **hru_tmax (*) (°C) - daily maximum temperature.**
- **hru_tmin (*) (°C) - daily minimum temperature.**
- **QdroDext (*) (MJ/m^2*d) - daily ExtraTerrestrial direct.**
- **QdflatD (*) (MJ/m^2*d) - daily clear-sky Qdro (with diffuse) on horizontal surface.**
- **QdflatE (*) (W/m^2) - 'Qdro' on horizontal surface, no atmosheric attenuation.**
- **Qdro (*) (W/m^2) - clear-sky direct with GSL and ASL applied.**
- **Qdfo (*) (W/m^2) - clear-sky diffuse with GSL and ASL applied.**
- **QdfoD (*) (MJ/m^2*d) - daily average clear-sky diffuse.**
- **SunMax (*) (h) - maximum sunshine hours.**

Observation Variables

- **QsiA (W/m^2) - interval synthesized short-wave derived from Annandale method.**
- **QsiD (W/m^2) - daily synthesized short-wave derived from Annandale method.**
- **QsiS (W/m^2) - synthesized short-wave on slope derived from global Qdro and Qdfo, and the Annandale method.**

netall

This module defined in Classnetall, models net all-wave radiation from the calculated short-wave radiation and the Brunt equation. Climatic data required are temperature, vapour pressure and actual sunshine hours.

Classnetall is an enhancement of the original class which provides interval radiation for the the day to allow modules to partition calculated daily evaporation over the day to intervals with positive net radiation. These calculations must be done at the beginning of the day to be available throughout the day.

Observations

none

Variables

- net (MJ/m^2 Δt) - average all-wave net over time step.
- Rn (mm/m^2 Δt) - average all-wave net over time step.
- netD (MJ/m^2 day) - average all-wave net over time day.
- RnD (mm/m^2 day) - average all-wave net over time day.
- RnD_Pos (mm/m^2 day) - sum of positive all-wave net over time day.
- cum_net (MJ/m^2) - cummulative all-wave net.

Parameters

- Albedo () - areal albedo. // pseudo

Variable Inputs

- hru_t (obs)
- hru_ea (obs)
- Tday (obs) t interval values for entire day.
- EAday (obs) ea interval values for entire day.
- hru_SunAct (obs)
- SunMax (global)
- Qdro (global)
- Qdfo (global)
- pQdro (global)
- pQdfo (global)
- albedo (albedo)

Returns

- returns (netlongwave + netshortwave*(1-hru_alb))/Freq

  where:

  netlongwave = -0.85 + 0.97*SB*pow(hru_t+273.0, 4)*(-0.39+0.093*sqrt(hru_ea)*(0.26+0.81*(hru_SunAct/SunMax))

  if(hru_SunAct)

  ```
  netshortwave = (0.024+0.974*pow(hru_SunAct/SunMax, 1.35))*Qdro*0.0864
  +(2.68+2.2*(hru_SunAct/SunMax)-3.85*sqr(hru_SunAct/SunMax))*Qdfo*0.0864
  ```

  else

  ```
  netshortwave = (0.024*Qdro + 2.68*Qdfo)*0.0864
  ```

  Freq is the number of intervals in the day.

long

This module defined in Classlong calculates long-wave radiation. Incoming Longwave radiation to Melting Snow: Observations, Sensitivity and Estimation in Northern Environments (Sicart et al., 2005?).

Observations

- Qsi (W/m^2) - incident short-wave.

Observation Functions

- QsiDtot(W/m^2*d) - daily total (cumulative) of average interval incident short-wave fluxes, Qsi.

observationVariables

- Qli# (W/m^2) - calculated incoming long-wave.

Variables

- tau () - atmospheric transmissivity.
- Long (W/m^2) - calculated incoming long-wave. Same as generated observation.

Parameters

- none.

Variable Inputs

- hru_t (°C) - air temperature.
- hru_ea (kPa) - vapour pressure.
- hru_rh (%) - relative humidity.
- QdroDext (global)(MJ/m^2*d) - daily extraTerrestial direct.
- QdroD (global) (MJ/m^2*d) - daily clear-sky direct.

Notes

tau = QsiDtot*(864000/10E6)/QdroDext.

Qli = 1.24*(e/T)^(1/7) (1 + 0.44*RH - 0.18*tau)sbcT^4 where RH lie in the range of 0 and 1 and T is °K.

The declared observation, Qsi# cannot be used as input as the daily total cannot be calculated. If an attempt is made by the module to use a declared observation it will be automatically converted to the simple observation, i.e. Qsi# converted to Qsi.

History.

March 2008 - temperature, vapour pressure and relative humidity changed from using observations to the HRU variables.

calcsun

This module is defined in Classcalcsun and can estimate sunshine hours from the the incoming short-wave observation Qsi and the values QdroD, QdfoD and SunMax calculated by the module global using the HRU's latitude.

Observations

- **Qsi (W/m^2) - average incident short-wave flux for time interval (optional). Used only if sunshine hours are not available.**
- **SunAct (h) - sunshine hours (optional). When not available the sunshine hours are estimated from Qsi, if available.**

Observation Functions

- **Qsitot(W/m^2*d) - daily total (cumulative) of average interval incident short-wave fluxes, Qsi.**

Variables

- **hru_SunAct (h) - calculated sunshine hours.**

Parameters

- **none**

Variable Inputs

- **QdfoD (global) (MJ/m^2*d)**
- **QdflatD (global) (MJ/m^2*d)**
- **SunMax (global) (h)**

Notes.

When the actual sunshine hours are not available as an observation, an estimate is made from the following,

hru_SunAct = (daily sum of Qsi*WtoMJ_D/Freq - QdfoD) * (theoretical maximum sunshine hours)/(QdflatD - QdfoD) where WtoMJ_D = 86400/10^6 and Freq is intervals/day,

otherwise the observation SunAct is used.

NeedleLeaf.

This module defined in Classneedleleaf, models net all-wave radiation at the snow surface under a needleleaf forest canopy from the observed incoming short-wave radiation and the Satterlund (1979) modified form of the Brunt equation. Other climatic data required are air temperature, relative humidity/vapour pressure and windspeed.

Observations

- Qsi (W/m2) - incident short-wave.
- Qli (W/m2) - incident long-wave. Optional.

Observation Variable.

- Ts (°C) - snow surface temperature.
- Qnsn (W/m2) - net all-wave at snow surface.
- Qsisn (W/m2) - incident short-wave at snow surface.
- Qlisn (W/m2) - incident long-wave at snow surface.
- Qlisn (W/m2) -refected long-wave at snow surface.

Variables

- k () - extinction coefficient.
- Tauc () - short-wave transmissivity through the canopy.
- ra (s/m) - resistance.
- Pa (kPa) - atmospheric pressure calculated from the elevation.

Parameters

- Ht (m) - canopy height.
- Albedo () - initial average HRU snow cover albedo.
- hru_elev (m) - HRU mean elevation.
- Z0snow (m) - roughness length.
- Zref (m) - temperature measurement height.
- Zwind (m) - wind measurement height.
- LAI (m2/m2) - leaf area index.

Variable Inputs

- hru_t (°C) - (*) air temperature.
- hru_rh (%) - (*) relative humidity.
- hru_ea (kPa) - (*) vapour pressure. Often calculated by an observation filter from t and rh.
- hru_u (m/s) - (*) average wind speed over time step.
- SWE (mm) - (*) snow water equivalent.
- albedo (albedo) - (*)
- SolAng (radians) - (*) from global or similar module.

Notes.

```
Pa[hh] = 101.3*pow((293.0-0.0065*hru_elev[hh])/293.0, 5.26); // kPa

Exposure = Ht[hh] - DepthofSnow(SWE[hh]); /* depths(m) SWE(mm)

  if(Exposure < 0.0) Exposure = 0.0

LAI_ = LAI[hh]*Exposure/Ht[hh]

Vf_ = Vf[hh] + (1.0 - Vf[hh])*sin((Ht[hh] - Exposure)/Ht[hh]*M_PI_2)

if(beta[hh] > 0.001) {

        k[hh] = 1.081*beta[hh]*cos(beta[hh])/sin(beta[hh]);

        Tauc[hh] = exp(-k[hh]*LAI_[hh]);

}

else {

        k[hh] = 0.0; Tauc[hh] = 0.0;

}

rho[hh] = Pa[hh]*1000/(Rgas*(t[0] + 273.15));
```

```
float U1 = max(u[0], 1.0e-3); // Wind speed (m/s)

ra[hh] = sqr(log(z1[hh]/z0[hh]))/(sqr(kappa)*U1);

delta[hh] = 0.622*Ls*Qstar[hh]/(Rgas*sqr(t[0] + 273.15));

float q = (rh[0]/100)*Qs(Pa[hh], T1); // specific humidity (kg/kg)

Ts[hh] = T1 + (emiss*(Qli[Qlimax] - sbc*pow(T1, 4)) + Ls*(q - Qs(Pa[hh], T1))*rho/ra[hh])/(4*emiss*sbc*pow(T1, 3) + (Cp + Ls*delta)*rho/ra[hh])

// Canopy temperature is approximated by the air temperature.

Qn[hh] = Vf_[hh]* Qli[Qlimax] + (1.0 - Vf[hh])*emiss_c*sbc*pow(T1, 4.0) - emiss*sbc*pow(Ts[hh] + Tm, 4.0) + Qsi[0]*Tauc[hh]*(1-Albedo[hh])
```

Constants

sbc = 5.67E-8 (W/m2/k4) Stephan-Boltzmann constant.

Ls = 2.845e6 (J/kg) latent heat of sublimation.

Lf = 0.334e6 (J/kg) latent heat of fusion.

Rgas = 287.0 (J/kg/K) gas constant for dry air.

rho = 1.23 (kg/m3) density of dry air.

Cp = 1005 (J/kg/K) volumetric heat capacity of dry air.

Tm = 273.15 (K) melting point.

Kevin

This module is defined in ClassKevin. Snow accounting is handled in PBSM using snow fall, transport and sublimation.

Observations

- **hru_Qn (obs) - net radiation (W/m^2). Optional.**

Variables

- **albedo () - areal albedo.**
- **snowmelt (mm/int) - depth/amount of snowmelt in time step, ∆t.**
- **snowmeltD (mm/d) - daily snowmelt.**
- **cumsnowmelt (mm) - index day melt.**
- **sca () - snow covered fraction. State variable.**
- **winter () - winter = 0/1 - false/true. State variable.**
- **SnowStat () - EARLY/MATURE/HOLD 0/1/2. State variable.**

Parameters

- **Asnow1 () - early snow albedo,. fresh snow, newly-fallen snow, snow at the start of ablation or "active melt".**
- **Asnow2 () - mature snow albedo, wet snow, metamorphosed snow.**
- **Asoil () - soil albedo.**
- **cv () - coefficient of variation.**
- **tfactor (mm/d*/ºC) - degree-day melt factor.**
- **nfactor (mm*m^2/MJ*d) - net radiation factor. Converts daily net radiation flux (MJ/m2) to an equivalent average depth of water**
- **meltthresh(mm/d) - amount of daily melt that must be exceeded in order for a snowcover to become a mature pack.**
- **hru_lat (°) - latitude. Adjusts empirical Julian constants for Northern/Southern Hemisphere.**

Variable Inputs

- **hru_t (obs) - air temperature (°C) from module obs.**
- **hru_tmean (obs) - air temperature (°C) from module obs.**
- **hru_eamean (obs) - mean daily vapour pressure (kPa) from module obs.**
- **hru_SunAct(obs) - actual sunshine hours (h).**
- **SunMax (global) - maximum sunshine hours (h).**
- **Qdro (global) - calculated clear-sky direct (W/m^2).**
- **Qdfo (global) - calculated clear-sky diffuse (W/m^2).**
- **SWE (pbsm) - Snow water equivalent (mm).**

Internal Variables

- **SWEpeak (mm) - maximum SWE before melt.**
- **SWElast (mm) - last SWE before new snow fall during melt.**
- **daymelt (mm) - cumulated melt during a day. Used to determine when to go from the Early to the Mature phase.**

Melt Algorithms

- **tfactor OR nfactor greater than zero.**

  melt (mm/int) = (hru_t*tfactor + hru_Qn*nfactor*WtoMJ_D)/Freq individual terms used when positive.

- **tfactor AND nfactor both equal to zero.**

  netlong = -0.85 + 0.97*SB*pow(hru_tmean+273.0, 4)*(-0.39+0.093*sqrt(hru_eamean))*(0.26+0.81*(hru_SunAct/SunMax))

  if(hru_SunAct > 0.0)

  shortw = (0.024+0.974*pow(hru_SunAct/SunMax, 1.35))*Qdro*WtoMJ_D
  +(2.68+2.2*(hru_SunAct/SunMax)-3.85*sqr(hru_SunAct/SunMax))*Qdfo*WtoMJ_D

  else

  shortw = (0.024*Qdro + 2.68*Qdfo)*WtoMJ_D

  net = (shortw*(1.0-albedo) + netlong)/Freq; // MJ/day to MJ/interval.

  where albedo = asnow1 for EARLY and = asnow2 for MATURE

  melt (mm/int) = net/Hf // Hf = 0.3336 Latent heat of fusion, MJ/kg.

Snow Covered Area (SCA)

This is calculated by a routine SWE_prob from Shook 1995. The lognormal probability density function is used to describe the frequency distribution of the snow water equivalent. The routine calculates the exceedence probability for a value of SWE (Melt), i.e., it is the probability of a given SWE-value being exceeded. The value SWEpeak is the arithmetic mean of the snow pack before the onset of melt.

Model Flow Structure

The season is handled as three separate states

1. EARLY - snow accumulation during early winter.
2. MATURE - snow accumulation in spring after a significant melt.
3. HOLD - occurs during the Mature state when a heavy snow fall occurs causing the snow pack to increase again.

EARLY: is the default state entered in the fall. SCA is 100% and the snow albedo is set to "Asnow1". After the spring melt it is re-entered when SCA is less than 1%.

MATURE: is entered when a melt has exceeded the value of "meltthresh". The albedo is set to "Asnow2". The SCA is calculated from the peak SWE (SWEpeak) and the remaining snow water equivalent (SWEpeak - SWElast).

HOLD: is entered from the Mature state when the SWE has exceeded the last interval SWE (SWElast). It will stay in this state until the SWE becomes less than value when the state was entered (SWElast) or SWE becomes greater than SWEpeak. SCA is 100% and the snow albedo is set to "Asnow1"

Status

- incomplete

Slope_Qsi.

This module estimates Qsi for a slope from the measured incoming short-wave Qsi on the level. The ratio of measured Qsi and the calculated clear sky direct and diffuse on a horizontal plane is used to adjust the calculated clear sky value on the slope.

Observations

- **Qsi (W/m^2) - incident short-wave measured on a level plane.**

Variables

- **Short (W/m^2) - Calculated Qsi for the slope equal to Qdro + Qdfo.**
- **ShortC (W/m^2) - Qsi corrected for slope from the ratio of (measured Qsi)/Qdflat.**

Parameters

- **none.**

Variable Inputs

- **Qdro (global) (W/m^2).**
- **Qdfo (global) (W/m^2).**
- **Qdflat (global)(W/m^2).**

Observation Variables

- **Qsi# (W/m^2) - Qsi corrected for slope.**

Truncation note.

In this implementation there is truncation at either sunrise or sunset since the alogorithm is only active when the incoming net short wave radiation on the level is positive. This means that any incoming radiation due to the slope at before dawn and after dusk is ignored unless the incoming is positive on the level.

evap (Granger, 1989; Granger and Pomeroy, 1997)

This module defined in Classevap, calculates the evapotranspiration using interval values from after snowmelt to fall snow cover.

It has been enhanced to partition calculated daily Granger evaporation over the day to intervals with positive net radiation.

Other evaporation algorithm available is Priestley-Taylor.

Observations

- **none**

Variables

- **hru_evap (mm/int) - average depth of evaporation from an HRU in a time step, Δt.**
- **hru_cum_evap (mm) - cumulative evaporation – calculated as the sum of interval estimates of hru_evap.**
- **hru_evapD (mm/d) - daily evaporation – calculated as the daily sum of interval estimates of hru_evap. Should equal evap_GrangerD**
- **evap_GrangerD (mm/d) - *daily* evaporation calculated from daily means of t, u and vapour pressure.**
- **evap_G () - relative evaporation from Granger.**
- **evap_D () - relative drying power from Granger.**
- **hru_actet (mm/int) - actual evapotranspiration over the HRU. Evaporation and evapotranspiration are limited by the amount of soil moisture available.**
- **hru_cum_actet (mm) - cumulative actual evapotranspiration over the HRU.**

Parameters

- **evap_type (flag) - Evaporation method for HRU, 0 - Granger, 1 - Priestley Taylor.**
- **Ht (m) - crop height.**
- **hru_elev (m) - altitude.**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **F_Qg () - fraction to ground flux. Qg = F_Qg*Rn.**
- **inhibit_evap (flag) - 0/1 enable/inhibit.**

Variable Inputs

The source of the variable is given in brackets after the name. A full description of the variable may be found there.

- **hru_t (obs) (°C)**
- **hru_tmean (obs) (°C)**
- **hru_umean (*) (m/s)**
- **hru_eamean (obs) (kPa)**
- **Rn (*) (mm/m^2*int)**
- **RnD (*) (mm/m^2*d)**
- **RnD_Pos (*) (mm/m^2*d)**

Returns

Daily calculation of Granger.

- **Ea = fdaily(u, Ht)*(e*-ea)**
- **D = Ea/(Ea+(Rn-Qg-Qs))**
- **G = 1/(0.793 + 0.2*exp(4.902*D) + 0.006*D**
- **E = (delta(t)*G*(Rn-Qg-Qs) + gamma(Pa, t)*G*Ea)/(delta(t)*G + gamma(Pa, t))**

  where t, u and ea are the mean of interval values and e* is calculated using the mean daily temperature t.

Priestley Taylor

- **1.26*delta(t)*(Rn-Qg-Qs)/((delta(t) + gamma(Pa, t)) for positive incoming net radiation,**

  where values are interval. Evaporation is zero when incoming net radiation is <= 0.0.

Atmospheric Pressure

- **P = P0*exp((293.0-0.0065*elevation)/293.0, 5.26),**

  where P0 = 101.3.

Granger Daily Evaporation Partitioned over daily intervals when Rn is positive.

- **hru_evap = hru_evapD*Rn[t_n] / RnD_POS,**

  where RnD_POS is the sum of daily intervals when Rn[Δt] is positive.

evapX - Evaporation calculated using Granger and Pomeroy, 1997, Priestley-Taylor and Penman-Monteith methods.

This module defined in ClassevapX, calculates the evapotranspiration from after snowmelt to fall snow cover.

It calculates the daily evapotranspiration using Granger and partitions it over the day to interval values using intervals with positive net radiation.

The interval values are calculated for the Priestley-Taylor and Penman-Monteith methods.

Observations

- RnObs (W/m^2) - all-wave net radiation used by Penmon_Monteith and Dalton Bulk transfer method (optional).
- Qsi (W/m^2) - incident short-wave (optional).

Variables

- hru_evap (mm/int) - average depth of evaporation from an HRU in a time step, Δt.
- hru_cum_evap (mm) - cumulative evaporation – calculated as the sum of interval estimates of hru_evap.
- hru_evapD (mm/d) - daily evaporation – calculated as the daily sum of interval estimates of hru_evap. Should equal evap_Granger_D etc.
- hru_actet (mm/int) - actual evapotranspiration over the HRU. Evaporation and evapotranspiration are limited by the amount of soil moisture available.
- hru_cum_actet (mm) - cumulative actual evapotranspiration over the HRU.
- Granger_D (mm/d) - daily evaporation calculated from daily means of t, u and vapour pressure using Granger_D method.
- Priestley_Taylor_D (mm/d) - daily evaporation calculated from daily means of t, u and vapour pressure using Priestley_Taylor method.
- rc (s/m) - stomatal resistance (used by Penman-Monteith and Dalton).
- Pa (kPa) - Atmospheric pressure.

Parameters

- evap_type (flag) - Evaporation method for HRU, 0 = Granger, 1 = Priestley-Taylor, 2 = Penman-Monteith.
- Ht (m) - crop height.
- hru_elev (m) - altitude.
- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- F_Qg () - fraction to ground flux. Qg = F_Qg*Rn.
- inhibit_evap (flag) - 0/1 enable/inhibit.
- Zwind () - wind measurement height (used by Penman-Monteith).
- rcs (s/m) - stomatal resistance (used by Penman-Monteith).
- soil_type () - HRU soil type (used by Penman-Monteith) [1->11]: sand/loamsand/sandloam/loam/siltloam/sascllloam/clayloam/sicllloam/sandclay/siltclay/clay.
- soil_Depth (m) - depth of soil column (used by Penman-Monteith).
- Htmax (m) - maximum vegetation height (used by Penman-Monteith).
- LAImax (m^2/m^2) - maximum leaf area index (used by Penman-Monteith).
- LAImin (m^2/m^2) - minimum leaf area index (used by Penman-Monteith).
- s () - seasonal growth index (used by Penman-Monteith).
- PM_method () - Penman-Monteith method (used by Penman-Monteith), 0 = RC min, 1 = LAI, 2 = bulk.

Variable Inputs

The source of the variable is given in brackets after the name. Afull description of the variable may be found there.

- hru_t (obs) (°C)
- hru_tmean (obs) (°C)
- hru_u (*) (m/s)
- hru_umean (*) (m/s)
- hru_ea (obs) (kPa)
- hru_eamean (obs) (kPa)
- Rn (*) (mm/m^2*int)
- RnD (*) (mm/m^2*d)
- RnD_Pos (*) (mm/m^2*d)

Evaporation calculations Daily/Interval.

Interval observations or daily mean observations can be used in Priestley-Taylor and Penman-Monteith methods. The selection is made by the parameters PT_Daily, PM_Daily and DB_Daily. When

Daily calculation of Granger.

- Ea = fdaily(u, Ht)*(e*-ea)
- D = Ea/(Ea+(Rn-Qg-Qs))
- G = 1/(0.793 + 0.2*exp(4.902*D) + 0.006*D
- E = (delta(t)*G*(Rn-Qg-Qs) + gamma(Pa, t)*G*Ea)/(delta(t)*G + gamma(Pa, t))

  where t, u and ea are the mean of interval values and e* is calculated using the mean daily temperature t.

The Granger daily evaporation is partitioned over the daily intervals when Rn is positive.

- hru_evap = hru_evapD*Rn[t_n] / RnD_POS,

  where RnD_POS is the sum of daily intervals when Rn[Δt] is positive.

Priestley Taylor

- 1.26*delta(t)*(Rn-Qg-Qs)/((delta(t) + gamma(Pa, t)) for positive incoming net radiation,

  where values are interval. Evaporation is zero when incoming net radiation is <= 0.0.

Atmospheric Pressure

- P = P0*exp((293.0-0.0065*elevation)/293.0, 5.26),

  where P0 = 101.3.

evapD (Granger, 1989; Granger and Pomeroy, 1997)

This module defined in ClassevapD, calculates the evapotranspiration using daily values from after snowmelt to fall snow cover.

