GAPS

_G_eneral-purpose _A_tmosphere-_P_lant-_S_oil Simulator

Version 3.0

User's Manual

July 1994

Susan J. Riha
David G. Rossiter
Patrick Simoens

Department of Soil, Crop, & Atmospheric Sciences
Cornell University
Ithaca, New York, USA

Correspondence regarding this manual or the GAPS program should be addressed to:
GAPS Project, c/o Dr Susan J. Riha
Department of Soil, Crop & Atmospheric Sciences
140 Emerson Hall
Cornell University
Ithaca, NY 14853
USA

Telex: WUI 6713054, Attention: Riha, Soils
FAX: (607) 255-8615, Attention: Riha
Email: sjr4@cornell.edu

Table of Contents

1. Introduction

GAPS is a **G**eneral-purpose simulation model of the **A**tmosphere-**P**lant-**S**oil system. Its outstanding features include the following:

1. It represents soil, plant, and atmospheric processes in a variety of ways. These representations can be selected independently of each other, subject to some limitations. Thus the model user can compare the effects of different ways of simulating the same phenomena.

2. It is menu-driven and includes an editing module for preparing data and parameter files.

3. It can display graphs during the simulation run, as well as after the run is complete, allowing a dynamic understanding of the processes being simulated.

4. It can simulate a sequence of crops and climates in a single simulation run.

5. It can be run on any personal computer with a hard disk or high-capacity diskette which uses the MS-DOS operating system.

6. A batch version, with the same simulation procedures but a command-line user interface, can be used for large numbers of repetitive simulations, and easily integrated with other computer programs for data entry, output, or analysis.

7. Version 3.5 is able to model plant competition

GAPS is intended for use in research and especially in teaching the principles and practice of dynamic simulation modelling of the soil-plant-atmosphere system. We assume that GAPS users have some background in such models, or are using GAPS in conjunction with other instructional materials on simulation in general, and the environmental processes being modelled in particular.

This manual is organized as follows. Chapter 2 explains how to install GAPS on your system. Chapters 3 and 4 describe how to run interactive and batch GAPS. Chapter 5 explains the scientific basis of each model procedure in the simulator. Chapter 6 has some notes of interest to programmers. The manual concludes with a bibliography.

Appendix A discusses the most common error messages. A list of the content of the distribution diskettes is given in appendix B. Appendix C contains examples for each input and output file with comments. A flowchart of the program is presented in appendix D. Appendix E is a tutorial, designed to be read at the computer, which introduces the user to the main features of GAPS.

2. Installing GAPS

2.1. System Requirements

Operating system: GAPS requires a computer capable of running one of the Microsoft operating systems MS-DOS or PC-DOS, V3.0 or later, or another operating system that emulates MS-DOS either directly (e.g. DR-DOS) or in a 'compatibility' mode (e.g. the 'vpix' program running on Unix/386).

CPU: GAPS will run on processors of the Intel 8086 family or functional equivalent ; for reasonable performance we recommend at least a 12MHz 80286.

Main memory: GAPS requires a minimum 480Kbytes of free RAM, but will run faster with more free memory, since less swapping of overlays will be required. GAPS does not use expanded (LIM/EMS) or extended memory. If you have extended memory, you might want to set up a RAM ('virtual') disk in this memory, and set up GAPS to read and write its data files from this virtual disk.

Secondary memory: The GAPS executable programs and support files require about 900Kb of disk space, and so can be installed on 1.2Mb, or 1.44Mb diskette, as well as on a hard disk with 0.9Mb free space. GAPS input files are in the range of 1Kb to 30Kb each (a complete year's climate, requiring 30Kb, is the largest input file); the size of the output files depends on the amount of output requested by the user, in particular, the number of days and hours within each day to save. Thus GAPS can be run on a system with no hard disk, if need be.

Coprocessor: A math coprocessor (Intel 80x87 or functional equivalent) is not required, but will speed up the program *considerably*, especially if computation-intensive model procedures such as the Richards' Equation or Soil Temperature are selected. The installation procedure automatically selects the correct version of the GAPS executables for your system, depending on whether or not a coprocessor is installed. Numerical results are *identical* on systems with and without a coprocessor, since floating-point calculations on non-coprocessor systems are performed by emulating the 80x87. (Borland International's 8087 emulation library).

Monitor: A graphics monitor is not required to run GAPS, but is required to see the run-time and post-run graphics. The graphics software uses Borland International's '.bgi' graphics descriptor files, and you need only install the one that matches your graphics adapter. CGA, EGA, VGA, Hercules, IBM 8514, AT&T 400, and IBM PC3270 graphics are all supported.

The batch version of GAPS, called GSB, requires 250Kb of free RAM. It writes text to the screen using DOS calls, and so does not require a graphics adapter.

Multitasking operating systems: GAPS can run under several popular multitasking operating systems, as a DOS task.

Under DESQVIEW 2.x, GAPS requires a 300KB partition. Other information for DESQVIEW setup: GAPS writes text directly to the screen and displays graphics. It does not use serial ports; it only uses diskettes if you specify data files on these. It does not require or use any extended or expanded memory, and it uses its own color schemes (not DESQVIEW's).

GSB can run as a background task under DESQVIEW, if you redirect its standard output to a disk file (so that it is not writing to the screen while running in the background).

2.2. Installation

GAPS is distributed on one 1.44Mb 3.5" diskette, one 1.2Mb 5.25" diskette, two 720Kb 3.5" diskettes, or four 360Kb 5.25" diskettes. To install GAPS, put the first distribution diskette into a diskette drive, connect to that drive (e.g. with the DOS command 'a:' or 'b:'), and type 'install', for example:

```
A:

install
```

Note that there is no harm in running the installation program several times, e.g. to load parts of the distribution that you hadn't loaded previously, or to update to a new version.

You will be presented with a short data entry screen. To move around press the Tab key or use the mouse if installed. To toggle a selection on or off, press the space bar. The amount of disk space required for each selection is given on the right of it. By default everything is selected and the drive and path where GAPS will be installed is 'C:\GAPS3'.

Specify:

(1) the drive and path where you want GAPS to be installed (the default is 'c:\gaps3'). The installation program will create this directory, if necessary; if it already exists, any earlier version of GAPS in the directory will be overwritten.

(2) GAPS 3.0 executable (protected/real mode):
If selected, the GAPS 3.0 executable with all the support files needed to run GAPS will be installed. The executables and support files require about 600Kb (protected mode version) or 400Kb (real mode version) of disk space. The installation program determines whether or not your computer can use the protected mode version and installs the correct version of GAPS.

(3) GSB 3.0 command-line executable:
GSB 3.0 is the command-line version of GAPS 3.0. Together with the executable, all the necessary files needed to run GSB are loaded. 200 Kb of disk space are needed. The executable is in real mode; so it runs on all PC's.

(4) Tutorial examples:
These files require about 200Kb of disk space. They are required to complete the tutorial examples in this manual. These files are installed in subdirectory '.\tutor', under whatever directory you specified for the program.

(5) Converting program (convert.exe):
The conversion program (convert.exe) allows you to update all the input files used in a previous version of gaps (older than version 3.0). The soil files of versions older than 2.1 must be converted first using the older converting program sol_conv.exe that came with release 2.1. This program requires 60Kb of disk space. See § 2.5. Updating input files for a description on how to use convert.exe.

(6) Source code:
We recommend that this be loaded so that you can examine or print the source code, which is the final authority on how the program works. The source code files require about 720Kb of disk space. These files are installed in subdirectory '.\src', under whatever directory you specified for the program.

(7) Graphics drivers:

This is a list of the seven possible types, and you can load any, all, or none of them. You must load the one(s) that correspond to your system's graphics adapter(s) in order to see run-time and post-run graphs on the screen. However, you don't need any graphics drivers to set up simulation runs, prepare input files, print input and output files, print post-run graphs, or run the batch version of the simulation program. Refer to your computer's hardware documentation to determine what kind of graphics adapter is present on your system. Each graphics driver requires about 6Kb of disk space.

When you are done filling in the form, select the Install button to proceed with the installation. You will be prompted to insert other distribution diskettes if necessary .

If you don't want to proceed with the installation, press the **ESC** key or select the Quit button.

2.3. Printers

GAPS produces two kinds of output: (1) listings of input and output files, and (2) post-run graphs. The listings can be produced on a wide variety of printers. As distributed, the printing output is directed to a TEXT-file. To change the printer setting, select the 'Printer Setup...' option in the 'File' menu. You will be shown a list of printers; select the one that is actually attached to your system.

GAPS assumes that the printer is DOS device 'LPT1:'. Usually this is associated with the first parallel port. If your printer is actually connected to another port, you will have to use the DOS 'assign' command to associate the logical device 'LPT1:' with the physical device to which the printer is attached. For example, the command:

```
assign lpt1: com2:
```

will direct GAPS printer output (and any other that uses LPT1) to the second serial port. GAPS does not set any communications parameters for serial ports, so these should be set up with the DOS 'mode' command (you'd have to do this for other programs as well).

2.4. Monitors

If you have a graphics adapter, but a monochrome monitor, the autodetection logic used by the Turbo Pascal run-time library will think that a color monitor is installed and you will be unable to see things correctly on the screen. This is often a problem with laptops and other computers with LCD screens. To tell GAPS to display output in monochrome regardless of the adapter, type at the DOS prompt the command:

```
SET GAPSMODE=MONO
```

You can also put this command in your 'autoexec.bat' or a batch file which calls GAPS. GAPS will not display any run-time graphs if this environment variable is set.

2.5. Updating input files

The format of all the input files (*.sol, *.cli, *.loc, *.plt, *.sav & *.set)) has changed with the new release of GAPS. Old input files (except soil files from versions older than 2.1) can be updated to the new format with the conversion program convert.exe.

The wildcards '*' and '?' can be used to specify a group of files. Only the files in the current directory can be updated. A backup of the old input file will be saved under the same filename with extension '.??k'.

Examples:

To convert all the input files in the current directory type the following command at the DOS-prompt :

```
convert  *.*
```

To convert all the plant files in the current directory type:

```
convert  *.plt
```

To convert all the input files starting with 'tut2' type:

```
convert tut2*.*
```

3. Running GAPS

This chapter describes how to use interactive GAPS.

3.1. Starting GAPS

GAPS is invoked from the DOS command line like any other executable program, with the command syntax:

```
[drive:][path]gaps[.exe] [filename.set]
```

When GAPS starts up, the simulation parameters are initially null (no procedures, no input or output files, no graphs) unless a scenario file is specified at the command line.

3.2. Interacting with GAPS

The GAPS interface has three visible components: the menu bar at the top, the status line at the bottom, and the desktop between the menu bar and the status line.

3.2.1. The menu bar

To activate the menu bar, press the F10 function key. When activated one of the menu options will be highlighted. Use the left or right arrow key to highlight another menu option. To deactivate the menu bar press the Escape key or press the F10 key again.

To select one of the menu options, first highlight it and then press Enter. Except for the 'Run' option, which starts the simulation on selection, a submenu will popup with more options. Use the up and down arrow keys, the End and the Home key to highlight another submenu option within the same menu option. Use the left and right arrow keys to popup another submenu. Submenu options followed with an arrow lead to another subsubmenu with related options. The Escape key brings you back to the previous submenu. Options followed by three dots (...) lead to a dialog box.

Shortcuts:

To select a menu option, simply press the highlighted letter of the menu option in combination with the Alt key. If the menu bar is already activated, the Alt key can be omitted. To select an option within a submenu simply press the highlighted letter of that option.

Some menu options are followed by a key combination. To select those options simply press the key combination (e.g. to cut a selected text from a file press the Shift and Del key together). Some of these shortcuts are mentioned on the status line.

If you have a mouse installed, simply click on the desired menu and submenu options. You can also drag straight from the menu option to the submenu option.

When a particular option doesn't make sense in the current context, it will be disabled and appear dim.

3.2.2. The status line

The status line appears at the bottom of the screen. Besides reminding you of some of the most important hot keys it offers one-line descriptions about the selected menu option.

You can click on the hotkeys on the status line to carry out the action instead of selecting the command from the menu or pressing the hot key.

Example:

To quit GAPS you can either:
- Press F10, select the File option using the arrow keys + the Enter key or by pressing Alt+F or F, and select Exit using the arrow or End keys + the Enter key or by pressing Alt+X or X
- Click on the File option in the menu bar and then click on the Exit option in the submenu.
- Click on the File option in the menu bar, drag straight down to the Exit option and release the mouse button.
- Press the hot key Alt-X
- Click on Alt+X Quit on the status line.

3.2.3. The desktop

Except for the runtime and postrun graphs, all you see and do happens in a window or dialog box.

Windows:

The editing of the input and output files occurs within windows. The windows are opened on selection of a file for editing or on creation of a new input file. A window is a screen area that can be moved, resized, tiled, overlapped and closed. More than one window can be open at any time, but only one can be active, the one you are currently working in.

The topmost horizontal bar of a window contains a close box on the left, the filename in the middle and a zoom box on the right. To quickly close the window, click in the close box with the mouse. To enlarge or shrink the window click in the zoom box.

The current position of the cursor (line : column) is indicated on the bottom line of the window. An asterisk on the left of the line and column numbers indicates the file has been modified since it was opened or last saved.

The resize corner in the lower right corner of the window allows you to make the window smaller or larger by dragging it with the mouse.

The contents of a window can be easily scrolled using the horizontal and vertical scroll bars on the lower and right border of the window. To scroll one line at a time , click the arrow at either end. For continuous scrolling keep the mouse button pressed. To scroll one page at a time, click the shaded area to either side of the scroll box. For quickly moving around, drag the scroll box to any spot on the bar.

Without a mouse these actions can be done using the options in the Window menu, or their corresponding hot key. To scroll the text in a window use the arrow, Page Up, and Page Down keys.

Dialogs:

Most selections in GAPS are made in dialog boxes. As with windows, dialog boxes can be moved around and have a close box which is equivalent to pressing the escape key. Five basic types of onsceen controls can be present in a dialog box : radio buttons, check boxes, action buttons, input boxes and list boxes.

Press the Tab key or Shift+Tab to move the cursor from one group of controls to the other. Use the arrow keys to move the cursor to the next or previous control. To toggle a check box or radio button on or off, press the space bar. To select an action button (e.g. Ok, or Cancel), press the Enter key with the cursor on that button. Selecting the cancel button, pressing Escape or clicking the close box will put away the dialog box without saving any changes.

With a mouse simply click on a button or box to select it.

3.3. The menu

The main menu contains the following options: File, Edit, Graph, Scenario, Run and Window. The menu options with their submenu options are discussed briefly below.

3.3.1. File

The File menu contains the options for creating, editing and saving input and output files, selecting the printer, printing of any file, shell to DOS and quit the program.

- About... : displays some general information about GAPS 3.0. Press Esc or Spacebar or click Ok (or press Enter) to close the box.

- New : pops up a submenu with the 5 different types of input files (climate, location, soil, plant and save file). After selection of one of these options the Open File dialog appears, prompting for the .def file. The different directories are listed. To move to a subdirectory, double click on the directory name or highlight the name of the directory and select Open. To move a directory up towards the root, select the ..\ directory. After selection of a .def file (climate.def, location.def, soil.def, save.def, pconst.def, pcorn.def, plant.def, psorkam.def, pepic.def, ptree.def, or pwheat.def), a new file 'Untitled' will be created with the content of the selected .def file. You can put your own comments in the .def file so that all the new files you create will contain those comments. To save the new file under a filename with the appropriate extension (.cli, .sol, .loc, .plt, or.sav), while in the 'Save file as' dialog, move to the directory where you want to save the file, enter the new name and select Ok. The current directory is displayed at the bottom of the 'Save file as' dialog.

- Open : pops up a submenu with the 6 different types of input files and the 6 types of output files. After selection of a filetype, the Open file dialog appears, prompting for the file to open. For selection and moving around in the Open file dialog see 'New'.

- Save : saves the file of the current active window to disk.

- Save as : allows you to save the file of the current active window under a different name. Enter the new name, optionally with drive and directory, and click or choose Ok. If you pick an existing filename , that file will be overwritten.

- Printer Setup... : pops up a dialog box with a list of all the printers described in the gaps.prt file. The default is 'TEXT-FILE' which allows printing to text files instead of a printer. Highlight a printer and select Ok to select it. The number of the selected printer is saved in the Gaps configuration file gaps.cfg.

- Print... : pops up a submenu with all the filetypes that can be printed.After selection of a filetype, the Print file dialog box appears, with the filename filter set to the selected filetype (eg. *.cli for climate files). Move around and select the file you want to print.

- DOS shell : leaves GAPS temporarily to enter commands at the DOS command line. To return to GAPS, type EXIT at the command line and press Enter. This option may not work when in real mode.

- Exit : exits GAPS and returns you to the DOS command line.

3.3.2. Edit

The Edit menu lets you cut, copy, and paste text in windows. You can also open the Clipboard window to view or edit its content.

To select text, press Shift while pressing any arrow key. The selected text will be highlighted. To select text with a mouse, drag the mouse pointer over the desired text. To select a whole line, double-click anywhere in the line. To extend or reduce the selection, Shift-click anywhere in the document.

- Undo : takes back the last editing command.

- Cut : removes the selected text from the document and places it temporarily in the clipboard. You can then paste it into one or more documents by choosing Paste.

- Copy : copies the selected text from the document in the clipboard. It can be pasted then into one or more other documents by choosing Paste.

- Paste : inserts the text from the clipboard into the current window at the cursor position.

- Clear : removes the selected text from the document but does not put it into the clipboard.

- Show clipboard : opens the Clipboard window, which stores the text you cut and copy from other windows.

3.3.3. Graph

The Graph menu lets you display the postrun results graphically or, if Print variables is checked, output will be directed to the print queue (printer or text file).

- Daily summary : pops up the open file dialog. After selection of a daily summary output file (extension .sum) the PostRun Graphs dialog box appears. Enter the time range, and the graphs to view (maximum 4 graphs can be displayed). Press any key to return to the menu after viewing the graphs.

- Layers - soil : pops up the open file dialog. After selection of a layers output file (extension .lay) the PostRun Graphs dialog box appears. Enter the time range, the graphs to view (maximum 4 graphs can be displayed) and the layers. Press any key to see the graphs for the next selected layer.

- Soil flux: pops up the open file dialog. After selection of a soil flux output file (extension .sof) the PostRun Graphs dialog box appears. Enter the time range, and the graphs to view (maximum 4 graphs can be displayed). Press any key to return to the menu after viewing the graphs.

- Plant flux : pops up the open file dialog. After selection of a plant flux output file (extension .plf) the PostRun Graphs dialog box appears. Enter the time range, and the graphs to view (maximum 4 graphs can be displayed). Press any key to return to the menu after viewing the graphs.

- Climate flux : pops up the open file dialog. After selection of a climate flux output file (extension .clf) the PostRun Graphs dialog box appears. Enter the time range, and the graphs to view (maximum 4 graphs can be displayed). Press any key to return to the menu after viewing the graphs.

- Settings : pops up the Postrun graph settings dialog box.
  Print variables : redirects the output to the selected printer instead of graphing the results to the screen. If printing is set to text-file, you will be prompted for a file name.
  Show grid : displays the grid on the graphs.
  Show symbols : adds symbols to the graphs.

3.3.4. Scenario

The Scenario menu contains all the options needed to specify the input and output files, the models to simulate and the graphs to display at runtime. It also contains the commands to view the current selected specifications and status.

- Load scenario : pops up the Load Scenario dialog for selection of a scenario file (extension .set). On selection of a valid scenario file, the input and output files, the model procedures, the selected graphs, and the time specifications will be set according to those described in the scenario file.

- Save scenario : pops up the Save scenario dialog box. Type a scenario filename or select an existing file. The current settings will be saved in that scenario file.

- Input files : pops up the Select dialog box for the selection of the climate, location, soil, plant file 1, plant file 2 and save input files. If Escape or Cancel is selected, the previous selection is cleared out.

- Output files : pops up the Select dialog box for the selection of the Soil layer, Soil flux, Plant flux, Climate flux, Daily summary and Model summary output files.If Escape or Cancel is selected, the previous selection is cleared out.

- Model procedures : pops up the Model procedures dialog box for selection of the ETP, Flow, Water Uptake and Temperature procedures, the Field hours and input models.

- Time : the simulation time dialog lets you enter the time step (in seconds), the first and last day of the simulation and the numbers of years to simulate (maximum 11). By default the first and last day of simulation are set to the first and last day specified in the scenario file.

- Runtime Graphs :
  Select Graphs : the first 4 selected graphs will be displayed at runtime. Use the Tab and/or arrow keys to move to another selection.
  Settings : lets you change the scale of the selected graphs and toggle the grid on or off.

- Status : displays the current status of the computer (coprocessor, memory, loaded scenario...).

- Specifications : displays the current settings (input and output files, simulation models, time specifications, and selected graphs).

3.3.5. Run

The Run menu option causes the simulation to start with the current settings.

3.3.6. Window

The Window menu contains the window management commands.

- Tile : arranges all the open windows one next to the other so that none overlap.

- Cascade : stacks all open windows so that the file names are visible and the active window is on top.

- Close all : closes all the windows.

- Size/Move : resizes or repositions the active window in response to the arrow keys. To change the size of a window use the arrow keys in combination with the Shift key, Press Enter when done. With a mouse you can also resize a window by dragging its title bar.

- Zoom : resizes a window to the maximum size. If already zoomed to the maximum, it will restore it to its previous size. With a mouse, double click the windows top line to zoom it.

- Next : makes the next window active.

- Previous : makes the previous window active.

- Close : closes the active window.

3.4. The input and output files

3.4.1. The input files

Several kinds of input can be provided to run a simulation in GAPS:

(1) one or more **climates**, specifying the weather on a daily basis.

(2) a **plant**, specifying various parameters associated with the plant. Multipleplant files may be specified in a cropping sequence (see (6), below).

(3) a **soil**, describing a one-dimensional soil profile.

(4) a **location**, specifying various parameters associated with the site being simulated, but not a specific soil or a particular year's weather.

(5) a **save specification**, specifying the kind and quantity of output that you want GAPS to produce during the simulation run.

(6) a **cropping sequence**, describing multiple crops (relay or inter-cropping)

Not all of these input files are necessarily required; this will depend on the specific simulation you are attempting.

These files can be created and edited with the GAPS Editor (**F**ile/**N**ew/a_file or **F**ile/**O**pen/a_file menu options) or by another DOS program (e.g. as output from a spreadsheet macro). They may be printed, nicely formatted, with the **F**ile/**P**rint/a_file menu option.

The actual file structures can be interpreted by comparing the input file with the appropriate procedures in source file 'fileio.pas', along with the structure definitions from 'global.pas'. You can print the sample input files, unformatted, supplied with GAPS, directly from DOS or a program editor to see their actual structure. The following list gives the default DOS file extensions, input/output procedures in 'fileio.pas', and structure names in 'global.pas':

(1) climates: default file extension: `cli` ; i/o procedures 'load_climate', structure 'climate_info'.

(2) plants: default file extension: `plt` ; i/o procedures 'load_crop_options_in_plant' and the crop object Inits; structure 'plant_info'

(3) soil: default file extension: `sol` ; i/o procedures 'load_soil'; structure 'soil_info'

(4) location: default file extension: `loc` ; i/o procedures 'load_location'; structure 'location_info'

(5) save specification: default file extension: `sav` ; i/o procedures 'load_save'; structure 'save_output_info'

(6) cropping sequence: default file extension: seq; i/o procedures 'load_sequence'; structure 'dates' and 'plantFileNames'.

The order and format of the data in the input files is important. Comments can be added, preceded with a '#' sign. All characters on an line following a '#' sign are ignored at input. Empty strings must be replaced by a '-' sign (eg. when no save file is selected). The data are followed by their variable name, datatype, and units of measure.

Since GAPS allows you to select which model procedures to include in a simulation, you may not need all of these files, or you may not need to fill in certain fields in a file. For example, if you are simulating the soil water balance under fallow (i.e. no crop is being simulated), you would not need to specify a plant file. If you are simulating potential evapotranspiration using any method other than 'Pan', your climate file does not need the field for 'Pan evaporation', and your location file need not specify the pan and crop coefficients. Refer to the description of each model procedure for the input data required for each one.