Other evaporation algorithm available is Priestley-Taylor.

Observations

- **none**

Variables

- **hru_evapD (mm/d) - average depth of evaporation from an HRU.**
- **hru_cum_evap (mm) - cumulative evaporation – calculated as the sum of the daily estimates of hru_evap.**
- **evap_G () - relative evaporation from Granger.**
- **evap_D () - relative drying power from Granger.**
- **hru_actet (mm/int) - actual evapotranspiration over the HRU. Evaporation and evapotranspiration are limited by the amount of soil moisture available.**
- **hru_cum_actet (mm) - cumulative actual evapotranspiration over the HRU.**

Parameters

- **evap_type (flag) - Evaporation method for HRU, 0 - Granger, 1 - Priestley Taylor.**
- **Ht (m) - crop height.**
- **hru_elev (m) - altitude.**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **F_Qg () - fraction to ground flux. Qg = F_Qg*Rn.**
- **inhibit_evap (flag) - 0/1 enable/inhibit.**

Variable Inputs

- **hru_tmean (obs) (°C)**
- **hru_umean (*) (°C)**
- **hru_eamean (obs) (°C)**
- **RnD (*) (mm/m^2*d)**

Granger Daily calculation

- **Ea = fdaily(u, Ht)*(e*-ea)**
- **D = Ea/(Ea+(Rn-Qg-Qs))**
- **G = 1/(0.793 + 0.2*exp(4.902*D) + 0.006*D**
- **E = (delta(t)*G*(Rn-Qg-Qs) + gamma(Pa, t)*G*Ea)/(delta(t)*G + gamma(Pa, t))**

  where t, u and ea are the mean of interval values and e* is calculated using the mean daily temperature t.

Priestley Taylor

- **1.26*delta(t)*(Rn-Qg-Qs)/((delta(t) + gamma(Pa, t))f or positive incoming net radiation,**

  where values are interval. Evaporation is zero when incoming net radiation is <= 0.0.

Atmospheric Pressure

- **P = P0*exp((293.0-0.0065*elevation)/293.0, 5.26),**

  where P0 = 101.3.

lake evaporation (Meyer Formula with revisions by PFRA)

This module defined in Classlake, calculates the monthly evaporation and distributes it uniformly over every interval of the month.

Observations

- none

Variables

- hru_evap (mm/int) - average depth of evaporation from an HRU in a time step, Δt.
- hru_cum_evap (mm) - cumulative evaporation – calculated as the sum of interval estimates of hru_evap.
- lake_evap_month (mm) - monthly evaporation from rh.
- hru_actet (mm/int) - actual evapotranspiration over the HRU. Evaporation and evapotranspiration are limited by the amount of soil moisture available.
- hru_cum_actet (mm) - cumulative actual evapotranspiration over the HRU.
- Va (mm) - water vapour pressure, Vw*rh (Meyer).
- Vw (mm) - air vapour pressure (Meyer).
- hru_t_mean (°C) - avereage monthly temperature.
- hru_rh_mean ( ) - avereage monthly relative humidity.
- hru_u_mean (m/s) - avereage monthlywind speed.
- hru_t_acc (°C*N) - accumulated monthly temperature.
- hru_rh_acc (N) - accumulated monthly relative humidity.
- hru_u_acc (m/s*N) - accumulated monthly wind speed.

Parameters

- Meyer_C () - Meyer coefficient.
- Meyer_B () - Meyer monthly intercept coefficient.
- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- start_open_water () - Julian date of start of lake evaporation.
- end_open_water () - Julian date of end of lake evaporation.

Variable Inputs

The source of the variable is given in brackets after the name. Afull description of the variable may be found there.

- hru_t (obs) (°C)
- hru_rh (obs) ()
- hru_u (obs) (m/s)

Equations.

```
hru_t_Mmean = hru_t_acc/N_intervals; // where N_intervals is the number of time steps in the month.
hru_rh_Mmean = hru_rh_acc/N_intervals;
hru_u_Mmean = hru_u_acc/N_intervals;

float Tw = 0.6*hru_t_Mmean + Meyer_B[ThisMonth-1];

Vw = pow(10,(-7.903*(373.16/(Tw+273.16)-1)+(5.028*log10(373.16/(Tw+273.16)))
-(0.0000001382*(pow(10, (11.34*(1-(373.16/(Tw+273.16)))))-1))
+(0.008133*pow(10, (-3.491*((373.16/(Tw+273.16))-1)))))+3.006);

Va = Vw*hru_rh_Mmean/100.0;

lake_evap_month = 0.75002*Meyer_C*(Vw-Va)*(1+0.06214*hru_u_Mmean);

hru_evap = lake_evap_month/N_intervals;
hru_cum_evap += hru_evap;
```

Note.

The months containing the Julian date of start of evaporation and end of operation must be complete months for the correct calculation of the monthly evaporation.

Ht_obs.

This module is defined in ClassHtobs. It reads a series of vegetation height observations into the parameter Ht. Parameters are normally constant during a model run. This module makes the parameter Ht follow the current value of the height in the observation file.

Observations

- **Ht_obs (m) - average vegetation height over the time interval. Optional, when not available the parameter value Ht retains its initial.**

Variables

- **Ht_var (m) - Observation values as a variable for monitoring.**

Parameters

- **Ht (m) - has initial default value. The initial value is used if no observations are available or until the first available observation is read.**

Variable Inputs

- **none.**

Notes.

The variable Ht_var is provided for debugging. Since parameters cannot be displayed by CRHM (assumed to be constant), Ht may be monitored by displaying the variable Ht_var.

ebsm (Gray and Landine, 1987)

This module is defined in Classebsm and is an energy-budget snowmelt model applicable to the Canadian Prairies.

Observations

- **QnD (W/m^2 day) - net radiation. Optional.**

Variables

- **snowmeltD (mm/d) - snowmelt.**
- **cumsnowmelt (mm) - cummulative snowmelt.**
- **LW_ebsm (mm/d) - liquid water in snowpack. State variable.**
- **u_ebsm (MJ/m^2*d) - snowpack energy deficit. State variable.**
- **Qmelt (MJ/m^2*d) - Qmelt = Qn+Qh+Qe+Qp.**
- **Qn_ebsm (MJ/m^2*d) - net radiation.**
- **Qh_ebsm (MJ/m^2*d) - sensible heat.**
- **Qe_ebsm (MJ/m^2*d) - latent heat.**
- **Qp_ebsm (MJ/m^2*d) - input from rainfall.**

Parameters

- **tfactor (mm/d*/ºC) - degree day melt factor.**
- **nfactor (mm*m^2/MJ*d) - net radiation factor.**
- **delay_melt () - inhibits any melt before this date. Only used when tfactor and nfactor are both zero. Default value not suitable for Southern Hemisphere**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **Albedo () - areal albedo.**

Variable Inputs

- **hru_tmean (obs) (ºC)**
- **hru_tmax (obs) (ºC)**
- **hru_tmin (obs) (ºC)**
- **hru_umean (obs) (m/s)**
- **hru_rhmean (obs) (%)**
- **hru_Sun_Act (obs) (hr)**
- **net_rain (*) (mm Δt)**
- **QdroD (global) (MJ/m^2)**
- **QdfoD (global) (MJ/m^2)**
- **SunMax (global) (h)**
- **SWE (*) (mm)**
- **albedo (albedo) ()**
- **meltflag (albedo) ()**

Useage.

tfactor and nfactor NOT equal to zero

For comparison there is an option to use the simplified energy budget method proposed by Kustas et al.(1994). In this degree-day algorithm the increased melting later in the season is simulated using a net radiation term.

Melt = tfactor*t_{max} + nfactor*Q_nD, where the observation Q_nD is the net all-wave radiation calculated from a module.

tfactor and nfactor equal to zero

If both parameters tfactor and nfactor are set equal to zero for an HRU, the module uses the original ebsm.f algorithm otherwise the simplified degree-day algorithm is used using the values set in tfactor and nfactor.

Observation QnD is used if available, otherwise Qn is calculated as in the original algorithm.

Over snow during melt;

Qn = -0.53+0.47*(QdroD+QdfoD)*(0.52+0.52*sunact/sunmax)*(1.0-Albedo)

An external observation QnD may be generated using modules long, NeedleLeaf and macro modules Slope_Qsi.mcr and Convert_Qsn_QnD.mcr.

MSM.

This module defined in ClassMSM is the minimal snow model.

Observations

- t (°C) - air temperature.
- rh (%) - relative humidity.
- u (m/s) - average wind speed over time step.
- Qsi (W/m^2) - incident short-wave.
- Qli (W/m^2) - incident long-wave.

Variables

- alb () - snow albedo.
- snowmelt (kg/m^2*int) - snowmelt.
- cumsnowmelt (kg/m^2) - cummulative snowmelt.
- snowmeltD (kg/m2*d) - daily snowmelt.
- sursubl (kg/m^2*int) - surface sublimation.
- T0 (°C) - surface temperature.
- sursubl (kg/m^2) - cummulative surface sublimation.
- LE (W/m^2) - latent heat flux .
- H (W/m^2) - sensible heat flux.
- Hsm (W/m^2) - snowmelt heat flux.
- LWn (W/m^2) - net long-wave radiation.
- SWn (W/m^2) - net short-wave radiation.
- Pa (kPa) - atmospheric pressure calculated from the elevation.

Parameters

- a1 (s) - Albedo decay time constant for cold snow.
- a2 (s) - Albedo decay time constant for melting snow.
- amin () - Minimum albedo for aged snow.
- amax () - Maximum albedo for fresh snow.
- smin (mm) - Minimum snowfall to refresh snow albedo.
- Z0snow (m) - snow roughnessength.
- Zref (m) - reference height.
- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- hru_elev (m) - altitude.

Variable Inputs

- SWE (*) (mm).
- net_snow (*) (mm).

snobal

This module is defined in Classsnobal.

snobal is an interactive point model using the energy balance to calculate snowmelt, and to predict runoff, from input data on snow properties, measurement heights & depths, and energy exchanges. Similar to the approach used by *Anderson (1976)*, **and** *Morris (1982)*, **but designed to run on simpler, more generalizable inputs. The model was first presented by** *Marks (1988)*, **described conceptually by** *Marks, et. al (1992)* **and** *Marks and Dozier (1992)*, **and then described in great detail by** *Marks, et. al (1997)*.

The model approximates the snow cover as being composed of two layers, a surface fixed-thickness active layer and a lower layer, solving for the temperature (C) and specific mass (kg/m^2) or mass per unit area (from density * depth (kg/m^3 * m)) for each, and computing total snowcover temperature and specific mass from these.

Melt is computed in either layer when the accumulated energy exceeds the "cold content" or when the "cold content" is > 0.0. Cold content is the energy required to bring the snow cover temperature up to freezing (0 C). Runoff is estimated when the accumulated melt and liquid H2O content exceed a specified threshold.

Trouble with the routine "hle1" iterative solution not converging. After ITMAX iterations an attempt is made to solve using a simplified stability function. This is tested by constraining the sensible heat to < 600 (W/m^2) and the latent heat to > -300. If either is outside the range, both are set to zero. The simple stability function can be selected instead of the iterative solution with the parameter "M_O_iter".

The simple stability function is $Ustar^2 t_a/(kgz_a(t_a-t_s)$, where k = 0.41, g = 9.81 m/s^2, z_a is the instrument height and t_a and t_s are the air and surface temperatures respectively.

If the snowfall is less than 1 (kg/m^2/s), it is immediately melted. This amount is accumulated and printed in the log screen as "melt_direct_cum".

To minimise the amount of snowfall immediately being melted because it was less then the 1 mm threshold, the parameter option "ppt_daily_distrib" was added to the module "obs" to dump all of the daily snowfall into the first interval of the day instead of distributing it evenly over the day.

Observations

- **none**

Variables

- **layer_count () - number of layers in snowcover.**
- **isothermal () - melting: 0/1**
- **snowcover () - snow on ground at start of current timestep: 0/1**
- **R_n (W/m^2) - net allwave radiation.**
- **H (W/m^2) - sensible heat transfer.**
- **L_v_E (W/m^2) - latent heat transfer.**
- **G (W/m^2) - heat transfer by conduction & diffusion from soil to snowcover.**
- **M (W/m^2) - advected heat from precip.**
- **delta_Q (W/m^2) - change in snowcover's energy.**
- **G_0 (W/m^2) - transfer by conduction & diffusion from soil or lower layer to active layer.**
- **delta_Q_0 (W/m^2) - change in active layer's energy.**
- **cc_s (J/m^2) - snowcover's cold content.**
- **cc_s_0 (J/m^2) - active layer cold content.**
- **cc_s_l (J/m^2) - lower layer cold content.**
- **E_s_int (kg/m^2*int) - mass of evap into air & soil from snowcover.**
- **E_int (kg/m^2*int) - mass flux by evap into air from active layer.**
- **melt_int (kg/m^2*int) - specific melt (kg/m^2 or mm.**
- **snowmelt_int (kg/m^2*int) - snow melt at bottom of pack.**
- **snowmeltD 9mm/d) - daily snow melt at bottom of pack.**
- **cumsnowmelt (mm) - cumulative snow melt at bottom of pack.**
- **z_s (m) - total snowcover thickness.**
- **z_s_0 (m) - active layer depth.**
- **z_s_l (m) - lower layer depth.**
- **rho (kg/m^3) - average snowcover density.**
- **SWE (kg/m^2) - snowcover's specific mass.**
- **m_s_0 (kg/m^2) - active layer specific mass.**
- **m_s_l (kg/m^2) - lower layer specific mass.**
- **T_s (°C) - average snowcover temp.**
- **T_s_0 (°C) - active layer temp.**
- **T_s _l (°C) - lower layer temp.**
- **h2o_sat () - %of liquid H2O saturation (relative water).**
- **h2o_vol () - liquid h2o content as volume ratio: V_water/(V_snow - V_ice).**
- **h2o_max (kg/m^2) - max liquid h2o content as specific mass.**
- **z_snow (m) - depth of snow in precip.**
- **h2o_sat_snow () - snowfall's %of liquid H2O saturation.**

- **precip_now () - precipitation in current timestep - 0/1.**
- **T_rain (°C) - temperature of rain.**
- **T_rain (°C) - temperature of snowfall.**

Parameters

- **relative_hts () - measurements heights, z_T and z_u, are relative to snow - 0/1.**
- **z_g (m) - depth of soil temperature measurement.**
- **z_u (m) - height of wind measurement.**
- **z_T (m) - height of air temperature & vapour pressure measurements.**
- **z_0 (m) - roughness length.**
- **max_z_s_0 (m) -maximum active layer thickness.**
- **max_h2o_vol () - max liquid h2o content as volume ratio: V_water/(V_snow - V_ice).**
- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **hru_elev (m) - altitude.**
- **M_O_iter () - Monin-Obukhov stability fuction. 0/1 - simplified/ original iterated solution.**

Variable Inputs

- **hru_S_n (W/m^2) - net solar radiation.**
- **hru_l_lw (W/m^2) - incoming longwave (thermal) radiation.**
- **hru_t (°C) - air temperature at height z_T.**
- **hru_ea (kPa) - vapour pressure at height z_T.**
- **hru_u (m/s) - wind speed at height z_u.**
- **hru_T_g(°C) - soil temp at depth z_g.**
- **hru_T_pp(°C) - temperature of precipitation.**
- **net_p (mm/int) - net precipitation below canopy.**
- **net_rain (mm/int) - net rain below canopy.**
- **net_snow (mm/int) - net snow below canopy.**
- **hru_drift (mm/int) - specific mass of drifting snow.**
- **hru_subl(mm/int) - specific mass of drifting snow.**
- **hru_rho_snow (kg/m^3) - density of snowfall.**

Notes

NAME

g_soil - conduction heat flow between snow and soil

SYNOPSIS

```
double g_soil(
    double  rho,    /* snow layer's density (kg/m^3)          */
    double  tsno,   /* snow layer's temperature (K)           */
    double  tg,     /* soil temperature (K)                   */
    double  ds,     /* snow layer's thickness (m)             */
    double  dg,     /* dpeth of soil temperature measurement (m) */
    double  pa)     /* air pressure (Pa)                      */
```

DESCRIPTION

g_soil calculates the heat flow between a snow layer and the soil accounting for both conduction and vapor transport. See pages 45 - 47 in the reference below.

RETURN VALUE

heat transfer between soil and snow (J/m^2)

HISTORY

Aug 1984
written by D. Marks, CSL (GSFC), UCSB;
May 1995
Converted from QDIPS to IPW by J. Domingo, OSU

SEE ALSO

Anderson 1976

NAME

g_snow - conduction heat flow between two snow layers

SYNOPSIS

```
double g_snow(
    double  rho1,  /* upper snow layer's density (kg/m^3)  */
    double  rho2,  /* lower "    "       "    (kg/m^3)  */
    double  ts1,   /* upper snow layer's temperature (K)   */
    double  ts2,   /* lower "    "       "      (K)   */
    double  ds1,   /* upper snow layer's thickness (m)     */
    double  ds2,   /* lower "    "       "     (m)     */
    double  pa)    /* air pressure (Pa)                    */
```

DESCRIPTION

g_snow calculates the heat flow between two snow layers for both conduction and vapor transport. See pages 46 and 47 of the reference below.

RETURN VALUE

heat transfer between snow layers (J/m^2)

HISTORY

Aug 1986
written by D. Marks, CSL, UCSB;
May 1995
Converted to IPW by J. Domingo, OSU

SEE ALSO

Anderson 1976

NAME

hle1 - sensible and latent heat from data at 1 height

SYNOPSIS

```
int hle1(
    double  press,  /* air pressure (Pa)           */
    double  ta,     /* air temperature (K) at height za    */
    double  ts,     /* surface temperature (K)          */
    double  za,     /* height of air temp measurement (m)  */
    double  ea,     /* vapor pressure (Pa) at height zq    */
    double  es,     /* vapor pressure (Pa) at surface      */
    double  zq,     /* height of spec hum measurement (m)  */
    double  u,      /* wind speed (m/s) at height zu       */
```

```
    double   zu,     /* height of wind speed measurement (m) */
    double   z0,     /* roughness length (m)     */
  /* output variables */
    double  *h,      /* sens heat flux (+ to surf) (W/m^2)   */
    double  *le,     /* latent heat flux (+ to surf) (W/m^2) */
    double  *e)      /* mass flux (+ to surf) (kg/m^2/s)     */
```

DESCRIPTION

hle1 computes sensible and latent heat flux and mass flux given measurements of temperature and specific humidity at surface and one height, wind speed at one height, and roughness length. The temperature, humidity, and wind speed measurements need not all be at the same height.

RETURN VALUE

0	successful calculation
-1	no convergence
-2	bad input

HISTORY

Jun 1987
Written as a Qdips program by J. Dozier, CRSEO, UCSB;
Oct 1990
Translated into IPW by K. Longley, OSU, ERLC;

SEE ALSO

Brutsaert 1982

CRHM implementation.

Trouble experienced with iterative solution not converging. Added simplified non-iterative solution. Better, but still blowing up. Added check to return h = 0.0 and le = 0.0, if H > 600 or Le < -300. This allows the model to complete the run. Values of h, le,t, ts, u, ea and es are output to allow for further investigation.

snobalCRHM

This module is defined in ClasssnobalCRHM.

snobal is an interactive point model using the energy balance to calculate snowmelt, and to predict runoff, from input data on snow properties, measurement heights & depths, and energy exchanges. Similar to the approach used by *Anderson (1976)*, **and** *Morris (1982)*, **but designed to run on simpler, more generalizable inputs. The model was first presented by** *Marks (1988)*, **described conceptually by** *Marks, et. al (1992)* **and** *Marks and Dozier (1992)*, **and then described in great detail by** *Marks, et. al (1997)*.

The model approximates the snow cover as being composed of two layers, a surface fixed-thickness active layer and a lower layer, solving for the temperature (C) and specific mass (kg/m^2) or mass per unit area (from density * depth (kg/m^3 * m)) for each, and computing total snowcover temperature and specific mass from these.

Melt is computed in either layer when the accumulated energy exceeds the "cold content" or when the "cold content" is > 0.0. Cold content is the energy required to bring the snow cover temperature up to freezing (0 C). Runoff is estimated when the accumulated melt and liquid H2O content exceed a specified threshold.

Trouble with the routine "hle1" iterative solution not converging. After ITMAX iterations an attempt is made to solve using a simplified stability function. This is tested by constraining the sensible heat to < 600 (W/m^2) and the latent heat to > -300. If either is outside the range, both are set to zero. The simple stability function can be selected instead of the iterative solution with the parameter "M_O_iter".

The simple stability function is $Ustar^2 t_a/(kgz_a(t_a-t_s)$, where k = 0.41, g = 9.81 m/s^2, z_a is the instrument height and t_a and t_s are the air and surface temperatures respectively.

If the snowfall is less than 1 (kg/m^2/s), it is immediately melted. This amount is accumulated and printed in the log screen as "melt_direct_cum".

To minimise the amount of snowfall immediately being melted because it was less then the 1 mm threshold, the parameter option "ppt_daily_distrib" was added to the module "obs" to dump all of the daily snowfall into the first interval of the day instead of distributing it evenly over the day.

Observations

- **Qsi (W/m^2) - incoming solar radiation**
- **Qli (W/m^2) - incoming longwave (thermal) radiation.**

Variables

- **layer_count () - number of layers in snowcover.**
- **isothermal () - melting: 0/1**
- **snowcover () - snow on ground at start of current timestep: 0/1**
- **R_n (W/m^2) - net allwave radiation.**
- **H (W/m^2) - sensible heat transfer.**
- **L_v_E (W/m^2) - latent heat transfer.**
- **G (W/m^2) - heat transfer by conduction & diffusion from soil to snowcover.**
- **M (W/m^2) - advected heat from precip.**
- **delta_Q (W/m^2) - change in snowcover's energy.**
- **G_0 (W/m^2) - transfer by conduction & diffusion from soil or lower layer to active layer.**
- **delta_Q_0 (W/m^2) - change in active layer's energy.**
- **cc_s (J/m^2) - snowcover's cold content.**
- **cc_s_0 (J/m^2) - active layer cold content.**
- **cc_s_l (J/m^2) - lower layer cold content.**
- **E_s_int (kg/m^2*int) - mass of evap into air & soil from snowcover.**
- **E_int (kg/m^2*int) - mass flux by evap into air from active layer.**
- **melt_int (kg/m^2*int) - specific melt (kg/m^2 or mm.**
- **snowmelt_int (kg/m^2*int) - snow melt at bottom of pack.**
- **snowmeltD 9mm/d) - daily snow melt at bottom of pack.**
- **cumsnowmelt (mm) - cumulative snow melt at bottom of pack.**
- **z_s (m) - total snowcover thickness.**
- **z_s_0 (m) - active layer depth.**
- **z_s_l (m) - lower layer depth.**
- **rho (kg/m^3) - average snowcover density.**
- **SWE (kg/m^2) - snowcover's specific mass.**
- **m_s_0 (kg/m^2) - active layer specific mass.**
- **m_s_l (kg/m^2) - lower layer specific mass.**
- **T_s (°C) - average snowcover temp.**
- **T_s_0 (°C) - active layer temp.**
- **T_s _l (°C) - lower layer temp.**
- **h2o_sat () - %of liquid H2O saturation (relative water).**
- **h2o_vol () - liquid h2o content as volume ratio: V_water/(V_snow - V_ice).**
- **h2o_max (kg/m^2) - max liquid h2o content as specific mass.**
- **z_snow (m) - depth of snow in precip.**

- h2o_sat_snow () - snowfall's %of liquid H2O saturation.
- precip_now () - precipitation in current timestep - 0/1.
- T_rain (°C) - temperature of rain.
- T_rain (°C) - temperature of snowfall.

Parameters

- relative_hts () - measurements heights, z_T and z_u, are relative to snow - 0/1.
- z_g (m) - depth of soil temperature measurement.
- hru_T_g (°C) - assume constant temperature at depth z_g during melt.
- z_u (m) - height of wind measurement.
- z_T (m) - height of air temperature & vapour pressure measurements.
- z_0 (m) - roughness length.
- max_z_s_0 (m) -maximum active layer thickness.
- max_h2o_vol () - max liquid h2o content as volume ratio: V_water/(V_snow - V_ice).
- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- hru_elev (m) - altitude.
- M_O_iter () - Monin-Obukhov stability fuction. 0/1 - simplified/ original iterated solution.
- hru_rho_snow (kg/m^3) - density of snowfall.
- rain_soil_snow () - 0 - handle only snow (infiltration routine handles rain), 1 - handle snow and rain (rain is added to the snowpack)..

Variable Inputs

- albedo () - snow surface albedo.
- hru_t (°C) - air temperature at height z_T.
- hru_ea (kPa) - vapour pressure at height z_T.
- hru_u (m/s) - wind speed at height z_u.
- hru_T_pp(°C) - precipitation temperature - assumed to be the air temperature.
- net_p (mm/int) - net precipitation below canopy.
- net_rain (mm/int) - net rain below canopy.
- net_snow (mm/int) - net snow below canopy.
- hru_drift (mm/int) - specific mass of drifting snow.
- hru_subl(mm/int) - specific mass of drifting snow.

Notes

NAME

g_soil - conduction heat flow between snow and soil

SYNOPSIS

```
double g_soil(
    double  rho,    /* snow layer's density (kg/m^3)          */
    double  tsno,   /* snow layer's temperature (K)           */
    double  tg,     /* soil temperature (K)                   */
    double  ds,     /* snow layer's thickness (m)             */
    double  dg,     /* dpeth of soil temperature measurement (m) */
    double  pa)     /* air pressure (Pa)                      */
```

DESCRIPTION

g_soil calculates the heat flow between a snow layer and the soil accounting for both conduction and vapor transport. See pages 45 - 47 in the reference below.

RETURN VALUE

heat transfer between soil and snow (J/m^2)

HISTORY

Aug 1984
written by D. Marks, CSL (GSFC), UCSB;

May 1995
Converted from QDIPS to IPW by J. Domingo, OSU

SEE ALSO

Anderson 1976

NAME

g_snow - conduction heat flow between two snow layers

SYNOPSIS

```
double g_snow(
    double  rho1,  /* upper snow layer's density (kg/m^3)  */
    double  rho2,  /* lower "    "       "    (kg/m^3)  */
    double  ts1,   /* upper snow layer's temperature (K)   */
    double  ts2,   /* lower "    "       "      (K)   */
    double  ds1,   /* upper snow layer's thickness (m)     */
    double  ds2,   /* lower "    "       "     (m)    */
    double  pa)    /* air pressure (Pa)                    */
```

DESCRIPTION

g_snow calculates the heat flow between two snow layers for both conduction and vapor transport. See pages 46 and 47 of the reference below.

RETURN VALUE

heat transfer between snow layers (J/m^2)

HISTORY

Aug 1986
written by D. Marks, CSL, UCSB;
May 1995
Converted to IPW by J. Domingo, OSU

SEE ALSO

Anderson 1976

NAME

hle1 - sensible and latent heat from data at 1 height

SYNOPSIS

```
int hle1(
    double  press,  /* air pressure (Pa)           */
    double  ta,     /* air temperature (K) at height za    */
    double  ts,     /* surface temperature (K)          */
    double  za,     /* height of air temp measurement (m)  */
    double  ea,     /* vapor pressure (Pa) at height zq    */
    double  es,     /* vapor pressure (Pa) at surface      */
    double  zq,     /* height of spec hum measurement (m)  */
```

```
    double   u,      /* wind speed (m/s) at height zu        */
    double   zu,     /* height of wind speed measurement (m) */
    double   z0,     /* roughness length (m)     */
  /* output variables */
    double  *h,      /* sens heat flux (+ to surf) (W/m^2)   */
    double  *le,     /* latent heat flux (+ to surf) (W/m^2) */
    double  *e)      /* mass flux (+ to surf) (kg/m^2/s)     */
```

DESCRIPTION

hle1 computes sensible and latent heat flux and mass flux given measurements of temperature and specific humidity at surface and one height, wind speed at one height, and roughness length. The temperature, humidity, and wind speed measurements need not all be at the same height.

RETURN VALUE

0	successful calculation
-1	no convergence
-2	bad input

HISTORY

Jun 1987
Written as a Qdips program by J. Dozier, CRSEO, UCSB;
Oct 1990
Translated into IPW by K. Longley, OSU, ERLC;

SEE ALSO

Brutsaert 1982

CRHM implementation.

Trouble experienced with iterative solution not converging. Added simplified non-iterative solution. Better, but still blowing up. Added check to return h = 0.0 and le = 0.0, if H > 600 or Le < -300. This allows the model to complete the run. Values of h, le,t, ts, u, ea and es are output to allow for further investigation.

frozen

This module defined in Classfrozen calculates the 'Limited' infiltration into soils where the infiltrability is governed by the soil moisture content (water + ice) and soil temperature at the start of snow ablation and the infiltration opportunity time.

Observations

- **t0_inhibit () - optional. If its value is > 0.0 the incrementing of t0_Acc during the calibration cycle is inhibited. This input may be controlled using a macro.**

Variables

- **infil (mm/int) - depth of infiltration in time step, $\triangle t$ – expressed as an average depth of water on an HRU.**
- **cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.**
- **snowinfil (mm/int) - amount of daily infiltration – expressed as an equivalent depth (m^3/m^2).**
- **cumsnowinfil (mm) - cummulative infiltration.**
- **meltrunoff (mm/int) - amount of interval runoff – expressed as an equivalent depth (m^3/m^2).**
- **cummeltrunoff (mm) - cummulative runoff.**
- **runoff (mm/int) - amount of interval runoff – expressed as an equivalent depth (m^3/m^2).**
- **cumrunoff (mm) - cummulative runoff.**
- **t0_Var (h) - value of T0 used by model.**
- **t0_Acc (h) - infiltration opportunity time accumulator.**
- **Inf (mm) - infiltration into a frozen soil calculated from parametric relationship.**
- **infiltype() - infiltration type. PREMELT/RESTRICTED/LIMITED/UNLIMITED/SATURATED - 0/1/2/3/4 respectively.**
- **snowmeltD(mm/d) - yesterday's snowmelt.**

Parameters

- **t0 (h) - Infiltration opportunity time calculated and loaded after a model run with t0[HRU 1] <= 0.**
- **Reset_t0 () - Reset t0 to zero on this Julian date. If zero does nothing. Used for multiple year runs.**
- **S0 (m^3/m^3) - surface saturation.**
- **Si (m^3/m^3) - initial soil saturation.**
- **C () - coefficient.**
- **hru_tsoil (ºK) - soil average temperature at start of frozen infiltration.**
- **t_ice_lens(ºC) - overnight minimum to cause ice lens after major melt.**
- **soil_moisture_max (mm) - soil moisture maximum. Common to module *smbal*.**
- **basin_area (km^2) - area of watershed.**
- **hru_area(km^2) - area of HRUs.**

Variable Inputs

- **snowmeltD (*) (mm/d).**
- **hru_tmin (obs) (°C).**
- **SWE (*) (mm).**
- **net_rain (*) (mm/int).**
- **soil_moist (*) (mm).**

Calculation

- **c*pow(S0, 2.92)*pow(1.0-Si, 1.64)*pow((273.15-tmean)/273.15, -0.45)*pow(t0, 0.44) *10**

Reference

- **Zhao and Gray (1999)**

Notes.

1. **Calibration. Since the opportunity time is not known until the model is run through the complete frozen infiltration period, the model has to be run twice. The user, by setting t0[HRU 1] to <= 0.0 causes the module to calculate the infiltration opportunity time. During this run the module forces all HRUs to be restricted. At the end of the model run the newly calculated opportunity times are transferred to the parameter. If the run is terminated early the partially calculated values will be used. During succeeding model runs the earlier calculated infiltration opportunity times will be used until t0[HRU 1] is set to <= 0.0 again.**
2. **Maximum infiltration is limited to the lesser of Inf calculated from the equation above and the available storage in the module smbal, i.e. (soil_moist - soil_moist_max).**
3. **Accumulation of the opportunity time when calculating. Normally 24 hours is added for every day when melt occurs. However, this may be refined by using the observation input t0_inhibit. When this input is not connected or its value is zero, the interval is added to the infiltration opportunity time. When t0_inhibit is greater than 0.0 the interval is not added.**
4. **When t0 is zero or is reset to zero using the Reset_t0 parameter, CRHM calculates ahead to the end of melt to determine the 'Infiltration opportunity time'. It then returns to the beginning of melt and handles infiltration**

over the melt period.

frostdepth

This module is defined in Classfrostdepth. The number of soil layers is defined by NLAY. The snow cover is handled as a separate layer whose depth is calculated from the SWE of the HRU using the function depthofsnow.

Observations

- none

Variables

- frostdepth (m) - frostdepth. State variable.
- snowdepth (mm) - calculated from SWE. State variable.
- Findex (°C) - cumulative degrees of frost in (day.°C). State variable.
- Tfreeze (d) - days of frost. State variable.
- Lacc (MJ/m3) - cumulative effective latent heat. State variable.
- Cacc (MJ/(m3.K)) - cumulative effective heat capacity. State variable.
- Kacc (W/(m K)) - cumulative effective thermal conductivity. State variable.

Parameters

- Ta (°C) - annual air temperature.
- soil_type () - 1/sand 2/loam 3/clay 4/organic (only used to differentiate between mineral and organic layers).
- por (m3/m3) - porosity.
- theta (m3/m3) - degree of saturation.
- d (m) - layer depths 1 ... NLAY.
- hru_lat (°) - latitude. Adjusts empirical Julian constants for Northern/Southern Hemisphere.

Variable Inputs

- hru_tmean (obs) (°C)
- SWE (pbsm) (mm)

Other Constants

- Lsnow = 0.35*Ci MJ/(m3) - latent heat of snow.
- csnow = 0.35*ki + 0.65*ka MJ/(m3.°C) - heat capacity of snow.
- ko = 0.21 W/(m K) organic material
- km = 0.27 W/(m K) mineral
- ka = 0.025 W/(m K) air
- ki = 2.24 W/(m K) ice
- kw = 0.57 W/(m K) water
- Cm = 2.000 MJ/(m3.K) mineral
- Cw = 4.185 MJ/(m3.K) water
- Ca = 0.001 MJ/(m3.K) air
- Co = 0.110 MJ/(m3.K) organic
- Ci = 1.950 MJ/(m3.K) organic

Returns

FrostDepth = sqrt((24*Kacc*Findex)/(Lacc + Cacc(Ta + Findex/(2.0*Tfreeze)))) - SnowDepth

where for individual layers

Ln = theta*por*333.0

Kn = (1.0 - por) * km + theta*por*kw + (por - theta*por)*ka

Cn = (1.0 - por) * Cm + theta*por*Cw + (por - theta*por)*Ca

for organic material use ko and Co instead of km and Cm.

For snow assuming a snowpack density of (250kg/m3) use

Ln = 0

Ks = 0.25*ki + 0.75*ka

Cs = 0.25*Ci

For the combined layers use

Lacc = (Ls*Ds + L1*D1 + L2*D2 ... Ln*Dn)/Dsum

Cacc = (Cs*Ds + C1*D1 + C2*D2 ... Cn*Dn)/Dsum

Kacc = Dsum/(Ds/Ks + D1/K1 + D2/K2 ... Dn/Kn)

where Ds (m) is SWE/250.0 assuming a snowpack density of (250kg/m3) and D1, D2 .. Dn, 1 ...NLAY are the respective layer thicknesses. The last layer only to the depth frozen.

Reference

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

Melt Routine Features.

This description of melt routine features will help to clarify the differences in operation of the modules, *ebsm*, *MSM* **and** *Kevin*.

Temperature Index and Simplified Energy Budget Method.

If either tfactor or nfactor is non zero this mode is selected.

Temperature Index.

Units of tfactor are (mm/d*°C).

1. *ebsm* **- Daily calculation. Melt = tmax[0]*tfactor (mm/day). Uses daily maximum temperature.**
2. *Kevin* **- Interval calculation. Melt = hru_t[hh]*tfactor/Freq (mm/int). Is set to zero unless hru_t is > 0°C. Where Freq is the number of intervals in a day.**

Simplified Energy Budget Method proposed by Kustas et al. (1994).

Units of nfactor are mm/(MJ/m^2)/day. This converts the energy input to mm (kg/m2) of melt.

1. *ebsm* **- Daily calculation. Melt = QnD*nfactor (mm/day).**
2. *Kevin* **- Interval calculation. Melt = hru_Qn *WtoMJ*nfactor/Freq (mm $\triangle$t).**

nfactor converts energy to melt. It is the latent heat of fusion, 0.3336 MJ/kg. However, it may be adjusted by the thermal quality B = 0.95.

Simplified Energy Budget Sources.

1. **QnD (W/m^2) - may be calculated using modules** *slope_Qsi, long*, *albedo, NeedleLeaf* **and** *Convert_Qnsn_QnD.*
2. **hru_Qn (MJ/d) - may be calculated using modules** *slope_Qsi, long*, *albedo, NeedleLeaf* **and Convert_Qnsn_hru_Qn.**

Adjusting for Cloud Cover.

Hours of bright sunshine used to be measured at meteorology stations. Today that is rarely done. Earlier algorithms for calculating short wave and long wave radiation used the ratio of actual sunshine hours to the maximum possible in the day (n/N). The module ebsm uses this ratio. When the sunshine hours are not measured the module *calcsun* **can estimate (n/N) from the measured incoming short-wave Qsi and the clear sky Qsi calculated by the module** *global*.

Adjusting Calculated Incoming Short-wave on a Slope for Cloud Cover.

Since radiation measurements are not normally made on a slope, it is necessary to calculated the incoming short-wave using the module global and then reducing the value for cloud cover. This is done by adjusting the value using the ratio of Qsi measured on the level divided by the calculated Qsi for the level. The module MSM uses these inputs.

ebsm (tfactor and nfactor both zero).

Qm = Qsn + Qnl + Qh + Qe + Qg + Qp - du/dt

During melt Qn = Qsn + Qsl is a linear function of the daily net short-wave radiation.

Qn = -0.53+0.47*(QdroD + QdfoD)*(0.52+0.52*sunact/sunmax)*(1.0-Albedo).

MSM (tfactor and nfactor both zero).

to be completed.

Kevin (tfactor and nfactor both zero).

to be completed.

Infiltration module summary.

	Winter Infiltration (frozen ground).		Summer and Winter (unfrozen ground) Infiltration.		
	Crack	Frozen	Everything	Ayers	GreenAmpt
Classcrack ("crack")	*		*		
ClassPrairieInfil ("PrairieInfiltration")	*			*	
ClassAyers ("Ayers")				Summer and Winter (unfrozen)	
ClassFrozenAyers ("FrozenAyers")		*		*	
ClassGreenAmpt ("GreenAmpt")					Summer and Winter (unfrozen)
ClassGreencrack ("Greencrack")	*				*
Classfrozen ("frozen")		*	*		

Comments.

ClassAyers ("Ayers") for unfrozen soil.

Attempts to infiltrate snowmelt before handling rainfall.

ClassGreenAmpt ("GreenAmpt") for unfrozen soil.

This module immediately combines snowmelt and net_rain before using the GreenAmpt method. It is not recommended.

Classcrack ("crack") and Classfrozen ("frozen") for frozen soil.

These module were developed for the Canadian prairies. During the summer it is assumed that everything infiltrates.

ClassPrairieInfil ("PrairieInfiltration") and ClassAyers ("FrozenAyers") for frozen soil.

These modules can limit summer infiltration using soil texture/groundcover and maximum permissible infitration rates.

ClassGreencrack ("Greencrack") for frozen soil.

GreenAmpt controls summer infiltration and is not recommended.

GreenAmpt.

This module is defined in ClassGreenAmpt. It handles unfrozen soil infiltration.

Observations

- **none**

Variables

- **infil (mm/int) - depth of infiltration in time step, Δt – expressed as an average depth of water on an HRU.**
- **cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.**
- **runoff (mm/int) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).**
- **cumrunoff (mm) - cummulative runoff.**
- **meltrunoff (mm/int) - amount of interval melt runoff. .**
- **cummeltrunoff (mm) - cumulative interval melt runoff.**
- **snowinfil (mm/int) - amount of interval infiltration – expressed as an equivalent depth (m^3/m^2).**
- **cumsnowinfil (mm) - cumulative interval infiltration**

Parameters

- **basin_area (km^2) - area of watershed.**
- **hru_area(km^2) - area of HRUs.**
- **soiltype () - 0 through 12 for water/sand/loamsand/sandloam/loam/siltloam/sasclloam/clayloam/siclloam/sandclay/siltclay/clay/pavement.**
- **soil_moist_max (mm) - maximum available water holding capacity of soil profile.**
- **soil_moist_init (mm) - initial value of available water in soil profile.**

Variable Inputs

- **net_rain (mm/int) (*) - depth of rain received by the ground surface in time step, Δt. Variable available from module *intcp*, etc..**
- **snowmeltD (mm/d) (*)**
- **soil_moist (mm) - from module *Soil, smba*l, etc.**

Notes on Green-Ampt.

The module is used when unlimited infiltration is not possible.

When the surface ponding is negligible,

$$f \text{ (mm/hr)} = k^*(\psi^*\Delta\theta/F + 1), \quad \text{(eq. 1.)}$$

where:

k (mm/hr) = saturated hydraulic conductivity.

ψ (mm) = capillary suction at the wetting front.

$\Delta\theta = (\theta s - \theta i)$. change in volumetric soil moisture content.

When ponded,

$$F1 \text{ (mm)} = k^*\Delta t + \psi^*\Delta\theta * \ln((F1 + \psi^*\Delta\theta)/(F0 + \psi^*\Delta\theta)). \quad \text{(eq. 2.)}$$

where F0 (mm) = cumulative infiltation at the beginning of the time interval i.e. t,

f0 (mm/hr) = infiltration rate at the beginning of the time interval i.e. t,

F1 (mm) = cumulative infiltration at the end of the time interval i.e. t+Δt,

f1(mm/hr) = infiltration rate at the end of the time interval i.e. t+Δt,

It (mm/hr) = rainfall intensity during the interval.

The cumulative infiltration (Fp) at instant of ponding time is

$$Fp = k^*\psi^*\Delta\theta/(It - k). \quad \text{(eq. 3.)}$$

The portion of the interval, after which ponding occurs is $\Delta t'$,

$$\Delta t' = (Fp - F0)/It. \quad \text{(eq. 4.)}$$

Flow Chart

Calculate the maximum infiltration rates, f0 and f1 at the beginning and end of the time step using equation 1. using F1 = F0+∆t*It.

Case 1: When f1 is >= rainfall intensity, all the rainfall infiltrates with F1 = F0+∆t*It.

Case 2: When the rainfall intensity > f0, all the rainfall ponds and F is calculated using an iterative solution of equation 2.

Case 3: When the rainfall intensity <= f0 but the rainfall intensity > f1, ponding occurs during the interval and the solution consists of solving for Fp using equation 3, then using equation 4 to solve the time into the step during which ponding occurs. Next, using equation 2 with Fp as F0 the cumulative infiltration F1 after the ponding time of (∆t - ∆t') is calculated.

Assigned Soil Properties

All, except fcap are specified at 0%saturation from Chow pg 115.

Value for wilt is from Chow.

Values for por, fcap is from Floods in Canada

Soil curves of form K = Ks / (A*Psi^3 + 1)

Soil Type	psi	k(mm/h)	wilt	fcap	porosity for Soil	porosity for Evap	air entry tension	pore size
water	{ 0.0	999.9	0.000	0.00	1.000	1.000	0.000	0.0
sand	{ 49.5	117.8	0.020	0.10	0.437	0.395	0.121	4.05
loamsand	{ 61.3	29.9	0.036	0.16	0.437	0.41	0.09	4.38
sandloam	{110.1	10.9	0.041	0.23	0.453	0.435	0.218	4.9
loam	{ 88.9	3.4	0.029	0.26	0.463	0.451	0.478	5.39
siltloam	{166.8	6.5	0.045	0.38	0.501	0.485	0.786	5.3
sacIloam	{218.5	1.5	0.068	0.38	0.398	0.420	0.299	7.12
clayloam	{208.8	1.0	0.155	0.39	0.464	0.476	0.63	8.52
sicIloam	{273.3	1.0	0.039	0.40	0.471	0.477	0.356	7.75
sandclay	{239.0	0.6	0.110	0.41	0.430	0.426	0.153	10.4
siltclay	{292.2	0.5	0.056	0.43	0.479	0.492	0.49	10.4
clay	{316.3	0.3	0.090	0.46	0.475	0.482	0.405	11.4

	psi(mm)	k(mm/hr)	wilt	fcap	por	A-Soil moist	A-Hyd cond
water	0.0	999.9	0.000	0.00	1.100	1.000000000	1.0000000}
sand	49.5	117.8	0.020	0.10	0.437	0.000010000	0.0002000}
loamsand	61.3	29.9	0.036	0.16	0.437	0.000006000	0.0001000}
sandloam	110.1	10.9	0.041	0.23	0.453	0.000001124	0.0000200}
loam	88.9	3.4	0.029	0.26	0.463	0.000000900	0.0000150}
siltloam	166.8	6.5	0.045	0.38	0.501	0.000000300	0.0000050}
sacIloam	218.5	1.5	0.068	0.38	0.398	0.000000200	0.0000030}
clayloam	208.8	1.0	0.155	0.39	0.464	0.000000150	0.0000020}
sicIloam	273.3	1.0	0.039	0.40	0.471	0.000000080	0.0000015}
sandclay	239.0	0.6	0.110	0.41	0.430	0.000000070	0.0000010}
siltclay	292.2	0.5	0.056	0.43	0.479	0.000000010	0.0000002}
clay	316.3	0.3	0.090	0.46	0.475	0.000000006	0.0000001}

Ayers .