The following sections describe each file in more detail.

3.4.1.1. Climate file

File suffix: '.cli'.

Climate Data Files are used to tell the GAPS simulator about the daily weather. These files can contain a maximum of one year of data (365 days); for multi-year simulations, you must create a separate climate file for each year.

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Minimum air temperature	MinTemp	°C	[AirTemperature], Soil_Temperature, Harmonic_Soil_Temp, Linacre_ETP, Penman_ETP, SR_model: Growth_Stages, Max_Photosynthesis, Critical_Leaf_Water_Potential
2	Maximum air temperature	MaxTemp	°C	same as MinTemp

3.	Solar radiation	SolRad	MJ m^{-2} d	[Atmos_Trans], Priestley_Taylor ETP, Penman_ETP, SR_model: Max_Photosynthesis, SK_model: dry_matter_accumulation
4.	Precipitation	Precip	mm d^{-1}	[Distribute_Precipitation], Tipping_Bucket, Richards_Equation, Matric_Flux_Potential, SR_model: Water_Interception
5.	Relative humidity	RelHumid at Tmax	dimension–less	Penman_ETP, Richards_Equation, Matric_Flux_Potential
6.	Wind speed	WindSpeed	m s^{-1}	Penman_ETP
7.	Pan evaporation	PanEV	mm d^{-1}	Pan_ETP
8.	Saturated vapor density	VaporDensity	g m^{-3}	Penman_ETP (optional, if not entered, will be estimated from the simulated air temperature).
9.	Snow	SnowPack	cm	SRw_model

In the computer code, all input climate data are in the record 'clim', and so their variable names are prefixed by the record name, e.g. 'clim.MaxTemp'.

A climate file with all this information for all 365 days in a year requires 30Kb disk space.

3.4.1.2. Soil file

File suffix: '.sol'.

A soil data file specifies a one-dimensional soil profile as a function of depth. The layers are numbered from 2 (the layer whose upper boundary is the surface) to an arbitrary maximum, which is layer 19 in the versions of GAPS and GSB as distributed[1] . In the soil data file, the user specifies the actual last layer, which may be between 2 (i.e. a one-layer soil) and 19. The program itself uses layer 1 for the air, and a layer immediately below the last one specified for a lower boundary. So in GAPS as distributed, a soil profile may have up to 18 layers.

The number of layers which should be specified by the user depends on two factors: the profile description from the field or laboratory, and the requirements of the GAPS program. The natural approach would be to specify one GAPS soil layer for each sampled layer. However, in some cases it may be desirable to split a sampled layer into several GAPS layers. This is most important when using one of the numerical soil water flow procedures (Richards Equation or Matric Flux Potential). Adjacent thick layers, especially with

[1] See section 7.1, "Changing GAPS' limits" for information on how to increase the number of allowable layers

contrasting saturated hydraulic conductivities or moisture release curves, may lead to instability in the numerical methods during periods with highly-contrasting water contents (e.g. infiltration into a dry soil). In this case you will get an error message, 'Non-convergence of iterative solution'. To avoid this, specify thin layers (1 to 2 cm thickness) around the boundary between contrasting layers.

The most common problem with non-convergence occurs at the surface, where infiltration and evaporation are most active. We highly recommend specifying layers of increasing thickness at the surface, e.g. of 1cm, 2cm, and 5cm, even if the surface is actually homogeneous to some greater depth, as is typical of cultivated soils.

The thickness of the surface layers is also important when estimating field hours. The depth to which you specify that the soil must be workable (see 'location' file description) should correspond to the lower boundary of a soil layer.

Root growth in GAPS is entirely by layer: when the model predicts that the rooting depth has exceeded the lower boundary of a layer, the roots 'instantly' fill the next layer. So for realistic water uptake, the layers should be thin enough so that this is not a bad assumption.

Within the GAPS simulation procedures, there are two ways of considering the soil profile: by *layer* and by *node*. Layers are the soil as described in the input file, and are used for calculations of volumetric water content. Nodes are considered to be points in the middle of layers, and are where the water potentials are measured. The node depths are calculated from the layer boundaries in 'Init_Soil' (source file 'misclib.pas').

Defaults for all variables are set in procedure 'clear_soil' (source file 'fileio.pas'). Each soil profile, as a whole, may have the following fields:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Number of soil layers, less 1	LastLayer	count	required for layer descriptions that follow
2.	Shortwave absorptivity of the surface	SwAbs	fraction Default: 0.78	Priestly_Taylor_ETP, Penman_ETP
3.	Curve number	CN2	index on [0 – 100]	Runoff_CN
4.	Site slope	Slope	m / m	Runoff_CN
5.	Water Uptake Distribution	WUD	[0-10]	EPIC_Water_Uptake (see comments there)

Each layer of the soil may have the following fields:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Lower boundary of soil layer	LowBound	m	Soil_Temperature, Harmonic_Soil_Temp, Tipping_Bucket, EPIC_Water_Uptake, Is_Workable, crop_model: Daily_Growth_Begin
		NodeDepth (derived from LowBound)	m	WaterUptake, Harmonic_Soil_Temp
		LayThick (derived from LowBound)	m	Soil_Temperature, Tipping_Bucket, Simple_Water_Uptake, WaterUptake
2.	Bulk density	BulkDensity	Mg m^{-3} Default: 1.2	Soil_Temperature
3.	Particle density	PartDensity	Mg m^{-3} Default: 2.6	Soil_Temperature
4.	Saturated hydraulic conductivity	HydCond	kg s m^{-3} Default: 0.003	WaterUptake, Richards_Flow, Matric_Flux_Potential
5.	Air Entry potential	AirEntryPot	J kg^{-1} Default: -0.3	WaterUptake, Richards_Flow, Matric_Flux_Potential
6.	B-value (slope of moisture release curve plotted on a log-log scale)	BValue	dimension–less Default: 7.8	WaterUptake, Richards_Flow, Matric_Flux_Potential
7.	Initial volumentric water content	InitWater	m^3 m^{-3} Default: DUL	initial conditions for Tipping_Bucket, Richards_Flow, Matric_Flux_Potential
8.	Field capacity volumetric water content (‘drained upper limit’)	DUL	m^3 / m^3 Default: 0.3	Tipping_Bucket, EPIC_Water_Uptake, Is_Workable (if flow method is Tipping_Bucket), SK_model: Soil_moisture_index

9.	Permanent wilting point volumetric water content ('lower limit')	DLL	m^3 / m^3 Default: 0.1	SimpleWaterUptake, EPIC_Water_Uptake
10.	Initial soil temperature	InitSoilTemp	° C Default: 20	initial conditions for Soil_Temperature, evaporation procedures of Richards_Equation, Matric_Flux_Potential
11.	Clay content (<2 μ)	Clay	cg / kg of the fine-earth fraction Default: 25	Soil_Temperature
12.	Silt content (2 – 20 μ)	Silt	cg kg^{-1} of the fine-earth fraction Default: 25	** not used in any currently-implemented module, included for completeness
13.	Saturated volumetric water content (i.e. total pore space)	SatWaterCon	m^3 m^{-3} Default: (1-BulkDensity/PartDensity)	Tipping_Bucket, Richards_Equation, Matric_Flux_Potential
14.	Coarse fragments	CoarseFrag	m^3 / m^3 Default: 0	

In the computer code, all soil data are in the record 'soil', and so their variable names are prefixed by the record name, e.g. 'soil.SatWaterCon'.

The disk space required for a soil file depends on the number of layers and the data recorded for each layer. The maximum size of a soil file is 3.5Kb.

3.4.1.3. Location file

File suffix: '.loc'.

A 'location' refers to a place on the earth's surface, without reference to the soil or crop. At a given location there are typically a variety of soils, and experimental treatments on the various soils. Defaults for all variables are set in procedure 'clear_location' (source file 'fileio.pas'). The location input data file contains the following parameters:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Location name	LocationName	string	
2.	Latitude	Latitude	° N or S Default: 0	[SolarAngle], Linacre_ETP, SK_model: emergence (for tropical cultivars only)
3.	Time of solar noon	TSN	s Default: 43200	[SolarAngle]
4.	'Alpha' factor for Priestly-Taylor equation	Alpha	dimension–less Default: 1.26	Priestly_Taylor_ETP
5.	Pan coefficient	Kp	dimension–less Default: 1.0	Pan_ETP
6.	Crop coefficient	Kc	dimension–less Default: 1.0	Pan_ETP
7.	Boundary layer conductance	BLC	W m^{-2} / °K Default: 20.0	Soil_Temperature
8.	Hour of day rain or irrigation starts	RainFirst	h (0..24) Default: 0	[Distribute_Precipitation]
9.	Hour of day rain or irrigation stops	RainLast	h (0..24) Default: 24	[Distribute_Precipitation]
10.	Height above canopy of wind speed measurement	WindHeight	m Default: 2	Penman_ETP

11.	Elevation	Elevation	m above mean sea level Default: 0	Linacre_ETP
12.	Depth to which soil must be dry in order to till, to traffic	WorkMaxDepth [till], WorkMaxDepth [traffic]	m Defaults: 0.15, 0.05	Is_workable
13.	Water content at which soil must be below in order to till, to traffic	WorkLimitWN [till], WorkLimitWN [traffic]	$m^3\ m^{-3}$ Defaults: 0.37	Is_workable (if water flow method is Tipping_Bucket)
14.	Water potential at which soil must be below in order to till, to traffic	WorkLimitWP [till], WorkLimitWP [traffic]	J / kg Defaults: -9.8	Is_workable (if water flow method is Richards_Equation or Matric_Flux_Potential)
15.	Maximum depth to which evaporation can dry soil to less than the DUL	depth_of_evap	m Default: 0.05	Tipping_Bucket
16.	CO_2 concentration of the atmosphere	CO2ext	g / m^3 Default: 0.54	SR_maize_model: Critical_Leaf_Water_Potential

In the computer code, all location parameters are in the record ‘loca, and so their variable names are prefixed by the record name, e.g. ‘loca.Latitude’.

A location file requires about 1.4Kb on disk.

3.4.1.4. Plant file

File suffix: ‘.plt’.

A plant data file describes the crop’s management and static parameters, e.g. its genetic characteristics. Defaults for some variables are set in procedure ‘clear_plant’ (source file ‘fileio.pas’).The plant data file contains the following parameters:

Global variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1	Crop model	Crop_option	from the fixed list of crop models, default none	Simula
2.	Plant simulation model	proc_status[1..9]	dimension-less	

Crop-record variables are global and loaded at crop initialization:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Sowing date[2]	SowingDate	Day of year Default: 1	Simula, SR_model: Growth_stages, is_active; crop_model: is_transpiring, is_active
2.	Harvest date	HarvestDate	Day of year Default: 365	Simula, SR_model: is_active; SK_model: is_active; crop_model: is_transpiring, is_active
3.	Initial and limiting root water potential	RootWP	J / kg Default: -1500	WaterUptake, SR_model: Critical_Leaf_Water_Potential; crop_model: constructor (but only used in SR_model) The root water potential is not allowed to drop below this value.
4.	Root radius	RootRad	m Default: 0.002	WaterUptake
5.	Root resistance to water flow	RootRes	m^4 / kg s Default: 2.5 x 10^{10}	WaterUptake

[2] These fields are ignored if there is a cropping sequence ('.seq') file for the simulation run. In this case, the dates in the sequence file are used instead. The dates in the plant file are retained for backwards compatibility in single-plant simulations that do not need a cropping sequence file.

6.	First (shallowest) soil layer that can contain roots	FRoot	soil layer number, 2..MaxLayer Default: 2	SimpleWaterUptake, WaterUptake, SK_model: Soil_Moisture_Index; crop_model: Daily_Growth_Begin
7.	Last (deepest) soil layer for which RootDens (field 11) is specified	NRoot	soil layer number, 2..MaxLayer Default: MaxLayer-1	crop_model: Daily_Growth_Begin Limited at run time to the last layer in the soil profile
8.	Root length density, for each layer from FRoot to NRoot	RootDens [FRoot .. NRoot]	m (of root length) / m^3 (soil volume) Default: 5.0 x 10^4	SimpleWaterUptake, WaterUptake
9.	Maximum height of canopy	CanopyHeight	m Default: 1.5	Penman_ETP, SR_model, SE_model, SK_model, SC_model Note: >= 0.01
10.	Aerodynamic resistance	RA	s / m Default: 30.0	Penman_ETP (only used if daily clim.WindSpeed is not known); SR_model: Critical_Leaf_Water_Potential
11.	Resistance of canopy when stomates open	RCopen	s / m Default: 20.0	Penman_ETP
12.	Shortwave absorptivity of plant canopy	AS	dimension–less Default: 0.75	Penman_ETP (when crop is growing; cf. loca.AS)

Plant-record variables are local to the general crop model and are loaded when the new crop is initialized:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Initial Leaf Area Index	LAI	m^2 / m^2 Default: 0.07	crop_model: constructor (Initialize for SK_model: dry_matter_accumulation, daily_plant_growth; SR_model: Growth_Stages, Max_Photosynthesis; *_model: GetTransFrac Note: inital LAI >= 0.07
2.	Initial dry matter	InitialDryMatter	kg / ha Default: 0	crop_model: constructor; initializes for SR_model: Dry_Matter_Accumulation; SK_model: dry_matter_accumulation
3.	Maximum rooting depth	MaxRooting Depth	m Default: 1.20	SK_model: daily_plant_growth; SR_model: Dry_Matter_Accumulation; note: for water uptake, roots are considered to exploit the entire layer in which MaxRootingDepth occurs
4.	Sowing depth	SowingDepth	m Default: 0.025	SR_model: Dry_Matter_Accumulation; SK_model: emergence, daily_plant_growth

All the following variables are local to specific crop models.

Stockle-Riha crop model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Maximum water on canopy	plant_Canopy Max	kg / m^2 leaf Default: 2	
2.	Critical leaf water potential	plant_Critical LeafWP	J / Kg Default: -1500	Critical_Leaf_Water_Potential
3.	Planting density	plant_Plant Density	plants / ha Default: 62500	Yield
4.	Maximum harvest index	plant_Harvest Index	kg (dry grain) / kg (above-ground dry matter) Defaults: 0.5 (crop_model), 0.44 (SR_model)	constructor (Initializes for yield [limits simulated harvest index], End_growth [used to estimate yield from dry matter if not simulating yield)
5.	Number of kernels per plant	plant_Kernels_per_ plant	count Default: 570 (SR_model)	constructor (Initializes for yield)
6.	Kernel growth rate	plant_Kernel_ growth_rate	mg / d Default: 8 (SR_model)	constructor (Initializes for yield)
7.	Power factor for critical leaf water potential equation	plant_CLWP_ Power	dimension–less Default: 7	Critical_Leaf_Water_Potential
8.	Maximum rate of photosynthesis	plant_Pmax	mgCO2/ m^2Leaf.sec Default: 1.0	Max_Photosynthesis

9.	Initial slope of photosynthetic response curve	plant_P_Effic	mgCO2/J Default: 0.007	Max_Photosynthesis
10.	Curvature of photosynthetic response curve	plant_P_Curve	Default: 0.75	Max_Photosynthesis
11.	Temperature minimum for photosynthesis	plant_PT_min	Default: 0 °C	Max_Photosynthesis
12.	Temperature optimum for photosynthesis	plant_PT_Opt	Default: 25 °C	Max_Photosynthesis

Stockle-Riha corn model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Varietal coefficients	plant_Coeff[1..5]	Defaults: 3, 49, 56, 58, 65	SRm_model
2.	1st point height-vs-GDD curve (x1, f1)	heightS_y1, as fitted by compete.FitSCurve	Default: (0.3, 0.2)	SRm_model.GetHeight
3.	2nd point height-vs-GDD curve (x2, f2)	heightS_y2, as fitted by compete.FitSCurve	Default: (0.5, 0.8)	SRm_model.GetHeight

Stockle-Riha wheat model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Row spacing	plant_Row Spacing	m Default: 0.20	constructor (Initializes for dry_matter_accumulation)

2.	Varietal coefficients	plant_Coeff[1..13]	Defaults: 100, 176, 328, 460, 602, 744, 886, 1028, 1170, 1312, 1454, 1514, 1892	Daily_Growth_End, Growth_Stages
2.	1st point height-vs-GDD curve (x1, f1)	heightS_y1, as fitted by compete.FitSCurve	Default: (0.3, 0.2)	GetHeight
3.	2nd point height-vs-GDD curve (x2, f2)	heightS_y2, as fitted by compete.FitSCurve	Default: (0.5, 0.8)	GetHeight

Stockle-Riha fast growing tree model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Accumulated stem dry matter	plant_AccStemDM	kg / m^2 Default: 0.0142	Dry_Matter_Accumulation
2.	Accumulated leaf dry matter	plant_AccLeaf DM	kg / m^2 Default: 0.0193	Dry_Matter_Accumulation
3.	Daily elongation rate of sinker roots	plant_Sink GrowRate	m / d Default: 0.024	Dry_Matter_Accumulation: Grow_Sinkers
4.	Number of sinker roots per plant	plant_NSink	Default: 3	Dry_Matter_Accumulation: Grow_Sinkers
5.	Sinker root length to weight ratio	plant_LW_Sink	m / kg Default: 10000	Dry_Matter_Accumulation: Grow_Sinkers
6.	Fine root length to weight ratio	plant_LW_Fine	m / kg Default: 13000	Dry_Matter_Accumulation: Maintain
7.	Initial depth of roots	plant_Sink Length	m Default: 0.18	Dry_Matter_Accumulation: Grow_Sinkers, Distribute
8.	Coefficient relating stem diameter to wood biomass	plant_aWood	Default: -4.358	Dry_Matter_Accumulation: Partition, Form_Aboveground

9.	Coefficient relating stem diameter to wood biomass	plant_bWood	Default: 2.9	Dry_Matter_Accumulation: Partition, Form_Aboveground
10.	Coefficient relating diameter and biomass to height	plant_aHeight	Default: -4.1	Dry_Matter_Accumulation: Form_Aboveground
11.	Coefficient relating diameter and biomass to height	plant_bHeight	Default: 1.949	Dry_Matter_Accumulation: Form_Aboveground
12.	Coefficient relating diameter and biomass to height	plant_cHeight	Default: 0.895	Dry_Matter_Accumulation: Form_Aboveground
13.	Reference temperature for maintenance respiration	plant_RefTemp	°C Default: 25	Dry_Matter_Accumulation: Maintain
14.	Specific leaf area	plant_SLA	m^2 / kg Default: 6	Dry_Matter_Accumulation: Form_Aboveground
15.	Tree age at planting	plant_Init_Age	day Default: 365	Dry_Matter_Accumulation: Form_Aboveground
16.	Maximum age for a leaf	plant_MaxLeafAge	day Default: 1095	Dry_Matter_Accumulation: Form_Aboveground
17.	Initial suberized root biomass	OldRtBio[2..5]	kg/m^3	Dry_Matter_Accumulation

Sorkam crop model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Row spacing	plant_Row Spacing	m Default: 0.20	constructor (Initializes for dry_matter_accumulation)
2.	Number of leaves	plant_NLeaves	Default: 19	emergence: Tropical_DD

EPIC crop model specific variables:

No.	Data field	Variable name	Units of measure	Modules in which field is used
1.	Harvest index	plant_Harvest Index	Default: 0.31	Daily_Growth_End
2.	Minimum harvest index	plant_MinHI	Default: 0.15	Daily_Growth_End
3.	Base temperature	plant_Base Temp	°C Default: 10.0	Daily_Growth_End
4.	Optimal temperature for growth	plant_Optimum Temp	°C Default: 25.0	Daily_Growth_End
5.	Potential heat units req. for maturation	plant_PotUnits	Default: 2000	Daily_Growth_End
6.	Biomass energy ratio	plant_Biomass Energy	kg/MJ Default: 25.0	Daily_Growth_End
7.	Fraction of season LAI declines	plant_Decline Fraction	Default: 0.60	Daily_Growth_End
8.	Maximum potential LAI	plant_MaxPotLAI	Default: 9.0	Daily_Growth_End
9.	1st point LAI-vs-heat unit curve (x1, f1)	plant_Point1LAI, as fitted by compete.FitSCurve	Default: 0.25, 0.20	Daily_Growth_End
10.	2nd point LAI-vs-heat unit curve (x2, f2)	plant_Point2LAI, as fitted by compete.FitSCurve	Default: 0.45, 0.80	Daily_Growth_End
11.	rate of LAI decline	plant_LAI DeclineRate	Default: 1.00	Daily_Growth_End
12.	Extinction coefficient	plant_ExtinctionCoeff	Default: 0.65	GetTransFrac
13.	1st point max. LAI-vs-population density curve (x1, f1)	plant_MaxPotLAI is adjusted with this and the next field	Default: (10000, 0.2)	Daily_Growth_End

14.	2nd point max. LAI-vs-population density curve (x2, f2)	plant_MaxPotLAI is adjusted with this and the previous field	Default: (40000, 0.8)	Daily_Growth_End
15.	Maximum height that can be attained per unit of total dry matter	plant_MaxHeightPerWeight (s) [m kg-1]	Default: 13.8	Daily_Growth_End

Constant crop model specific variables:

No.	*Data field*	*Variable name*	*Units of measure*	*Modules in which field is used*
1.	Maximum top dry matter	plant_MaxTop DM	kg / ha Default: 5	Begin_growth
2.	Maximum water on canopy	plant_Canopy Max	kg / m^2 Default: 0	Water_Interception
3.	Base temperature	plant_Base Temp	°C	Daily_Growth_End
4.	Extinction coefficient	plant_ExtinctionCoeff	Default: 0.65	GetTransFrac

Note that rooting depth is specified by two fields of the plant input file: explicitly, by *MaxRootingDepth*, and implicitly, by *NRoot*. The latter depends on the layer boundaries in the selected soil input file. The rooting depth is limited by the *MaxRootingDepth* t and by *NRoot* (the deepest layer for which *RootDens* is specified in the plant input file).

A note on the S-curve parameters used in Stockle and EPIC for various purposes: these are two points on an S-shaped curve that predict proportion of some (0…1) as a function of proportion of some other units to the maximum(0…1). If we assume that the S-shaped curve is valid, any two points should work. However, it seems best to avoid the tails, and look for points where the response variable is between 20% and 80% of maximum.

A plant file requires about 3 to 5.5 Kb on disk.

3.4.1.5 Save file

File suffix: '.sav'.

GAPS can periodically save its state to disk file for later printing, plotting, or input to other programs. To get GAPS to do this, the user must first specify the days and hours within the day to save and second specify the names of the output files to which to save the output, using the menu option **S**cenario/ **O**utput files/ a_file (see following section on 'output files).

An output specification file requires about 1Kb on disk.

3.4.1.6 Cropping sequence file

File suffix: '.seq'.

This file is required if GAPS is to simulate more than one crop, either in sequence (relay cropping) or at the same time (intercropping or competition). It lists the crops by their corresponding plant files ('.plt'), and their starting and stopping year and day. It also specifies the competition model and options. See the Appendix for this file's format.

A cropping sequence file requires about 1Kb on disk.

3.4.2. Output files

GAPS can periodically save its state to disk file for later printing, plotting, or input to other programs. To get GAPS to do this, the user must first specify the days and hours within the day to save, and second specify the names of the output files to which to save the output, using the menu option **S**cenario/ **O**utput files/ a_file .

3.4.2.1. Simulation log and summary file

File suffix: '.det'.

Menu option: '**S**cenario/ **O**utput files/ **M**odel summary '

This file is a record of the model procedures, input files, times of simulation, critical events (e.g. dates at each crop growth stage) and final simulation output such as crop yield and water budget. Its contents may be printed using the menu option:

- **F**ile/ **P**rint/ **M**odel summary

3.4.2.2. Daily summary file

File suffix: '.sum'.

Menu option: '**S**cenario/ **O**utput files/ **D**aily summary'

This file contains daily output variables at the dates specified in the output specification. Its contents may be printed or graphed using the menu options:

- **F**ile/ **P**rint/ **D**aily Summary
- **G**raph/ **D**aily Summary

To see the variables included in this file, and their output formats, see source file 'simio.pas', procedure 'save_sum_data', along with the appropriate variable definitions in 'global.pas'. This file is read and printed by procedures 'printSum' of source file 'printout.pas'.

3.4.2.3. Hourly summary files

File suffixes:

- '.lay': soil layers
- '.sof': soil flux
- '.plf': plant flux
- '.clf' : climate flux

Menu option:
- '**S**cenario/ **O**utput files/ **S**oil layer (hourly)'
- '**S**cenario/ **O**utput files/ **S**oil flux (hourly)'
- '**S**cenario/ **O**utput files/ **P**lant flux (hourly)'
- '**S**cenario/ **O**utput files/ **C**limate flux (hourly)'

These files contain output variables at each hour specified in the output specification, for each day specified in the output specification. Their contents may be printed or graphed using the menu options:

- **F**ile/ **P**rint/ **L**ayers (soil)
- **F**ile/ **P**rint/ **S**oil flux
- **F**ile/ **P**rint/ **P**lant flux
- **F**ile/ **P**rint/ **C**limate flux
- **G**raph/ **L**ayers - soil
- **G**raph/ **S**oil flux
- **G**raph/ **P**lant flux
- **G**raph/ **C**limate flux

To see the variables included in these file, and their output formats, see source file 'simio.pas', procedures 'save_sim_lay_data', 'save_sim_sol_data', 'save_sim_plt_data', 'save_sim_cli_data', along with the appropriate variable definitions in 'global.pas'. These files are read and printed by procedures 'printLayer' , 'printPlf , 'printSof' , 'printClf of source file 'printout.pas'.

3.5. The GAPS simulator

This screen allows you to control the simulation. Typically you will (1) select input files (**S**cenario/ **I**nput files/ ...), (2) select the model procedures (**S**cenario/ **M**odel procedures)(F7), (3) specify the names of output files (**S**cenario/ **O**utput files/ ...), (4) specify the time of simulation , both its length in days and the duration of each time step (**S**cenario/ **T**ime ...), (5) specify the viewgraphs you want to see while the simulation is running (**S**cenario/ **R**untime graphs.../**S**elect graphs...)(F8), and (6) run the simulation (**R**un)(F9).

3.5.1. Run-time graphs

A major feature of GAPS is its ability to display a dynamic view of up to four (out of a possible 40) aspects of the plant-soil-atmosphere system as this is being simulated. You can select up to four run-time graphs at a time, using the **S**cenario/ **R**untime graphs.../**S**elect graphs...) menu option or by pressing F8.

There is also a second selection screen, because only 42 options will fit on a screen. You access this screen using the **S**cenario/ **R**untime graphs.../**M**ore graphs... menu option. The graphs selected with this screen also count towards the four-graph limit.

If more than four graphs are selected on the two screens, the first four (in order of appearance on the screens) will be shown and saved in the scenario file.

3.6. Scenarios

File suffix: '.set'

GAPS makes it easy to re-run simulations, by allowing the program user to save the set of model procedures, input and output file specifications, view graphs, simulation dates, and time step, in a *scenario* file.

The **S**cenario menu option **S**ave scenario is used to save the scenario; you are asked for a file name (the extension '.set' is automatically appended), and GAPS writes the simulation specifications to this file.

To re-run a simulation, select menu option **S**cenario /**L**oad scenario, and then select an existing scenario file from the list that is presented. This file will be read, setting the simulation options, and in addition the input files named in the scenario file will be read into memory. At this point menu option **R**un will start the simulation.

A scenario file requires about 2.5 Kb on disk.

4. Running GSB

Program 'gsb.exe' is a stripped-down version of GAPS, with the same simulation procedures as the interactive version, but a non-iteractive command-line user interface, intended for batch processing. This will be of interest to you if you need to run large numbers of simulations with only slightly changed parameters. For example, we used this program in a project to investigate the effects of climate change on US agriculture, using a large number of synthetic climates.

You run program 'gsb.exe' from the DOS command line. Without any parameters, it will show the correct usage, as follows:

```
>GSB
GAPS V 3.0 - Batch simulation
Usage: gsb <scenario file name> [-<filetype>[<sequence>]<filename>]*

Scenario file names have the default extension .set


Valid filetypes and their default extensions are:
 -c : climate  : .cli
 -p : plant    : .plt
 -s : soil     : .sol
 -l : location : .loc
 -x : save     : .sav
 -q : sequence : .seq
 -o : output   : .sum and .det, can't be overridden

Code -c may optionally be followed by a sequence number
For example:
 -c-2 : second year's climate
 -c by itself is equivalent to -c-1

For multiple plants, use a cropping sequence file
```

As this shows, GSB must be run with at least one parameter: the name of a simulation scenario file (GAPS '.set' file). The scenario file contains the names of the input and output files, as well as the simulation specifications, such as the model procedures. The easiest way to create a scenario file is with interactive GAPS: set up the simulation run as you wish, and then save the scenario file with the '**S**cenario / **S**ave scenario' menu.

In addition, GSB accepts up to six additional parameters, which allow you to override the names of input and output files given in the scenario file. This allows you to use a single scenario file to specify the model procedures and invariant data files, and a series of variant input and output files. If you don't specify file extensions, GSB will automatically use the same ones as GAPS, as shown above.

If any of the files which are listed as parameters for GSB are not in the current directory, you must specify their absolute or relative pathname as well as their name.

The optional parameters for GSB begin with a hyphen '-' and a key letter, which indicates the type of file name being specified, and then continue with the actual file name. For example:

```
-c-3aclimate.cli
```

specifies that, no matter what the scenario file says, the plant parameters for the third climate should be taken from file 'aclimate.cli'.

Here is an example of a DOS batch file that invokes GSB several times, illustrating some of the possibilities:

```
rem run GAPS for five climates, with separate outputs
rem 'quito.set' specifies model procedures, and the
rem   plant, soil, location, and save specification
gsb quito -ccuec0001 -oc1s1
gsb quito -ccuec0002 -oc2s1
gsb quito -ccuec0003 -oc3s1
gsb quito -ccuec0004 -oc4s1
gsb quito -ccuec0005 -oc5s1
rem now use a different soil at the same location, for the
rem  same set of climates.  This soil is described in file
rem  'j3.sol'
gsb quito -sj3 -ccuec0001 -oc1s2
gsb quito -sj3 -ccuec0002 -oc2s2
gsb quito -sj3 -ccuec0003 -oc3s2
gsb quito -sj3 -ccuec0004 -oc4s2
gsb quito -sj3 -ccuec0005 -oc5s2
rem there are now ten output files
```

You may also run GSB from another DOS program, using the 'exec' system call. This is a convenient way to run large numbers of simulations: the control program builds the command line for GSB and then executes it.

The batch simulations should not be used blindly. Use the interactive GAPS to explore several of the combinations and ensure that the simulations are doing what you expect. Then you can use GSB to run large numbers of closely-related simulations.

5. The GAPS models

The core of GAPS is the set of simulation modules. These provide a flexible framework for simulating the atmosphere-plant-soil system. Modules may be selected and combined according to (1) input data requirements, (2) components of the system that will be simulated , and (3) desired model of a system component.

We encourage GAPS users to *examine the source code* for each module. To this end, each section of this chapter begins with a list of the module's procedures, and the source file in which they may be found. Any text editor may be used to examine the source files; however, a good programmer's editor such as Epsilon[3] along with utilities such as Borland's text search utility 'grep' (supplied with the 'Turbo' languages) will greatly simplify such browsing.

5.1. Structure of the simulation driver

The simulation is time-driven, with procedures being called by the driver at various time steps: yearly, daily, and sub-daily (e.g.hourly). This latter time step is the one that can be controlled with the **S**imulation_Specs / **T**imeStep menu choice. The following psuedo-code shows the conceptual structure of the simulation (the actual code can be found in source file 'simula.pas', procedure 'manage'):

[3] Lugaru Software Ltd., Pittsburgh, PA, `http://www.lugaru.com/`

```
var
        year            : integer;
        day             : 1..366; { day in year }
        time_step       : 1..MaxTimeStep; { time step in day }

begin
        Begin_simulation_procedures;
        for year := first_year to last_year do begin
                Begin_year_procedures;
                for day := first_day to last_day do begin
                        Begin_day_procedures;
                        for time_step := 1 to num_of_time_steps do
                                TimeStep_procedures;
                        End_day_procedures
                end; { for each day }
                End_year_procedures
        end; { for each year }
        End_simulation_procedures;
end;
```

Each of procedures '..._procedures' (referred to as *subdrivers*) in turn contains a list of the simulation procedures to be carried out at that time resolution, depending on the model options selected by the user. For example, the subdriver for the elementary time step conceptually is as follows (the actual code is in source file 'simula', procedure 'TimeStep_Procedures'):

```
{ subdriver called at each time step }
procedure TimeStep_procedures;
        begin
        { atmospheric values }
                SolarAngle; Air_Temperature; Distribute_Precipitation;
        { soil temperature }
                case SoilTemp_method of
                        Variable: Numeric_Soil_Temperature_TimeStep;
                        Harmonic: Harmonic_Soil_Temperature_TimeStep;
                end;
        { potential ETP }
                case ETP_method of
                        Priestley    :      Priestley_Taylor_ETP_TimeStep;
                        Penman       :      Penman_ETP_TimeStep;
                        Linacre      :      Linacre_ETP_TimeStep;
                        Pan          :      Pan_ETP_TimeStep;
                end;
                PartitionETP;
                if ActiveCrops > 0 then begin  { at least one crop is active }
                { competition model }
                        TheCompModel^.TimeStep;
                { grow the plants }
                        for crop_i := 1 to global.MaxCrops do
                                if (TheCrop[crop_i] <> nil) and (TheCrop[crop_i]^.is_transpiring) then
                                        TheCrop[crop_i]^.TimeStep_Growth;
                { take up water }
                        case Uptake_method of
                                Potential_driven : WaterUptake_TimeStep;
                                Plant_available : SimpleWaterUptake_TimeStep;
                                Epic_water_uptake : EPICWaterUptake_TimeStep;
                        end; { Uptake_method }
                        end; { at least one crop is active }
        { soil water flow }
                case Flow_method of
                        Richards_Equation : Richards_Equation_TimeStep;
                        Tipping_Bucket : Tipping_Bucket_TimeStep;
                        Matric_Flux : Matric_Flux_Potential_TimeStep;
                end; { Flow_method }
        { summarize water relations for the time step }
                WaterBudget;
        end; { TimeStep_Procedures }
```

From this organization it follows that a particular simulation process, e.g. water flow in the soil, may be made up of several procedures, each corresponding to a time resolution. The several procedures for each module are linked through private 'state' variables shared by the procedures of the module.

The simulation driver also serves to summarize and report simulation variables, in `End_Daily_Procedures`.

5.1.1 DepthAndHeight

This procedure is called by `End_Daily_Procedures` at the end of each day. It adjusts the overall maximum rooting depth and maximum canopy height, by examining the state variables *RootingDepth*, *LRoot*, and *crop.CanopyHeight* for each crop, accessed by its pointer *TheCrop[]^*. The maximum or minimum, as appropriate, over all crops is used to set the state variables *sim.RootingDepth* (deepest rooting of any active crop), *sim.LRoot* (highest-numbered soil layer with roots of any crop), and *sim.CanopyHeight* (top of canopy for all crops). These state variables are then available for non-plant procedures, e.g. Penman ETP.

5.2. Atmospheric processes

This collection of modules simulate processes that occur exclusively in the atmosphere, i.e. computation of air temperature, solar angle and distribution of precipitation throughout the day. They provide values that are used throughout the other time-step processes, such as water flow, heat flow, photosynthesis, and evapotranspiration.

5.2.1. Function Air temperature

This function procedure is called at every time step to determine the mean air temperature (°C) for the time step from the daily values of minimum and maximum measured air temperature provided in the climate input file. The current air temperature is determined by fitting a sine function to the maximum and minimum, and then interpolating from this function over the time step. The daily maximum temperatures *clim.MaxTemp[]* (°C) are assumed to occur at 1500 hours and minimum temperatures *clim.MinTemp[]* (°C) at 0300 hours. To assure smooth interpolation, the minimum air temperature of the following day is used after 1500 hours, and the maximum of the preceding day prior to 300 hours. The formula used is:

$$\text{Air\_Temperature} := 0.5 * ((\text{MaxTemp} + \text{MinTemp}) + (\text{MaxTemp} - \text{MinTemp}) * \cos((\pi/12) * (\text{elapsed\_hour} - 15))) \quad (1)$$

where *elapsed_hour* is the elapsed hour in the day, 0..23. In this formula, the factor $\pi/12$ ensures a period of 24 hours, the factor *(elapsed_hour–15)* ensures a maximum temperature at 1500 hours, *(MaxTemp + MinTemp)/2* is the mean, and *(MaxTemp – MinTemp)/2* the amplitude of the cosine function.

5.2.2. Computation of solar angles

```
source file: atmoslib.pas
procedures: SolarAngle, SolarAngle_Init, SolarAngle_Daily,
            SolarAngle_Done
```

This module calculates the solar elevation angle for any given latitude, day of the year and hour of the day. First, the solar declination angle (angle which the sun's rays make with the earth's equatorial plane) is calculated knowing the day of the year. Then, the solar elevation angle is calculated on an hourly basis from the solar declination angle, latitude and time of day in relation to solar noon (when the sun is directly north or south of the point of observation). Knowing the solar elevation angle and the solar constant, i.e. the mean annual radiant flux density outside the earth's atmosphere and normal to the solar beam (1360 W m^{-2}), the theoretical irradiance above the atmosphere can be estimated.

The irradiance above the atmosphere *SpaRad* (W m^{-2}) at any time during the day can be calculated according to Campbell (1977):

$$SpaRad = SinElevAngle * SolarConstant \quad (1)$$

where: *SolarConstant* is the solar constant (1360 W m^{-2}), and *SSolElA* is the sine of the solar elevation angle at the time of the day.

In turn, the sine of the solar elevation angle *SSolElA* for any time of the day can be calculated as (Campbell 1977, 1985):

$$SinElevAngle = SolDcA * \sin(loca.Lat*0.01745) + \cos(loca.Lat*0.01745) * \cos(0.2618*(Time - loca.Tsn)/3600)) * CSolDcA \quad (2)$$

where: *SolDcA* is the sine of the solar declination angle, *CSolDcA* is the cosine of the solar declination angle, *loca.Lat* is the latitude (˚), *loca.Tsn* is the time of solar noon (sec, default 43 200), and *Time* is the time of the day (sec).

To determine solar noon accurately for a particular geographic location see Gates (1980), Appendix 4, p. 572.

The sine of the solar declination angle *SolDcA* can be computed according to Swift (1976) as:

$$SolDcA = 0.39785 * \sin [4.869 + 0.0172\ JulDay + 0.03345 \sin(6.224 + 0.0172 JulDay)] \quad (3)$$

where *JulDay* is the day-of-year number or Julian date, i.e. January 01 = 1, December 31 = 365.

Solar elevation angles are calculated for every time step of the day and, if the angles are positive, the theoretical solar radiation for the hour is calculated.

On a daily basis, `SolarAngle_Daily` estimates the total theoretical solar radiation for the day, *SpaceSum*. This value is needed to estimate the fractional cloud cover, in procedure `AtmosTrans`.

5.2.3. Distribution of precipitation over the day

```
source file: rainlib.pas
procedures: Distribute_Precipitation,
            Distribute_Precipitation_Init
```

This module allocates the daily precipitation and/or irrigation specified in the climate input file. Two variables contained in the location input file determine during which time interval water will be applied: *FirstRainHour* and *LastRainHour*. For example, to apply all the daily precipitation within one hour starting at 6:00 pm, specify *FirstRainHour* = 18 and *LastRainHour* = 19. Any application of water according to the climate file can be altogether withheld by setting *FirstRainHour* = *LastRainHour*.

Before distributing the precipitation, it is decreased by any simulated runoff.

In the initialization procedure, the rainfall hours are transformed into a partitioning factor *rain_fac*, which is the proportion of the daily rainfall to apply in each of the time steps when water is to be applied. Then during each time step, if the time step is one in which water should be applied, the rate of application *WInput* (kg m^{-2} s^{-1}) to the top layer of the soil *MinLayer* during the time step is calculated as:

$$WInput_{MinLayer} := rain\_frac * (clim.precip[real\_day] - SumRunoff)$$

where *Precip* is the daily precipitation, and *SumRunoff* is the simulated daily runoff, both expressed as kg m^{-2} or mm of water. *WInput* is now available for the various soil water procedures as the upper boundary rate.

5.3. Vapor exchange: evapotranspiration

GAPS includes four methods for computing potential evapotranspiration (ETP). Selecting between them is primarily on the basis of input data availability. In addition, a shared module partitions potential ETP into actual evaporation and transpiration.

5.3.1. Priestley-Taylor

```
source file: atmoslib.pas
procedures:: Priestly_Taylor_ETP
```

Priestley and Taylor (1972) developed an equation for calculating potential evapotrans–piration (ETP) for short vegetation, well-supplied with water under non–advective conditions. This equation incorporates a proportionality factor known as the Priestley-Taylor factor ALPHA which is multiplied by equilibrium evapotranspiration to predict ETP. The proportionality factor is supposed to compensate for the elimination of the aerodynamic component from the Penman equation (see the description of the Penman module). Despite the empirical nature of the proportionality factor ALPHA, the Priestley Taylor equation is based on reasonable physical grounds. It reduces input data requirements and can be used in situations where wind speed data and therefore aerodynamic resistances are not available. It also does not require knowing relative humidity/vapor density.

Priestley and Taylor experimentally determined an average value for ALPHA of 1.26, which is supported by the observation that the radiation component is generally four to five times as large as the aerodynamic component. An ALPHA very close to this value has since been confirmed by several investigators (Stewart and Rouse, 1977; Davies and Allen, 1973), when water supply to the evaporating surface is not a limiting factor. To improve estimates under advective conditions several attempts were made to estimate ALPHA from available climatic data such as air temperature (Jury and Tanner, 1975) or air temperature and net radiation (Nakayama and Nakamura, 1982). Others attempted to correlate ALPHA with soil surface moisture (Davies and Allen, 1973).The Priestley-Taylor equation has also been used to predict evapotranspiration from forests (Shuttleworth and Calde, 1979).

The Priestley-Taylor equation for potential evapotranspiration can be written as:

$$sim.ETP = loca.ALPHA * (NetRad-G) * (SSVD/ (SSVD+PSYCON)) / LAMB \quad (1)$$

where *sim.ETP* is the simulated potential evapotranspiration (kg/m2 s), *loca.ALPHA* is the Priestley-Taylor factor (1.08 to 1.50), *NetRad* is the net radiation (W/m2), *G* is the soil heat flux (W/m2), *SSVD* is the slope of the saturation vapor density function (kg/m3 K), *PSYCON* is the psychrometric constant (0.494 g m-3 K-1), and *LAMB* is the latent heat of vaporization of water (2450 J g-1, at 20 °C).

Net radiation *NetRad* (W m-2) is calculated as:

$$NetRad = LWR + SwAbs * clim.SolRad * 11.574 * (SpaRad / SpaceSum) \quad (2)$$

where *LWR* is net longwave radiation (W m-2), *soil.AS* is the short wave absorptivity of the soil surface (default 0.78), *clim.SolRad* is measured solar radiation (MJ m^{-2} d^{-1}) being converted to (W m^{-2}) by

multiplication with 11.574 (= 1/86400 s d^{-1} * 1,000,000), *SpaRad* is the theoretical solar radiation above the atmosphere for a given time step (W m^{-2}) [calculated in module `SolarAngles]`, and *SpaceSum* is the theoretical solar radiation for a day (W m^2) [calculated in module `SolarAngles]`.

Net longwave radiation *LWR* (W m-2) is calculated assuming the soil or plant surface is equal to air temperature:

$$LWR = (EA - ES) * ST * (AirTemp+273)^4 \quad (3)$$

where *EA* is the long-wave atmospheric emissivity, *ES* is the long-wave emissivity of the soil or canopy surface (constant 0.97), *ST* is the Stephen-Boltzmann constant (5.67E-08 W m-2 K-4), *AirTemp* is the air temperature (°C),being converted to (K) by adding 273.

Atmospheric emissivity can be calculated according to Campbell (1985) as:

$$EA = (1-0.84*Clouds) * (0.72+0.005 * AirTemp) + 0.84 * Clouds \quad (4)$$

where *Clouds* is the fractional cloud cover taking values between 0 and 1 and is calculated in module `Atmos_Trans`.

The slope of the saturation vapor density function *SSVD* (kg m-3 K-1), i.e. the change in saturation vapor density with a change in temperature, is given by Fuchs et al. (1978) as:

$$SSVD = SVD * (((LAMB*MW)/R) /AirTempK-1)/AirTempK \quad (5)$$

where *LAMB* is the latent heat of vaporization of water (2450 J g-1), *MW* is the molecular weight of water (0.018 kg mol-1), *R* is the gas constant (8.3143 J mol-1 K-1), *AirTempK* is the air temperature (K). and *SVD* is the saturation vapor density (g m-3). This latter is given by Campbell (1985) as:

$$SVD = (EXP(31.3716 - (6014.79 / AirTempK) - 0.00792495 * AirTemp K)) / AirTempK \quad (6)$$

5.3.2. Penman

source file: atmoslib.pas
procedures:: Penman_ETP, Penman_ETP_Init, Penman_ETP_Daily

The Penman-Monteith equation combines a vapor diffusion and energy budget approach to predict evapotranspiration from plant canopies. Monteith (1964) applied the Penman equation (1948) to crop canopies, arguing that the resistances to vapor diffusion from inside the leaves through the stomates, leaf boundary layer and canopy, could be incorporated into a single canopy resistance. Net radiation must be known, change in soil heat storage either known or assumed to be negligable, air temperature, vapor density, and windspeed must be known.

The Penman-Monteith equation may be applied to both soil and cropped surfaces. First we consider the computation of variables that are needed to predict evapotranspiration from either soil surfaces or crop canopies. These are found in procedure `Penman_ETP`.

The saturation vapor density *SVD* (g m-3) and the slope of the saturation vapor density function *SSVD* (kg m-3 K-1) are calculated as a function of air temperature, as described in the Priestly-Taylor method, above.

The actual vapor density *VD* (g m-3) is updated on a daily basis using one of three methods, in the following order of preference. First, if there is a measured value of vapor density for the day,

clim.VaporDens[real_day] in the climate file, this is used. If not, but if there is a measured value of relative humidity for the day, *clim.RelHumid[real_day]* in the climate file (which should be the value of relative humidity at the maximum temperature), *VD* is calculated multiplying the daily relative humidity by the saturated vapor density at the daily maximum temperature:

VD = clim.RelHumid[real_day] * SVD(clim.MaxTemp[real_day]) (1)

If neither of these values are known, we assume that the minimum daily air temperature is usually dew point temperature, and calculate *VD* using equation (6) in section 6.3.1.1 .

To apply the Penman-Monteith equation to a soil surface several variables must be calculated. These are found in sub-procedure `Soil_Res` of procedure `Penman_ETP_Daily`. First, the aerodynamic resistance to vapor transfer *RA* (s m-1) for a soil surface is calculated based on the theory of turbulent transport if a windspeed at a specified height is known. For a soil surface, the zero plane displacement *D* is assumed equal to 0 m. The momentum roughness parameter *ZM* is assumed equal to 0.01 m. The vapor roughness parameter *ZV* is assumed to be 20% of the height of the momentum roughness parameter. The calculation is thus:

RA = (LN((loca.WindHeight - D + ZV) / ZV) *
ln((loca.Windheight - D + ZM) / ZM)) /(K^2 * clim.Windspeed) (2)

where: *loca.Windheight* is the height above soil surface at which windspeed was measured (m), *D* is the zero plane displacement (m), *ZM* is the momentum roughness parameter (0.01 m), *ZV* is the vapor roughness parameter (0.2 * ZM), *K* is the von Karman constant (0.4), and *clim.Windspeed* is the measured windspeed (m s-1).