This module is defined in ClassAyers . It handles only unfrozen soil infiltration.

Observations

- **none**

Variables

- **infil (mm/int) - depth of infiltration in time step, Δt – expressed as an average depth of water on an HRU.**
- **cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.**
- **runoff (mm/int) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).**
- **cumrunoff (mm) - cummulative runoff.**
- **meltrunoff (mm/int) - amount of interval melt runoff. .**
- **cummeltrunoff (mm) - cumulative interval melt runoff.**
- **snowinfil (mm/int) - amount of interval infiltration – expressed as an equivalent depth (m^3/m^2).**
- **cumsnowinfil (mm) - cumulative interval infiltration**
- **melt(mm/int) - interval melt equal to snowmeltD/Freq.**
-

Parameters

- **basin_area (km^2) - area of watershed.**
- **hru_area(km^2) - area of HRUs.**
- **texture () - 1 through 4 for texture:- 1 - coarse/medium over coarse, 2 - medium over medium, 3 - medium/fine over fine, 4 - soil over shallow bedrock..**
- **groundcover () - 1 through 6 for groundcover: 1 - bare soil, 2 - row crop, 3 - poor pasture, 4 - small grains, 5 - good pasture, 6 - forested..**

Variable Inputs

- **net_rain (mm/int) (*) - amount of rain received by the ground surface in time step, Δt. Variable available from module *intcp*, etc..**
- **snowmeltD (mm/d) (*)**

Notes on Ayers.

textureproperties[texture] [groundcover] in mm/hour.

bare soil	row crop	poor pasture	small grains	good pasture	forested	
7.6	12.7	15.2	17.8	25.4	76.2	coarse over coarse
2.5	5.1	7.6	10.2	12.7	15.2	medium over medium
1.3	1.8	2.5	3.8	5.1	6.4	medium/fine over fine
0.5	0.5	0.5	0.5	0.5	0.5	soil over shallow bedrock

Greencrack.

This module is derived from two other CRHM modules, crack and Greenampt and is defined in ClassGreencrack. It handles frozen soil infiltration during the spring snow melt and during the remainder of the year uses Green-Ampt for unfrozen soil infiltration.

Observations

- **none**

Variables

- **infil (mm/int) - depth of infiltration in time step, Δt – expressed as an average depth of water on an HRU.**
- **cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.**
- **snowinfil (mm/int) - amount of daily infiltration – expressed as an equivalent depth (m^3/m^2).**
- **cumsnowinfil (mm) - cummulative infiltration.**
- **runoff (mm/int) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).**
- **cumrunoff (mm) - cummulative runoff.**
- **meltrunoff (mm/int) - melt runoff.**
- **cummeltrunoff (mm) - cummulative melt runoff.**
- **crackstat () - infiltration status. Values: 0 - not started, 1 to 6 - #events, >6 - *Limited* ended and 10 - ice lens. State variable.**
- **crackon () - indicates when the crack frozen soil routine is enabled for an HRU. 0 - disabled/1 - enabled.**
- **RainOnSnow (mm) - cumulative rain on snow.**

Parameters

- **basin_area (km^2) - area of watershed.**
- **hru_area(km^2) - area of HRUs.**
- **fallstat () - fall status <= 0.0 or -1 for unlimited/cracked, >= 100.0 for restricted and < 0.0 limited < 100.0. Pore saturation as a percentage.**
- **major (mm) - threshold for major melt. Default is 5 mm/day.**
- **soiltype - 0 through 11 for water/sand/loamsand/sandloam/loam/siltloam/sascllоam/clayloam/sicllоam/sandclay/siltclay/clay/pavement.**
- **soil_moist_init (mm) - initial soil moisture content --- used to set F1.**
- **soil_moist_max (mm) - maximum soil moisture content.**
- **PriorInfiltration () - allow limited melt to infiltrate prior to major melt.**

Variable Inputs

- **hru_tmax (°C) - from module *obs*.**
- **snowmeltD (mm/d) (*) - from melt module, e.g. *ebsm*, etc..**
- **SWE (mm) (*) - Snow water equivalent from module, e.g. *pbsm*, etc..**
- **net_rain (mm/int) (*) - depth of rain received by the ground surface in time step, Δt. Variable available from module *intcp*, *brushintcp* etc..**
- **soil_moist (mm) - from module *smba*l, etc.**

Notes on Crack.

The Division of Hydrology at the University of Saskatchewan (Granger et al. 1984; Gray et al., 1985), postulated that the infiltration potential of frozen soils may be grouped in three broad categories, namely: restricted, limited and unlimited.

Restricted **- Infiltration is impeded by an impermeable layer, such as an ice lens on the soil surface or within the soil close to the surface. For all practical purposes, the amount of meltwater infiltration can be assumed to be negligible and that the melt goes directly to runoff and to evaporation.**

Limited **- Infiltration is governed primarily by the snow-cover water equivalent and the frozen water content of the top 30 cm. of soil.**

Unlimited **- A soil with a high percentage of large, air-filled macropores at the time of melt. Examples of soils having these properties are dry, heavily cracked clays and coarse, dry sands, organics and others.**

Within the model there is not sufficient information to determine these classifications automatically and as a result the user is required to specify these properties for each HRU every fall and input this information to the model as a parameter. The one case the model does handle is when there is an early melt and the subsequent re-freezing causing an ice lens to form. This will change both Limited to Restricted. Implementation of the infiltration to frozen soils routines is described below.

Definitions

1. Index = INF/SWE where INF=5(1-θ p)*SWE^0.584.

2. Potential = INF/6.

3. MELT_THRESHOLD = 5 mm. Minimum daily meltwater at which the melt routine is enabled. Lower meltwater levels are not counted as one of the six major melt events.

4. A major melt is a day when the amount of meltwater generated is greater than the MELT_THRESHOLD.

5. Six major daily melts are allowed before the limited infiltration category is changed to Restricted.

The Frozen Soil Infiltration routine

1. The Frozen Infiltration routine is enabled in an HRU when its SWE is greater than 50mm. Each HRU is handled independently.

2. Limited infiltration is triggered into operation by the first major melt. At this time, Index and Potential are calculated from the soil moisture (θ p) and the SWE of the snowpack.

3. The Frozen Infiltration routine is disabled in an HRU when its SWE reaches zero.

The three frozen soil categories are described below.

LIMITED

1. Only six major over-winter snowmelt events are possible before the infiltration potential becomes Restricted.

2. Meltwater amounts less than the MELT_THRESHOLD is handled as runoff unless the parameter *PriorInfiltration*are is set, then all of the melt is permitted to infiltrate into the soil. Once the MELT_THRESHOLD has been exceeded it is assumed that spring snowmelt has begun and only an amount of meltwater equal to MELT*Index will infiltrate and the remainder will be handled as runoff.

3. Index and Potential are recalculated if another major melt occurs with a greater SWE.

4. If the temperature the day following a major melt event is colder than -10ºC, the category is changed from Limited to Restricted assuming an ice lens has formed.

5. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

UNLIMITED

1. All meltwater is allowed to infiltrate.

2. If the temperature the day after a major melt event is less than -10ºC, the category remains unchanged and all the melt water still infiltrates into the soil.

3. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

RESTRICTED

1. No meltwater is allowed to infiltrate.

2. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

Rain on Snow.

The energy component of Rain on Snow was not handled when the Rain on Snow occurs before melt. All the Rain on Snow is accumulated before melt and released when the first melt occurs, i.e. snowmelt is greater than zero. This is can be seen by an inspection of RainOnSnow as it remains at zero until there is melt in the HRU. The Rain on Snow is accumulated before melt in an internal variable RainOnSnowA which is not accessible to the user.

Notes on Green-Ampt.

This portion of the module replaces the module srunoff which is used to simulate prairie infiltration when the assumption is made that unlimited infiltration is possible.

When the surface ponding is negligible,

$$f \text{ (mm/hr)} = k*(\psi*\Delta\theta/F + 1), \quad \text{(eq. 1.)}$$

where:

k (mm/hr) = saturated hydraulic conductivity.

ψ (mm) = capillary suction at the wetting front.

$\Delta\theta = (\theta s - \theta i)$. change in volumetric soil moisture content.

When ponded,

F1 (mm) = k*Δt + ψ*Δθ * ln((F1 + ψ*Δθ)/(F0 + ψ*Δθ)). (eq. 2.)

where F0 (mm) = cumulative infiltation at the beginning of the time interval i.e. t,

f0 (mm/hr) = infiltration rate at the beginning of the time interval i.e. t,

F1 (mm) = cumulative infiltration at the end of the time interval i.e. t+Δt,

f1(mm/hr) = infiltration rate at the end of the time interval i.e. t+Δt,

It (mm/hr) = rainfall intensity during the interval.

The cumulative infiltration (Fp) at instant of ponding time is

Fp = k*ψ*Δθ/(It - k). (eq. 3.)

The portion of the interval, after which ponding occurs is Δt',

Δt' = (Fp - F0)/It. (eq. 4.)

Flow Chart

Calculate the maximum infiltration rates, f0 and f1 at the beginning and end of the time step using equation 1. using F1 = F0+Δt*It.

Case 1: When f1 is >= rainfall intensity, all the rainfall infiltrates with F1 = F0+Δt*It.

Case 2: When the rainfall intensity > f0, all the rainfall ponds and F is calculated using an iterative solution of equation 2.

Case 3: When the rainfall intensity <= f0 but the rainfall intensity > f1, ponding occurs during the interval and the solution consists of solving for Fp using equation 3, then using equation 4 to solve the time into the step during which ponding occurs. Next, using equation 2 with Fp as F0 the cumulative infiltration F1 after the ponding time of (Δt - Δt') is calculated.

Assigned Soil Properties

All, except fcap are specified at 0%saturation from Chow pg 115.

Value for wilt is from Chow.

Values for por, fcap is from Floods in Canada

Soil curves of form K = Ks / (A*Psi^3 + 1)

	psi(mm)	k(mm/hr)	wilt	fcap	por	A-Soil moist	A-Hyd cond
water	0.0	999.9	0.000	0.00	1.100	1.000000000	1.0000000}
sand	49.5	117.8	0.020	0.10	0.437	0.000010000	0.0002000}
loamsand	61.3	29.9	0.036	0.16	0.437	0.000006000	0.0001000}
sandloam	110.1	10.9	0.041	0.23	0.453	0.000001124	0.0000200}
loam	88.9	3.4	0.029	0.26	0.463	0.000000900	0.0000150}
siltloam	166.8	6.5	0.045	0.38	0.501	0.000000300	0.0000050}
sacllоam	218.5	1.5	0.068	0.38	0.398	0.000000200	0.0000030}
clayloam	208.8	1.0	0.155	0.39	0.464	0.000000150	0.0000020}
sicllоam	273.3	1.0	0.039	0.40	0.471	0.000000080	0.0000015}
sandclay	239.0	0.6	0.110	0.41	0.430	0.000000070	0.0000010}
siltclay	292.2	0.5	0.056	0.43	0.479	0.000000010	0.0000002}
clay	316.3	0.3	0.090	0.46	0.475	0.000000006	0.0000001}

crack (Granger et al. 1984; Gray et al., 1986)

This module is defined in Classcrack and handles frozen soil infiltration during the spring snow melt. This module is based upon 15 years of study of the snow hydrology of the Prairie region and results reported in the former USSR.

Observations

- **none**

Variables

- **snowinfil (mm/d) - amount of daily infiltration – expressed as an equivalent depth (m^3/m^2).**
- **cumsnowinfil (mm) - cummulative infiltration.**
- **runoff (mm/d) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).**
- **cumrunoff (mm) - cummulative runoff.**
- **infil (mm/int) - infiltration in time step, Δt – expressed as an average depth of water on an HRU.**
- **cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.**
- **crackstat () - HRO infiltration status. Values: 0 - not started, 1 to 6 - #events, >6 - *Limited* ended and 10 - ice lens.**
- **crackon () - indicates when the crack frozen soil routine is enabled for an HRU. 0 - disabled/1 - enabled.**
- **RainOnSnow (mm) - cumulative rain on snow.**

Parameters

- **fallstat () - fall status <= 0.0 or -1 for unlimited/cracked, >= 100.0 for restricted and < 0.0 limited < 100.0. Pore saturation as a percentage.**
- **basin_area (km2) - area of watershed.**
- **hru_area(km2) - area of HRUs.**
- **major (mm) - threshold for major melt. Default is 5 mm/day.**
- **PriorInfiltration () - allow 'limited" melt to infiltrate prior to first major melt event. 0 - No/1 - Yes.**

Variable Inputs

- **hru_tmax (obs)**
- **snowmeltD (mm/d) (*)**
- **SWE (mm) (*)**
- **net_rain (mm/int) (*)**

Notes

The Division of Hydrology at the University of Saskatchewan (Granger et al. 1984; Gray et al., 1985), postulated that the infiltration potential of frozen soils may be grouped in three broad categories, namely: restricted, limited and unlimited.

Restricted **- Infiltration is impeded by an impermeable layer, such as an ice lens on the soil surface or within the soil close to the surface. For all practical purposes, the amount of meltwater infiltration can be assumed to be negligible and that the melt goes directly to runoff and to evaporation.**

Limited **- Infiltration is governed primarily by the snow-cover water equivalent and the frozen water content of the top 30 cm. of soil.**

Unlimited **- A soil with a high percentage of large, air-filled macropores at the time of melt. Examples of soils having these properties are dry, heavily cracked clays and coarse, dry sands, organics and others.**

Within the model there is not sufficient information to determine these classifications automatically and as a result the user is required to specify these properties for each HRU every fall and input this information to the model as a parameter. The one case the model does handle is when there is an early melt and the subsequent re-freezing causing an ice lens to form. This will change both Limited to Restricted. Implementation of the infiltration to frozen soils routines is described below.

Definitions

1. Index = INF/SWE where INF=5(1-θ p)*SWE^0.584.

2. Potential = INF/6.

3. MELT_THRESHOLD = 5 mm. Minimum daily meltwater at which the melt routine is enabled. Lower meltwater levels are not counted as one of the six major melt events.

4. A major melt is a day when the amount of meltwater generated is greater than the MELT_THRESHOLD.

5. Six major daily melts are allowed before the limited infiltration category is changed to Restricted.

The Frozen Soil Infiltration routine

1. The Frozen Infiltration routine is enabled in an HRU when its SWE is greater than 50mm. Each HRU is handled independently.

2. Limited infiltration is triggered into operation by the first major melt. At this time, Index and Potential are calculated from the soil moisture (θ p) and the SWE of the snowpack.

3. The Frozen Infiltration routine is disabled in an HRU when its SWE reaches zero.

The three frozen soil categories are described below.

LIMITED

1. Only six major over-winter snowmelt events are possible before the infiltration potential becomes Restricted.

2. Meltwater amounts less than the MELT_THRESHOLD is handled as runoff unless the parameter *PriorInfiltration***are is set, then all of the melt is permitted to infiltrate into the soil. Once the MELT_THRESHOLD has been exceeded it is assumed that spring snowmelt has begun and only an amount of meltwater equal to MELT*Index will infiltrate and the remainder will be handled as runoff.**

3. Index and Potential are recalculated if another major melt occurs with a greater SWE.

4. If the temperature the day following a major melt event is colder than -10ºC, the category is changed from Limited to Restricted assuming an ice lens has formed.

5. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

UNLIMITED

1. All meltwater is allowed to infiltrate.

2. If the temperature the day after a major melt event is less than -10ºC, the category remains unchanged and all the melt water still infiltrates into the soil.

3. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

RESTRICTED

1. No meltwater is allowed to infiltrate.

2. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

Rain on Snow.

The energy component of Rain on Snow was not handled when the Rain on Snow occurs before melt. All the Rain on Snow is accumulated before melt and released when the first melt occurs, i.e. snowmelt is greater than zero. This is can be seen by an inspection of RainOnSnow as it remains at zero until there is melt in the HRU. The Rain on Snow is accumulated before melt in an internal variable RainOnSnowA which is not accessible to the user.

frozenAyers .

This module is defined in ClassfrozenAyers . It handles unfrozen soil infiltration using Ayers and frozen soil using Zhao and Gray (1999)..

Observations

- t0_inhibit () - optional. If its value is > 0.0 the incrementing of t0_Acc during the calibration cycle is inhibited. This input may be controlled using a macro.

Variables

- infil (mm/int) - depth of infiltration in time step, Δt – expressed as an average depth of water on an HRU.
- cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.
- runoff (mm/int) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).
- cumrunoff (mm) - cummulative runoff.
- meltrunoff (mm/int) - amount of interval melt runoff. .
- cummeltrunoff (mm) - cumulative interval melt runoff.
- snowinfil (mm/int) - amount of interval infiltration – expressed as an equivalent depth (m^3/m^2).
- cumsnowinfil (mm) - cumulative interval infiltration
- t0_Var (h) - value of T0 used by model..
- t0_Acc (h) - infiltration opportunity time accumulator.
- Inf (mm) - infiltration into a frozen soil calculated from parametric relationship.
- infiltype() - infiltration type. PREMELT/RESTRICTED/LIMITED/UNLIMITED/SATURATED - 0/1/2/3/4 respectively.
- snowmeltD(mm/d) - yesterday's snowmelt.

Parameters

- basin_area (km^2) - area of watershed.
- hru_area(km^2) - area of HRUs.
- t0 (h) - Infiltration opportunity time calculated and loaded after a model run with t0[HRU 1] <= 0.
- Reset_t0 () - Reset t0 to zero on this Julian date. If zero does nothing. Used for multiple year runs.
- S0 (m^3/m^3) - surface saturation.
- Si (m^3/m^3) - initial soil saturation.
- C () - coefficient.
- hru_tsoil (ºK) - soil average temperature at start of frozen infiltration.
- t_ice_lens(ºC) - overnight minimum to cause ice lens after major melt.
- texture () - 1 through 4 for texture:- 1 - coarse/medium over coarse, 2 - medium over medium, 3 - medium/fine over fine, 4 - soil over shallow bedrock..
- groundcover () - 1 through 6 for groundcover: 1 - bare soil, 2 - row crop, 3 - poor pasture, 4 - small grains, 5 - good pasture, 6 - forested..
- soil_moist_max (mm) - Maximum available water holding capacity of soil profile. Soil profile is surface to bottom of rooting zone.
- reset_t0 () - reset t0 to 0 at this Julian date. 0 implies disable. Default incorrect for the Southern Hemisphere (~210).

Variable Inputs

- net_rain (mm/int) (*) - amount of rain received by the ground surface in time step, Δt. Variable available from module *intcp*, etc..
- snowmeltD (mm/d) (*).
- soil_moist (mm) (*)
- SWE (mm) (*)
- net_rain (mm/int) (*)
- hru_tmin (°C) (*)

Notes on Ayers.

textureproperties[texture] [groundcover] in mm/hour.

bare soil	row crop	poor pasture	small grains	good pasture	forested	
7.6	12.7	15.2	17.8	25.4	76.2	coarse over coarse
2.5	5.1	7.6	10.2	12.7	15.2	medium over medium
1.3	1.8	2.5	3.8	5.1	6.4	medium/fine over fine
0.5	0.5	0.5	0.5	0.5	0.5	soil over shallow bedrock

Calculations used in frozen.

- c*pow(S0, 2.92)*pow(1.0-Si, 1.64)*pow((273.15-tmean)/273.15, -0.45)*pow(t0, 0.44) *10

Reference

- Zhao and Gray (1999)

Notes.

1. Calibration. Since the opportunity time is not known until the model is run through the complete frozen infiltration period, the model has to be run twice. The user, by setting t0[HRU 1] to <= 0.0 causes the module to calculate the infiltration opportunity time. During this run the module forces all HRUs to be restricted. At the end of the model run the newly calculated opportunity times are transferred to the parameter. If the run is terminated early the partially calculated values will be used. During succeeding model runs the earlier calculated infiltration opportunity times will be used until t0[HRU 1] is set to <= 0.0 again.
2. Maximum infiltration is limited to the lesser of Inf calculated from the equation above and the available storage in the module smbal, i.e. (soil_moist - soil_moist_max).
3. Accumulation of the opportunity time when calculating. Normally 24 hours is added for every day when melt occurs. However, this may be refined by using the observation input t0_inhibit. When this input is not connected or its value is zero, the interval is added to the infiltration opportunity time. When t0_inhibit is greater than 0.0 the interval is not added.
4. When t0 is zero or is reset to zero using the Reset_t0 parameter, CRHM calculates ahead to the end of melt to determine the 'Infiltration opportunity time'. It then returns to the beginning of melt and handles infiltration over the melt period.

PrairieInfiltration .

This module is defined in ClassPrairieInfiltration . It handles unfrozen soil infiltration using Ayers and frozen soil using Granger et al. 1984; Gray et al., 1986..

Observations

- none

Variables

- infil (mm/int) - depth of infiltration in time step, Δt – expressed as an average depth of water on an HRU.
- cuminfil (mm) - cumulative infiltration - expressed as an average depth of water on an HRU.
- runoff (mm/int) - amount of daily runoff – expressed as an equivalent depth (m^3/m^2).
- cumrunoff (mm) - cummulative runoff.
- meltrunoff (mm/int) - amount of interval melt runoff. .
- cummeltrunoff (mm) - cumulative interval melt runoff.
- snowinfil (mm/int) - amount of interval infiltration – expressed as an equivalent depth (m^3/m^2).
- cumsnowinfil (mm) - cumulative interval infiltration
- crackstat () - HRO infiltration status. Values: 0 - not started, 1 to 6 - #events, >6 - *Limited* ended and 10 - ice lens.
- crackon () - indicates when the crack frozen soil routine is enabled for an HRU. 0 - disabled/1 - enabled.
- RainOnSnow (mm) - cumulative rain on snow.

Parameters

- basin_area (km^2) - area of watershed.
- hru_area(km^2) - area of HRUs.
- texture () - 1 through 4 for texture:- 1 - coarse/medium over coarse, 2 - medium over medium, 3 - medium/fine over fine, 4 - soil over shallow bedrock..
- groundcover () - 1 through 6 for groundcover: 1 - bare soil, 2 - row crop, 3 - poor pasture, 4 - small grains, 5 - good pasture, 6 - forested..
- fallstat () - fall status <= 0.0 or -1 for unlimited/cracked, >= 100.0 for restricted and < 0.0 limited < 100.0. Pore saturation as a percentage.
- major (mm) - threshold for major melt. Default is 5 mm/day.
- PriorInfiltration () - allow 'limited" melt to infiltrate prior to first major melt event. 0 - No/1 - Yes.

Variable Inputs

- hru_tmax (obs)
- snowmeltD (mm/d) (*)
- SWE (mm) (*)
- net_rain (mm/int) (*) - amount of rain received by the ground surface in time step, Δt. Variable available from module *intcp*, etc..

Notes on Ayers.

textureproperties[texture] [groundcover] in mm/hour.

bare soil	row crop	poor pasture	small grains	good pasture	forested	
7.6	12.7	15.2	17.8	25.4	76.2	coarse over coarse
2.5	5.1	7.6	10.2	12.7	15.2	medium over medium
1.3	1.8	2.5	3.8	5.1	6.4	medium/fine over fine
0.5	0.5	0.5	0.5	0.5	0.5	soil over shallow bedrock

Notes on frozen routine.