If windspeed is not known, the aerodynamic resistance to vapor transport is assigned a constant value of 90 s m-1. Also in this procedure, shortwave absorptivity *ABS* is assigned a value from the soil input file.

To apply the Penman-Monteith equation to a crop surface these same variables must also be determinated. These are found in sub-procedure `Crop_Res` of procedure `Penman_ETP_Daily`. First, an aerodynamic resistance for the boundary layer above the crop is calculated based on the same equation used for calculating the aerodynamic resistance of the boundary layer above a soil. However, in this case the height of zero plane displacement *D* is no longer zero but assumed to be 64% of the canopy height (m) assigned in the plant input file updated. The momentum roughness parameter *ZM* is assumed to be 13% of the canopy height and again the vapor roughness parameter *ZV* is assumed to be 20% of ZM (Campbell 1977). If windspeed is not known, then some assumed value, *plant.RA*, for the aerodynamic resistance is called from the plant input file. The canopy resistance can either be assigned and held constant or can be made a function of plant (leaf water potential), environmental (vapor density deficit, radiation, air temperature) or soil (soil water potential) factors, depending on the crop being modeled either in this procedure or the null sub-procedure `**Bulk Canopy Resistance`. Shortwave absorptivity is assigned a value from the plant input file.

The Penman-Monteith equation is applied in the sub-procedure `Penman_ETP`. First, net radiation *sim.NetRad* (W m-2) is calculated as the sum of absorbed shortwave radiation and net longwave radiation:

sim.NetRad := LWR + SwAbs * clim.solrad[real_day].value * 11.574 *
SpaRad/SpaceSum (3)

where *LWR* is net longwave radiation (W m-2), *SwAbs* is the shortwave absorptivity of the soil (*soil.SwAbs*) or crop (*plant.AS*), *clim.SolRad* is daily solar radiation (MJ m^{-2} d^{1}), being converted to (W m^{-2}) by multiplication with 11.574 (= 1/86400 s d^{-1} * 1000), *SpaRad* is irradiance above the atmosphere (W m-2), *SpaceSum* is theoretical solar radiation for the day (W m^{-2}).

Next, latent heat of evapotranspiration *LE* (J m^{-2} s^{-1})is calculated as

```
LE = (SSVD*(sim.NetRad  G)+(RO*(SVD-VD)
          /RA))/((PSYCON*(1+RC/RA))+SSVD)                    (4)
```

where *RO* is the specific heat capacity of air (1200 J m^{-3} K^{-1}), *RA* is the canopy resistance to heat transport (s m^{-1}), *PSYCON* is the psychrometric constant (0.459 g m^{-3} K^{-1}), *RC* is the canopy resistance to vapor transport (s m^{-1}),and *SSVD* is the slope of the saturation vapor density curve (g m^{-3} K^{-1}).

Latent heat of evapotranspiration is then converted into evapotranspiration *sim.ETP* (kg m^{-2} s-1) by dividing by the latent heat of vaporization.

```
sim.ETP = LE / (LAMB * 1000)                                 (5)
```

where *LAMB* is the latent heat of vaporization (2430 J g^{-1}), and multiplication by 1000 converts g into kg.

5.3.3. Pan

```
source file: atmoslib.pas
procedures:: Pan_ETP, Pan_ETP_Init
```

The procedure Pan_ETP uses daily measured values for pan evaporation provided in the climate input file (*clim.PanEV[]*) and distributes the evaporative demand during the day using a sine wave function.

A pan coefficient *loca.Kp* and a crop coefficient *loca.Kc* can be specified in the location input file. These are ratios of actual to measured evapotranspiration, which are needed because water evaporates from the pan differently than from the soil surface or a crop. Typical values of the pan coefficient are 0.67 – 0.81 (Shaw 1988, p. 56), but these may vary widely by season, even exceeding 1. Ideally the pan coefficient should be determined by long-term studies at the site, and the simulation should only be run for time periods with reasonably constant pan coefficients.

The crop coefficient *loca.Kc* further corrects (typically, reduces) the pan evaporation to account for a growing crop. If a crop is being simulated, this should be set to 1, as the crop model will adjust potential ETP to actual evaporation and transpiration. This coefficient is useful when no crop is being simulated, but it is assumed that there is a uniform crop cover at the site. Again, the value of crop coefficient should be determined experimentally.

Procedure Pan_ETP_Init uses the pan and crop coefficients to compute a time-invariant coefficient:

```
PanCoeff := 2.3 * loca.Kp * loca.Kc / secs_in_day            (2)
```

where *secs_in_day* is 86,400 s. This coefficient is used as the amplitude of the sine wave at the time-step resolution:

```
sim.ETP:= PanCoeff * clim.PanEV[real_day].value
          * (0.05 + POW(SIN(0.0175*7.5 * ellapsed_hour),4))   (2)
```

5.3.4. Linacre

```
source file: atmoslib.pas
procedures:: Linacre_ETP
```

This module uses a simple empirical formula (Linacre, 1977) to estimate potential evapotranspiration from mean daily air temperature, mean daily dew-point temperature, elevation and latitude. If dew-point temperature is unavailable, minimum daily temperature is used as an approximation. Thus this method of computing ETP has the least data requirements of the four methods found in GAPS.

The daily mean air temperature *TMean* (°C)is calculated as:

```
TMean := (clim.MinTemp[real_day] + clim.MaxTemp[real_day] / 2          (1)
```

where *clim.MinTemp[]* and *clim.MaxTemp[]* are the measured daily minimum and maximum air temperatures in °C.

Then, daily potential evaporation *ETPdaily* is calculated as:

```
ETPdaily:= ( 700 * Tm / (100 – ABS(loca.latitude)) + 15*(TMean – Td) )
           / (80 – TMean)                                              (2)
```

where: *Td* is the dew-point temperature (°C), *loca.latitude* is latitude (˚),*TMean* is mean temperature (°C), and *Tm* is a factor which compensates for elevation, computed as:

```
TMean + 0.006 * loca.elevation                                         (3)
```

where *loca.elevation* is the station elevation in m, from the location input file.

The daily evaporative demand *ETPdaily* is then distributed over the day using a sine function. The instantaneous rate of potential evapotranspiration *sim.ETP* (kg m^{-2} s^{-1}) is computed as:

```
sim.ETP :=  2.3 * ETPdaily * (0.05 + POW(SIN(0.0175*7.5 * ellapsed_hour),4))
           / secs_in_day                                               (4)
```

where *ellapsed_hour* is the elapsed time in a day (h) and *secs_in_day* is 86,400 s.

If the sine of the solar elevation angle is less than 0, *sim.ETP* is set to 0.

5.3.5. Partitioning evapotranspiration

```
source file: atmoslib.pas
procedures:: PartitionETP
```

This procedure is called by the simulation driver each time step, after the rate of potential ETP *sim.ETP* has been determined by one of the four methods listed above. It partitions the estimate of potential evapotranspiration *sim.ETP* into potential transpiration *sim.PotTrans* and potential soil evaporation *sim.PotEva* according to the fraction of solar radiation intercepted by the canopy.

This procedure first obtains *TransFrac*, the fraction of light absorbed by the total canopy, by a call to the competition model:

```
TransFrac := TheCompModel^.GetTransFrac
```

See the description of the competition module for the computation of this transmission fraction.

Using this transpiration fraction *TransFrac*, the simulated rate of ETP is partitioned as follows:

sim.PotTrans := sim.ETP * TransFrac (1)

sim.PotEva := sim.ETP * (1.0 – TransFrac) (2)

or equivalently,

sim.PotEva := sim.ETP – sim.PotTrans (2')

The potential transpiration is further divided by the competition module, into the potential transpiration of each active crop model.

5.4. Plant processes

The crop models in GAPS are arranged in a hierarchy, using the inheritance feature of the object-oriented language Turbo Pascal 7.0. There is a generic crop model *crop_model*, which defines the fundamental structure of all GAPS crop models, but which does not itself implement a complete crop model, and can not be selected by the GAPS user. Descended from this are four main crop models: *Stockle-Riha*, based on the work of Stockle & Campbell (1985) and adapted by I. Buttler and S. Riha (maize), D. Rossiter and S. Riha (wheat) and J. Phillips (fast growing tree), *SORKAM*, adapted by D. Rossiter and S. Riha from the work of Rosenthal *et al.* (1989), *EPIC*, adapted by P. Simoens and S. Riha from the work of Williams*et al*. (1989) and *CONSTANT* by P. Simoens. These in turn have sub-models, specialized by crop species (Stockle-Riha) or cultivar group (SORKAM).

In GAPS, crop model objects are created and destroyed *dynamically*, that is, during the execution (not the compilation) of the program. Access to the objects is by means of *pointer* variables, which contain a memory address of the object (not the object itself), and which must be *dereferenced* using the '^' Pascal operator to access the object itself.

5.4.1 Crop model: generic

```
source file: croplib.pas
public procedures:: Begin_Growth, Daily_Growth_Begin,
            TimeStep_Growth, Daily_Growth_End, End_Growth,
            TimeStep_Stress, open_det_file, write_summary,
            check_model
public functions: GetTransFrac, is_active, is_transpiring,
            has_matured
private procedures: grain_moisture, report_progress
```

This model implements the skeleton of a crop model, and in addition implements behavior that is shared by all the descendent crop models: status reporting based on plant and harvest date, exploitation of soil layers by roots, and grain drying.

Constructor Init

This procedure is called by the simulation driver whenever any crop object is created at run-time. Its purpose is to initialize the state of variables shared by all crop models, such as the accumulated and daily dry matter, rooting depth, etc.

Destructor Done

This procedure is called by the simulation driver whenever any crop object of is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object. Even though the Pascal procedure is empty, the compiler generates additional code to release the correct amount of memory from the heap.

Public procedure Begin_Growth

This procedure is called by the simulation driver at the beginning of crop growth. It logs this fact to the simulation summary file.

Public procedure Daily_Growth_Begin

This procedure is called by the simulation driver at the beginning of each day. Its only function is to determine the soil volume exploited by roots. As rooting depth (calculated by the various descendent crop models) increases, additional layers of the soil profile containing roots are activated. The root length density in each layer does not change from the values *plant.RootDens[]* specified in the plant input file. This means that once the rooting depth exceeds the upper boundary of a soil layer, this layer is instantly fully exploited by roots according to the fixed root root density.

Public procedure TimeStep_Growth

This procedure is called by the simulation driver at each time step when the crop is actively growing. It is called after determining the current soil temperature and potential evapotranspiration, but before the plant water uptake and soil water flow. Thus the model has a current estimate of potential ETP, but only the previous time step's estimates of root water potentials.

In the generic crop model, this procedure does nothing. It is a 'placeholder' for the procedure of the same name in descendent crop model object types.

Public procedure Daily_Growth_End

This procedure is called by the simulation driver at the end of each day, after the descendent crop model has finished simulating daily growth. It accumulates the total dry matter, and, if there is any grain accumulated on a matured grain crop, calls a procedure to simulate grain drying.

Public procedure End_Growth

This procedure is called by the simulation driver at the end of crop growth, i.e. at harvest or the end of the simulation. It logs this fact to the simulation summary file, and resets some state variables for the run-time graphs.

Public Function GetTransFrac

The proportion of light intercepted by the canopy for this crop as a monoculture is computed as a simple function of the leaf area index:

$$GetTransFrac := 1 - \exp(-0.5 * LAI) \quad (1)$$

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing crop. Note that when LAI=0, i.e. there is no crop, *GetTransFrac* is 0, i.e.all light reaches the ground; as LAI increases, *GetTransFrac* increases exponentially towards 1.

Public Function is_active

This function returns True if and only if the crop object is active, i.e. if the simulation driver should call any plant processes at all. The crop is assumed to be active between the planting and harvest dates specified in the plant input file.

Public Function is_transpiring

This function returns True if and only if the crop object is transpiring, i.e. actively growing, in which case the simulation driver will call the crop's 'GetTransFrac' function before partitioning potential evapotranspiration into evaporation and transpiration. In the absence of more specific information this generic crop model assumes that the crop is transpiring during its entire lifetime, i.e. between planting and harvest.

Public Function has_matured

This function returns True if and only if the crop object is a grain crop which has matured, i.e. grain filling has stopped, but grain drying is proceeding, in which case the generic crop model's *Daily_Growth_End* procedure will simulate grain drying. In the absence of more specific information this generic crop model assumes that the crop is not a grain crop, and so never matures, therefore this function always returns False.

Private procedure grain_moisture

This procedure is called by the generic crop model's *Daily_Growth_End* procedure at the end of each day for a mature grain crop, i.e. one in which the grain is drying on the plant, which is determined by a call to the active crop object's *has_matured* function. It follows the empirical formula of Gerik *et al.* (1988). A variable used only by this procedure, *GrainDryingET* (kg m^{-2}) accumulates the potential ETP since grain drying began at maturity:

$$\text{GrainDryingET} := \text{GrainDryingET} + \text{sim.PotET} \quad (1)$$

This sum is used to decrease the maximum grain moisture, i.e. at the beginning of maturity, *GrainMoistureMax* (kg water kg^{-1} grain dry matter), assumed to be 48%, by the empirical formula:

$$\text{GrainMoisture} := \text{GrainMoistureMax} * \exp(-0.00591 * \text{GrainDryingET}) \quad (2)$$

5.4.2. Crop model: Stockle-Riha

```
source file: stockle.pas
object name: SR_model
public procedures:: Daily_Growth_Begin, TimeStep_Growth,
            TimeStep_Stress, open_det_file, write_summary,
            Daily_Growth_End, End_Growth, PSTempFac
public functions:
private procedures: Water_Interception, Max_Photosynthesis,
            Limit_Photosynthesis, Light_Interception,
            Critical_Leaf_Water_Potential,
            Simple_Water_Stress, Dry_Matter_Accumulation,
            LimitRootingDepth
private functions:
```

This model is adapted from the simulation model developed by Stockle & Campbell (1985) to predict the effects of water stress on corn yield, and the earlier model by Stockle to predict the effect of water and nitrogen stress on wheat. Although there are differences between the two submodels, they share a common structure and some common procedure, which are grouped in this generic model. The generic model can

not be selected by the GAPS user; it merely serves as a common ancestor of the submodels (maize and wheat) in the crop model hierarchy.

Constructor Init

This procedure is called by the simulation driver whenever an object of type *SR_model* is created at run-time. Its purpose is to initialize the state of the crop model.

Destructor Done

This procedure is called by the simulation driver whenever an object of type *SR_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Daily_Growth_Begin

This procedure is called by the simulation driver at the beginning of each simulated day. It initializes the daily total photosynthesis *SumPS* and the daily accumulated stress index *StressIndex*, and then calls its ancestor object's procedure of the same name. It also computes the canopy resistance to vapor transport, *RA*, based on the windspeed if available, otherwise from a constant *plant.RA* taken from the plant file, as in the Penman-Monteith evapotranspiration method, for use in the critical leaf water potential method of computing water stress.

Public procedure TimeStep_Growth

This procedure is called by the simulation driver at each time step when the crop is actively growing. It is called after determining the current soil temperature and potential evapotranspiration, but before the plant water uptake and soil water flow. Thus the model has a current estimate of potential ETP, but only the previous time step's estimates of root water potentials.

This procedure calls various private procedures to carry out the actual simulation, namely `Water_Interception` to determine how much water is intercepted by the plant canopy (only if the 'Water Interception' simulation option was selected by the model user) , `Max_Photosynthesis` to determine the maximum photosynthetic rate and non-stressed resistances, either `Critical_Leaf_Water_Potential` (if the 'Critical Leaf Water Potential' simulation option was selected by the model user) or `Simple_Water_Stress` (otherwise) to determine a water stress coefficient, and `Limit_Photosynthesis` to limit photosynthesis by this coefficient. Finally, this procedure keeps a running total of the daily photosynthesis and stress factor.

Public procedure Daily_Growth_End

This procedure is called by the simulation driver at the end of each simulated day. It controls the major plant growth processes, by calling the following private procedures and functions:

- `Growth_Stages` : transition between growth stages
- `Dry_Matter_Accumulation` : accumulate dry matter
- `Yield` : accumulate grain.

This procedure also accumulates stress indices for reporting at the end of the simulation, and calls the generic crop object's end-of-day procedure.

Public procedure End_Growth

This procedure is called by the simulation driver at the end of crop growth, i.e. at harvest or the end of the simulation. This procedure sets the growth stage to 'none', to stop crop simulation, and calls the generic crop model's end growth procedure.

Public Function is_active

This function returns True if and only if the crop object is active, i.e. if the simulation driver should call any plant processes at all. The crop is assumed to be active between the planting and harvest dates specified in the plant input file, but in addition this function checks whether the crop simulation has been stopped by some other means, in which case the growth stage is 'none'.

Public Function is_transpiring

This function returns True if and only if the crop object is transpiring, i.e. actively growing, in which case the simulation driver will call the crop's 'GetTransFrac' function before partitioning potential evapotranspiration into evaporation and transpiration. In this model, the crop is transpiring only during the growth stages *vegetative, pollination, late lag, early filling,* and *senescing*.

Public Function has_matured

This function returns True if and only if this grain crop has matured, i.e. grain filling has stopped, but grain drying is proceeding, in which case the generic crop model's *Daily_Growth_End* procedure will simulate grain drying. In this model, this function is a direct translation of the *matured* growth stage.

Private procedure Water_Interception

This procedure is called at each time step if the 'Water Interception' simulation option was selected by the model user. It computes the amount of water intercepted by and evaporated from the canopy and the amount of throughfall, i.e. water reaching the soil surface. It adjusts the rainfall during the time step, so that less water is applied to the soil surface for soil water balance calculations.

If there is any rain during the time step, this procedure is called to compute the water intercepted by the canopy, the total amount of water on the canopy *CanopyWater* (kg m^{-2} ground) and the rate of throughfall to the soil *WInput[MinLayer]* (i.e. the adjusted precipitation, kg m^{-2} s^{-1}). The amount of water on the canopy is initially calculated as the amount of water previously on the canopy (at the end of the previous time step) plus the current time step's rainfall. The throughfall to the soil is computed as the difference between this amount and the amount that the canopy is storing on its surface. This amount is computed as the product of the *plant.Canopymax* (the amount of water potentially held per m^2 of surface area of leaves which is input in the plant file) and the current leaf area. So, if the canopy can store all this time step's water, no precipitation reaches the soil, otherwise, the overflow amount from the canopy reaches the soil.

Water can then be evaporated from the canopy. If the amount of potential transpiration (kg m^{-2} ground) is greater than or equal to the amount of water stored on the canopy *CanopyWater* (kg m^{-2} ground), then we assume that all the water on the canopy is evaporated, and that potential transpiration is reduced by this

amount, due to the energy needed to evaporate the water. Otherwise, the potential transpiration rate is set equal to zero and canopy water reduced by the amount evaporated.

Private procedure Light_Interception

In procedure Light_Interception, the sunlit and shadelit leaf areas are calculated and direct and diffuse photosynthetically active radiation, (PAR), interception is summed. It is only called in the daytime, i.e. when the solar elevation angle is positive. At night, photosynthesis is assumed to be zero and the stomatal resistance is set at a high value representative of closed stomates (3200 s m^{-1}).

This procedure begins by determining the amount of leaf area that is sunlit. An extinction coefficient *K* for a spherical (random) leaf angle distribution is calculated as a function of the sine of the solar elevation angle *SSolElA*, as follows:

$$K = 0.5 / SSolElA \quad (1)$$

This equation, as well as others for calculating extinction coefficients for different types of leaf inclination angles, is presented and discussed in Campbell (1977).

The sunlit leaf area index *LAISun* is then calculated as a function of the total leaf area and extinction coefficient:

$$LAISun = (1 - EXP(-K * LAI)) / K \quad (2)$$

where *LAI* is the leaf area index (m2 leaves m-2 ground), and *K* is the canopy extinction coefficient computed according to equation (1).

The shaded leaf area index *LAIShade* is computed as the difference between the total leaf and the sunlit leaf area:

$$LAIShade = LAI – LAISun \quad (3)$$

Following this, photosynthetically active radiation *PAR* (W m^{-2}) is calculated from the daily solar radiation, assuming that half of the short wave spectrum is PAR:

$$PAR = 0.5 * DailyRad * SpaRad \quad (4)$$

where *DailyRad* is today's proportion of theoretical solar radiation, and was computed in the `daily_growth_begin` procedure as:

```
DailyRad := (TheCompModel^.MySolRad(MyCropI) * MJd_to_W) / SpaceSum
```

where the function *MySolRad* is the portion of *clim.SolRad* (measured daily solar radiation) (MJ m^{-2} d^{-1}) allocated to this crop by the competition module, which is then converted to (W m^{-2}) by multiplication with 11.574 (= 1/86400 s d^{-1} * 1000), and *SpaceSum* is the theoretical solar radiation for a given day (W m^{-2}). Thus *DailyRad* is a proportion (0..1) of theoretical daily radiation.

Returning to equation (4)*SpaRad* is the theoretical solar radiation above the atmosphere during the time step (W m^{-2}), so that when this is multiplied by *DailyRad*, we have the radiation flux density to this crop during this time step (W m^{-2}).

This computed *PAR* (W m^{-2})is then used to calculate direct and diffuse photosynthetically active radiation *PARSun* (W m^{-2}) and *PARShade* (W m^{-2}), by the method of Norman (1982).

The direct PAR for sunlit leaves *PARSun* is assumed to be a function of the proportion of the total transmission coefficient of the atmosphere *TTrans* that is direct radiation, i.e. *(1 - DTrans/TTrans)*, multiplied by the canopy extinction coefficient *K* plus the proportion of the total transmission coefficient of the atmosphere that is diffuse radiation, i.e. *(DTrans/TTRans)* :

PARSun = PAR * (K * (1 - DTrans / TTrans) + DTrans / TTrans) (5)

The diffuse PAR for shaded leaves *PARShade* is calculated as a function of the proportion of the total transmission coefficient of the atmosphere *TTrans* that is the diffuse radiation coefficient multiplied by diffuse transmission coefficient of the canopy *Kd* plus the scattered irradiance within the canopy *ScIrr* :

PARShade = PAR * Kd * DTrans / TTrans + ScIrr (6)

The scattered irradiance *ScIrr* is estimated as:

$$ScIrr = (1-DTrans/TTrans) * PAR * 0.07 * (1.1 - 0.1*LAI) * e^{-SSolElA} \quad (7)$$

where *DTrans* is diffuse transmission coefficient of the atmosphere, *TTrans* is total transmission coefficient of the atmosphere, *PAR* is photosynthetically active radiation (W m^{-2}), *LAI* is the leaf area index (m^2 leaves m^{-2} ground), and *SSolElA* is the sine of solar elevation angle.

Private procedure Max_Photosynthesis

This procedure is called hourly to calculate the rate of photosynthesis as a function of leaf temperature and irradiance using a non-rectangular hyperbola (Thornley and Johnson, 1990). Sunlit and shadelit rates are calculated separately and summed.

First a maximum photosynthetic rate given the current leaf temperature (assumed to be equal to air temperature) is computed using *Pmax* (mg CO2 $m^{-2}s^{-1}$), the maximum rate of photosynthesis, *Topt*, the optimal temperature for photosynthesis, and *Tmin*, the minimum temperature at which photosynthesis is possible, in degrees c. These parameters are all species specific and are defined by the user in the plant input file. The function `PSTempFac` is called to compute the temperature factor which varies between zero and 1 and is multiplied by Pmax to get the temperature corrected photosynthetic rate. If the air temperature is between *Tmin* and *Topt*, *PSTempFac* increases quadratically:

$$PSTempFac = (T-T_{min}).(2T_{opt}-T-T_{min}) / (T_{opt} - T_{min})^2$$

Below the minimum there is zero photosynthesis and above the optimum it is assumed to be constant at the maximum. Thus, the possibly harmful effect of exceedingly high air temperatures are not directly reflected in the photosynthesis routine.

Gross sunlit and shadelit photosynthesis (in mg $m^{-2}s^{-1}$) is computed as a function of the photosynthetic parameters input by the user, *plant_P_curve*,_, *plant_P_effic*,_, the modified maximum photosynthetic rate, *Pmax* and the leaf area in sun or shade:

PSSun := 1/2_[_PARSun+Pmax-[(_PARSun+Pmax)^2-4__PARSun Pmax]$^{1/2}$]

where the *P_curve* ,_, defines the curvature of the relationship between light and gross photosynthesis as the maximum rate, *Pmax*, is approached, and *P_effic*, _, defines the initial slope of the relationship, or the efficiency with which light is converted to photosynthate. After converting mg to g, crop photosynthesis for the hour is then obtained by multiplying the photosynthetic rate for sunlit leaves *PSSun* (g m^{-2} s^{-1}) by the sunlit leaf area index *LAISun* and adding to this the photosynthetic rate for shaded leaves *PSShade* (g m^{-2} s^{-1}) multiplied by the shaded leaf area index *LAIShade*, this rate then being multiplied by the time step*time_step* (s) to obtain a quantity:

PS = (PSSun*LAISun + PSShade*LAIShade) * time_step (10)

In the final section of this procedure, a non-stressed stomatal resistance to vapor loss is calculated for both shaded and sunlit leaves, using Ohm's equation to model CO_2 diffusion from the atmosphere into the leaves:

PSxxx = (COext – COint) / (RACO + RESxxx) (11)

where: *xxx* refers to either *Sun* or *Shade*, *ResSun* is the sunlit stomatal resistance to water vapor transfer (s m-1), *ResShade* is the shadelit stomatal resistance to water vapor transfer (s m-1), *COext* is the atmospheric CO_2 concentration (0.54 g m-3), *COint* is the CO_2 concentration internal to the leaf (assumed here to be 0.20 g m-3), *RaCO* is the leaf boundary layer resistance to CO_2 transfer, *PSSun* is the photosynthesis rate of sunlit leaves (g m-2 s-1), and *PSShade* is the photosynthesis rate of shadelit leaves (g m-2 s-1).

This equation is rearranged to solve for *RESxxx* and also divided by 1.65 to convert from CO_2 to water vapor resistance (Campbell 1977):

ResSun = ((COext – COint) / PSSun – RaCO) / 1.65 (12)

ResShade = ((COext – COint / PSShade – RaCO) / 1.65 (13)

To obtain the whole canopy non-stress resistance to water vapor transfer, the sun and shade lit resistances are weighted according to the proportion of sun and shade lit leaf area and added in parallel (Stockle and Campbell 1985):

NonStressRes = LAI / (LAISun / ResSun + LAIShade / ResShade) (14)

This is an optimization approach to predicting water vapor loss from a plant canopy, since there is an implicit assumption that the plant only opens its stomates the amount necessary to achieve maximum photosynthesis and is never losing more water than required to achieve this rate.

Private procedure Limit_Photosynthesis

This procedure is called every hour to limit the photosynthetic rate *PS* (g m^{-2} s^{-1}) calculated in procedure `Max_Photosynthesis`, according to the stress factor *PSStressFact* calcaulated in procedure `Critical_Leaf_Water_Potential` (if the 'Critical Leaf Water Potential' simulation option was selected by the model user) or `Simple_Water_Stress` (otherwise), as follows:

PS := PS * PSStressFact

This procedure is redefined in the maize model (see below) to also stress the LAI.

Private procedure Critical_Leaf_Water_Potential

This procedure is called at each time step to compute the stress coefficient *PSStressFact* if the 'Critical Leaf Water Potential' simulation option was selected by the model user. It is only called whenever there is simulated potential transpiration, i.e. *sim.PotTrans* is greater than zero. It uses an empirically derived equation that relates decreases in leaf water potentials to increases in stomatal resistance. The increase in stomatal resistance contributes to a simulated decrease in actual transpiration *ActTrans* below potential transpiration *sim.PotTrans*. Using Ohm's Law to model water flow through plants:

$$\text{ActTrans} = (\text{RootWP} - \text{LeafWP}) / \text{LeafRes} \quad (1)$$

a new leaf water potential *LeafWP* (J kg^{-1}) based on the simulated actual transpiration, is calculated. This leaf water potential, in turn, is used to calculate a new stomatal resistance, which then results in a new simulated *ActTrans* and *LeafWP*. This process is repeated until *LeafWP* changes less than 10 J kg^{-1} with successive iterations. The details of this iterative process are now presented.

Initially, the saturation vapor density *SVD* (g m^{-3}) is calculated according to Campbell (1981) and the slope of the saturation vapor density function *SSVD* (g m^{-3} K^{-1}) is calculated according to Fuchs et al. (1978), exactly as in the Priestly-Taylor ETP equation.

Next, the stressed stomatal resistance *StressRes* is calculated, following the work of Fisher et al. (1981), using values for corn given by Stockle and Campbell (1985):

$$\text{StressRes} = \text{NonStressRes} * (1 + (\text{LeafWP} / \text{plant.CriticalLeafWP})^{\text{plant.CLWP\_Power}}) \quad (2)$$

where *NonStressRes* (s m-1) is the non-stressed stomatal resistance calculated in the procedure `MaxPhotosynthesis`, *LeafWP* is the simulated leaf water potential (J kg-1) calculated in the procedure `Water_Uptake`, *plant.CriticalLeafWP* is a species dependent, empirically derived value (J kg-1), and *plant.S* is a species dependent, empirically derived constant.

The reduction in transpiration that occurs due to an increase in stomatal resistance is related to the contribution of stomatal resistance to the total resistance to vapor transport given by Campbell (1977). The ratio of evaporation from leaves with non-stressed and stressed resistances *PSStressFact* is determined by the following relation, from Campbell (1977):

$$\text{PSStressFact} := (\text{SSVD} + \text{PSCON} * (\text{NonStressRes} + \text{RA})/\text{Re}) / (\text{SSVD} + \text{PSYCON} * (\text{StressRes} + \text{RA})/\text{Re}) \quad (3)$$

where *SSVD* is the slope of the saturation vapor density function (g m^{-3} K^{-1}), *PSYCON* is the psychrometer constant (0.494 g m^{-3} K^{-1}), *RA* is the daily crop boundary layer resistance (s m^{-1}) calculated at the beginning of the day in `Daily_Growth_Begin`, and *Re* is the combined resistance for convection and longwave radiation heat transfer from the canopy surface, assumed to be a constant 40 s m^{-1} (Campbell 1977).

Knowing the transpiration ratio *PSStressFact*, the simulated actual transpiration *ActTrans* can be calculated as:

$$\text{ActTrans} = \text{PSStressFact} * \text{sim.PotTrans} \quad (4)$$

Then a new simulated leaf water potential *LeafWP* (J kg^{-1}) is calculated as:

$$LeafWP = RootWP - ActTrans * LeafRes \quad (5)$$

where *RootWP* is the root water potential calculated in the module `WaterUptake` (J kg^1), and *LeafRes* is the leaf resistance to liquid water transport (m^4 s^{-1} kg^{-1}) and is assumed to be 2 x 10^6 (Campbell 1985).

Finally, the ratio of actual to potential transpiration *sim.TRatio* (dimensionless) is calculated, using the adjusted value of actual transpiration from the preceding parts of the procedure, as:

$$sim.TRatio := sim.ActTrans / sim.PotTrans \quad (6)$$

Private procedure Simple_Water_Stress

This procedure is called at each time step to compute the stress coefficient *PSStressFact* if the 'Critical Leaf Water Potential' simulation option was <u>not</u> selected by the model user. The stress factor is set equal to the transpiration ratio *TRatio* of the actual transpiration rate *ActTrans* (kg m^{-2} s^{-1}) to the potential transpiration rate *PotTrans* (kg m^{-2} s^{-1}), which is calculated in the `time_step_procedures` of the simulation driver:

$$sim.TRatio := sim.ActTrans / sim.PotTrans \quad (1)$$

Private procedure LimitRootingDepth

This procedure is called by submodels, once they have determined a proposed rooting depth, to adjust this depth for two things: sowing depth and the user-defined maximum. First, the rooting depth predicted by the submodels is increased by the sowing depth specified in the 'plant' input file:

$$RootingDepth := RootingDepth + plant.SowingDepth \quad (1)$$

Then, if the depth is greater than another user input in the 'plant' file, namely *plant.MaxRootingDepth*, it is limited to that depth. This would be appropriate if the user had field observations of a maximum, or wanted to artificially limit rooting depth to certain horizons to simulated water stress.

5.4.2.1. Crop model: Stockle-Riha maize

```
source file: stockle.pas
object name: SRm_model
public procedures:: Daily_Growth_End, End_Growth,
            open_det_file, check_model, PSTempFac
public functions: GetTransFrac, PSTempFac, is_active,
            is_transpiring, has_matured
private procedures: Limit_Photosynthesis,
            Max_Photosynthesis, Dry_Matter_Accumulation,
            Yield, GetTransFrac
private functions: Growth_Stages
```

This submodel implements specific procedures for modelling the maize crop.

There are several differences between the original model and its implementation in GAPS, which are summarized in the following table.

Process / Parameter	Current Implementation	Stockle & Campbell (1985)
Water Stress Correction on partitioning of dry matter	(not included)	
Root resistances as a function of time	(not included)	
Root Density as function of root dry matter and thermal time	(not included)	
Critical leaf water potential	-1400 J kg^{-1} (1)	-1800 J kg^{-1}
Conversion factor photosynthesis to dry matter	0.40 (2)	0.46 – 0.50 (3), 0.31–0.34 (4)
Exponent of F for reduction of photosynthetic rate	1.0	0.8 - 1.2

(1) : after Stockle (1983)
(2) : Monteith (1981)
(3) : during early vegetative phase
(4) : during late vegetative phase

A key concept of this model is the growth stage. The plant is considered to be in one of a set of mutually-exclusive stages, from planting through harvest. The stages control or modify various behaviors of the plant, e.g. the rate of dry matter production. The stages are listed in the enumeration data types *SRm_GrowthStages*, and are: pre-emergence, vegetative, pollination, late lag, early grain filling, senescing (i.e. late grain filling), and mature. The plant moves from stage to stage according to accumulated heat units and days, as explained in private function `Growth_Stages`.

Constructor Init

This procedure is called by the simulation driver whenever an object of type *SRm_model* is created at run-time. Its purpose is to initialize the state of the crop model.

Destructor Done

This procedure is called by the simulation driver whenever an object of type *SRm_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Daily_Growth_End

This procedure is called by the simulation driver at the end of each day. It first accumulates a stress factor which is used to limit LAI. It then calls private function `Growth_Stages` to accumulate degree-days and possibly move to a new growth stage. Then the current day's water stress index (accumulated during the day in the `Critical_Leaf_Water_Potential` or `Simple_Water_Stress` procedures) is added into the current growth stage's cumulative stress index; these will be used to determine grain numbers at flowering.

If the crop has emerged but not yet matured, private procedure `Dry_Matter_Accumulation` is called to simulate the conversion of photosynthate into dry matter. This is simulated by calling private procedure `Yield`.

Finally, the `Daily_Growth_End` of the generic Stockle-Riha model is called to accumulate a stress index.

Public Function GetTransFrac

The proportion of light intercepted for this crop as a monoculture is computed according to Stockle (1985) as:

$$\text{GetTransFrac} := 1 - \exp(-0.823 * \text{LAI} + 0.0286 * \text{LAI}^2) \quad (1)$$

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing maize crop.

Public Function PSTempFac

This procedure computes a correction factor for the effects of air temperature on photosynthetic rate of maize. It assumes that the leaf temperature *LeafTemp* (°C) equals air temperature (argument *AirTemp*), and uses an equation from Stockle and Campbell (1985) derived from data of Hofstra and Hesketh (1969):

$$\text{PSTempFac} = -1.37893 + 0.184573*\text{LeafTemp} - 7.6341\text{E-}03 * \text{LeafTemp}^2 + 1.98485\text{E-}04 * \text{LeafTemp}^3 - 2.15152\text{E-}06 * \text{LeafTemp}^4 \quad (1)$$

Private Function Growth_Stages

This function is called at the end of each day, and it returns the current growth stage, which may be changed by the passage of time or the accumulation of heat units.

First, if the plant has not yet emerged, and forty days have passed, it is assumed that the seed has rotted in the ground. A message is written to the simulation summary file, and the growth stage is set to 'none', which stops the crop simulation.

Next, the function checks whether the plant has been killed by frost. This occurs when the daily minimum temperature is below –2.2°C, and the growing point is above the ground. If the plant is killed, and there has been no grain accumulation, the growth stage is set to 'none', which stops the crop simulation. If there is any grain, the growth stage is set to 'mature', so that grain drying can continue to be simulated.

The heart of this procedure is the computation of heat units, or accumulated thermal time as implemented in the corn growth model of Stockle and Campbell (1985) and based on data by Coelho et al. (1980) The

heat units *DD* accumulated during each day are computed as a piecewise continuous function of mean daily air temperature, truncated to the next-lower integral degree °C:

```
case trunc(TMean) of
        6..20 : DD := 0.027 * TMean – 0.162;
        21..27 : DD := 0.086 * TMean – 1.41;
        28..32 : DD := 1;
        33..43 : DD := –0.083 * TMean + 3.67;
        else DD := 0;
end; { case }                                                    (1)
```

This function allows for maximum heat units in the range 28–32°C, correspondingly less heat units from 6–27°C and 33–43°C, and no heat units (i.e. effectively no plant development) above and below this range.

The daily heat units are added into the accumulated heat units since emergence, *AccDD*. Then, the accumulated heat units are compared to the number of units necessary to move to the next growth stage, which of course is dependent on the current growth stage. These 'transition' heat units are stored in the constant array *AccDD_to[growth_stage]*. (These are the 'crop coefficients' of the plant data input file, and are read in when the crop is initialized, in the constructor *SR_model.Init*.)

However, the transition from growth stage 'senescing' to 'mature' is not based on heat units, since the plant is not actively growing; instead this transition is effected when the simulated LAI drops below 0.01; LAI is adjusted in private procedure `Dry_Matter_Accumulation`. In addition, even if the plant is not yet in the senescing stage, but is past the vegetative stage, it will be forced to maturity if the LAI has dropped below 0.01 because of water stress.

If the growth stage is past emergence but before senescence, the number of days since emergence is incremented. This number of days is used in the piecewise dry matter partitioning function in `Dry_Matter_Accumulation`.

Finally, the 'standard' base-10°C growing degree days is computed. This is not used in the simulation, but may be shown on a graph. The actual function is the 'modified' growing degree days, based on Barger (1969), which constrains the maximum temperature to 30°C and the minimum to 10°C.

Private procedure Max_Photosynthesis

Both sunlit and a shaded rates of photosynthesis (*PSSun and PSShade*) (g m^{-2} s^{-1}) are then calculated assuming that photosynthesis is related to PAR according to an equation given for maize by Hesketh and Baker (1969). This rate is then modified for leaf temperature, using a temperature factor *TempFac*, which is obtained by calling the correct virtual procedure for the submodel. The code:

```
TempFac := PSTempFac(AirTemp)
```

calls one of the virtual functions, e.g. *SRw_model.PSTempFac* for wheat, which returns the temperature factor. See the submodels for a description of the computation of the temperature correction factors. This factor, which depends on leaf temperature, is further multiplied by another linear factor that does not vary with temperature, *TempCoeff*. This is crop-specific, and is set by the constructor of each Stockle-Riha submodel. For maize, this is (6.2527E-05 / 0.7), and for wheat (1.6 * 2.23583E-05). So the product of the linear factors is on the order of 1. Another crop-specific factor is *TempPower*, which is the exponent to which PAR is raised. This is set by the constructor of each Stockle-Riha submodel. For maize, this is 0.507578, and for wheat 0.695407.

Given these factors, PAR is converted to PS by these equations:

$$PSSun = TempFac * TempCoeff * PARSun^{TempPower} \quad (8)$$

$$PSShade = TempFac * TempCoeff * PARShade^{TempPower} \quad (9)$$

Because of the way the factor, coefficient, and power are constructed, photosynthesis is somewhat less than PAR.

Crop photosynthesis for the hour *PS* (g m^{-2}) is then obtained by multiplying the photosynthetic rate for sunlit leaves *PSSun* (g m^{-2} s^{-1}) by the sunlit leaf area index *LAISun* and adding to this the photosynthetic rate for shaded leaves *PSShade* (g m^{-2} s^{-1}) multiplied by the shaded leaf area index *LAIShade*, this rate then being multiplied by the time step*time_step* (s) to obtain a quantity:

$$PS = (PSSun*LAISun + PSShade*LAIShade) * time\_step \quad (10)$$

In the final section of this procedure, a non-stressed stomatal resistance to vapor loss is calculated for both shaded and sunlit leaves, using Ohm's equation to model CO_2 diffusion from the atmosphere into the leaves:

$$PSxxx = (COext - COint) / (RACO + RESxxx) \quad (11)$$

where: *xxx* refers to either *Sun* or *Shade*, *ResSun* is the sunlit stomatal resistance to water vapor transfer (s m-1), *ResShade* is the shadelit stomatal resistance to water vapor transfer (s m-1), *COext* is the atmospheric CO_2 concentration (0.54 g m-3), *COint* is the CO_2 concentration internal to the leaf (assumed here to be 0.20 g m-3), *RaCO* is the leaf boundary layer resistance to CO_2 transfer, *PSSun* is the photosynthesis rate of sunlit leaves (g m-2 s-1), and *PSShade* is the photosynthesis rate of shadelit leaves (g m-2 s-1).

This equation is rearranged to solve for *RESxxx* and also divided by 1.65 to convert from CO_2 to water vapor resistance (Campbell 1977):

$$ResSun = ((COext - COint) / PSSun - RaCO) / 1.65 \quad (12)$$

$$ResShade = ((COext - COint / PSShade - RaCO) / 1.65 \quad (13)$$

To obtain the whole canopy non-stress resistance to water vapor transfer, the sun and shade lit resistances are weighted according to the proportion of sun and shade lit leaf area and added in parallel (Stockle and Campbell 1985):

$$NonStressRes = LAI / (LAISun / ResSun + LAIShade / ResShade) \quad (14)$$

This is an optimization approach to predicting water vapor loss from a plant canopy, since there is an implicit assumption that the plant only opens its stomates the amount necessary to achieve maximum photosynthesis and is never losing more water than required to achieve this rate.

Private procedure Limit_Photosynthesis

This procedure is called every hour to limit the photosynthetic rate *PS* (g m^{-2} s^{-1}) calculated in procedure `Max_Photosynthesis`, according to the stress factor *PSStressFact* calcaulated in procedure `Critical_Leaf_Water_Potential` (if the 'Critical Leaf Water Potential' simulation option was selected by the model user) or `Simple_Water_Stress` (otherwise), as follows:

```
PS := PS * PSStressFact
```

This procedure also accumulates a factor *LAIStressFact* which is used in daily procedure `Dry_Matter_Accumulation` to accelerate the decline in leaf area during the pollination, late lag, and early grain filling growth stages, if actual transpiration is less than 90% of potential transpiration::

LAIStressFact := LAIStressFact + 0.8 * sim.TRatio (1)

where *sim.TRatio* is the ratio of actual to potential transpiration.

Private procedure Dry_Matter_Accumulation

This procedure is called at the end of each day to compute the amount of new dry matter produced during the day. In this model, daily dry matter accumulation *DryMatter* (kg m^{-2} d^{-1}) is directly related to the daily photosynthesis *SumPS* (g m^{-2} d^{-1}) using average conversion factor *PSFact* which depends on the growth stage:

DryMatter := PSFact * (SumPS / 1000) (1)

Daily photosynthesis *SumPS* is accumulated throughout the day in procedure `Max_Photosynthesis`, based on the time step's photosynthetic rate, and possibly reduced by water stress in procedure `Limit_Photosynthesis`.

The conversion factor is assumed to be 0.40, as suggested by Monteith (1981), during the vegetative growth stage. During the pollination, late-lag, and early filling growth stages, this factor is assumed to be 0.33, after which it is zero, i.e. no dry matter is accumulated during senescence.

The daily accumulated dry matter is partitioned into top and root dry matter following the method of Foth (1962). A partitioning factor *PartFactor* is calculated based on the number of days since plant emergence *DAE* (days) and the partitioning stage *SR_PartitioningStage* maintained by procedure `Growth_Stages`:

```
case SR_PartioningStage of:
        early : PartFactor := 0.125 * DAE;
        mid : PartFactor := 2;
        late : PartFactor := 0.135 * DAE;
end { case } (2)
```

This factor is in turn used to compute a the ratio *PartRatio*:

PartRatio := PartFactor / (1 + PartFactor) (3)

PartRatio is then used to partition the daily gain in dry matter into top dry matter *TopDryMatter* (kg m^{-2}) and root dry matter *RootDryMatter* (kg m^{-2}):

```
TopDryMatter := DryMatter * PartFactor
RootDryMatter := DryMatter * (1 – PartFactor) (4)
```

or equivalently,

RootDryMatter := DryMatter – TopDryMatter (4')

Rooting depth *RootingDepth* (cm) is empirically related to root dry matter (Acevedo, 1975), using a piecewise function of root dry matter by current depth:

RootingDepth := 1174.8 * AccRootDryMatter;
if RootingDepth > 82 then RootingDepth := 7.28 + 121.3 * AccRootDryMatter;
if RootingDepth > 165 then RootingDepth := 112.6 + 70.4 * AccRootDryMatter; (5)

The rooting depth is corrected for the depth to which the seed was sowed:

RootingDepth := RootingDepth + plant.SowingDepth (6)

No correction for water stress on partitioning as included in Stockle and Campbell (1985) was used for these simulations. When the maximum rooting depth *plant.MaxRooting–Depth* as specified in the plant input file is reached, the roots are assumed to stop.

The development of leaf area *LAI* (m^2 leaf m^{-2} ground) depends on the growth stage. During the vegetative stage, the relationships originally included in Stockle and Campbell's model, and based on independent data sets of Acevedo (1975) have been synthesized to the following conversion from above-ground dry matter *AccTopDryMatter* (kg m^{-2}):

LAI := 7.6 * AccTopDryMatter (7)

Once vegetative LAI reaches a set value (4.1 in this model), the expansion of leaf area slows down dramatically from the vegetative rate of 7.6 m^2 leaf kg^{-1} dry matter, continuing at a rate of 0.5 m^2 leaf kg^{-1} dry matter:

LAI := 3.85 + 0.5 * AccTopDryMatter (8)

These two equations intersect at LAI = 4.1197, but are not continuous in any derivative.

Development of LAI during the vegetative stage is not directly affected by water stress, except as this affects photosynthesis and dry matter production. By contrast, during the pollination, late lag, and early grain filling growth stages, leaf area is reduced as a function of attained leaf area index and thermal time since silking (Dale et al., 1980), with a correction for water stress as proposed by Stockle and Campbell (1985). The more water stress has been accumulated during the vegetative phase, the more rapid will be the decline of leaf area. The relation is:

LAI := LAI_at_silking – (0.035 * FillingDD * (1 + AccLAIStressFact) (9)

where *AccLAIStressFact* is the accumulation (in procedure `Daily_Growth_End`) over the day of each time step's *LAIStressFact*, calculated in procedure `Limit_Photosynthesis`, and *FillingDD* is the number of growing degree days since silking, calculated in procedure `Growth_Stages`.

During the senescing growth stage, leaf area declines from its maximum to zero as a function of time (Dale et al., 1980):

LAI := LAI_at_leaf_maturity – (0.15 * (real_day – leaf_maturity_day)) (10)

where *leaf_maturity_day* is the day of the year on which the plant began to senesce.

Private procedure Yield

This procedure is called at the end of each day, after the daily dry matter accumulation has been determined, to compute the current grain dry matter and moisture.