The Division of Hydrology at the University of Saskatchewan (Granger et al. 1984; Gray et al., 1985), postulated that the infiltration potential of frozen soils may be grouped in three broad categories, namely: restricted, limited and unlimited.

Restricted - Infiltration is impeded by an impermeable layer, such as an ice lens on the soil surface or within the soil close to the surface. For all practical purposes, the amount of meltwater infiltration can be assumed to be negligible and that the melt goes directly to runoff and to evaporation.

Limited **- Infiltration is governed primarily by the snow-cover water equivalent and the frozen water content of the top 30 cm. of soil.**

Unlimited **- A soil with a high percentage of large, air-filled macropores at the time of melt. Examples of soils having these properties are dry, heavily cracked clays and coarse, dry sands, organics and others.**

Within the model there is not sufficient information to determine these classifications automatically and as a result the user is required to specify these properties for each HRU every fall and input this information to the model as a parameter. The one case the model does handle is when there is an early melt and the subsequent re-freezing causing an ice lens to form. This will change both Limited to Restricted. Implementation of the infiltration to frozen soils routines is described below.

Definitions

1. Index = INF/SWE where INF=5(1-θ p)*SWE^0.584.

2. Potential = INF/6.

3. MELT_THRESHOLD = 5 mm. Minimum daily meltwater at which the melt routine is enabled. Lower meltwater levels are not counted as one of the six major melt events.

4. A major melt is a day when the amount of meltwater generated is greater than the MELT_THRESHOLD.

5. Six major daily melts are allowed before the limited infiltration category is changed to Restricted.

The Frozen Soil Infiltration routine

1. The Frozen Infiltration routine is enabled in an HRU when its SWE is greater than 50mm. Each HRU is handled independently.

2. Limited infiltration is triggered into operation by the first major melt. At this time, Index and Potential are calculated from the soil moisture (θ p) and the SWE of the snowpack.

3. The Frozen Infiltration routine is disabled in an HRU when its SWE reaches zero.

The three frozen soil categories are described below.

LIMITED

1. Only six major over-winter snowmelt events are possible before the infiltration potential becomes Restricted.

2. Meltwater amounts less than the MELT_THRESHOLD is handled as runoff unless the parameter *PriorInfiltration* **are is set, then all of the melt is permitted to infiltrate into the soil. Once the MELT_THRESHOLD has been exceeded it is assumed that spring snowmelt has begun and only an amount of meltwater equal to MELT*Index will infiltrate and the remainder will be handled as runoff.**

3. Index and Potential are recalculated if another major melt occurs with a greater SWE.

4. If the temperature the day following a major melt event is colder than -10ºC, the category is changed from Limited to Restricted assuming an ice lens has formed.

5. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

UNLIMITED

1. All meltwater is allowed to infiltrate.

2. If the temperature the day after a major melt event is less than -10ºC, the category remains unchanged and all the melt water still infiltrates into the soil.

3. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

RESTRICTED

1. No meltwater is allowed to infiltrate.

2. When the SWE of the snowpack is zero, the model returns to the module that handles unfrozen infiltration.

Rain on Snow.

The energy component of Rain on Snow was not handled when the Rain on Snow occurs before melt. All the Rain on Snow is accumulated before melt and released when the first melt occurs, i.e. snowmelt is greater than zero. This is can be seen by an inspection of RainOnSnow as it remains at zero until there is melt in the HRU. The Rain on Snow is accumulated before melt in an internal variable RainOnSnowA which is not accessible to the user.

Soil

This module is defined in ClassSoil and handles soil moisture accounting throughout the year. When snow cover is present, the input to this module is the infiltration (snowinfil) generated by the modules crack, frozen etc. From the end of snow melt till late fall the infiltration (infil) is generated by the modules Greencrack, PrairieInfiltration etc. The soil is handled as two layers. The upper layer is called the recharge layer and represents the top soil. Evaporation can occur from the recharge layer, soil_moist (all layers) or none. The surface infiltration satisfies the recharge layer before being used by the lower layer. The excess water from both soil layers contribute to the ground water flow , depressional storage and then to sub surface flow. The depressional storage is allowed to drain to ground water and sub surface flow.The parameters soil_rechr_max and soil_moist_max represents the maximum soil moisture capacity for the two layers i.e. field capacity. Wilt occurs when the state variables soil_rechr and soil_moist are equal to zero.

Evaporation and subsurface runoff are handled after infiltration. The maximum amount of evapotranspiration is calculated by the module *evap*. There is a variation (#1) to handle HRU runoff through a culvert.

Observations

- none

Variables

- Sd (mm) - depression storage. State variable.
- gw (mm) - ground water storage. State variable.
- soil_rechr (mm) - soil moisture content of recharge zone, i.e., the portion of the soil profile from which water is withdrawn by evaporation - expressed as an equivalent depth (m^3/m^2). State variable.
- soil_moist (mm) - soil moisture content of the rooting zone of the major vegetation type on the HRU - expressed as an equivalent depth (m^3/m^2). State variable.
- soil_ssr (mm/int) - subsurface runoff from soil_rechr and depressional storage.
- cum_soil_ssr (mm) - cumulative subsurface runoff from soil_rechr.
- soil_ssr_D (mm/d) - daily accumulation of soil_ssr.
- rechr_ssr (mm/int) - Portion of excess soil water from a HRU that enters subsurface reservoirs.
- cum_rechr_ssr (mm) - Accumulation of Portion of excess from a HRU to ssr.
- soil_gw (mm/int) - excess from soil_moist and depressional storage that enters groundwater reservoirs.
- cum_soil_gw (mm) - cumulative portion of excess soil water from a HRU that enters groundwater reservoirs.
- soil_gw_D (mm/d) - portion of excess soil water from a HRU that enters groundwater reservoirs in a day.
- soil_runoff (mm/int) - remainder of excess soil water.
- cum_soil_runoff (mm) - cumulative surface runoff.
- soil_runoff_D (mm/d) - daily accumulation of soil_runoff.
- cum_runoff_to_Sd (mm/int) - cumulative portion of runoff to depression storage.
- infil_act (mm/int) - Actual amount of water infiltrating the soil on each HRU.
- infil_act_D (mm/d) - Daily actual amount of water infiltrating the soil on each HRU.
- cum_infil (mm) - cumulative potential amount of water infiltrating the soil on each HRU. Contents of the variable "infil" are calculated in the infiltration module used.
- cum_infil_act (mm) - cumulative actual amount of water infiltrating the soil on each HRU.
- gw_flow (mm/int) - Drainage from HRU ground water reservoir.
- gw_flow_D (mm/d) - Daily drainage from HRU ground water reservoir.
- cum_gw_flow (mm) - Accumulation of excess soil water from a HRU that enters groundwater reservoirs.
- redirected_residual (mm*km^2/int) - redirected residual after topping up Sd and soil_rechar in Netroute/Netroute_D/Netroute_M/Netroute_M_D. Average over the basin.
- cum_redirected_residual (mm*km^2/int) - cumulative redirected residual after topping up Sd and soil_rechar in Netroute/Netroute_D/Netroute_M/Netroute_M_D. Average over the basin.

Culvert variables only in variation #1.

- culvert_Q (m^3/s) - flow in culvert.
- culvert_water_H (m) - depth of pond at culvert inlet.
- culvert_water_A (m^2) - surface area of culvert pond
- culvert_water_V (m^3) - volume of water in culvert pond
- culvert_water_O (m^3) - volume of water overflowing road
- culvert_evap (mm/int) - Depth of water evaporating.
- cum_culvert (mm) - Cumulative culvert HRU flow.
- cum_culvert_over (mm) - Cumulative culvert HRU overflow.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- Sdmax (mm) - Maximum depression storage.
- Sdinit (mm) - Initial depression storage.
- soil_rechr_max (mm) - maximum moisture content of the soil recharge zone - expressed as an equivalent depth (mm^3/mm^2). Must be less than or equal to soil_moist.
- soil_rechr_init (mm) - initial moisture content of the soil recharge zone - expressed as an equivalent depth (mm^3/mm^2). Must be less than or equal to soil_moist_init.

- soil_moist_max (mm) - maximum available water holding capacity of rooting zone - expressed as an equivalent depth (mm^3/mm^2).
- soil_moist_init (mm) - initial moisture content of rooting zone - expressed as an equivalent depth (mm^3/mm^2).
- gw_max (mm) - Maximum available water holding capacity of ground water reservoir.
- gw_init (mm) - Initial value of available water in ground water reservoir.
- rechr_ssr_K (mm/d) - recharge ssr drainage factor.
- lower_ssr_K (mm/d) - lower column (soil_moist - rechr_mois) ssr drainage factor.
- soil_gw_K (mm/d) - The maximum amount of the excess soil water for an HRU that is routed directly to the associated groundwater reservoir each day- expressed as an equivalent depth (mm^3/mm^2).
- Sd_ssr_K (mm/d) - depression storage ssr factor.
- Sd_gw_K(mm/d) - depression storage groundwater factor.
- gw_K (mm/d) - daily ground water drainage from gw reservoir.
- soil_withdrawal () - HRU evaporation withdrawal for soil type: 1= sand, 2= loam, 3= clay. Water availability is used to limit evaporation as decribed below.
- cov_type () - Vegetation cover type designation for HRU: "0 = no evaporation, 1 = bare soil or shallow crops (evaporation from the recharge layer only), 2 = crops, grasses, shrubs and trees (evaporation from all soil moisture).
- transp_limited () - Inhibits transpiration from the soil moisture layer when set to 1.
- soil_ssr_runoff () - soil column excess to interflow(ssr)/runoff (and possibly SD) interflow-0/runoff-1

Culvert variables only in variation #1.

- channel_slope () - soil slope to culvert.
- side_slope () - side soil slope mormal to culvert slope
- culvert_diam (m) - culvert diameter.
- culvert_water_Dmax (m) - maximum depth of pond at culvert inlet
- number_culverts () - number of culverts and efficiency factor. Zero = no culvert

Variable Inputs

- hru_evap (evap) (mm/int) or (mm/d). Programmed for an interval or daily value.
- infil (*) (mm/int).
- snowinfil (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- runoff (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- meltrunoff (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- hru_actet (*) (mm/int). Programmed as a *put* value.
- hru_cum_actet (*) (mm).Programmed as a state *put* value.

Notes

Set Sdmax to zero and soil_moist to non- zero to model HRU with no Depression or Pond storage.

Set Sdmax to non-zero and soil_moist to non- zero to model HRU with Depression storage.

Set Sdmax to non-zero and soil_moist to zero to model HRU with Pond storage.

The significant difference between Pond and Depression storage is that soil_runoff to Pond storage is unlimited to its maximum capacity whereas depression storage is also limited to (Sdmax-Sd)*(1-exp(-soil_runoff/Sdmax)).

The layer soil_rechr is part of the soil_moist zone. It is not a separate identity.

Evaporation from bare soil and shallow crops only occurs from the recharge layer. All other cover types have evaporation/transpiration from recharge and soil moisture layers.

transp_limited is provided to limit grasses, shrubs and trees to only the recharge layer when the vegetation limits transpiration during drought conditions.

infil, *snowinfil* and condensation are added to the recharge layer and then to soil_moist. The excess is next used to satisfy groundwater. The surplus is designated as soil_runoff. The incoming runoff and meltrunoff are added to soil_runoff. If depression storage is specified any soil_runoff is used to fill the depression storage to its maximum.

Soil Withdrawal Types.

Water availability controls evaporation withdrawal in 'smbal' and 'Soil'. Modified from Zahner(1967) by G.H.Leavesley.

1 - Sand, at pcts < 0.25, limited to 0.25*pcts*avail_evap,

2 - loam, at pcts < 0.5, limited to 0.5*pcts*avail_evap,

3 - clay, at pcts < 0.67 && pcts > 0.33, limited to pcts*avail_evap and at pcts < 0.33 limited to 0.5*pcts*avail_evap,

4 organic, unlimted availability,

where pcts = (soil moist)/(soil moist max).

SoilX

This module is defined in ClassSoil and handles soil moisture accounting throughout the year. When snow cover is present, the input to this module is the infiltration (snowinfil) generated by the modules crack, frozen etc. From the end of snow melt till late fall the infiltration (infil) is generated by the modules Greencrack, PrairieInfiltration etc. The soil is handled as two layers. The upper layer is called the recharge layer and represents the top soil. Evaporation can occur from the recharge layer, soil_moist (all layers) or none. The surface infiltration satisfies the recharge layer before being used by the lower layer. The excess water from both soil layers contribute to the ground water flow , depressional storage and then to sub surface flow. The depressional storage is allowed to drain to ground water and sub surface flow.The parameters soil_rechr_max and soil_moist_max represents the maximum soil moisture capacity for the two layers i.e. field capacity. Wilt occurs when the state variables soil_rechr and soil_moist are equal to zero.

Evaporation and subsurface runoff are handled after infiltration. The maximum amount of evapotranspiration is calculated by the module *evap*. There is a variation (#1) to handle HRU runoff through a culvert.

In this version evaporation is taken from the runoff until the recharge layer of the soil has begun to thaw.

Observations

- **none**

Variables

- **Sd (mm) - depression storage. State variable.**
- **gw (mm) - ground water storage. State variable.**
- **soil_rechr (mm) - soil moisture content of recharge zone, i.e., the portion of the soil profile from which water is withdrawn by evaporation - expressed as an equivalent depth (m^3/m^2). State variable.**
- **soil_moist (mm) - soil moisture content of the rooting zone of the major vegetation type on the HRU - expressed as an equivalent depth (m^3/m^2). State variable.**
- **soil_ssr (mm/int) - subsurface runoff from soil_rechr and depressional storage.**
- **cum_soil_ssr (mm) - cumulative subsurface runoff from soil_rechr.**
- **soil_ssr_D (mm/d) - daily accumulation of soil_ssr.**
- **rechr_ssr (mm/int) - Portion of excess soil water from a HRU that enters subsurface reservoirs.**
- **cum_rechr_ssr (mm) - Accumulation of Portion of excess from a HRU to ssr.**
- **soil_gw (mm/int) - excess from soil_moist and depressional storage that enters groundwater reservoirs.**
- **cum_soil_gw (mm) - cumulative portion of excess soil water from a HRU that enters groundwater reservoirs.**
- **soil_gw_D (mm/d) - portion of excess soil water from a HRU that enters groundwater reservoirs in a day.**
- **soil_runoff (mm/int) - remainder of excess soil water.**
- **cum_soil_runoff (mm) - cumulative surface runoff.**
- **soil_runoff_D (mm/d) - daily accumulation of soil_runoff.**
- **cum_runoff_to_Sd (mm/int) - cumulative portion of runoff to depression storage.**
- **infil_act (mm/int) - Actual amount of water infiltrating the soil on each HRU.**
- **infil_act_D (mm/d) - Daily actual amount of water infiltrating the soil on each HRU.**
- **cum_infil (mm) - cumulative potential amount of water infiltrating the soil on each HRU. Contents of the variable "infil" are calculated in the infiltration module used.**
- **cum_infil_act (mm) - cumulative actual amount of water infiltrating the soil on each HRU.**
- **gw_flow (mm/int) - Drainage from HRU ground water reservoir.**
- **gw_flow_D (mm/d) - Daily drainage from HRU ground water reservoir.**
- **cum_gw_flow (mm) - Accumulation of excess soil water from a HRU that enters groundwater reservoirs.**
- **redirected_residual (mm*km^2/int) - redirected residual after topping up Sd and soil_rechar in Netroute/Netroute_D/Netroute_M/Netroute_M_D. Average over the basin.**
- **cum_redirected_residual (mm*km^2/int) - cumulative redirected residual after topping up Sd and soil_rechar in Netroute/Netroute_D/Netroute_M/Netroute_M_D. Average over the basin.**
- **depth_upper (m) - depth of soil in upper layer. Calculated from soil_rechr_max and porosity_upper**
- **depth_lower (m) - depth of soil in lower layer. Calculated from soil_max and porosity_lower.**
- **thaw_upper () - thawed fraction of upper layer. Controls infiltration, rechr_ssr_K, soil_gw_K and evaporation.**
- **thaw_lower () - thawed fraction of lower layer. Controls lower_ssr_K.**
- **thaw_all () - thawed fraction of combined upper and lower layer. Controls Sd_ssr_K.**

Culvert variables only in variation #1.

- **culvert_Q (m^3/s) - flow in culvert.**
- **culvert_water_H (m) - depth of pond at culvert inlet.**
- **culvert_water_A (m^2) - surface area of culvert pond**
- **culvert_water_V (m^3) - volume of water in culvert pond**
- **culvert_water_O (m^3) - volume of water overflowing road**
- **culvert_evap (mm/int) - Depth of water evaporating.**
- **cum_culvert (mm) - Cumulative culvert HRU flow.**
- **cum_culvert_over (mm) - Cumulative culvert HRU overflow.**

Parameters

- **basin_area (km^2) - basin area.**

- hru area (km^2) - HRU area.
- Sdmax (mm) - Maximum depression storage.
- Sdinit (mm) - Initial depression storage.
- soil_rechr_max (mm) - maximum moisture content of the soil recharge zone - expressed as an equivalent depth (mm^3/mm^2). Must be less than or equal to soil_moist.
- soil_rechr_init (mm) - initial moisture content of the soil recharge zone - expressed as an equivalent depth (mm^3/mm^2). Must be less than or equal to soil_moist_init.
- soil_moist_max (mm) - maximum available water holding capacity of rooting zone - expressed as an equivalent depth (mm^3/mm^2).
- soil_moist_init (mm) - initial moisture content of rooting zone - expressed as an equivalent depth (mm^3/mm^2).
- gw_max (mm) - Maximum available water holding capacity of ground water reservoir.
- gw_init (mm) - Initial value of available water in ground water reservoir.
- rechr_ssr_K (mm/d) - recharge ssr drainage factor.
- lower_ssr_K (mm/d) - lower column (soil_moist - rechr_mois) ssr drainage factor.
- soil_gw_K (mm/d) - The maximum amount of the excess soil water for an HRU that is routed directly to the associated groundwater reservoir each day- expressed as an equivalent depth (mm^3/mm^2).
- Sd_ssr_K (mm/d) - depression storage ssr factor.
- Sd_gw_K(mm/d) - depression storage groundwater factor.
- gw_K (mm/d) - daily ground water drainage from gw reservoir.
- soil_withdrawal () - HRU evaporation withdrawal for soil type: 1= sand, 2= loam, 3= clay. Water availability is used to limit evaporation as decribed below.
- cov_type () - Vegetation cover type designation for HRU: "0 = no evaporation, 1 = bare soil or shallow crops (evaporation from the recharge layer only), 2 = crops, grasses, shrubs and trees (evaporation from all soil moisture).
- transp_limited () - Inhibits transpiration from the soil moisture layer when set to 1.
- soil_ssr_runoff () - soil column excess to interflow(ssr)/runoff (and possibly SD) interflow-0/runoff-1
- porosity_upper(m^3/m^3) - upper soil porosity (recharge layer).
- porosity_loweer(m^3/m^3) - lower soil porosity.

Culvert variables only in variation #1.

- channel_slope () - soil slope to culvert.
- side_slope () - side soil slope mormal to culvert slope
- culvert_diam (m) - culvert diameter.
- culvert_water_Dmax (m) - maximum depth of pond at culvert inlet
- number_culverts () - number of culverts and efficiency factor. Zero = no culvert

Variable Inputs

- hru_evap (evap) (mm/int) or (mm/d). Programmed for an interval or daily value.
- infil (*) (mm/int).
- snowinfil (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- runoff (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- meltrunoff (*) (mm/int) or (mm/d). Programmed for an interval or daily value.
- hru_actet (*) (mm/int). Programmed as a *put* value.
- hru_cum_actet (*) (mm).Programmed as a state *put* value.
- frostdepth (*)(m) - depth of frost table from surface.

Notes

Set Sdmax to zero and soil_moist to non- zero to model HRU with no Depression or Pond storage.

Set Sdmax to non-zero and soil_moist to non- zero to model HRU with Depression storage.

Set Sdmax to non-zero and soil_moist to zero to model HRU with Pond storage.

The significant difference between Pond and Depression storage is that soil_runoff to Pond storage is unlimited to its maximum capacity whereas depression storage is also limited to (Sdmax-Sd)*(1-exp(-soil_runoff/Sdmax)).

The layer soil_rechr is part of the soil_moist zone. It is not a separate identity.

Evaporation from bare soil and shallow crops only occurs from the recharge layer. All other cover types have evaporation/transpiration from recharge and soil moisture layers.

transp_limited is provided to limit grasses, shrubs and trees to only the recharge layer when the vegetation limits transpiration during drought conditions.

infil, *snowinfil* and condensation are added to the recharge layer and then to soil_moist. The excess is next used to satisfy groundwater. The surplus is designated as soil_runoff. The incoming runoff and meltrunoff are added to soil_runoff. If depression storage is specified any soil_runoff is used to fill the depression storage to its maximum.

Soil Withdrawal Types.

Water availability controls evaporation withdrawal in 'smbal' and 'Soil'. Modified from Zahner(1967) by G.H.Leavesley.

1 - Sand, at pcts < 0.25, limited to 0.25*pcts*avail_evap,

2 - loam, at pcts < 0.5, limited to 0.5*pcts*avail_evap,

3 - clay, at pcts < 0.67 && pcts > 0.33, limited to pcts*avail_evap and at pcts < 0.33 limited to 0.5*pcts*avail_evap,

4 organic, unlimted availability,

where pcts = (soil moist)/(soil moist max).

K_Estimate.

The rates for lateral flow rate in soil layers and groundwater layer (i.e. subsurface and groundwater discharges) as well as vertical flow of excess soil water to groundwater (i.e. groundwater recharge) are controlled by several drainage factors: rechr_ssr_K [mm day-1], lower_ssr_K [mm day-1], gw_K [mm day-1], sd_ssr_K [mm day-1], sd_gw_K [mm day-1] and soil_gw_K [mm day-1]. rechr_ssr_K, lower_ssr_K, gw_K and sd_ssr_K are the drainage factors for lateral flows in soil recharge, lower soil, groundwater layers and depressions storage, respectively. soil_gw_K and sd_gw_K are the drainage factors for the vertical flow from soil and from depression storage to groundwater layer. Darcy's law for unsaturated flow is used to calculate these drainage factors.

Observations

- **none**

Variables

- **v_L_upper (m/s) - Darcy's lateral flow velocity in upper soil column (ie. recharge layer).**
- **v_L_lower (m/s) - Darcy's lateral flow velocity in lower soil column.**
- **v_V_sd (m/s) - Darcy's vertical flow velocity for sd (ie. depression).**
- **v_V_soil (m/s) - Darcy's vertical flow velocity for soil column.**
- **v_L_gw (m/s) -Darcy's lateral flow velocity for groundwater reservoir.**

Parameters

- **Ks_lower (m/s) - saturated hydraulic conductivity for lower soil.**
- **Ks_upper (m/s) - saturated hydraulic conductivity for upper soil layer.**
- **Ks_gw (m/s) - saturated hydraulic conductivity for groundwater layer.**
- **hru_GSL (°) - ground slope - increasing the slope positively, tilts the plane to the north with ASL = 0.**
- **porosity (m^3/m^3) - soil porosity.**
- **PSD () - pore size distribution.**
- **soil_rechr_max (mm) - Maximum value for soil recharge zone (upper portion of soil_moist where losses occur as both evaporation and transpiration). Must be less than or equal to soil_moist.**
- **soil_moist_max (mm) - Maximum available water holding capacity of soil profile. Soil profile is surface to bottom of rooting zone.**
- **gw_max (mm) - Maximum available water holding capacity of ground water profile.**
- **inhibit_evap (flag) - 0/1 enable/inhibit. Is indicator to when there is snowcover.**

The following parameters are "declputparam" (i.e. the parameter values can be changed by this module.)

- **rechr_ssr_K (m/d) - daily ssr drainage from recharge.**
- **lower_ssr_K (m/d) - daily ssr drainage from soil column.**
- **Sd_ssr_K (m/d) - daily depression storage ssr drainage factor.**
- **Sd_gw_K (m/d) - daily depression storage gw drainage.**
- **soil_gw_K (m/d) - daily maximum amount of the soil water excess for an HRU that is routed directly to the associated groundwater reservoir each.**
- **gw_K (m/d) - daily ground water drainage from gw reservoir.**

Variable Inputs

- **soil_rechr (*) (mm).**
- **soil_moist (*) (mm).**
- **gw (*) (mm).**

SetSoil

Defined in ClassSetSoil. This module contains no active code. It performs the same functions as the original Classbasin but adds the capability of setting ClassSoil parameters from soil type and volumetric water content. It declares general parameters for the model. Examples of the parameters declared are: basin area, HRU area, latitude, elevation, ground slope (GSL) and aspect angle (ASL).

Observations

- none

Variables

- run_ID () - run identification.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- hru_lat (°) - latitude.
- hru_elev (m) - altitude.
- hru_GSL (°) - ground slope.
- hru_ASL (°) - aspect.
- basin_name - text string.
- hru_names - text strings.
- RUN_ID - integer number. Used to uniquely identify "CRHM_output" log files when RUN_ID is positive.
- RUN_START (d) - run start day. Only used on batch execution.
- RUN_END (d) - run end day. Only used on batch execution.
- INIT_STATE - initial state file (Automation).
- Vol_h2o_content () - fractional volumetric water content.
- soil_Depth_rechr (m) - recharge depth.
- soil_Depth (m) - soil column depth.
- soil_type_rechr () - recharge layer soil type. 0 - 12, water/sand/loamsand/sandloam/loam/siltloam/sasclloam/clayloam/siclloam/sandclay/siltclay/clay/pavement.
- soil_type () - soil column. 0 - 12, water/sand/loamsand/sandloam/loam/siltloam/sasclloam/clayloam/siclloam/sandclay/siltclay/clay/pavement.

The following parameters are set by this module from the above information and need not be set.

- soil_rechr_max = soil_Depth_rechr*SetSoilproperties[soiltype][field capacity - wilt].
- soil_rechr_init = soil_Depth_rechr*(Vol_h2o_content*SetSoilproperties[soiltype_rechr][field] - SetSoilproperties[soiltype_rechr][wilt]).
- soil_moist_max = soil_Depth*SetSoilproperties[soiltype][field capacity - wilt].
- soil_moist_init = soil_Depth*(Vol_h2o_content*SetSoilproperties[soiltype][field] - SetSoilproperties[soiltype][wilt]).
- soil_rechr_init is limited to the range >= 0 and <= soil_rechr_max.
- soil_moist_init is limited to the range >= 0 and <= soil_moist_max.
- soil_rechr_init cannot be > soil_moist_init.
- When the two soil layers are of different types the values are only approximate.

Variable Inputs

- none

Notes

1. This module only executes once as an initialization routine before the model execution..

available (mm)	wilt (mm)	field capacity (mm)	pore space ()	
1000.0	0.0	1000.0	1000.0	0 water
84.0	40.0	124.0	395.0	1 sand
80.0	60.0	140.0	410.0	2 loamsand
130.0	100.0	230.0	435.0	3 sandloam
157.0	110.0	267.0	451.0	4 loam
162.0	130.0	292.0	485.0	5 siltloam
170.0	140.0	310.0	420.0	6 sacIloam
167.0	150.0	317.0	476.0	7 clayloam

150.0	190.0	340.0	477.0	8 siclloam
150.0	200.0	350.0	426.0	9 sandclay
150.0	210.0	360.0	492.0	10 siltclay
145.0	215.0	360.0	482.0	11 clay

Volumetric

Defined in ClassVolumetric. The purpose of this module is to display the volumetric soil moisture as calculated from the variables in the module "Soil" and the properties of the soil. Additionally, on a selected date in the fall it sets the parameter " fallstat" handle the infiltration into frozen soil for the following spring as determined from the soil properties and the soil moisture variables in the module "Soil".

Observations

- **none**

Variables

- **Volumetric () - fractional volumetric soil moisture and equals (soil_moist[hh]/soil_Depth[hh] + SetSoilproperties[soiltype[hh]][1])/1000.0).**

Parameters

- **soil_Depth (m) - soil column depth.**
- **soil_type () - soil column. 0 - 12, water/sand/loamsand/sandloam/loam/siltloam/sasclloam/clayloam/siclloam/sandclay/siltclay/clay/pavement.**
- **set_fallstat () - Julian day when the fallstat parameter is set from the current soil moisture**

The following parameters are set by this module from the above information and need not be set.

- **fallstat () - percentage of pore space holding water and equals Volumetric*100.0.**

Variable Inputs

- **soil_moist (*) (mm) - from module Soil etc.**

Notes

1. **During the first interval of execution, if set_fallstat is greater than the current Julian date, fallstat is set. Otherwise, only on the last interval of the actual Julian date.**

available (mm)	wilt (mm)	field capacity (mm)	pore space ()	
1000.0	0.0	1000.0	1000.0	0 water
84.0	40.0	124.0	395.0	1 sand
80.0	60.0	140.0	410.0	2 loamsand
130.0	100.0	230.0	435.0	3 sandloam
157.0	110.0	267.0	451.0	4 loam
162.0	130.0	292.0	485.0	5 siltloam
170.0	140.0	310.0	420.0	6 sacllloam
167.0	150.0	317.0	476.0	7 clayloam
150.0	190.0	340.0	477.0	8 siclloam
150.0	200.0	350.0	426.0	9 sandclay
150.0	210.0	360.0	492.0	10 siltclay
145.0	215.0	360.0	482.0	11 clay

Netroute

This module defined in ClassNetroute, handles the routing of surface runoff, subsurface runoff and HRU routing using the lag and route method described by Clark(1945). Outflow from a HRU is calculated by lagging its inflow by the travel time through the HRU, then routing it through an amount of linear storage defined by the storage constant, K. The outflow from a HRU can be diverted to the inflow of another HRU or directed to the basin outlet. The parameter 'order' ensures that the outflows from the various HRUs are calculated in the correct order from upstream to downstream. The parameter 'whereto' defines the destination of an HRU outflow. A non zero value indicates flow to another HRU and a zero value specifies to the basin outflow.

The sum of runoff and snowmelt runoff may be delayed using independent Clark routing objects before being handled as input to the main HRU Clark object.

Similarly the subsurface runoff may also be delayed using an independent Clark object before being handled as input to the main HRU Clark routing object.

The delayed runoff and subsurface runoff are added and applied to the common HRU Clark routing object.

The output flow of an HRU may be diverted to another HRU by using the *whereto* parameter. This flow is used to top up the depression storage (Sd in the Soil module) and the soil moisture recharge layer (soil_rechr in the Soil module) of the HRU during the current time step. Any surplus is returned to the soil module by the *put* variable redirected_residual and is added to the soil_runoff variable during the next time step. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

The groundwater flow from an HRU may be diverted to another HRU by using the 'gwwhereto' parameter. The gw flow is added to the input flow of the particular HRU after the recharge layer (soil_rechr in the Soil module) and the HRU free depressional storage (Sd in the Soil module) has been replenished. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

Observations

- none

Variables

- inflow (mm*km^2/int) - sum of the optional inflow from other HRUs and delayed surface and subsurface runoff from this HRU.
- cuminflow (mm*km^2) - HRU cumulative inflow.
- outflow (mm*km^2/int) - HRU outflow after Clark(Kstorage, Lag) delay.
- cumoutflow (mm*km^2) - HRU cumulative outflow.
- outflow_diverted (mm*km^2) - HRU outflow diverted to another HRU.
- cumoutflow_diverted (mm*km^2) - HRU cumulative outflow diverted to another HRU.
- cum_to_Sd (mm) - cumulative flow from other HRUs to the depression storage of this HRU.
- cum_to_soil_rechr (mm) - cumulative other HRU to soil_rechr of this HRU.
- ssrinflow (mm*km^2/int) - subsurface runoff (ssr) to this HRU.
- ssrcuminflow (mm*km^2) - cumulative ssrinflow.
- ssroutflow (mm*km^2/int) - ssr outflow after Clark(ssrKstorage, ssrLag) delay.
- ssrcumoutflow (mm*km^2) - cumulative ssr outflow.
- runinflow (mm*km^2/int) - sum of snowmelt runoff and runoff to this HRU.
- runcuminflow (mm*km^2) - cumulative runinflow.
- runoutflow (mm*km^2/int) - runoff outflow after Clark(runKstorage, runLag) delay.
- runcumoutflow (mm*km^2) - cumulative runoutflow.
- gwinflow (mm*km^2) - gw inflow.
- gwcuminflow (mm*km^2) - cumulative gwinflow.
- gwoutflow (mm*km^2/int) - gw outflow after Clark(gwKstorage, gwLag) delay.
- gwcumoutflow (mm*km^2) - cumulative gw outflow.
- gwoutflow_diverted (mm*km^2/int) - gw outflow diverted to other HRU.
- gwcumoutflow_diverted (mm*km^2/int) - cumulative gw outflow diverted to other HRU.
- basinflow (m^3/int) - average outflow rate of surface and subsurface runoff from the basin.
- basinflow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the basin.
- cumbasinflow (m^3) - cumulative basinflow.
- basingw (m^3/int) - average ouflow rate of groundwater from the basin.
- basingw_s (m^3/s) - average ouflow rate of groundwater from the basin.
- cumbasingw (m3) - cumulative basingw.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- Kstorage (d) - storage constant for HRU.
- Lag (h) - Lag for HRU.
- ssrKstorage (d) - storage constant for subsurface runoff to HRU routing.

- ssrLag (h) - Lag for subsurface runoff to HRU routing.
- runKstorage (d) - storage constant runoff to HRU routing.
- runLag (h) - Lag for runoff to HRU routing.
- whereto () - '0' for basin outflow or 1..n to be directed toanother HRU.
- gwwhereto () - 0 - basin gw (basingw), >0 other HRU surface input. <0 other -HRU gw input, or (< -HRUmax or > +HRUmax) - surface basin outflow (basinflow),
- order () - HRU routing process order.
- Sdmax (mm) - maximum depression storage.
- soil_rechar_max (mm) - soil recharge maximum.
- Sd_ByPass () - 0 - normal, 1 - Bypass Pond/Depressional storage (i.e. Sd). Also applies to redirected gw.
- soil_rechr_ByPass () - 0 - normal, 1 - Bypass recharge layer (i.e. soil_rechr). Also applies to redirected gw.
- gwKstorage (d) - storage constant for gw to HRU routing.
- gwLag (h) - Lag for gw to HRU routing.

Variable Inputs

- soil_ssr (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_runoff (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_gw (*) (mm/int) - from module Soil etc. Programmed as a daily value.
- Sd (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_rechr (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_moist (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- redirected_residual (*) (mm*km^2/int) - to module Soil etc. Programmed as a state *put* value.

Notes.

The redistribution of water between HRU's is handled by Netroute. The interval outflow from an HRU may be directed to another HRU or designated as basin flow using the whereto parameter. When directed to another HRU, any water is first used to top up the soil recharge layer if soil_rechr_ByPass is not enabled and then any surplus applied to Depression or Pond storage (Sd) if Sd_ByPass is not enabled and then the excess is added to the HRU's inflow. If the whereto parameter is 0, the outflow is accumulated in basin flow. The last HRU is basin flow, so whereto is always 0.

Netroute_D

This module defined in ClassNetroute_D, handles the routing of surface runoff, subsurface runoff and HRU routing using the lag and route method described by Clark(1945). Outflow from a HRU is calculated by lagging its inflow by the travel time through the HRU, then routing it through an amount of linear storage defined by the storage constant, K. The outflow from a HRU can be diverted to the inflow of other HRUs and to the basin outlet. The parameter 'distrib_Route' determines the outflow of an HRU to the other HRUs. The parameter 'distrib_Basin' determines the amount of outflow of an HRU to the the basin outflow.

The sum of runoff and snowmelt runoff may be delayed using independent Clark routing objects before being handled as input to the main HRU Clark object.

Similarly the subsurface runoff may also be delayed using an independent Clark object before being handled as input to the main HRU Clark routing object.

The delayed runoff and subsurface runoff are added and applied to the common HRU Clark routing object.

The output flow of an HRU may be diverted to another HRU by using the 'distrib_Route' parameter. This flow is used to top up the depression storage (Sd in the Soil module) and the soil moisture recharge layer (soil_rechr in the Soil module) of the HRUs during the current time step. Any surplus is returned to the soil module by the *put* variable redirected_residual and is added to the soil_runoff variable during the next time step. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

The groundwater flow from an HRU may be diverted to another HRU by using the 'gwwhereto' parameter. The gw flow is added to the input flow of the particular HRU after the recharge layer (soil_rechr in the Soil module) and the HRU free depressional storage (Sd in the Soil module) has been replenished. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

Observations

- **none**

Variables

- **inflow (mm*km^2/int) - sum of the optional inflow from other HRUs and delayed surface and subsurface runoff from this HRU.**
- **cuminflow (mm*km^2) - HRU cumulative inflow.**
- **outflow (mm*km^2/int) - HRU outflow after Clark(Kstorage, Lag) delay.**
- **cumoutflow (mm*km^2) - HRU cumulative outflow.**
- **cum_to_Sd (mm) - cumulative flow from other HRUs to the depression storage of this HRU.**
- **cum_to_rechr (mm) - cumulative flow from other HRUs to the soil_rechar of this HRU.**
- **ssrinflow (mm*km^2/int) - subsurface runoff (ssr) to this HRU.**
- **ssrcuminflow (mm*km^2) - cumulative ssrinflow.**
- **ssroutflow (mm*km^2/int) - ssr outflow after Clark(ssrKstorage, ssrLag) delay.**
- **ssrcumoutflow (mm*km^2) - cumulative ssr outflow.**
- **runinflow (mm*km^2/int) - sum of snowmelt runoff and runoff to this HRU.**
- **runcuminflow (mm*km^2) - cumulative runinflow.**
- **runoutflow (mm*km^2/int) - runoff outflow after Clark(runKstorage, runLag) delay.**
- **runcumoutflow (mm*km^2) - cumulative runoutflow.**
- **gwinflow (mm*km^2) - gw inflow.**
- **gwcuminflow (mm*km^2) - cumulative gwinflow.**
- **gwoutflow (mm*km^2/int) - gw outflow after Clark(gwKstorage, gwLag) delay.**
- **gwcumoutflow (mm*km^2) - cumulative gw outflow.**
- **gwoutflow_diverted (mm*km^2/int) - gw outflow diverted to other HRU.**
- **gwcumoutflow_diverted (mm*km^2/int) - cumulative gw outflow diverted to other HRU.**
- **cumscaling_boost (mm*km^2) - cumulative amout inflow boosted.**
- **basinflow (m^3/int) - average outflow rate of surface and subsurface runoff from the basin.**
- **basinflow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the basin.**
- **cumbasinflow (m^3) - cumulative basinflow.**
- **basingw (m^3/int) - average ouflow rate of groundwater from the basin.**
- **basingw_s (m^3/s) - average ouflow rate of groundwater from the basin.**
- **cumbasingw (m3) - cumulative basingw.**

Parameters

- **basin_area (km^2) - basin area.**
- **hru area (km^2) - HRU area.**
- **Kstorage (d) - storage constant for HRU.**
- **Lag (h) - Lag for HRU.**
- **ssrKstorage (d) - storage constant for subsurface runoff to HRU routing.**
- **ssrLag (h) - Lag for subsurface runoff to HRU routing.**
- **runKstorage (d) - storage constant runoff to HRU routing.**

- runLag (h) - Lag for runoff to HRU routing.
- distrib_Route () - route HRU flow array. None zero values indicate diversion from this HRU to another HRU.
- distrib_Basin () - HRU basin flow array. None zero values indicate diversion from this HRU to the basin flow.
- gwwhereto () - 0 - basin gw (basingw), >0 other HRU input. <0 other -HRU gw input, or (< -HRUmax or > +HRUmax) - basin outflow (basinflow),
- Sdmax (mm) - maximum depression storage.
- soil_rechr_max (mm) - maximum depression storage.
- Sd_ByPass () - 0 - normal, 1 - Bypass Pond/Depressional storage (i.e. Sd).
- soil_rechr_ByPass () - 0 - normal, 1 - Bypass recharge layer (i.e. soil_rechr).
- scaling_factor () - multiplies the inflow to Muskingum by this scaling factor. Experimental.

Variable Inputs

- soil_ssr (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_runoff (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_gw_D (*) (mm/d) - from module Soil etc. Programmed as a daily value.
- Sd (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_rechr (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_moist (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- redirected_residual (*) (mm*km^2/int) - to module Soil etc. Programmed as a state *put* value.

Notes.

The distrib_Route parameter if positive is used as is. However, if negative it is multiplied by the HRU area and made positive before being used. The last drainage HRU can only flow to the basin outlet. For the last drainage HRU the distrib_Route values must be zero..

The parameter distrib_Basin must always be zero or greater. If the last drainage HRU has distrib_Basin set to zero it is forced to one.

The redistribution of water between HRU's is handled by Netroute_D. The interval outflow from an HRU may be directed to another HRU or designated as basin flow using the distrib_Basin and distrib_Route parameters. When directed to another HRU, any flow is first used to top up the soil recharge layer (soil_rechr), if soil_rechr_ByPass is not set, then to top the Depression or Pond storage (Sd) if Sd_ByPass is not set and finally any excess is directed to the designated HRU's inflows. The absolute value of the parameter values determines the contribution to each destination. E.g, if the outflow of HRU 1 is Outflow1, then the diversion to HRU 2 would be N2*Outflow1/(B1 + N1 + N2 + N3 + N4 ...Nn), where B1 is the amount of HRU 1's output to basin flow and N1 to Nn the contribution of HRU1 to the other HRUs. Normally. Sinnce an HRU cannot recirculate its output to itself, in this example N1 would be zero.

Netroute_M

This module defined in ClassNetroute_M, handles the routing of surface runoff, subsurface runoff and HRU routing using Muskingum method. Outflow from a HRU is calculated by lagging its inflow by the travel time through the HRU, then routing it through an amount of linear storage defined by the storage constant, K. The outflow from a HRU can be diverted to the inflow of another HRU or directed to the basin outlet. The parameter 'order' ensures that the outflows from the various HRUs are calculated in the correct order from upstream to downstream. The parameter 'whereto' defines the destination of an HRU outflow. A non zero value indicates flow to another HRU and a zero value specifies to the basin outflow.

The sum of runoff and snowmelt runoff may be delayed using independent Clark routing objects before being handled as input to the main HRU Clark object.

Similarly the subsurface runoff may also be delayed using an independent Clark object before being handled as input to the main HRU Clark routing object.

The delayed runoff and subsurface runoff are added and applied to the common HRU Clark routing object.

The output flow of an HRU may be diverted to another HRU by using the *whereto* parameter. This flow is used to top up the depression storage (Sd in the Soil module) and the soil moisture recharge layer (soil_rechr in the Soil module) of the HRU during the current time step. Any surplus is returned to the soil module by the *put* variable redirected_residual and is added to the soil_runoff variable during the next time step. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

The groundwater flow from an HRU may be diverted to another HRU by using the 'gwwhereto' parameter. The gw flow is added to the input flow of the particular HRU after the recharge layer (soil_rechr in the Soil module) and the HRU free depressional storage (Sd in the Soil module) has been replenished. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

Observations

- none

Variables

- inflow (mm*km^2/int) - sum of the optional inflow from other HRUs and delayed surface and subsurface runoff from this HRU.
- cuminflow (mm*km^2) - HRU cumulative inflow.
- outflow (mm*km^2/int) - HRU outflow after Clark(Kstorage, Lag) delay.
- cumoutflow (mm*km^2) - HRU cumulative outflow.
- outflow_diverted (mm*km^2) - HRU outflow diverted to another HRU.
- cumoutflow_diverted (mm*km^2) - HRU cumulative outflow diverted to another HRU.
- cum_to_Sd (mm) - cumulative flow from other HRUs to the depression storage of this HRU.
- cum_to_soil_rechr (mm) - cumulative other HRU to soil_rechr of this HRU.
- ssrinflow (mm*km^2/int) - subsurface runoff (ssr) to this HRU.
- ssrcuminflow (mm*km^2) - cumulative ssrinflow.
- ssroutflow (mm*km^2/int) - ssr outflow after Clark(ssrKstorage, ssrLag) delay.
- ssrcumoutflow (mm*km^2) - cumulative ssr outflow.
- runinflow (mm*km^2/int) - sum of snowmelt runoff and runoff to this HRU.
- runcuminflow (mm*km^2) - cumulative runinflow.
- runoutflow (mm*km^2/int) - runoff outflow after Clark(runKstorage, runLag) delay.
- runcumoutflow (mm*km^2) - cumulative runoutflow.
- gwinflow (mm*km^2) - gw inflow.
- gwcuminflow (mm*km^2) - cumulative gwinflow.
- gwoutflow (mm*km^2/int) - gw outflow after Clark(gwKstorage, gwLag) delay.
- gwcumoutflow (mm*km^2) - cumulative gw outflow.
- gwoutflow_diverted (mm*km^2/int) - gw outflow diverted to other HRU.
- gwcumoutflow_diverted (mm*km^2/int) - cumulative gw outflow diverted to other HRU.
- basinflow (m^3/int) - average outflow rate of surface and subsurface runoff from the basin.
- basinflow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the basin.
- cumbasinflow (m^3) - cumulative basinflow.
- basingw (m^3/int) - average ouflow rate of groundwater from the basin.
- basingw_s (m^3/s) - average ouflow rate of groundwater from the basin.
- cumbasingw (m3) - cumulative basingw.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- Lag (h) - Lag for HRU.
- route_n () - Manning roughness coefficient.
- route_R (() -hydraulic radius.