The action of this procedure depends on the growth stage. If the plant is in the 'pollination' stage, the number of kernels per plant may be decreased if there is sufficient water stress, *PSStressFact*. If this is ≤ 0.25 on a scale of 1 (no stress) to 0 (complete stress), the kernels per plant are decreased by 10%. Note that this will happen each day there is this much stress. The stress factor is computed by one of the procedures `Simple_Water_Stress` or `Critical_Leaf_Water_Potential`, depending on whether the user selected the 'critical leaf water potential' method of limiting photosynthesis.

If the plant is past the lag stage, but not yet mature, it is considered that photosynthate is actively being translocated to the grain. The daily increase in grain dry matter *DailyYield* (kg m^{-2} d^{-1}) is then calculated as:

$$\text{DailyYield} := \text{Kernels\_Per\_Plant} * \text{Kernel\_Growth\_Rate} * (\text{plant.PlantDensity} * 10^{-4}) \quad (1)$$

where the factor 10^{-4} converts from hectares to m^2, the *Kernels_Per_Plant* is an input parameter in the plant file, possibly reduced by water stress as explained above, the *PlantDensity* (plants ha^{-1}) is an input parameter in the plant file, and the *Kernel_Growth_Rate* is an input parameter in the plant file, converted to kg d^{-1} in the constructor procedure `SR_Model.Init`. The daily grain yield is accumulated into a total grain yield:

$$\text{AccYield} := \text{AccYield} + \text{DailyYield} \quad (2)$$

and this is limited by the maximum harvest index, which is an input parameter in the plant file, and the current total dry matter:

$$\text{If } (\text{AccYield} / \text{AccTopDryMatter}) > \text{HarvestIndex then } \text{AccYield} := \text{AccTopDryMatter} * \text{HarvestIndex} \quad (3)$$

Thus the accumulated yield on any given day can never exceed the harvest index times the accumulated dry matter.

5.4.2.2. Crop model: Stockle-Riha wheat

```
source file: stockle.pas
object name: SRw_model
public procedures:: Daily_Growth_End, End_Growth,
            open_det_file,
public functions: GetTransFrac, PSTempFac, is_active,
            is_transpiring, has_matured, check_model
private procedures: Dry_Matter_Accumulation, Yield,
            Winter_kill, Max_Photosynthesis
private functions: Growth_Stages, Grain_Number
```

This submodel implements specific procedures for modelling a wheat crop. It uses the generic Stockle-Riha procedures to simulate photosynthesis and water stress. The wheat and maize models differ in their growth stages, dry matter accumulation, and grain yield formation.

This model can account for winter kill but does not account for vernalization.

This model has several important differences from the model presented in Stockle's PhD thesis. Most notably, we do not here attempt to model the nitrogen status of the soil or plant. Any equations involving

nitrogen have been simplified by assuming that N is non-limiting. Other simplifications are: (1) degree-days are accumulated daily from the average temperature, not hourly; (2) root growth is not adjusted for the soil water potential; (3) no carbon is translocated from the shoot to the grain.

A key concept of this model is the growth stage. The plant is considered to be in one of a set of mutually-exclusive stages, from planting through harvest. The stages control or modify various behaviors of the plant, e.g. the rate of dry matter production. The stages are listed in the enumeration data types *SRw_GrowthStages*, and are: pre-emergence, emerged, one leaf, two leaves, three leaves, jointed, 5 leaves, 6 leaves, 8 leaves, flag leaf extended, boot completed, heading completed, flowering completed, and ripe. The plant moves from stage to stage according to accumulated heat units, as explained in private function `Growth_Stages`.

Constructor Init

This procedure is called by the simulation driver whenever an object of type *SRw_model* is created at run-time. Its purpose is to initialize the state of the crop model.

Destructor Done

This procedure is called by the simulation driver whenever an object of type *SRw_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Daily_Growth_End

This procedure is called by the simulation driver at the end of each day. It first calls private function `Growth_Stages` to accumulate degree-days and possibly move to a new growth stage. Then the current day's water stress index (accumulated during the day in the `Critical_Leaf_Water_Potential` or `Simple_Water_Stress` procedures) is added into the current growth stage's cumulative stress index; these will be used to determine grain numbers at flowering.

If the crop has emerged but not yet done flowering, private procedure `Dry_Matter_Accumulation` is called to simulate the conversion of photosynthate into dry matter; otherwise, if the crop is not yet ripe, grain yield formation is simulated by calling private procedure `Yield`.

Finally, the `Daily_Growth_End` of the generic Stockle-Riha model is called to accumulate a stress index.

Public Function GetTransFrac

The proportion of light intercepted for this crop as a monoculture is computed according to Stockle (1985) as:

$$\text{GetTransFrac} := 1 - \exp(-0.82 * \text{LAI}) \qquad (1)$$

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing crop.

Public Function PSTempFac

This procedure computes a correction factor for the effects of air temperature on photosynthetic rate. It assumes that the leaf temperature *LeafTemp* (°C) equals air temperature (argument *AirTemp*), and then follows equation [53] from Stockle's thesis:

$$PSTempFac = (2 * (Tmax+A)^2 * (LeafTemp+A)^2 - (LeafTemp+A)^4) / (Tmax+A)^4) \quad (1)$$

where *Tmax* is the temperature for maximum photosynthesis, here set to 25°C, *A* is a constant which is adjusted to best fit the experimental response data; in this model it is 5°C.

Private Function Growth_Stages

This function is called at the end of each day, and it returns the current growth stage, which may change due to the accumulation of heat units. First, the day's effective degree-days are computed as the daily mean temperature. For wheat, a base temperature of 0°C is used, so that the numerical values of the daily mean and the degree-days are the same.

The degree-days are limited by a maximum temperature of 21°C in the early vegetative stages (up to the third leaf's appearance) and 35°C thereafter. The degree-days are accumulated into a running total that starts at zero when the crop is planted. When the accumulated degree-days exceed a variety-specific threshold, the crop enters a new growth stage. These degree-day thresholds may be set as varietal coefficients 1–13 in the plant input file; if they are omitted, the `Init` constructor of this crop model will use the following defaults for a 'standard' wheat variety:

Stage	Degree-days required to enter
Emergence	100
1 leaf	140
2 leaves	258
3 leaves	339
Jointing	425
5 leaves	506
6 leaves	561
8 leaves	698
Flag leaf extended	814
Boot completed	873
Heading completed	960
Flowering completed	984
Ripe	1790

In addition, at the end of heading, the initial number of grains set *GrainNumber* is determined by a call to the `Grain_Number` private function.

Private function Grain_Number

This function is called at the end of flowering to compute the initial grain set from the accumulated dry matter and the record of water stress accumulated in earlier growth stages. It first computes the potential number of grains *PotGrainNumber* (grains m^{-2}) as:

PotGrainNumber := (PotEarsInRow / (plant.RowSpacing * 100)) * PotGrainsPerEar (1)

where the ears in row and grains per ear are constant parameters. Then, multiplicative factors on 0..1 to account for the effects of cumulative water stress during the early vegetative stages *STFactor*, the late vegetative states *SJBHFactor*, and heading *SHFactor*, are determined by empirical relations. All three of these then multiply the potential grain number to arrive at the actual number, which is the function result:

Grain_Number := PotGrainNumber * STFactor * SJBHFactor * SFFactor (2)

Private procedure Dry_Matter_Accumulation

This procedure is called at the end of each day from emergence until flowering is complete, to compute the amount of new dry matter from the amount of photosynthate accumulated during the day, and the current carbon reserve, *CRes* (kg m^{-2}), which is adjusted by this procedure. Thus the plant is considered to have a fresh source of carbon each day (the photosynthate) and a labile (non-structural) carbon pool (the reserve).

First, the net photosynthesis *NetPS* (kg m^{-2}) is computed from the daily photosynthesis *SumPS* (g m^{-2}), depending on the growth stage:

NetPS := 0.60 * 12/44 * (SumPS / 1000) in the vegetative stages up to booting (1a)
NetPS := 0.80 * 12/44 * (SumPS / 1000) after booting (1b)

Next, the potential daily shoot growth *ShootGrowthMax* (kg m^{-2}) is determined by a piecewise function which depends on the growth stage. In the earlier stages, this is a fixed amount:

ShootGrowthMax := 7.2×10^{-4} from emergence until 2 leaves (2a)
ShootGrowthMax := 4.05×10^{-3} 2 or 3 leaves (2b)

In the later stages, the function also includes a linear term based on the heat units accumulated since a specific growth stage was entered. From jointing until the flag leaf is fully out this relation is:

ShootGrowthMax := 6.48×10^{-3}
+ 3.672×10^{-2} (AccDD – AccDD_to[Jointed])/389 (2c)

From when the flag leaf is fully out until the end of flowering this relation is:

ShootGrowthMax := 8.88×10^{-2}
+ 0.176 * (AccDD – AccDD_to[FlagLeafOut])/176 (2d)

From these functions it should be clear that potential shoot growth increases as the plant moves through the growth stages.

Maximum potential root growth is a constant 0.0025 kg m^{-2} d^{-1} up until the end of flowering.

Next, both the shoot and root potential growths are adjusted by a temperature function *TFG* on 0..1 which relates growth to optimal and sub-optimal temperature:

TFG(TOpt) := exp(-1.023 * TMeanK * ((TOpt–TMean)/TMeanK)^2) (3)

where *TMeanK* is today's mean temperature (°K), *TMean* is today's mean temperature (°C), and *TOpt* is the optimum temperature (°C) for growth, which is considered to be 25°C for shoots and 20°C for roots. Thus the potential growth is adjusted as:

ShootGrowth := ShootGrowthMax * TFG(25) (4a)
RootGrowth := RootGrowthMax * TFG(20) (4b)

These potential growth rates are then multiplied by a partitioning factor, which adjusts the demand for growth in the two sinks. These are both a fixed fraction of 0.42:

ShootDemand := 0.42 * ShootGrowth (5a)
RootDemand := 0.42 * RootGrowth (5b)

With the sources (photosynthate *NetPS* and C reserve *InitReserve*, set equal to *CRes*) and sinks (shoot and root demand, and residual C reserve *Reserve*), we then compute a mass balance *FRes* as:

FRes := NetPS + InitReserve – (Reserve * ParallelDemand) (6)

where *ParallelDemand* (kg m^{-2}) represents the total demand for C, computed as:

ParallelDemand := (ShootDemand / (Reserve + ShootSaturation))
+ (RootDemand / (Reserve + RootSaturation)) (7)

where the ..*Saturation* are C saturation constants, fixed at 10^{-6} for shoots, and 10^{-5} for roots, implying that the shoot demand contributes about an order of magnitude to the total demand more than the root demand.

The mass balance should be zero, but unless both the net photosynthesis and C reserve are initially zero, this is very unlikely from a direct computation of (6). What must occur is that *Reserve* is adjusted until the equation balances. Thus *FRes* is a function on *Reserve*, and can be solved by the Newton-Raphson method of root-finding. First, the derivative of the function with respect to *Reserve* is computed as:

dFRes := – ParallelDemand
– Reserve * (1 / (Reserve+ShootSaturation)^2 + 1 / (Reserve+RootSaturation)^2
– 1 (8)

and the new value of *Reserve* is computed as the intercept of the tangent to the function (i.e. the line with the derivate as its slope) with the *Reserve* axis:

Reserve := Reserve – FRes / dFRes (9)

This estimate of *Reserve* is then substituted into equations (6) and (7), and a new estimate is obtained by application of equation (9). The process stops when the mass balance error *FRes* is within 1ppm of 0, and the residual mass balance error is added into the new estimate of the reserve, which is then assigned to *CRes*.

With the revised C reserve value *CRes* obtained by mass balance, the actual shoot and root growth are computed as:

ShootGrowth := ShootDemand / 0.42 * (CRes / (CRes + ShootSaturation)) (10a)
RootGrowth := RootDemand / 0.42 * (CRes / (CRes + RootSaturation)) (10a)

where the 0.42 factor is the same as for equation (5). The actual growths are accumulated into the appropriate dry matters:

AccTopDryMatter := AccTopDryMatter + ShootGrowth (11a)
AccRootDryMatter := AccRootDryMatter + RootGrowth (11b)
AccTotalDryMatter := AccTotalDryMatter + ShootGrowth + RootGrowth (11c)

and the rooting depth is determined from the accumulated dry matter:

RootingDepth := 2.07 * AccRootDryMatter
/ (0.0427436 + AccRootDryMatter) (12)

and adjusted by calling the ancestor model's `LimitRootingDepth` procedure.

The carbon reserve *CRes* is limited to 1.5% of the total dry matter, on the theory that at most this proportion of the structural carbon can be labile:

CRes := Min(CRes, 0.015 * AccTotalDryMatter) (13)

Finally, the current leaf area index *LAI* is determined. From emergence until the flag leaf is extended, the LAI is a function of accumulated top dry matter:

LAI := LAI_K * AccTopDryMatter / (0.237614 + AccTopDryMatter) (14a)

where *LAI_K* adjusts for the effects of row spacing, and also includes an empirical growth factor. This is computed in the object constructor as:

LAI_K := 3.791 * (1 + (0.3 – (plant.RowSpacing / 100))) (15)

where *plant.RowSpacing* is the row spacing (cm). Note that this equation corrects for deviations from the standard 30cm rows used to calibrate the model.

Once the flag leaf is out, LAI declines slowly according to real time. If the flag leaf is out but the head is not yet in the boot, this decline is at the rate of 3% of the maximum LAI *LAIatFlagLeafOut* per day:

LAI := LAIatFlagLeafOut – 0.03 * ((real_day - transition_day[FlagLeafOut])) (14b)

The *transition_day* and maximum LAI are both set in the `GrowthStages` function when the 'FlagLeafOut' growth stage is entered.

Once booting is complete, the decline continues at a rate of 5.5% of the LAI at booting per day:

LAI := LAIatBooted – 0.03 * ((real_day - transition_day[Booted])) (14c)

The *transition_day* and LAI on which to base the decline are both set in the `GrowthStages` function when the 'Booted' growth stage is entered.

Private procedure Yield

This procedure is called at the end of each day during grain filling to compute the amount of new grain dry matter produced during the day and the change in the labile carbon reserve. It is very similar to the `Dry_Matter_Accumulation` procedure, with the difference that the carbon sink is the grain, and it is assumed that there is no root or shoot growth.

First, the net photosynthesis *NetPS* (kg m^{-2}) is computed from the daily photosynthesis *SumPS* (g m^{-2}), exactly as in the late vegetative stages:

NetPS := 0.80 * 12/44 * (SumPS / 1000) (1)

Next, the daily potential grain growth *GrainGrowth* (kg m^{-2}) is determined from a constant maximum daily growth of each grain *GrainGrowthMax* kg $grain^{-1}$ d^{-1} of $2.2x10^{-6}$ kg $grain^{-1}$ d^{-1} , the number of grains per unit area *GrainNumbetr* (grains m^{-2}) and a temperature correction *TFGG* as:

GrainGrowth := GrainNumber * GrainGrowthMax * TFGG (2)

where this temperature function is linear from 1 (optimum) at 20°C or above to 0 (no grain growth) at 3°C or below.

This potential growth is then multiplied by a fixed sink factor of 0.42:

GrainDemand := 0.42 * GrainGrowth (3)

and a mass balance is computed as in `Dry_Matter_Accumulation` equations (6) –(9), with the difference that during grain filling the sink term *ParallelDemand* is computed only from the grain demand:

ParallelDemand := GrainDemand / (Reserve + GrainSaturation) (7')

where *GrainSaturation* is a constant 10^{-6}, the same as for shoots, and the computation of the derivative in (8) changes accordingly.

With the revised C reserve value *CRes* obtained by mass balance, the actual grain growth are computed as:

GrainGrowth := GrainDemand / 0.42 * (CRes / (CRes + GrainSaturation)) (10')

where the 0.42 factor is the same as for equation (3). The actual growths is accumulated:

AccYield := AccYield + GrainGrowth (11a')
AccTotalDryMatter := AccTotalDryMatter + GrainGrowth (11c')

The carbon reserve *CRes* is limited to 10% of the total dry matter, on the theory that at most this proportion of the structural carbon can be labile:

CRes := Min(CRes, 0.010 * AccTotalDryMatter) (13')

Finally, the number of grains *GrainNumber* can be increased from the number determined at the beginning of grain filling (one-time procedure `GrainNumber`), if there is sufficient nitrogen. In this implementation of the Stockle model, we assume that N is never limiting, so set the C/N ratio of the grain to a constant 13:

GrainNumber := GrainNumber
+ (GrainGrowth + (CFracGrain / CNRatioGrain)) (4)

Private procedure Max_Photosynthesis

Both sunlit and a shaded rates of photosynthesis (*PSSun and PSShade*) (g m^{-2} s^{-1}) are then calculated assuming that photosynthesis is related to PAR according to an equation given for maize by Hesketh and

Baker (1969). This rate is then modified for leaf temperature, using a temperature factor *TempFac*, which is obtained by calling the correct virtual procedure for the submodel. The code:

TempFac := PSTempFac(AirTemp)

calls one of the virtual functions, e.g. *SRw_model.PSTempFac* for wheat, which returns the temperature factor. See the submodels for a description of the computation of the temperature correction factors. This factor, which depends on leaf temperature, is further multiplied by another linear factor that does not vary with temperature, *TempCoeff*. This is crop-specific, and is set by the constructor of each Stockle-Riha submodel. For maize, this is (6.2527E-05 / 0.7), and for wheat (1.6 * 2.23583E-05). So the product of the linear factors is on the order of 1. Another crop-specific factor is *TempPower*, which is the exponent to which PAR is raised. This is set by the constructor of each Stockle-Riha submodel. For maize, this is 0.507578, and for wheat 0.695407.

Given these factors, PAR is converted to PS by these equations:

$$PSSun = TempFac * TempCoeff * PARSun^{TempPower} \quad (8)$$

$$PSShade = TempFac * TempCoeff * PARShade^{TempPower} \quad (9)$$

Because of the way the factor, coefficient, and power are constructed, photosynthesis is somewhat less than PAR.

Crop photosynthesis for the hour *PS* (g m^{-2}) is then obtained by multiplying the photosynthetic rate for sunlit leaves *PSSun* (g m^{-2} s^{-1}) by the sunlit leaf area index *LAISun* and adding to this the photosynthetic rate for shaded leaves *PSShade* (g m^{-2} s^{-1}) multiplied by the shaded leaf area index *LAIShade*, this rate then being multiplied by the time step*time_step* (s) to obtain a quantity:

$$PS = (PSSun*LAISun + PSShade*LAIShade) * time\_step \quad (10)$$

In the final section of this procedure, a non-stressed stomatal resistance to vapor loss is calculated for both shaded and sunlit leaves, using Ohm's equation to model CO_2 diffusion from the atmosphere into the leaves:

$$PSxxx = (COext - COint) / (RACO + RESxxx) \quad (11)$$

where: *xxx* refers to either *Sun* or *Shade*, *ResSun* is the sunlit stomatal resistance to water vapor transfer (s m-1), *ResShade* is the shadelit stomatal resistance to water vapor transfer (s m-1), *COext* is the atmospheric CO_2 concentration (0.54 g m-3), *COint* is the CO_2 concentration internal to the leaf (assumed here to be 0.20 g m-3), *RaCO* is the leaf boundary layer resistance to CO_2 transfer, *PSSun* is the photosynthesis rate of sunlit leaves (g m-2 s-1), and *PSShade* is the photosynthesis rate of shadelit leaves (g m-2 s-1).

This equation is rearranged to solve for *RESxxx* and also divided by 1.65 to convert from CO_2 to water vapor resistance (Campbell 1977):

$$ResSun = ((COext - COint) / PSSun - RaCO) / 1.65 \quad (12)$$

$$ResShade = ((COext - COint / PSShade - RaCO) / 1.65 \quad (13)$$

To obtain the whole canopy non-stress resistance to water vapor transfer, the sun and shade lit resistances are weighted according to the proportion of sun and shade lit leaf area and added in parallel (Stockle and Campbell 1985):

NonStressRes = LAI / (LAISun / ResSun + LAIShade / ResShade) (14)

This is an optimization approach to predicting water vapor loss from a plant canopy, since there is an implicit assumption that the plant only opens its stomates the amount necessary to achieve maximum photosynthesis and is never losing more water than required to achieve this rate.

Private procedure Winter_kill

This procedure is meant to simulate the potential for death of winter wheat due to cold temperatures. The impact of air temperature and snow pack depth are considered. A decreasing fraction of the crop is considered to be vulnerable to damage as the daily maximum air temperature (temp) increases above -17 °C:

kill_temp := (1 / (1+ exp(4 * 0.20 * (temp + 17))));

Similarly a decreasing fraction of the crop is considered vulnerable to damage as the snow pack is increased:

kill_snow := (1 / (1 + exp(4 * 0.6 * (snow - 2.5))));

where snow is snow depth in cm. The *winter_kill_factor* is the product of the *kill_snow* and *kill_temp* factors. It is set to zero when either *kill_snow* or *kill_temp* is less than 0.05.

The *winter_kill_factor* is then used to reduce *AccTopDryMatter*:

AccTopDryMatter := AccTopDryMatter * (1 - winter_kill_factor);

5.4.2.3. Crop model: Stockle-Riha Fast Growing Tree

```
source file: stockle.pas
object name: SRt_model
public procedures:: Daily_Growth_Begin, Daily_Growth_End,
            End_Growth, TimeStep_growth, TimeStep_Stress,
            open_det_file
public functions: GetTransFrac, PSTempFac, is_active,
            is_transpiring
private procedures: Dry_Matter_Accumulation, Winter_kill,
            Light_Interception, LimitRootingDepth,
            Critical_Leaf_Water_Potential
```

This submodel uses its ancestor, the generic crop model Stockle-Riha, as a framework for modeling the growth of fast-growing, perennial plantation trees. In order to account for the differences between a densely planted annual grain crop and a widely spaced perennial woody plant which remains in the vegetative phase, many procedures are unique to this submodel.

The primary processes of interest are partitioning between wood, leaves and roots and root dynamics in the soil profile. Rather than using a growth stage concept as is key in the maize and wheat models, the fast growing tree modelled here remains in the vegetative phenological phase from planting until harvest. Respiration is modelled explicitly, and both root and leaf turnover are assumed to be responsive to the moisture environment. Since these dynamics are combined with a "functional equilibrium" approach to

partitioning with respect to supply and demand for water, biomass accumulation is sensitive to the water balance in the environment.

Constructor Init

This procedure is called by the simulation driver whenever an object of type *SRt_model* is created at run-time. Its purpose is to initialize the state of the crop model.

Destructor Done

This procedure is called by the simulation driver whenever an object of type *SRt_model* is destroyed at run-time. Its purpose is to deallocate the heap space used by the object.

Private procedure Daily_Growth_Begin

This procedure is called at the beginning of each day to initialize daily variables *Sum_SoilTemp[layer]*, *SumWN[layer]*, *StressIndex*, and *SumPS* to 0.

Private procedure LimitRootingDepth

This procedure does nothing and is called only to override the procedure called at the level of Stockle-Riha.

Procedure Light_Interception

Here, the simplified method of computing light interception by a plant canopy used at the level of Stockle-Riha is overridden to account for the non-homogeneity of the canopy in a tree plantation before canopy closure occurs. It is based on a model developed by Jackson and Palmer (1979; 1981) and assumes canopy units are shaped as ellipsoids with dimensions defined by the daily leaf area index (LAI) and leaf area density (LAD), and a height to width ratio of 3/1. The area of the shadow cast on a horizontal surface by the canopy, given the sun angle for that time step, is calculated as:

$$\text{Shadow\_Area} := W * \pi/4 * \sqrt{W^2 + H^2 * \cot^2 * \text{SolElA}}$$

where *W* is the crown width, *H* is the crown height and *SolElA* is the solar elevation angle. Since shadows will overlap at low sun angles, a maximum of 100% ground area in shadow is defined.