- route_S0 (() - longitudinal channel slope.
- route_L (m) - routing length.
- route_X_M () - dimensionless weighting factor.
- ssrKstorage (d) - storage constant for subsurface runoff to HRU routing.
- ssrLag (h) - Lag for subsurface runoff to HRU routing.
- runKstorage (d) - storage constant runoff to HRU routing.
- runLag (h) - Lag for runoff to HRU routing.
- whereto () - '0' for basin outflow or 1..n to be directed toanother HRU.
- gwwhereto () - 0 - basin gw (basingw), >0 other HRU input. <0 other -HRU gw input, or (< -HRUmax or > +HRUmax) - basin outflow (basinflow),
- order () - HRU routing process order.
- Sdmax (mm) - maximum depression storage.
- soil_rechar_max (mm) - soil recharge maximum.
- Sd_ByPass () - 0 - normal, 1 - Bypass Pond/Depressional storage (i.e. Sd). Also applies to redirected gw.
- soil_rechr_ByPass () - 0 - normal, 1 - Bypass recharge layer (i.e. soil_rechr). Also applies to redirected gw.
- gwKstorage (d) - storage constant for gw to HRU routing.
- gwLag (h) - Lag for gw to HRU routing.

Variable Inputs

- soil_ssr (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_runoff (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_gw (*) (mm/int) - from module Soil etc. Programmed as a daily value.
- Sd (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_rechr (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_moist (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- redirected_residual (*) (mm*km^2/int) - to module Soil etc. Programmed as a state *put* value.

Notes.

The redistribution of water between HRU's is handled by Netroute. The interval outflow from an HRU may be directed to another HRU or designated as basin flow using the whereto parameter. When directed to another HRU, any water is first used to top up the soil recharge layer if soil_rechr_ByPass is not enabled and then any surplus applied to Depression or Pond storage (Sd) if Sd_ByPass is not enabled and then the excess is added to the HRU's inflow. If the whereto parameter is 0, the outflow is accumulated in basin flow. The last HRU is basin flow, so whereto is always 0.

Netroute_M_D

This module defined in ClassNetroute_M_D, handles the routing of surface runoff, subsurface runoff and HRU routing using the Muskingum method. Outflow from a HRU is calculated by lagging its inflow by the travel time through the HRU, then routing it through an amount of linear storage defined by the storage constant, K. The outflow from a HRU can be diverted to the inflow of other HRUs and to the basin outlet. The parameter 'distrib_Route' determines the outflow of an HRU to the other HRUs. The parameter 'distrib_Basin' determines the amount of outflow of an HRU to the the basin outflow.

The sum of runoff and snowmelt runoff may be delayed using independent Clark routing objects before being handled as input to the main HRU Clark object.

Similarly the subsurface runoff may also be delayed using an independent Clark object before being handled as input to the main HRU Clark routing object.

The delayed runoff and subsurface runoff are added and applied to the common HRU Clark routing object.

The output flow of an HRU may be diverted to another HRU by using the 'distrib_Route' parameter. This flow is used to top up the depression storage (Sd in the Soil module) and the soil moisture recharge layer (soil_rechr in the Soil module) of the HRUs during the current time step. Any surplus is returned to the soil module by the *put* variable redirected_residual and is added to the soil_runoff variable during the next time step. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

The groundwater flow from an HRU may be diverted to another HRU by using the 'gwwhereto' parameter. The gw flow is added to the input flow of the particular HRU after the recharge layer (soil_rechr in the Soil module) and the HRU free depressional storage (Sd in the Soil module) has been replenished. The parameters soil_rechr_ByPass and Sd_ByPass may be used to prevent replenishing the recharge and depressional storage.

Observations

- none

Variables

- inflow (mm*km^2/int) - sum of the optional inflow from other HRUs and delayed surface and subsurface runoff from this HRU.
- cuminflow (mm*km^2) - HRU cumulative inflow.
- outflow (mm*km^2/int) - HRU outflow after Clark(Kstorage, Lag) delay.
- cumoutflow (mm*km^2) - HRU cumulative outflow.
- cum_to_Sd (mm) - cumulative flow from other HRUs to the depression storage of this HRU.
- cum_to_rechr (mm) - cumulative flow from other HRUs to the soil_rechar of this HRU.
- ssrinflow (mm*km^2/int) - subsurface runoff (ssr) to this HRU.
- ssrcuminflow (mm*km^2) - cumulative ssrinflow.
- ssroutflow (mm*km^2/int) - ssr outflow after Clark(ssrKstorage, ssrLag) delay.
- ssrcumoutflow (mm*km^2) - cumulative ssr outflow.
- runinflow (mm*km^2/int) - sum of snowmelt runoff and runoff to this HRU.
- runcuminflow (mm*km^2) - cumulative runinflow.
- runoutflow (mm*km^2/int) - runoff outflow after Clark(runKstorage, runLag) delay.
- runcumoutflow (mm*km^2) - cumulative runoutflow.
- gwinflow (mm*km^2) - gw inflow.
- gwcuminflow (mm*km^2) - cumulative gwinflow.
- gwoutflow (mm*km^2/int) - gw outflow after Clark(gwKstorage, gwLag) delay.
- gwcumoutflow (mm*km^2) - cumulative gw outflow.
- gwoutflow_diverted (mm*km^2/int) - gw outflow diverted to other HRU.
- gwcumoutflow_diverted (mm*km^2/int) - cumulative gw outflow diverted to other HRU.
- cumscaling_boost (mm*km^2) - cumulative amout inflow boosted.
- basinflow (m^3/int) - average outflow rate of surface and subsurface runoff from the basin.
- basinflow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the basin.
- cumbasinflow (m^3) - cumulative basinflow.
- basingw (m^3/int) - average ouflow rate of groundwater from the basin.
- basingw_s (m^3/s) - average ouflow rate of groundwater from the basin.
- cumbasingw (m3) - cumulative basingw.

Parameters

- basin_area (km^2) - basin area.
- hru area (km^2) - HRU area.
- Lag (h) - Lag for HRU.
- route_n () - Manning roughness coefficient.
- route_R (() -hydraulic radius.
- route_S0 (() - longitudinal channel slope.
- route_L (m) - routing length.

- route_X_M () - dimensionless weighting factor.
- ssrKstorage (d) - storage constant for subsurface runoff to HRU routing.
- ssrLag (h) - Lag for subsurface runoff to HRU routing.
- runKstorage (d) - storage constant runoff to HRU routing.
- runLag (h) - Lag for runoff to HRU routing.
- distrib_Route () - route HRU flow array. None zero values indicate diversion from this HRU to another HRU.
- distrib_Basin () - HRU basin flow array. None zero values indicate diversion from this HRU to the basin flow.
- gwwhereto () - 0 - basin gw (basingw), >0 other HRU input. <0 other -HRU gw input, or (< -HRUmax or > +HRUmax) - basin outflow (basinflow),
- Sdmax (mm) - maximum depression storage.
- soil_rechr_max (mm) - maximum depression storage.
- Sd_ByPass () - 0 - normal, 1 - Bypass Pond/Depressional storage (i.e. Sd).
- soil_rechr_ByPass () - 0 - normal, 1 - Bypass recharge layer (i.e. soil_rechr).
- scaling_factor () - multiplies the inflow to Muskingum by this scaling factor. Experimental.

Variable Inputs

- soil_ssr (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_runoff (*) (mm/int) - from module Soil etc. Programmed as an interval or daily value.
- soil_gw_D (*) (mm/d) - from module Soil etc. Programmed as a daily value.
- Sd (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_rechr (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- soil_moist (*) (mm) - from module Soil etc. Programmed as a state *put* value.
- redirected_residual (*) (mm*km^2/int) - to module Soil etc. Programmed as a state *put* value.

Notes.

The distrib_Route parameter if positive is used as is. However, if negative it is multiplied by the HRU area and made positive before being used. The last drainage HRU can only flow to the basin outlet. For the last drainage HRU the distrib_Route values must be zero..

The parameter distrib_Basin must always be zero or greater. If the last drainage HRU has distrib_Basin set to zero it is forced to one.

The redistribution of water between HRU's is handled by Netroute_D. The interval outflow from an HRU may be directed to another HRU or designated as basin flow using the distrib_Basin and distrib_Route parameters. When directed to another HRU, any flow is first used to top up the soil recharge layer (soil_rechr), if soil_rechr_ByPass is not set, then to top the Depression or Pond storage (Sd) if Sd_ByPass is not set and finally any excess is directed to the designated HRU's inflows. The absolute value of the parameter values determines the contribution to each destination. E.g, if the outflow of HRU 1 is Outflow1, then the diversion to HRU 2 would be N2*Outflow1/(B1 + N1 + N2 + N3 + N4 ...Nn), where B1 is the amount of HRU 1's output to basin flow and N1 to Nn the contribution of HRU1 to the other HRUs. Normally. Sinnce an HRU cannot recirculate its output to itself, in this example N1 would be zero.

REW_route

This module defined in ClassREWroute, handles the routing of surface runoff, subsurface runoff from RBs (representative basins) and RB routing using the lag and route method described by Muskingum. Outflow from a RB is calculated by lagging its inflow by the travel time through the RB, then routing it through an amount of linear storage defined by the storage constant, K. The outflow from a RB can be diverted to the inflow of another RB or directed to the watershed outlet. The parameter 'order' ensures that the outflows from the various RBs are calculated in the correct order from upstream to downstream. The parameter 'whereto' defines the destination of an RB outflow. Anon zero value indicates flow to another RB and a zero value specifies to the watershed outflow.

The delayed runoff and subsurface runoff are added and applied to the common RB routing object.

The flow output of an RB may be diverted to another RB by using the *whereto* parameter. Normally, this flow is added to the input flow of the particular RB and does not become part of the internal hydro;ogy of the RB.

This routine handles the groundwater from all RBs in an identical manner.

Observations

- **none**

Variables

- **WS_inflow (m^3/int) - sum of the optional inflow from other RBs and delayed surface and subsurface runoff from the current RB before Muskingum delay.**
- **cum_WSinflow (m^3) - HRU cumulative inflow.**
- **WS_outflow (m^3/int) - HRU outflow after Muskingum delayed WS_inflow.**
- **cum_WSoutflow (m^3) - HRU cumulative outflow.**
- **WS_flow (m^3/int) - average outflow rate of surface and subsurface runoff from the watershed.**
- **WS_flow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the watershed.**
- **cum_WSflow (m^3) - cumulative watershed outflow.**

- **WS_gwinflow (m^3/int) - sum of the optional inflow from other RBs and delayed surface and subsurface runoff from the current RB before Muskingum delay.**
- **cum_WSgwinflow (m^3) - HRU cumulative inflow.**
- **WS_gwoutflow (m^3/int) - HRU outflow after Muskingum delayed WS_gwinflow.**
- **cum_WSgwoutflow (m^3) - HRU cumulative outflow.**
- **WS_gwflow (m^3/int) - average outflow rate of surface and subsurface runoff from the watershed.**
- **WS_gwflow_s (m^3/s) - average outflow rate of surface and subsurface runoff from the watershed.**
- **cum_WSgwflow (m^3) - cumulative watershed outflow.**

Parameters

- **watershed_area (km^2) - Sum of RB areas.**
- **RB_area (km^2) - RB areas.**
- **route_n () - Manning roughness coefficient.**
- **route_R () - hydraulic radius.**
- **route_L (m) - routing length.**
- **route_H (m) - elevation change over route)L**

- **WS_Kstorage (d) - storage constant for RB.**
- **WS_Lag (h) - Lag for RB.**
- **WS_X_M () - dimentionless weighting factor.**
- **WS_whereto () - '0' for basin outflow or 1..n to be directed toanother HRU.**
- **WS_order () - HRU routing process order.**

- **WS_gwKstorage (d) - storage constant for RB.**
- **WS_gwLag (h) - Lag for RB.**
- **WS_gwX_M () - dimentionless weighting factor.**
- **WS_gwwhereto () - '0' for basin outflow or 1..n to be directed toanother HRU.**
- **WS_gworder () - HRU routing process order.**

Variable Inputs

- **inflow_All (*) (mm/int) - "basinflow" variables from all groups. Programmed as an interval or daily value. Associated variable "rew".**
- **gw_All (*) (mm/int) - "basingw" variables from all groups. Programmed as an interval or daily value. Associated variable "gwrew".**

Notes.

The variable inputs above are generated by a special call to " cnt = declgrpvar(Variable name, "search variable", "help information", "units", var, var_data)". The name given as "search variable" is searched for in all model groups. The variable "cnt" gives the number of occurrences found. The one dimensional variable "var" gives the group index of the occurrences found. Group A - 1, B - 2, etc. The two dimensional [RBindex][HRU], variable"var_data" gives the interval data. In the case of "basinflow" and "basingw" which are dimensioned as "basin", [HRU] is always [0].

Muskingum Routing.

The following is calculated from each RB's parameter and used for the Muskingum routing.

Vw[3] = {1.67, 1.44, 1.33}; // rectangular - 0/parabolic - 1/triangular - 2

WS_Kstorage = route_L/(Vw[route_Cshp]*Vavg) and

WS_gwKstorage = route_L/(Vw[route_Cshp]*Vavg) where

Vavg = (1/route_n)*pow(route_R, 2/3)*pow(route_H/route_L, 0.5)*86400.

Demo Project.

watershed.prj

Qmelt

This module defined in Classqmelt to calculate the ground flux for hummuck-covered hillslopes in the Artic tundra.

Observations

- **t(°C) - average air temperature during time step.**
- **ts(°C) - average surface/skin temperature during time step.**

Function Observations

- **tmean (°C) - mean daily air temperature.**
- **tpos (°C) - sum of interval air temperatures above 0°C during a day.**
- **tsmean (°C) - mean daily surface/skin temperature.**
- **tspos (°C) - sum of interval surface/skin temperatures above 0°C during a day.**

Variables

- **Qm (MJ/m2 ∆t) - snow melt during the time step, calculated using the mean daily air temperature.**
- **cum_Qm (MJ/m2 ∆t) - cumulative snow melt (from daily mean air temperature) over time.**
- **Qg (MJ/m2) -ground heat flux during the time step, calculated using the mean daily surface/skin temperature.**
- **cum_Qg (MJ/m2) - cumulative ground heat flux (from daily mean surface/skin temperature) over time.**

Parameters

- **Tla0Mn (°C) - offset in temperature - index model in calculating QmD.**
- **Tla1Mn (MJ/day/°C) - coefficient for estimating QmD.**
- **Tls0Mn (°C) - offset in temperature - index model in calculating QgD.**
- **Tls1Mn (MJ/day/°C) - coefficient for estimating QgD.**

Variable Inputs

- **none**

Notes

Snow melt from mean daily air temperature .

1. **if (tmean - Tla0Mn > 0.0) QmD = (tmean - Tla0Mn)*Tla1Mn,**
2. **if time step t > 0.0 Qm = t*QmD/tpos else Qm = 0.0.**
3. **cum_Qm = cum_Qm + Qm.**

Ground heat flux from mean daily surface/skin temperature

1. **if (tsmean - Tls0Mn > 0.0) QgD = (tsmean - Tls0Mn)*Tls1Mn,**
2. **if time step ts > 0.0 Qg = ts*QgD/tspos else Qg = 0.0.**
3. **cum_Qg = cum_Qg + Qg.**

Quinton

This module defined in Classquinton handles the subsurface drainage from hummock-covered hillslopes in the Arctic tundra. An HRU is a one-metre wide strip of land having defined hydrological properties that is aligned perpendicular to a selected channel. All measurements of depth are referenced using the ground surface as the datum unless otherwise stated and taken positive.

Observations

- **p (mm Δt) – depth of precipitation (rainfall) received at the ground surface in time step.**

Variables

- theta (m3/m3) - volumetric soil moisture.
- layerwater (m) - depth of liquid water in layer. It consists of melt, precipitation and inflow received from other HRUs.
- dmelt (m) - depth to the frost table.
- wdrained (m) – depth to the frozen saturated layer when dmelt is in the frozen unsaturated layer above. Otherwise 0.0.
- Depth (m) – height of liquid water above the frost table.
- watertable (m) - depth from the surface to the surface of the water table. Is equal to (dmelt - Depth).
- d_surface (m) - depth from the surface to the middle of the liquid water layer. This depth is used to determine the average horizontal hydraulic conductivity. Is equal to (dmelt - Depth/2).
- k (m/day) - horizontal hydraulic conductivity.
- flow (m $_{Dt}$) – volume of subsurface flow per unit plan area of hummock HRU in a time step – expressed as an average depth of water (m3/m2)
- flowm3 (m3) – volume of subsurface flow from hummock HRU in time step.
- cumflow (m) - cumulative volume of subsurface flow per unit plan area of hummock HRU – expressed as an average depth of water (m3/m2).
- runoff (m $_{Dt}$) - volume of surface runoff per unit plan area of hummock HRU in a time step – expressed as an average depth of water (m3/m2).
- runoffm3 (m3 $_{Dt}$) - volume of surface runoff from hummock HRU in time step.
- cumrunoff(m/m2) – cumulative volume of surface runoff per unit plan area of hummock HRU – expressed as an average depth of water (m3/m2).
- flowin (m $_{Dt}$) - volume of inflow from external sources (melt, drainage, other) entering an unit plan area of a hummock HRU in a time step – expressed as an average depth of water (m3/m2).
- flowinm3 (m3 $_{Dt}$) - volume of inflow from external sources (melt, drainage, other) entering a hummock HRU in a time step.
- cumflowin (m/m2) - cumulative volume of inflow from external sources (melt, drainage, other) entering an unit plan area of a hummock HRU – expressed as an average depth of water (m3/m2).
- loss (m $_{Dt}$) – loss of water from all layers of hummock HRU per unit plan area in a time step– expressed as an average depth of water (m3/m2) over the time step.
- capillary (m) –depth of the capilary water in the layer.
- tension (m) – head of water in the layer.
- transit (day) - time for a particle of water to traverse an HRU. It is equal to the length of the HRU.
- cump(mm) - total rainfall.
- Cvis (J/m3/K) - heat capacity of layer saturated, frozen condition.

- Cvisa (J/m3/K) - heat capacity of layer in unsaturated, frozen condition.
- Cvws (J/m3/K) - heat capacity of layer in saturated, unfrozen condition.
- Cvwsa (J/m3/K) - heat capacity of layer in unsaturated, unfrozen condition.
- lamis (W/m/K) - thermal conductivity of layer in saturated, frozen condition.
- lamws (W/m/K) - thermal conductivity of layer in saturated, unfrozen condition.
- lamwsa (W/m/K) - thermal conductivity of layer in saturated, unfrozen condition

Parameters

- Type () - HRU land type, 0=NOTUSED, 1=DRIFT, 2=HUMMOCK.
- length (m) - length of HRU. HRU has unit width.
- K_btm (m/d) - horizontal bottom hydraulic conductivity.
- K_top (m/d) - horizontal top hydraulic conductivity.
- ztrn (m) - transition depth.
- DrainTo () - If drift HRU, the Hummock HRU drained to, 0=NOWHERE or HRU# (1 to MAXHRU).
- soil_type() - 0=organic1, 1=organic2, 2=organic3, 3=sand, 4=clay.
- tinit (°C) - initial layer temperatures.
- slope (m/m) - HRU average slope.
- Residual () - organic non-drainable porosity. Also referred to as specific yield.
- d (m) - soil layer thickness.
- Drained (m) - depth of water table, i.e. saturated organic material (ice or water).
- FrozenTo (m) - initial depth of frost table.
- flowLag (hours) - lag inserted into flow out.
- flowstorage (days) - storage inserted into flow out.
- runoffLag (hours) - lag inserted into runoff.
- runoffstorage (days) - storage inserted into runoff.
- catchadjust () - precipitation calibration.
- Pors (m3/m3) - used to redefine the porosity a soil type if the value is greater than zero. Affects all HRU's using this soil type.
- n () - an empirical constant used in the Van Genuchten calculation. If zero Van Genuchten is not used to determine the soil tension.
- a () - an empirical constant used in the Van Genuchten calculation.

Variable Inputs

- Qg (MJ/m2/Δt) - surface ground flux from module **Qmelt**.
- driftmelt (m3/Δt) - drift melt and precipitation from module **Qdrift**.

Heat capacity of layer soil/organic matter:

frozen/saturated

Cvis = Cv_i*por_s[soil_type_lay] + Cv_s[soil_type_lay]*(1.0-por_s[soil_type_lay]), (J/kg/K).

frozen/unsaturated - drained to the irreducible water content

Cvisa = Cv_i*por_s[soil_type_lay]*Residual + Cv_s[soil_type_lay]*(1.0-por_s[soil_type_lay]) + Cv_a*(por_s[soil_type_lay] - Residual), (J/kg/K).

unfrozen/saturated

Cvws = Cv_w*por_s[soil_type_lay]) + Cv_s[soil_type_lay]*(1.0-por_s[soil_type_lay]), (J/kg/K).

unfrozen/unsaturated - drained to the irreducible water content

Cvwsa = Cv_w*Residual *por_s[soil_type_lay] + Cv_s[soil_type_lay]*(1.0-por_s[soil_type_lay]) + Cv_a*(por_s[soil_type_lay] - Residual), (J/kg/K).

Thermal capacity of layer soil/organic matter:

Xs = 1.0 - por_s[soil_type_lay]

Xw = por_s[soil_type_lay]-Residual

Xa = 1.0 - Xs - Xw

n = por_s[soil_type_lay]

if(Xw >= 0.09)

ga = 0.333-Xa/n*(0.333-0.035)

else

ga = 0.013 + 0.944*Xw

gc = 1.0 - 2.0*ga

Fs = 1.0/3.0*(2.0/(1 + (ks_s[soil_type_lay]/lam_w-1.0)*0.125)+ (1.0/((1 + (ks_s[soil_type_lay]/lam_w-1.0)*0.75))))

Fa = 1.0/3.0*(2.0/(1 + (lam_a/lam_w-1.0)*ga) + (1.0/((1 + (lam_a/lam_w-1.0)*gc))))

a = Farouki_a(por_s[soil_type_lay]) solution of the relationship: fractional porosity = $3a^2 - 2a^3$

frozen/saturated

lamis_lay = lam_i*a*a + lam_s[soil_type_lay]*sqr(1.0-a) + lam_s[soil_type_lay]*lam_i*(2*a-2*sqr(a))/(lam_s[soil_type_lay]*a + lam_i*(1.0-a))

unfrozen/saturated

lamws_lay = lam_w*a*a + lam_s[soil_type_lay]*sqr(1.0-a) + lam_s[soil_type_lay]*lam_w*(2*a-2*sqr(a))/(lam_s[soil_type_lay]*a+lam_w*(1.0-a))

unfrozen/unsaturated

lamwsa_lay = (Xw*lam_w + Fa*Xa*lam_a +Fs*Xs*lam_s[soil_type_lay])/(Xw + Fa*Xa + Fs*Xs)

Calculation of melt

Latent heat of fusion, Hf = 334.4E+3 (J/kg).

frozen/saturated

melt = Qm/(-tlayer_lay *Cvis + por_s[soil_type_lay]*Hf)/1000 (m),

frozen/unsaturated - drained to the irreducible water content

melt = Qm/(-tlayer_lay *Cvisa + Residual*Hf)/1000 (m)

Assuming drainage is controlled by the Van Genuchten estimation.

It is assumed that when the soil tension is less than the bubbling pressure that the soil drains by gravity. Otherwise is controlled by:

$\theta_f = (\phi - \theta_r)*(2^n)^{-m} + \theta_r$ where $\psi_b = 1/\alpha$. Refer to capillary for additional information.

For the unfrozen depth of a soil layer and the liquid water in the thawed layer the actual soil moisture is calculated. If $\theta >= \theta_f$, all of the melt is transferred to the variable, layer_water_lay otherwise, the amount transferred is determined by the soil tension and the Van Genuchten estimation.

excess = capillary_lay - θ_f * d_lay /por_s

Assuming all melt drains by gravity

layerwater_lay = layerwater_lay + melt distributed over applicable layers.

Depth of free liquid water in organic layers;

Depth = Sum((layerwater_lay)) (m).

Depth from surface of the water table;

d_surface = dmelt - Depth/2.0 (m).

Calculation of Horizontal Hydraulic Conductivity.

Three power series were fitted to the horizontal hydraulic conductivity versus depth data collected at Siksik, Granger Basin and Scotty Creek sites. One series used all the data and the other two power series were created using only the data greater than the first power series trendline and the second series used only the values less than the first series trendline. These two power series were created to simulate the two extremes, i.e. minimum and maximum possible horizontal hydraulic conductivities.

The continuous function for hydraulic conductivity k are:

$\log(K(z)) = \log(Kbtm) - (\log(Ktop) - \log(Kbtm))/(1 + (z/ztrn)^n)$

Calculation of Horizontal Flow.

lossD = k*Depth*slope (m/day/m2).

loss = k*Depth*fall/length/Freq (m/m2/int). Where FREQ is the number of intervals/day.

Water balance.

The horizontal drainage is taken from the uppermost layerwater first and any left is taken from the next lower layer.

layerwater_lay = layerwater_lay - Loss/(por_s[soil_type_lay]*(1-Residual)) (m)

The out flow is calculated from the losses in the layerwater layers;

flow = Sum(Loss) (m).

N.B. units of (m/m2) are used for convenience.

Replenishment from snow drift melt;

flowin = driftmelt*length_drift_hru/length_hummock_hru (m) or from a hummock HRU is

flowin = flow*length_contributing_hummock_hru/length_hummock_hru.

layer maximum replenish depth:

maxdepth = dmelt - top_of_layer,

maximum replenish water (m),

maxfree = maxdepth*por[soil_type_lay]*(1.0 - replenish),

surplus after satisfying maxfree for melted layers is runoff (m).

Calculation of theta for each layer.

layerwater_ht = sum(layerwater_lay/por[soil_type_lay] (m),

if(dmelt - layerwater_ht) <= top_of_layer then

theta = 1.0

else if(dmelt - layerwater_ht) > (top_of_layer + layer_depth) then

theta = residual

else

theta = (depth_w +(layer_depth - depth_w)*residual)/layer_depth,

where depth_w = top_of_layer + layer_depth - (dmelt - free_water_ht/por[soil_type_lay] (m).

Constants

g = 9.81 (m/s2)

visc_w = 0.0018 at 0°C (N.s/m2) (kg/(m.s) water

rho_a = 1.2 (kg/m3) air

rho_i = 920.0 (kg/m3) ice

rho_w = 1000.0 (kg/m3) water

c_a = 1010.0 (J/kg/K) air

c_i = 2120.0 (J/kg/K) ice

c_w = 4185.0 (J/kg/K) water

Cv_a = 1212.0 (J/m3/K) air

Cv_i = 1950400.0 (J/m3/K) ice

Cv_w = 4185000.0 (J/m3/K) water

lam_a = 0.025 (W/m/K) air

lam_i = 2.24 (W/m/K) ice

lam_w = 0.57 (W/m/K) water

order[LOAM1, LOAM2, LOAM3, SAND, CLAY]

rho_s[] = { 41.1, 75.2, 91.4, 1300.0, 1300.0} (kg/m3) density

c_s[] = {1920.0, 1920.0, 1920.0, 890.0, 890.0} (J/m3/K) specific heat

Cv_s[] = {78912.0, 144384.0, 175392.0, 1157000.0, 1157000.0} (J/m3/K) heat capacity

lam_s[] = {0.21, 0.21, 0.21, 2.50, 2.50} (W/m/K) thermal conductivity

ks_s[] = {450.0, 154.0, 13.0, 5.0, 3.0} (m/day) hydraulic conductivity

por_s[] = {0.96, 0.9, 0.87, 0.43, 0.43, } () porosity

Notes on HRUs.

The Quinton modules are not areal models but written to handle a one-metre wide strip of land that is aligned perpendicular to a selected channel The strip is made up of , each having a distinct personality that is determined by the parameters assigned to it. At present there are two types of HRUs', **drift** and **hummock**. Any number of **drift** and **hummock** HRUs can be included in a model. **Drift** HRUs must drain into **hummock** HRUs, not into another **drift** HRU. Multiple **drift** HRUs can drain into one **hummock** HRU. **Drift** HRUs must be defined before the **hummock** HRUs they drain into. **Hummock** HRUs may be cascaded, draining into one another before finally draining into the creek. At present it is assumed that if an areal water balance is required for a basin, the ouput of the CRHM model will be further processed in a spreadsheet.

Water Balance.

To confirm proper operation of the modules a water balance was added to the Quinton and Qdrift modules. When the model is initialised at the beginning of a model run the water content defined in the initial conditions is tabulated in the Log/Debug Output window. Similarly at the completion of the model run the final water content is tabulated. An example of the results of a model run follow.

The outputs from the HRUs are given twice. Once, as the volume (m3) from the HRU given that the area is equal to the length of the HRU times one metre width and secondly as the equivalent water depth over the area of the HRU.

02 05 19 00 30 Initial

HRU 1: (Drift) - water content (m3) (m/m2): 5.40 0.200

HRU 2: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 3: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 4: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 5: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 6: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 7: (Hummock) - water content (m3) (m/m2): 4.16 0.297

HRU 8: (Hummock) - water content (m3) (m/m2): 4.16 0.297

02 08 01 00 00 Final

HRU 1: (Drift) - water content (m3) (m/m2): 0.00 0.000

HRU 1: (Drift) - total precip (m3) (mm/m2): 0.00 0.000

HRU 1: (Drift) - water storage (m3) (m/m2): 0.00 0.000

HRU 2: (Hummock) - water content (m3) (m/m2): 3.11 0.222

HRU 2: (Hummock) - cumulative flowin (m3) (m/m2): 5.40 0.386

HRU 2: (Hummock) - cumulative precip (m3) (m/m2): 0.00 0.000

HRU 2: (Hummock) - cumulative flowout (m3) (m/m2): 3.90 0.278

HRU 2: (Hummock) - flowout in storage (m3) (m/m2): 0.00 0.000

HRU 2: (Hummock) - cumulative runoff (m3) (m/m2): 2.55 0.182

HRU 2: (Hummock) - runoff in storage (m3) (m/m2): 0.00 0.000

HRU 3: (Hummock) - water content (m3) (m/m2): 3.11 0.222

HRU 3: (Hummock) - cumulative flowin (m3) (m/m2): 6.45 0.460

HRU 3: (Hummock) - cumulative precip (m3) (m/m2): 0.00 0.000

HRU 3: (Hummock) - cumulative flowout (m3) (m/m2): 5.43 0.388

HRU 3: (Hummock) - flowout in storage (m3) (m/m2): 0.00 0.000

HRU 3: (Hummock) - cumulative runoff (m3) (m/m2): 2.06 0.147

HRU 3: (Hummock) - runoff in storage (m3) (m/m2): 0.00 0.000

Processing of parameters *Drained* and *FrozenTo*.

When the frost table is below the water table the initialization consists of moving the water contained between the two tables into the variable layerwater_lay**???** for the applicable layers. When the (frozen) water table is below the frost table the quantity of heat required to melt the frozen soil is less as instead of saturated s. **Don't understand?? Do you mean?** When the water table falls below the frost table the quantity of heat required to melt the drained soil is less because its' moisture content is set to the residual value

Derived from Letts et al. (2000).

	Fibric	Hemic	Sapric
	1.7 *	2.0 *	1.0 *

k_s (m/s) - hydraulic conductivity.	10^{-4}	10^{-6}	10^{-7}
θ_P (m^3/m^3) - porosity.	0.93	0.88	0.83
θ_{LIM} (m^3/m^3) - residual water content.	0.04	0.15	0.22
S_y (m^3/m^3) - specific yield.	0.5	0.3	
ψ_S (cm) - suction at saturation.	1.03	1.02	1.01
h () - soil texture exponent.	2.7	6.1	12

Qdrift

This module defined in Classqdrift calculates the amount of runoff that originates from melting of deep snow drifts on hummock-covered hillslopes in the Arctic tundra. An HRU is a one-metre wide strip of land having defined hydrological properties that is aligned perpendicular to a selected channel.

Observations

- **p (mm ∆t) - precipitation. Assumed to be rainfall.**

Variables

- **SWE (mm) - mean water equivalent of snowcover.**
- **driftmelt (m3 ∆t) - volume of snowmelt released by the drift in the time step.**
- **driftmeltD (m3) - daily snow melt from drift.**
- **cumdriftmelt (m3) - cumulative snow melt from drift.**

Parameters

- **Type () - HRU land type, 0=NOTUSED/1=DRIFT/2=HUMMOCK.**
- **length (m) - length of HRU.**
- **DrainTo () - the Hummock HRU drained to, 0=NOWHERE or HRU# (1 to MAXHRU).**
- **InitSWE (mm) - initial mean snow water equivalent. Assumes drift has a triangular distribution in depth (x-section) oriented normal to creek. All exposed area occurs on the down slope edge.**
- **meltLag (hours) - lag inserted into driftmelt.**
- **meltstorage (days) - storage inserted into driftmelt.**
- **catchadjust () - precipitation calibration.**

Variable Inputs

- **Qm (MJ/m2 ∆t) - snowmelt from module Qmelt in a time step ∆t.**
- **cump (mm) - cumulative rainfall. Is a *Put* variable.**

Notes

The snow is assumed to be distributed over the length of the HRU as a triangular distribution. The altitude of the triangle is twice the initial average snow water equivalent and the base the length of the HRU. As the snow melts, the triangle keeps the same proportions with all the melt occurring normal to the valley bottom, i.e. as the SWE depth decreases the snow covered base decreases in proportion.

To avoid round off errors the change in SWE is handled as follows: Note that the dimension of length (l) of the drift HRU is in the units.

```
float lastcumdriftmelt = cumdriftmelt[hh]; // following avoids round off error

cumdriftmelt[hh] = length[hh]*InitSWE[hh]*(1.0 - sqr(SWE[hh]/InitSWE[hh])); // (m3*1E3)

driftmelt[hh] = (cumdriftmelt[hh] - lastcumdriftmelt)/1E3; // (m3/Interval)
```

Creating a Macro to dynamically change parameters at model run time.

Normally parameters have an intial value which is held constant throughout a model run. This is the correct behaviour for parameters like elevation, area etc. However, vegetation height is a variable during the growing season. It should be handled as a variable. This is easily accomplished using a *macro*. Two examples follow. The first illustrates how the vegetation height may read from an observation file and the second if the height versus time relationship is known.

1) Using an observation file holding vegetation heights.

The following macro reads the observation *Heights* and writes it every interval to the parameter *Ht*. The variable Ht_Var was created so that the heights could be displayed as there is no method of charting parameters in CRHM (assumed to be constants).

```
Ht_file
declreadobs,Heights, NOBS, "Vegetation height", (m)
declparam, Ht, NHRU,0.1, 0.01, 10.0, description, (m)
declvar, Ht_Var, NHRU, "vegetation height", (m)
command
  Ht[hh] = Heights[hh]
  Ht_Var[hh] = Ht[hh]
end
```

2) Programming vegetation heights.

The following macro is written assuming the crop grows continuously from a Julian date to a maximum value.

```
Ht_Change
declparam, Ht, NHRU,0.1, 0.01, 100.0, description, (m)
declparam, Ht_change, NHRU,0.1, 0.01, 1.0, description, (m)
declparam, Ht_min, NHRU,0.1, 0.01, 3.0, description, (m)
declparam, Ht_max, NHRU,0.1, 0.01, 3.0, description, (m)
declparam, Ht_Julian, NHRU,195, 1, 366, description, ()
declvar, Ht_Var, NHRU, "vegetation height", (m)
declvar, Ht_Start, NHRU, "vegetation height", ()
command
  if(STEP == 1)
    Ht[hh] = Ht_min[hh]
  endif
  if(JULIAN >= Ht_Julian)
    Ht_Start[hh] = JULIAN
    if(LASTINT)
      Ht[hh] = Ht[hh] + Ht_change[hh]
    endif
    if( Ht[hh] > Ht_max[hh])
      Ht[hh] = Ht_max[hh]
    endif
  endif
  Ht_Var[hh] = Ht[hh]
end
```

Using the Height change macro modules.

After the macro code is added to the editor answer *YES* to the query asking if the modules should be loaded. The modules must be next added to the current model using the normal BUILD/CONSTRUCT screen. After adding the height macro module and during the build process, CRHM will ask if the module should be deleted, answer *NO*. This is necessary as CRHM is not able to determine ihow the module is used.

Units.

All observation, parameters and variables have assigned units. These are checked at run-time and any errors logged.

A sample observation file header follows.

Observation header file format

t 1 (°C) comment. e.g. air temperature.

rh 1 (%) comment. e.g. relative humidity.

u 1 (m/s) comment etc.

SunAct 1 (h)

ppt 1 (mm/d)

Qsi 1 (W/m^2)

Qso 1 (W/m^2)

Qn 1 (W/m^2)

$Qsi mul(Qsi\t 277.8) unit conversion

$Qso mul(Qso\t 277.8)

$Qn mul(Qn\t 277.8)

$ea ea(t\t rh) (kPa) vapour pressure

Note that there is one or more spaces or tabs between the number of observations and the units and after. The remainder of the line is treated as a comment.

Units must be enclosed in parentheses.

Case is significant in all units and multipliers..

° (degree) is not required but may be constructed using ALT key and Num Lock and 0176 or 0186 entered on the numeric key pad.

%(percentage) is ignored.

Multipliers.

Y	Yotta	1e+24
Z	Zetta	1e+21
E	Exa	1e+18
P	Peta	1e+15
T	Tera	1e+12
G	Giga	1e+9
M	Mega	1e+6
k	kilo	1e+3
K	Kilo	1e+3
h	hecto	1e+2
H	Hecto	1e+2
D	Deka	1e+1
d	deci	1e-1
c	cent	1e-2
m	mili	1e-3
u	micro	1e-6
µ	micro	1e-6
n	nano	1e-9
p	pico	1e-12

f	femto	1e-15
a	atto	1e-18
z	zepto	1e-21
y	yocto	1e-24

Units.

A	Ampere	1	
Bq	becquerel	1	l/s
Btu	InternationalTableBtu	1055.05585262	Kg*m^2/s^2
C	Coulomb	1	A*s
Ci	Curie	3.7*1010	l/s
F	Farad	1	A^2*s^4/Kg*m^2
Fdy	Faraday	96487	A*s
Gy	Gray	1	m^2/s^2
H	Henry	1	Kg*m^2/A^2*s^2
Hz	Hertz	1	s^-1
J	Joule	1	Kg*m^2/s^2
K	Kelvin	1	"),
Knot	NauticalMilesPerHour	463.0/900.0	m/s
N	Newton	1	Kg*m/s^2
Ohm	Ohm	1	Kg*m^2/A^2*s^3
OzUK	UKFluidOunce	2.8413075e-5	m^3
P	Poise	0.1	Kg/m*s
Pa	Pascal	1	Kg/m*s
Pdl	Poundal	0.13825495376	Kg*m/s^2
Pica	Pica	1.0/72.0	in
R	Roentgen	0.000258	A*s/Kg
S	Siemens	1	A^2*s^3/Kg*m^2
St	Stokes	0.0001	m^2/s
Sv	Sievert	1	m^2/s^2
T	Tesla	1	Kg/A*s^2
U	UnifiedAtomicMass	1.6605402e-27	Kg
V	Volt	1	Kg*m^2/A*s^2
W	Watt	1	Kg*m^2/s^3
Wb	Weber	1	Kg*m^2/A*s^2
acre	Acre	4046.87260987	m^2
arcmin	ArcMinute	2.9088820866e-4	r
arcs	ArcSecond	4.848136111e-6	r
atm	Attmosphere	101325	Kg/m*s^2
au	AstronomicalUnit	1.495979*1011	m
b	Barn	1e-28	m^2
bar	Bar	100000	Kg/m*s^2
bbl	Barrel	0.158987294928	m^3
bu	Bushel	0.03523907	m^3
c	LightSpeed	299792458	m/s
cal	Calorie	4.1868	Kg*m^2/s^2
cd	Candela	1	
chain	Chain	20.1168402337	m
ct	Carat	0.0002	Kg
cu	USCup	2.365882365e-4	m^3
d	day	86400	s
dyn	Dyne	0.00001	Kg*m/s^2
eV	ElectronVolt	1.60217733e-19	Kg*m^2/s^2
erg	Erg	0.0000001	Kg*m^2/s^2

fath	Fathom	1.82880365761	m
fbm	BoardFoot	0.002359737216	m^3
fc	Footcandle	10.7639104167	cd*sr/m^2
fermi	Fermi	1e-15	m
flam	Footlambert	3.42625909964	cd/m^2
ft	InternationalFoot	0.3048	m
ftUS	SurveyFoot	0.304800609601	m
g	Gram	1	
ga	StandardFreefall	9.80665	m/s^2
gal	USGallon	0.003785411784	m^3
galC	CanadianGallon	0.00454609	m^3
galUK	UKGallon	0.004546092	m^3
gf	GramForce	0.00980665	Kg*m/s^2
grad	Grade	1.57079632679e-2	r
grain	Grain	0.00006479891	Kg
h	Hour	3600	s
ha	Hectare	10000	m^2
hp	horsepower	745.699871582	Kg*m^2/s^2
in	Inch	0.0254	m
inH2O	InchesOfWater	248.84	Kg/m*s^2
inHg	InchesOfMercury	3386.38815789	Kg/m*s^2
int	CRHM time step	3600	s
kip	KilopoundForce	4448.22161526	Kg*m/s^2
kph	KilometersPerHour	5.0/18.0	m/s
l	Liter	0.001	m^3
lam	Lambert	3183.09886184	cd/m^2
lb	AvoirdupoisPound	0.45359267	Kg
lbf	PoundForce	4.44822161526	Kg*m/s^2
lbt	TroyPound	0.3732417216	Kg
lm	Lumen	1	cd*sr
lx	Lux	1	cd*sr/m^2
lyr	LightYear	9.46052840488*1015	m
m	Meter	1	
mho	Mho	1	A^2*s^2/Kg*m^2
mi	InternationalMile	1609.344	m
miUS	USStatuteMile	1609.34721869	m
mil	Mil	0.0000254	m
min	Minute	60	s
mmHg	MilimeterOfMercury	133.322368421	Kg/m*s^2
mol	Mole	1	
mph	MilesPerHour	0.44704	m/s
nmi	NauticalMile	1852	m
oz	Ounce	0.028349523125	Kg
ozfl	USFluidOunce	2.95735295625e-5	m^3
ozt	TroyOunce	0.0311034768	Kg
pc	Parsec	3.08567818585106	m
ph	Phot	10000	cd*sr/m^2
pk	Peck	0.0088097675	m^3
psi	PoundsPerSquareInch	6894.75729317	Kg/m*s^2
pt	Pint	0.000473176473	m^3
qt	Quart	0.000946352946	m^3
r	Radian	1	
rad	Rad	0.01	m^2/s^2
rd	Rod	5.02921005842	m
rem	Rem	0.01	m^2/s^2

s	Second	1	
sb	Stilb	10000	cd/m^2
slug	Slug	14.5939029372	Kg
sr	Steradian	1	
st	Stere	1	m^3
t	MetricTon	1000	Kg
tbsp	Tablespoon	1.47867647813e-5	m^3
therm	EECTherm	105506000	Kg*m^2/s^2
ton	ShortTon	907.18474	Kg
tonUK	UKLongTon	1016.0469088	Kg
torr	Torr	133.322368421	Kg/m^2
tsp	Teaspoon	4.92892159375e-6	m^3
yd	InternationalYard	0.9144	m
yr	Year	31556925.9747	s
°	Degree	1.74532925199e-2	r
°C	DegreeCelsius	1.0	K
°F	DegreesFahrenheit	1.0/1.8	K
°R	DegreesRankine	1.0/1.8	K
µ	Micron	1e-6	m
Å	Angstrom	1e-10	m

Table of Contents

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. Alocal 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). The parameter HRU_OBS can be declared in the macro to allow CRHM to access the correct observation but it is easier to omit this parameter and the default of 1, 2, 3 .. will be used..
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].

decllocal, 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.

decllocal, 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,

declputparameter, 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. Atypical 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. Amacro 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@Awill 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@Ais reduced to *.hru_t before searching. *.hru_t@Awill 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@Ais 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.

Changes to Modules for use in Groups and Structures.

Two changes are necessary to allow CRHM to handle modules in Groups and Stuctures.

Addition to the header (*.h) file of the module,

```
ClassOurClass* klone(string name) const; // where ClassOurClass is your class name.
```

and the addition to the source code (*.cpp) file of the module,

```
ClassOurClass* ClassOurClass::klone(string name) const{
return new ClassOurClass(name, "10/26/06"); // date is your date.
}
```

This code allows the CRHM plarform to create multiple copies of the module.

Structures modification.

Modules are designed to execute the same module code for every HRU. Provision had to be made to control the HRUs that a module executes for.

In the typical module there is a loop similar to;

```
for(int hh = 0; hh < nhru; ++hh){
  // code to be executed
}
```

The simplest approach is to modify it to;

```
for(hh = 0; chkStruct(); ++hh){
  // code to be executed
}
```

The function bool chkStruct(void) and the variable hh are declared in ClassModule. Now control is transferred to CRHM by chkStruct() and CRHM is able to skip the execution of any HRU by incrementing hh or by returning false, terminate execution of the entire loop.

In the *init* function of a module it is normal to use the former loop since everything must be initialised and in the *run* and *finish* functions use the latter loop to limit change of variable output to that determined by the allowed values of HRU. Care must be taken since the module variables are shared with other modules and nothing that the other modules set can be overwritten.

To handle unusual situations these alternatives are also provided.

```
for(int hh = 0; chkStruct(hh); ++hh){ // bool chkStruct(long &hh)
  // code to be executed
}
```

and

```
for(int hh = 0; chkStruct(hh, newmax); ++hh){ // bool chkStruct(long &hh, long newmax) where newmax <= nhru.
  // code to be executed
}
```

The former is to handle multiple loops and the latter is to handle a loop that handles a reduced range. These extensions were required for *pbsm*.