The "effective" leaf area index for purposes of light interception is then calculated as:

LAI_prime := plant.LAI / Shadow_Area

This accounts for the fact that there are large areas of the ground with no leaves above them, and where there are leaves, the leaf area density through which light must filter is much higher than is implied by the assumption that leaf area is evenly distributed in the horizontal dimension. A standard Beer's Law analogy is then used to compute the sunlit leaf area from LAI_prime:

LAISun_prime := (1 - (Shadow_area * EXP(-K * LAI_prime))) / K

Converting back to standard LAI:

LAISun := LAISun_prime * Shadow_area

The percentage of ground area in sunlight is calculated here as 1 - *shadow_area*, or the area between trees, plus the sunlit ground area below the trees:

Shadow_area * EXP(-K * LAI_prime)

This is used to separate evapotranspiration into evaporation and transpiration. In the calculation of diffuse light interception, *LAI_prime* is used in place of LAI in an equation otherwise identical to that described in `SR_model.Light_Interception`.

This version of `Light_Interception` is called hourly by the function `GetTransFrac` for the fast growing tree model.

Procedure TimeStep_Growth

in this procedure the soil water potentials from 5 a.m. (*Pre_Dawn_WP[layer]*) are stored for use in the root growth routine and `Water_Interception` (optional) and `Max_Photosynthesis` are called.

The sub-procedure `Accum_SoilTemp` accumulates time step soil temperatures by layer in order to calculate an average at the end of the day for root respiration. Similarly, *AccumWater* accumulates the volumetric water content by layer for computing daily averages elsewhere.

Private procedure Critical_Leaf_Water_Potential

The fast growing tree model uses a variation on the Stockle-Riha procedure `Critical_Leaf_Water_Potential`. As in Stockle-Riha, a stress index, *WaterStressFact*, is calculated according to the hourly leaf water potential and *CLWP_Power*. *WaterStressFact* is used to cut back potential transpiration, *sim.PotTrans*:

$$Z := NonStressRes * (1+(LeafWP/plant\_CritialLeafWP)^{plant\_CLWP\_Power}$$

$$WaterStressFact := (SSVD + PSYCON * (NonStressRes+RA) / Re) \; / \; (SSVD + PSYCON * (Z + RA) / Re)$$

sim.ActTrans := WaterStressFact * sim.PotTrans

where *SSVD* is the slope of the saturation vapor density function (g m^{-3} K^{-1}), *PSYCON* is the psychrometric constant (0.494 g m^{-3} K^{-1}), *RA* is the crop boundary layer resistance (s m^{-1}) and *Re* is the combined resistance for convection and longwave radiation heat transfer from the canopy surface, assumed to be a constant 40 s m^{-1} (Campbell, 1977).

A separate photosynthesis stress index, *PSStressFact*, is calculated in the same manner, but taking the differential diffusion rates of CO_2 and H_2O into consideration by dividing the resistances by 1.65:

PSStressFact := (SSVD + PYSCON * (NonStressRes/1.65+RA) / Re) /

(SSVD + PSYCON * (Z/1.65 + RA)/Re)

Note that, in the other versions of Stockle-Riha crop model, this distinction is not made for purposes of reducing photosynthesis, and *PSStressFact* is set equal to *WaterStressFact*.

Private procedure TimeStep_Stress

In this procedure, either `SRt.Critical_Leaf_Water_Potential` or `Simple_Water_Stress` are called, depending on which was selected by the user. Then `Limit_Photosynthesis` is called to multiply *PS*, the timestep potential photosynthesis, by *PSStressFact*.

Private procedure Dry_Matter_Accumulation

This procedure contains six subroutines and is called at the end of each day to calculate maintenance respiration, suberization of roots, growth of sinker roots, partitioning of photosynthate to roots, stem and leaves, and distribution of biomass above and below ground. Throughout this section, allometric equations defining the relationship between stem diameter and woody biomass or tree height are utilized. Log transformations are made of the general equation $Y = aD^b$ where Y is woody biomass in kg/tree and D is stem diameter in cm. The parameters a and b are often reported in the literature (Parde, 1980) and are input by the user according to the tree species being modelled.

ln(woody biomass) = aWood + bWood * (ln(diameter)

ln(woody biomass) = aHeight + bHeight * (ln(diameter) + cHeight * (ln(Height)

where *aWood* and *bWood* are the parameters relating diameter to tree woody biomass, and *aHeight*, *bHeight* and *cHeight* are defined for the log-transform of the equation relating height (m) and diameter (cm) to woody biomass (kg/tree): $Y = aD^bH^c$. This equation is inverted to find tree height through time.

Private procedure Maintain

The maintenance procedure computes the maintenance respiration costs of above and below ground biomass as a function of temperature and existing biomass. A Q_{10} of 2 is assumed. With a reference temperature (Tref) of 20 °C, and a standard maintenance factor at the reference temperature (MRef), a temperature corrected maintenance factor (*MaintFact*, kg kg^{-1} day^{-1}) is calculated for each plant part according to the following:

T_power := (Tm - Tref) / 10

MaintFact := MRef * $Q_{10}^{T\_power}$

Maintenance := Biomass * MaintFact

where *Tm* is the mean temperature of the soil layer or atmosphere, depending on the plant part, and the units for *Maintenance* are kg biomass per m^2 ground per day. Maintenance respiration rates for new roots and older, suberized roots are assumed to differ, with new active roots being more costly to maintain (Lambers et al, 1991). Growth respiration is calculated separately by decreasing that day's sum of

photosynthate by 25% (Johnson, 1990). With respect to stem biomass, it is assumed that only the outer, living tissue is respiring and therefore the amount of biomass used in the above calculation is scaled down from total stem biomass. The reference maintenance factors used are estimated from measurements of other woody species (Rauscher et al, 1990).

Maintenance respiration at 20°C, MRef	(kg $kg^{-1}day^{-1}$)
New, White Root	0.03
Suberized Roots	0.0003
Stem	0.001
Leaf	0.008

The maintenance requirement is then subtracted from the existing biomass of that plant part, with old and new root biomass calculated separately for each soil layer. This procedure therefore accounts for death and senescence of root biomass. If the growth allocated to each soil layer is less than the maintenance cost for that day, there will be a net decrease in roots for that layer.

Private Procedure Suberize

In order to account for the transformation of white fleshy roots to woody roots over time, this routine transfers a fraction (0.03/day) of the new root biomass in each layer to old root biomass. Only new roots are used in the calculation of root water uptake, but old roots, which have a lower respiration rate and are therefore more stable over time, provide the infrastructure for new root growth and water uptake.

Private Procedure Grow_Sinkers

This procedure creates the sinker, or tap root structure from which new root growth can occur. It is only called if today's accumulated photosynthate exceeds the amount required for the daily increment and the sinker roots have not yet reached the bottom of the profile. The *Sinker_Growth_Rate* in units of m day^{-1} is input by the user in the plant file and does not vary as a function of the environment. In this procedure the sinker roots are incremented in length and then the layer in which the roots now end is computed in order to update *sim.LRoot*, the layer number of the last soil layer with roots. In this step, the tap root may have reached beyond the edge of a new layer, so the amount of biomass to add to each layer's current root biomass is calculated and *NewRtBio[layer]* is updated.

This amount of biomass is calculated as the *No_of_Sinkers* per plant, input by the user, multiplied by the planting density, the *Sinker_Growth_Rate* and the inverse of the length to weight ratio for coarse roots, also input by the user in the plant file. This quantity is then subtracted from the available photosynthate, *sumPS*. Since this procedure is called before the daily partitioning routine is called, it implies that the model assumes that root extension growth will proceed regardless of other demands for carbon.

If the tap root has reached the bottom of the profile, it is assumed to continue growing and adding biomass to the current layer.

Private Procedure Partition

A variation on the theory of "functional equilibrium" (Brouwer, 1962; Davidson, 1969a,b; Wilson, 1988) was utilized in developing the procedure for partitioning biomass to new roots, stem and leaves. The essential feature of the theory is that above and below ground plant parts both supply and consume products necessary to sustain life and therefore must maintain some functional balance between their relative sizes. In this case we are concerned with the supply and demand for photosynthate and water. It is assumed here that the tree will maximize the amount of biomass it partitions to leaves within the constraints created by limitations on water uptake. As long as there is no water stress, the tree can continue to expand its leaf area and thereby increase the demand for water. If the demand exceeds the supply of water by the roots, photosynthate will be partitioned to roots until the stress is relieved.

Because the ability to take up water per unit root biomass will vary with soil water content and location in the profile, a means of integrating the limitation to soil water uptake over the day was necessary. As an approximation of the effective "demand" for root biomass, the ratio of potential to actual transpiration for that day is multiplied by the quantity of existing new root biomass summed over all layers, *SumNew*. A maximum limit of double the roots, and, if the tree is still young, a minimum of 1.115 times the root biomass were set:

```
if ActTrans < 0.5*SumPotTrans  then   ActTrans := 0.5*SumPotTrans

if (SumPotTrans/ActTrans) < 1.11 and (Tree_Age < 400)

        then   Pot_Root := SumNew * 1.11

Pot_Root := SumPotTrans /  SumActTrans * SumNew

RT_demand := Pot_Root - SumNew
```

Pot_Root is the amount of root biomass that would have been required to meet the potential demand for water given today's moisture and biomass distribution in the profile. The added constraint of a minimum potential root biomass reflects the limitations of the functional equilibrium model to provide a full explanation of partitioning above and below ground plant parts. The difference between existing roots and potential roots then equals the root "demand" for this day. Distribution within the profile is computed in the sub-routine `Distribute_Roots`.

A slightly different approach is taken in the calculation of the "demand" for stem biomass. While root demand is estimated as a function of the ratio of potential to actual transpiration and is therefore directly related to the demand for water, stem biomass is indirectly related to the demand for water through an assumed relationship with leaf area. In accordance with the "Pipe Model" theory (Shinozaki et al, 1964), for each increment of new leaf area a corresponding area of xylem must be added to conduct water to it. Thus, a relationship between new leaf area and new stem cross sectional area may be defined. The slope m was estimated to be 0.05 cm^2 stem area/m^2 leaf area, with an intercept of zero.

```
Delta_Area_Tree := 0.05 * New_LA/planting_dens
```

where *Delta_Area_Tree* is the change in stem cross sectional area per tree in cm^2, and *New_LA* (m^2 leaf area per m^2 of ground added yesterday) is divided through by the planting density (trees m^{-2}) to get leaf area per tree. Note that, because these units are on a per tree basis, you must multiply by the planting density (trees/m^2) to get stemdiameter (cm^2/m^2) per leaf area (m^2/m^2). Knowing the amount of stem

biomass required to meet the "demand" for water by the new leaf area, allometric parameters are used to calculate the stem biomass corresponding to that area:

```
Ln_NewBio := aWood + bWood * ln(Current_Diam + Delta_Diam)

St_Demand := exp(Ln_NewBio) * planting_dens - Stembio + St_Demand
```

where *Ln_NewBio* is the natural log of the estimate of stem biomass required (kg tree^{-1}). This biomass per tree is multiplied by the planting density to convert it to biomass m^{-2} ground. Then the existing biomass is subtracted to get the updated demand for stem, *St_Demand*. Since the demand for each day may not always be met with new stem biomass, the balance is carried over from day to day and added to the new stem demand.

If there is sufficient photosynthate available to meet the sum of the demands for root and stem biomass, each plant part is allocated its full requirement and any left over is partitioned to leaf dry matter. However, if there is less photosynthate than the demand, dry matter is divided according to the relative weights of the demands for stem and roots:

```
Root_Part := DryMatter * (Rt_Demand/(Rt_Demand + St_Demand))

Stem_Part := DryMatter - Root_Part
```

in which case no dry matter is allocated to leaves that day. This assures that the canopy will not continue to expand until the tree has the ability to supply sufficient water to the leaves.

Private Procedure Distribute_Roots

This procedure controls the distribution of new root biomass in the profile as a function of soil water potential and pre-existing biomass. The pre-existing biomass (*BaseRtBio[layer]*) is calculated as all of the new root biomass plus half of the old root biomass. The fraction to be allocated to each layer is calculated as:

```
PotGro[layer] := BaseRtBio[layer] * laythick[layer]
                                  * (1 - Pre_Dawn_WP[layer] /  -500)

RtGrowth[layer] := (PotGro[layer] / sumPotGro) * Root_Part
```

where *PotGro* is the potential root growth in that layer under the conditions of the function defined and *sumPotGro* is the sum of potential root growth in all layers calculated for the purpose of defining the fraction. Note that a pre-dawn soil water potential (*Pre_Dawn_WP*) of -500 J kg^{-1} is the lower limit for root growth, i.e., no growth will occur in that layer as long as the pre-dawn potential is -0.5 MPa or lower. This assures that root growth is preferentially distributed in the wetter regions of the soil profile.

New stem and leaf biomass are added to their respective plant parts here, and the sum of root biomass is recalculated.

Private Procedure Form_Aboveground

`Form_Aboveground` computes a new tree height and takes care of bookkeeping on leaf ages and senescence. Tree height is calculated using the allometric relationships defined by the parameters input by the user in the plant file:

ln_diam := ln(TreeWood) - aWood / bWood

ln_height := (ln(TreeWood) - aHeight - bHeight * ln_diam) / cHeight

crop.CanopyHeight := EXP(ln_height)

Leaf cohorts are represented in two arrays, one defining leaf biomass for each cohort (*Leaf_Cohort[i]*) and the other for the age in days of each leaf cohort (*Cohort_Age[j]*). At any point in time, some number of cohorts exists, each having an age and a quantity of biomass. Every 30 days, a new cohort is formed and becomes the recipient of all new leaf biomass. Cohort age is incremented as a function of both days passed and water stress as follows:

Cohort_Age[i] := Cohort_Age[i] + 1 + WaterStressFact

Once a cohort reaches the user-defined maximum leaf age, it is senesced from the canopy and the biomass is subtracted from the total leaf biomass. If the cohort did not experience any water stress during it's lifetime, it would remain on the tree for the number of days equal to *Max_Leaf_Age*. But stress speeds the aging process, causing leaves to fall more quickly, which decreases the transpiring surface and thus the demand for water. Under a period of prolonged drought, leaf area may decrease substantially, as has been observed in the field (Pook, 1985).

The final calculation is of the leaf area density, *LAD*, m^2 leaf area m^{-3} canopy volume, which is not constant throughout the life of the tree:

If Tree_Age <= 1095 days then
    LAD := -0.0137 * Tree_Age +15
else LAD := 1

This defines *LAD* as decreasing from 10 m^2 m^{-3} at planting to about 1 m^2 m^{-3} at three years. The *LAD* is used in procedure `Light_Interception` to calculate canopy volume.

5.4.3. Crop model: modified SORKAM

```
source file: sorkam.pas
object name: SK_model
public procedures:: Daily_Growth_End, End_Growth,
            TimeStep_Growth, open_det_file, write_summary
public functions: GetTransFrac, is_active, is_transpiring,
            has_matured
private procedures:: soil_moisture_index, emergence,
            dry_matter_accumulation, daily_plant_growth
private functions:: heat_units, water_stress, growth_stages
```

This model is a modification of the SORKAM grain sorghum crop growth model of Rosenthal *et al.* (1989). The main differences from the original SORKAM are:

- only the plant processes of SORKAM are included; the atmospheric, soil, and plant water uptake processes modelled in SORKAM have been removed, and the various GAPS methods of modelling these may be selected by the GAPS user. This allows the different crop models (e.g. SORKAM and Stockle-Riha) to be compared with the same atmospheric, soil, and water uptake models.

- individual leaf development is not modelled; instead, we replaced this part of SORKAM with a model of composite LAI development.

- tillering is not modelled.

- the code is structured so that this crop model is a descendent object type of the generic GAPS crop model object type.

Rosenthal *et al*. (1989) describe the objective of SORKAM as follows: "The simulation model SORKAM describes the morphological development of a well-fertilized single grain sorghum plant in response to the environment. Growth and development are assumed to be unaffected by insects and diseases ... SORKAM is intended as a guide for further research to evaluate growth and development of grain sorghum." Since the GAPS implementation of SORKAM does not predict individual leaf development, its use by crop physiologists may be limited. However the GAPS implementation does predict growth stages, photosynthesis, stress, dry matter and grain yield.

Throughout this description, we refer to the original SORKAM documentation, in particular to their subroutine names, like 'LEAF', and equation numbers, like [53].

Constructor Init

This procedure is called whenever an object of type *SK_model* is created at run-time. Its purpose is to initialize the state of the crop model, for example, setting the current growth stage to 'pre-emergence'. It also computes several parameters that do not vary during the course of the simulation, including:

- the number of heat units needed to reach growth stage 3 *GDD_to_3*; this replaces SORKAM subroutine LEAF, which keeps track of each leaf's development;

- for temperate cultivars, the heat units needed to reach growth stages 2 and 4;

- parameters X1, X2, and X3 used in SORKAM's empirical dry-matter accumulation equation.

Destructor Done

This procedure is called whenever an object of type *SK_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Daily_Growth_End

This is called at the end of each simulated day. First, the daily mean temperature *MeanTemp* is calculated from the daily maximum and minimum air temperatures. Next, if the plant has not yet emerged, the private procedure `Emergence` is called to simulate the seedling; if the plant has matured, no processes are simulated; otherwise, the plant is actively growing, and following private procedures are called in order to simulate the growing plant:

- `Soil_Moisture_Index` : determine today's composite soil moisture status

• `Growth_Stages` : determine if the growth stage has changed

• `Dry_Matter_Accumulation` : determine how much dry matter is accumulated today

• `Daily_Plant_Growth` : partition dry matter and grow organs

See the descriptions of these private procedures, below.

Public procedure TimeStep_Growth

This is called at each simulated time step. Although SORKAM only changes the plant's state once a day, other GAPS procedures, in particular the plant water uptake procedures, operate on a less-than-daily time step. Both water uptake methods are based on the time step's simulated potential transpiration, which is first calculated by the atmospheric procedure 'Partition_ETP', and then modified by the crop model to account for water stress that reduces transpiration.

This procedure simply reduces simulated potential transpiration by a fraction:

sim.PotTrans := sim.PotTrans * Watco (1)

where *Watco* is a fraction which is computed for just this purpose, in the private function 'SK_model_water_stress' each day (see below).

Public Function GetTransFrac

The proportion of light intercepted for this crop as a monoculture is computed according to SORKAM equation [51] as:

$$GetTransFrac := 0.53 * LAI^{0.5}$$

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing crop. This is >1 for LAI > 3.56, which is impossible; therefore, the result is limited to 1.0:

If (LAI > 1/(0.53 * 0.53)) then GetTransFrac := 1

SORKAM is unusual in that it doesn't estimate the fraction with a negative exponential function of increasing LAI..

Public Function is_active

This function returns True if and only if the crop object is active, i.e. if the simulation driver should call any plant processes at all. The crop is assumed to be active between the planting and harvest dates specified in the plant input file, but in addition this function checks whether the crop simulation has been stopped by some other means, in which case the growth stage is '6'.

Public Function is_transpiring

This function returns True if and only if the crop object is transpiring, i.e. actively growing, in which case the simulation driver will call the crop's 'GetTransFrac' function before partitioning potential

evapotranspiration into evaporation and transpiration. In this model, the crop is transpiring only during growth stages '1' through '4'.

Public Function has_matured

This function returns True if and only if this grain crop has matured, i.e. grain filling has stopped, but grain drying is proceeding, in which case the generic crop model's *Daily_Growth_End* procedure will simulate grain drying. In this model, this function is a direct translation of the *matured* growth stage.

Private Function heat_units

This function is called by various of the daily growth processes, to determine the daily heat units above some base temperature *BaseTemp*, °C. It is based on the SORKAM function HFUNC. The function is called with the base temperature and the daily mean, *MeanTemp*, both in °C.

If the daily maximum temperature is less than the specified base, there are no heat units.

If the daily minimum temperature is above the base value, the entire day contributes to the heat units, which are calculated as:

heat_units := MeanTemp – BaseTemp (1)

However, if *MeanTemp* is above 30°C, this latter value is used instead of the mean. This limits the heat units in the case of very warm days.

In the most general case, the daily maximum is above the base and the daily minimum is below it. In this case we must integrate a sine wave, which represents the instantaneous air temperature, over the period of time when this temperature is greater than the base. The amplitude *Amplitude* of the sine wave is *MaxTemp–MeanTemp*, and the cutoff value *Zeta* is *sin((BaseTemp–MeanTemp) / Amplitude)*. Then the integral is computed as:

heat_units := 1/π *
(Amplitude * Cos(Zeta) + (MeanTemp–BaseTemp) * (π/2 * Zeta)) (2)

Private Function water_stress

This function computes a daily water stress coefficient on 0..1 (extreme to no stress) from the daily potential evapotranspiration *ET* , the current soil moisture index *SMI* on 0..1 (very dry to saturated) and a threshold index *Threshold*. It is based on the SORKAM function WATCO.

First, the threshold index is adjusted so that increasing potential ET results in additional water stress, according to SORKAM equation [52]:

AdjustedThreshold := (Threshold + 0.11) + 0.028 * ET (1)

If this is above 0.3, there is no stress to any plant process, i.e. *water_stress := 1*. Otherwise, the stress is computed as the ratio of the moisture index to the threshold (SORKAM eqn [53]):

water_stress := SMI / AdjustedThreshold (2)

Private Procedure soil_moisture_index

This procedure computes the current soil moisture status as the index *SMI* on 0..1 (very dry to saturated), and with this calls the *water_stress* function to determine the current water stress coefficient *WATCO* on 0..1 (extreme to no stress). The soil moisture index is calculated from the weighted ratio of the current volumetric water contents *sim.WN[]* to the drained upper limits (field capacity) *soil.DUL[]* of the layers in the current rooting zone:

$$SMI := \frac{\sum_{layer := plant.FRoot}^{sim.LRoot} soil.WN_{layer} * KgFines_{layer}}{\sum_{layer := plant.FRoot}^{sim.LRoot} soil.DUL_{layer} * KgFines_{layer}} \quad (1)$$

where *KgFines[]* is the mass of the fine earth fraction in the layer, which weights the moisture contents appropriately.

Private Function Growth_Stages

SORKAM models the phenological development of the sorghum plant in seven growth stages:

0 : pre-emergence
1 : emergence to growing point differentiation
2 : growing point differentiation to the end of leaf growth
3 : end of leaf growth to anthesis
4 : anthesis to physiological maturity
5 : post-physiological maturity
6 : dead, i.e. frozen or harvested

The crop progresses from one growth stage to another according to accumulated heat units, possibly modified by water stress. This function is called at the end of each day to determine the current growth stage. It also computes the daily heat units for base temperatures of 1 and 7°C (*DailyUnits1* and *DailyUnits7*), and the accumulated heat units in growth stages 1..3 and 4 (*AccUnits1to3* and *AccUnits4*), which are used in the daily plant growth procedure as well as in this function.

First, we check if the plant has been killed by frost. This happens if the growing point is vulnerable (i.e. in growth stages 2 through 5) and the daily minimum temperature is below –2.2°C. Note that until growing-point differentiation, the growing point is protected below the soil surface.

Next, the daily heat units for base temperatures of 1 and 7°C (*DailyUnits1* and *DailyUnits7*) are determined by calls to the `heat_units` function. The base-7°C heat units are then adjusted for the effects of water stress (SORKAM eqns [58] and [59]). Before anthesis, they are multiplied by the water stress coefficient *WATCO*, thereby reducing the effective heat units (i.e. potential growth) with water stress. After anthesis, water stress hastens physiological maturity, which is accomplished in the model by increasing the effective heat units, if the soil moisture index *SMI* is less than 30%:

DailyUnits7 := DailyUnits7 / ((1.1 * SMI) + 0.67) (1)

Note that a smaller *SMI* , representing less soil water, results in an increase in heat units.

The daily heat units are then accumulated into the cumulative heat units. Before anthesis, the base-7 heat units are accumulated into *AccUnits1to3* . After anthesis but before physiological maturity, the base-1 heat units are accumulated into *AccUnits4* .

The heart of this procedure is the attempt to move to the next growth stage. This is based on a comparison of the accumulated heat units with threshold values *GDD_to_2, GDD_to_3, GDD_to_4*, and *GDD_4_to_5* that must be reached in order to move to the next stage. These threshold values are set as follows.

The base-1 degree-days from anthesis to maturity are a constant 880:

GDD_4_to_5 := 880 (2)

The base-7 degree-days to reach the end of leaf growth is computed in the object constructor `Init` from the number of leaves *NLeaves* (varietal coefficient 1, from the 'plant' input file) as:

GDD_to_3 := ((50 * NLeaves) – 40) + (5.71 + (14.29 * NLeaves)) (3)

where the first term expands all leaves but the last, at the rate of 50 degree-days per leaf expansion, and the last term expands the flag leaf (SORKAM eqn [26]). This expression substitutes for the simulation of individual leaf growth as found in SORKAM. Note that later-maturing varieties are those with more leaves.

For temperate cultivars (model option 'temperate') the remaining values are determined in the object constructor `Init` from the number of leaves, following SORKAM equations [3] and [6], as:

GDD_to_2 := 12.9 + (27.6 * NLeaves) (4)
GDD_to_4 := 289.72 + (45.84 * NLeaves) (5)

For tropical cultivars, these values can not be determined until emergence, since they depend on the daylength. See the description of procedure `Emergence` for this calculation.

Private Procedure Emergence

This procedure is called at the end of each day when the plant is in growth stage 0, i.e. pre-emergence, to simulate the development of the seed and seedling until emergence.

If the seed has not yet germinated, heat units base 6.3°C are accumulated until the threshold value of 18 is reached, at which time the radicle has penetrated the seed coat.

Once the seed has germinated, heat units base 11.4 °C are accumulated until the threshold value of 51 is reached, at which time the plumule has penetrated the seed coat.

Once the plumule has broken through the seed coat, heat units base 11.4 °C *Units11* are accumulated, and the plumule length is computed as:

Plumule_length := Plumule_length + (Units11 / DCoeff) (1)

where *DCoeff* is the plumule extension rate, a constant 3.1 cm $°C^{-1}$. When the plumule length reaches the sowing depth, the plumule has emerged, and the growth stage is changed to stage 1. Also, if this is a tropical cultivar (model option 'tropical'), the threshold degree-days to stages 2 and 4 are computed from the daylength, following SORKAM equations [4] and [7] as:

GDD_to_2 := 370 + (400 – (daylength–13.6)) (2)
GDD_to_4 := GDD_to_2 + 1020 + (120 – (daylength–13.6)) (3)

if the daylength is greater than 13.6 hours. This assumes that the daylength is increasing at emergence. If the daylength is shorter, these are set to the constants 370 and 1020, respectively, SORKAM eqns [5] and [8]. The daylength is determined as twice the difference from noon to sunrise, which in turn is determined from the solar declination on the emergence day, and the latitude.

For tropical cultivars, the actual number of leaves that will appear is adjusted for the heat units needed to reach anthesis, following SORKAM equation [9]:

NLeaves := trunc((GDD_to_4 – 5.71) / 64.29) (4)

Private Procedure dry_matter_accumulation

This procedure, based on SORKAM procedure 'PHOTO', is called at the end of each day from emergence to physiological maturity to determine the amount of dry matter produced by the plant during the day. Unlike the Stockle-Riha models, the SORKAM model does not explicitly model photosynthesis, but instead estimates the photosynthetically-active radiation (PAR) intercepted by the plant from the mean air temperature. This is then converted to dry matter by an empirical formula which takes into account the daily water stress.

First, the proportion of the PAR that is transmitted (i.e. not intercepted by the leaves) *PctTrans* is estimated from the leaf area index *LAI* and several empirical parameters that related LAI and row spacing. In growth stages 1 and 2:

PctTrans := 100 * exp(–0.7675 * LAI) if LAI < X3 (1a)
PctTrans := X1 * exp(X2 * LAI) if LAI >= X3 (1b)

where X1, X2, and X3 are calculated in the object constructor `Init` as:

X1 := Min(0.2341 * plant.RowSpacing + 67.9915, 100) (2)
X2 := 0.0010 * plant.RowSpacing – 0.322 (3)
X3 := ln (X1/100) / (–0.7675 – X2) (4)

where *plant.RowSpacing* is the distance between plant rows, cm. In growth stage 4, the transmitted PAR is computed as:

PctTrans := 70.1 * exp(–0.612 * LAI) (5)

and in growth stage 3, *PctTrans* is interpolated from that computed in formulas (1) and (5), for the first 7 days in stage 3, after which it is simply that computed by (5). The interpolation formula is:

$$PctTrans := PctTrans_1 - (PctTrans_1 - PctTrans_5) * (DaysIn3 / 7) \quad (6)$$

where *DaysIn3* is the number of elapsed days in growth stage 3. In all growth stages, *PctTrans* is at least 5%.

Knowing the percent transmission and the portion of the daily solar radiation (*clim.SolRad[]*) intercepted by this crop (function *MySolRad*),(MJ m^{-2}), the daily PAR *PARInr* (MJ m^{-2}) is computed by SORKAM equation [34] as:

PARInt := 0.45 * TheCompModel^.MySolRad(MyCropI) * (1 – (PctTrans /100)) (7)

Next, a temperature stress coefficient *Tempco* on 0..1 (no .. full growth) is determined from the daily mean temperature *MeanTemp* (°C) as follows:

MeanTemp < 5 : 0 (no growth)
MeanTemp ≥ 5 & MeanTemp < 25 : 0.05 * MeanTemp – 0.25
MeanTemp ≥ 25 & MeanTemp < 40 : 1 (full growth)
MeanTemp ≥ 40 & MeanTemp < 45 : –0.2 * MeanTemp + 9
MeanTemp > 45 : 0 (no growth) (7)

In growth stage 4, this coefficient is further adjusted by SORKAM equation [35] for the effects of high minimum temperature (i.e. warm nights during grain filling), where *clim.MinTemp[]* > 19°C:

Tempco := Min(Tempco, –0.0333 * clim.MinTemp[real_day] + 1.633) (8)

This temperature stress coefficient and the water stress coefficient *Watco* , which was computed in the `water_stress` function, are compared, and the numerically-lesser (i.e. more stressed) one is chosen as the stress factor *StressIndex* on 0..1. Then the daily top dry matter can be computed by SORKAM equation [36] as:

DailyTopDryMatter := Alpha * PARInt * StressIndex * 10^{-3} (9)

where *Alpha* is an empirical conversion from MJ to g of dry matter, a constant 3.2 g MJ^{-1}, and the factor of 10^{-3} converts from g to kg. This is then accumulated into *AccTopDryMatter* (kg m^{-2}).

Private Procedure daily_plant_growth

This procedure is called at the end of each day from emergence to the end of grain filling to grow the sorghum plant. It is based on SORKAM procedure 'GROW', but has been simplified: here we do not grow individual leaves, and do not partition photosynthate into plant organs other than the top, root, and grain.

First, a 'fractionation stage' (1..4) is determined from the growth stage and the number of elapsed days in stages 2 and 4. These are then used to partition the top dry matter *DailyTopDryMatter* determined in procedure `dry_matter_accumulation`, as follows:

DailyLeafDryMatter := 0.6 * DailyTopDryMatter (1a)
DailyRootDryMatter := 0.5 * DailyTopDryMatter
DailyGrainDryMatter := 0 in fractionation stage 1;

DailyLeafDryMatter := 0.22 * DailyTopDryMatter (1b)
DailyRootDryMatter := 0.25 * DailyTopDryMatter
DailyGrainDryMatter := 0 in fractionation stage 2;

DailyLeafDryMatter := 0 (1c)
DailyRootDryMatter := 0.25 * DailyTopDryMatter
DailyGrainDryMatter := 0 in fractionation stage 3;

DailyLeafDryMatter := 0 (1d)
DailyRootDryMatter := 0
DailyGrainDryMatter := 0.9 * DailyTopDryMatter in fractionation stage 4.

The grain yield, if any, is accumulated:

AccYield := AccYield + DailyGrainYield (2)

and similarly for the roots:

AccRootDryMatter := AccRootDryMatter + DailyRootDryMatter (2)

The rooting depth (cm) is determined as an empirical function of the total root dry matter and the base-7°C heat units accumulated since emergence *AccUnits1to3*, following SORKAM equation [44]:

RootingDepth := (plant.MaxRootingDepth
* (0.5 + (0.5 * sin((3.03 * (AccUnits1to3 / GDD_to_4)) – 1.487))))
+ plant.SowingDepth (3)

where *GDD_to_4* is the number of heat units needed to reach anthesis. This depth is of course limited to the maximum rooting depth specified in the 'plant' input file.

Next, the leaf area index is determined. In growth stages 1 and 2 (i.e. until the end of leaf growth), it is determined from the specific leaf area and the daily leaf dry matter accumulation. The specific leaf area *SpecificLeafArea* is computed from the daily mean temperature and PAR by SORKAM equation [37] as:

SpecificLeafArea := MeanTemp * 10^{-1}
/ ((0.00178 * MeanTemp) + (0.000809 * MeanTemp * PARInt) + 0.0308) (4)

where the factor 10^{-1} converts from $cm^2\ g^{-1}$ to $m^2\ kg^{-1}$. This specific leaf area is then decreased by water stress. In SORKAM this is done for each leaf individually in procedure 'LEAF'; here we apply the same principle to the entire top. The SLA is decreased linearly from its value as calculated in (4) at soil moisture indices ≥ 0.5 (no significant stress) to no new leaf area at SMI = 0:

SpecificLeafArea := SpecificLeafArea * (2 * SMI) if SMI < 0.5 (5)

The SLA is limited to at most 40. This is then used to determine LAI:

LAI := LAI + (SpecificLeafArea * DailyLeafDryMatter) (6a)

In growth stage 3, the LAI is constant, i.e. from end of leaf growth to anthesis. In stage 4, i.e during grain filling, the LAI declines linearly by the base-1°C heat units:

LAI := LAI – ((LAILossIn4 / GDD_4_to_5) * DailyUnits1) (6b)

where *LAILossIn4* is the amount of LAI lost for each heat unit, a constant 0.4, normalized by the base-1°C heat units needed to go from anthesis to physiological maturity, a constant 880. The LAI must be non-negative.

5.4.4. Crop model: CONSTANT

```
source file: constant.pas
object name: SC_model
public procedures:: Begin_growth, Daily_Growth_Begin,
            Daily_Growth_End, TimeStep_Growth, End_Growth,
            open_det_file, write_summary
public functions: GetTransFrac, is_active, is_transpiring,
            has_matured
private procedures:: Water_Interception
```

This model simulates a constant crop that grows 'immediately' to final dimensions. From planting to harvest, the crop is active, transpiring and mature, and has a constant height, rooting depth, LAI and biomass. It is useful to simulate situations with a more-or-less 'steady state' vegetative cover, and also to examine the non-dynamic effects of competition on another crop model.

Constructor Init

This procedure is called whenever an object of type *SC_model* is created at run-time. Its purpose is to initialize the state of the crop model.

Destructor Done

This procedure is called whenever an object of type *SC_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Begin_Growth

This procedure is called at the beginning of crop growth. It computes several parameters that do not vary during the course of the simulation, including:

- the accumulated total dry matter, *AccTotalDryMatter* which is set to the maximum value of *Plant_MaxTopDM* and *plant.InitialDryMatter*,
- the accumulated top dry matter, *AccTopDryMatter* := *Plant_MaxTopDM*,
- the accumulated root dry matter, *AccRootDryMatter* := *AccTotalDryMatter* - *AccTopDryMatter*,
- the rooting depth, *RootingDepth* := *plant.MaxRootingDepth*,
- and the leaf area index *LAI* := *plant.LAI* (or minimum 0.07).

Public procedure Daily_Growth_End

This is called at the end of each simulated day. The growing degrees days is calculated as:

GDD := (Tmax + Tmin)/2 - plant_BaseTemp and GDD >=0

The growing degrees days are accumulated into *AccDD*.

Public procedure TimeStep_Growth

This is called at each simulated time step. This procedure calls the `Water_Interception` procedure if Canopy Water Interception (proc_status[1]) in the plant input file was checked.

Public Function GetTransFrac

The proportion of light intercepted for this crop as a monoculture is computed as:

GetTransFrac := 1 - exp(-plant_ExtinctionCoeff * LAI)

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing crop.

Public Function is_active

This function returns True between the planting and harvest dates specified in the plant input file.

Public Function is_transpiring

This function returns True between the planting and harvest dates specified in the plant input file.

Public Function has_matured

This function returns True between the planting and harvest dates specified in the plant input file.

Private Procedure Water_Interception

This procedure is identical to the Water_Interception procedure called in the Stockle-Riha model.

5.4.5. Crop model: EPIC model

```
source file: epic.pas
object name: SE_model
public procedures:: Begin_growth, Daily_Growth_End,
            End_Growth
public functions: GetTransFrac
```

This model is adapted from the EPIC growth model developed by Williams, J.R., Jones, C.A. & Dyke, P.T. (1989) and modified by Williams, J.R. (1994). The EPIC crop model was designed to simulate growth for many different crops. Each crop has unique values for the model parameters.

The main differences with the original EPIC crop growth model are:

- the biomass energy conversion factor is not adjusted by vapor pressure deficit and by atmospheric CO2 level,

- the nutrient, aeration, soil strength, soil temperature and aluminum toxicity stresses are not simulated,

• winter dormancy and tillage are not simulated.

The development of the crop is based on the daily heat unit accumulation, not on physiology.

Constructor Init

This procedure is called whenever an object of type *SE_model* is created at run-time. Its purpose is to initialize the state of the crop model. In particular, it computes *plant_Point1LAI* and *plant_Point2LAI*, the two parameters of the S-shaped LAI-vs-heat unit curve, from two points on the curve, read from the input file.

The Initializer also adjusts the maximum potential LAI, *plant_MaxPotLAI*, based on the plant density:

```
plant_MaxPotLAI := plant_MaxPotLAI * compete.SCurve(plant_PlantDensity, y1, y2)
```

where *y1* and *y2* are the S-curve parameters, fitted from two points on the curve (*x1*,*f1*) and (*x2*,*f2*), read from the input file:

```
compete.FitSCurve(x1, f1, x2, f2, y1, y2)
```

Destructor Done

This procedure is called whenever an object of type *SC_model* is destroyed at run-time. Its only purpose is to deallocate the heap space used by the object.

Public procedure Begin_Growth

This procedure is called at the beginning of crop growth. It is only used to set *SE_GrowthStage* to *SE_leafIncrease*.

Public procedure Daily_Growth_End

This is called at the end of each simulated day. Growing degrees days are calculated as:

```
GDD :=  (Tmax + Tmin)/2 - plant_BaseTemp;          and GDD >=0
```

The growing degrees days are accumulated into *AccDD*.

The daily heat unit index *AccUnits*, ranging from 0 at planting to 1 at physiological maturity is computed as Acc*DD* divided by *plant_PotUnits*, the potential heat units required for the maturation of the crop.

The leaf area index is simulated as a function of heat units, crop stress, and the development stage. From planting to the start of leaf decline, (= while the *AccUnits* is less than the *plant_DeclineFraction)*, the *LAI* is calculated as follows:

```
LAI := LAI + DeltaHUF * plant_MaxPotLAI
            * (1 - exp(5 * (LAI - plant_MaxPotLAI))) * sqrt( CropStressFactor )
```

where *DeltaHUF* is the daily change in the heat unit factor *HeatUnitFactor, plant_MaxPotLAI* is the maximum possible LAI for the crop, and *CropStressFactor* is the value of the minimum crop stress factor: WaterStress or Temperature Stress.

Water stress factor is calculated as:

```
WaterStress := sumActTrans / sumPotTrans;          and WaterStress <= 1.
```

with *sumActTrans* the daily actual transpiration (= daily root water uptake) and *saumPotTrans* the daily potential transpiration.

The temperature stress factor is estimated with the equation:

```
TemperatureStress := sin (   π/2 * (AirTemp - plant_BaseTemp)
                                    /  (plant_OptimumTemp - plant_BaseTemp)   );
```

where *plant_BaseTemp* is the base temperature for the crop in °C and *plant_OptimumTemp* is the optimum temperature for the crop in °C.

The heat unit factor, *HeatUnitFactor* is computed using the equation:

```
HeatUnitFactor := AccUnits / (AccUnits
        + exp( plant_Point1LAI - plant_Point2LAI * AccUnits );
```

where *AccUnits* is the daily heat unit index, and *plant_Point1LAI* and *plant_Point2LAI* are parameters of the crop, computed during object initialization to fit an S-shaped growth curve.

As soon as the *AccUnits* is more than the *plant_DeclineFraction* the *LAI* is calculated as follows:

$$LAI := DeclineLAI * ((1 - AccUnits)/(1 - DeclineUnits))^{plant\_LAIDeclineRate}$$

where *DeclineLAI* and *DeclineUnits* are the LAI and the accumulated units at the day the LAI starts to decline and plant_DeclineRate is a crop parameter that governs the LAI decline rate.

The crop height *CanopyHeight* , is estimated with the function:

```
CanopyHeight := crop.CanopyHeight * Sqrt( HeatUnitFactor);
```

where *crop.CanopyHeight* is the maximum height for the crop.

The daily increase in rooting depth is calculated as:

```
RootingDepth := RootingDepth + 2.5 * plant.MaxRootingDepth * DeltaHUF;
```

where *plant.MaxRootingDepth* is the maximum root depth for the *crop. RootingDepth* is also limited by the *MaxRootingDepth* and the soil profile depth.

The daily increase in biomass *DailyTotalDM* in t/ha is calculated as a function of the photosynthetically active radiation, *PAR,* and the crop parameter for converting energy to biomass *plant_BiomassEnergy,* in $kg.m^{-2}.ha^{-1}.MJ^{-1}$. It is adjusted with the crop growth regulating factor *CropStressFactor* and the CO_2-enhancement factor *ARatio*:

```
DailyTotalDM := 0.001 * plant_BiomassEnergy * PAR * CropStressFactor * ARatio;
```

Photosynthetic active radiation in turn is estimated as:

```
PAR := 0.5 * TheCompModel^.MySolRad(MyCropI);
```

where *PAR* is the photosynthetic active radiation in MJ/m^2, the *MySolRad* function returnes the solar radiation in MJ/m. See the Competition module for a description of how the solar radiation is divided among multiple crops; in the case that there is only one crop (in this case, EPIC), the function is equivalent to the daily solar radiation times this crop's transpiration fraction (see function `GetTransFrac`).

Daily change in root weight (in t/ha) is computed with the equation:

```
DailyRootDM := DailyTotalDM * (0.4 - 0.2 * AccUnits);
```

The daily changes in dry matter are accumulated in *AccTopDryMatter*, *AccRootDryMatter* and *AccTotalDryMatter* after division by 10 to convert to kg/m^2.

Finally the yield *AccYield* (in t/ha) is calculated as:

```
AccYield := AccTopDryMatter * [ (HIA - plant_MinHI) *
WUR/(WUR + exp(6.13 - 8.83 * WUR) / 100)  +  plant_MinHI ];
```

and

```
HIA := plant_HarvestIndex * AccUnits / (AccUnits + exp(11.1 - 10*AccUnts) /100);

WUR := AccActTrans / AccPotTrans,        WUR <= 1;
```

where *HIA* is the simulated potential harvest index, *plant_MinHI* is the minimum harvest index for the crop, *WUR* is the water use ratio, *plant_HarvestIndex* is the potential harvest index, *AccActTrans* is the actual plant water use rate in mm/d and *AccPotTrans* is the potential water use rate in mm/d.

Plant height is limited by plant weight, so that plants can't be unrealistically thin. First, the specific mass of the plant, in kg pl^{-1}, is computed as:

```
PlantSpecMass := (AccTotalDryMatter { kg m-2 } / plant_PlantDensity { pl ha-1 })
        * 10000 { m2 ha-1 }; { -> kg pl-1 }
```

Then, the model checks whether this specific mass is too small to support the plant height that was calculated previously; if so, it limits the height accordingly:

```
if PlantSpecMass > 0.0001 then
        if crop.CanopyHeight / PlantSpecMass > plant_MaxHeightPerWeight then
                crop.CanopyHeight := PlantSpecMass * plant_MaxHeightPerWeight;
```

Public Function GetTransFrac

The proportion of light intercepted for this crop as a monoculture is computed as:

```
GetTransFrac := 1 - exp(- plant_ExtinctionCoeff * LAI)
```

where *LAI* (m^2 leaf m^{-2} ground) is the leaf area index of the growing crop and *plant_ExtinctionCoeff* is the extinction coefficient.

5.5 Competition

GAPS V3.5 and later provide for multiple crops to be growing at once (the limit is set by the constant *global.MaxCrops*). The competition module, found in file `compete.pas`, has some non-object procedures, but is mainly made up of a hierarchy of objects and their methods, similar to the organization of the crop models. The *TheCompModel* variable is a pointer to any competition model, and is always initialized to a non-nil value. Currently there are two competition models:

1. **comp_model** : associated with Comp_option = Comp_none : No competition
2. **AL_model** : associated with Comp_option = Comp_AL : ALMANAC

Some aspects of competition are found in other modules:

- plant water uptake with multiple crops were implemented as generalizations of the existing plant water uptake procedures `SimpleWaterUptake` and EPICWaterUptake.

5.5.1 S-curve procedures

The competition module has two utility procedures associated with the S-shaped growth curves that are used in several crop models:

```
source file: compete.pas
public procedure: FitSCurve
public function: SCurve
```

Here is a MathCAD worksheet that may help visualize the S-curve:

S-curves
General form: $S(y_1, y_2, x) := \frac{x}{\left[x + e^{\left[y_1 - (y_2 \cdot x)\right]}\right]}$

Analytical solution for curve parameters

$$a(x, f) := \ln\left[\frac{-x \cdot (f - 1)}{f}\right] \qquad y2(x_1, f_1, x_2, f_2) := \frac{a(x_2, f_2) - a(x_1, f_1)}{x_1 - x_2}$$

$$y1(x_1, f_1, x_2, f_2) := a(x_1, f_1) + y2(x_1, f_1, x_2, f_2) \cdot x_1$$

Enter the two calibration points:

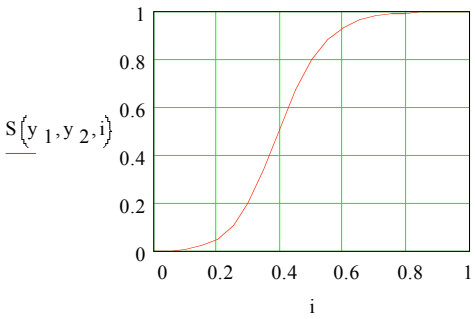
$x_1 := .3$ $\quad f_1 := .2$ $\quad x_2 := .5$ $\quad f_2 := .8$

Calculated **S-curve parameters**:

$a(x_1, f_1) = 0.182$ $\quad y_2 := y2(x_1, f_1, x_2, f_2)$ $\quad y_2 = 11.309$

$a(x_2, f_2) = -2.079$ $\quad y_1 := y1(x_1, f_1, x_2, f_2)$ $\quad y_1 = 3.575$

Graph of this function $\quad i := 0, .05 .. 1$



FitSCurve

```
procedure FitSCurve(x1, f1, x2, f2 : single;
                    var y1, y2 : single)
```

This procedure fits an S-shaped curve of the form:

f = x / (x + exp(y1 - y2 * x))

from any two points on the curve: (x1, f1) and (x2, f2). It does this with the explicit solution:

y2 := (a(x2, f2) - a(x1,f1)) / (x1 - x2)
y1 := a(x1, f1) + (y2 * x1)

where

a(x, f) := ln((-x * (f - 1)) / f)

SCurve

```
function SCurve(x, y1, y2 : single) : single
```

This function determines the functional value of an S-shaped curve of the form:

f = x / (x + exp(y1 - y2 * x))

5.5.2 Competition model: none

```
source file: compete.pas
public procedures: Init, Done, BeginDay, TimeStep
public functions: MySolRad, GetTransFrac, GetPotTrans
```

This is the base competition model, and provides a skeleton for other competition models, as well as default implementations of all public procedures and functions.

Constructor Init

This constructor is called by the simulation driver as part of its initialization (`Init_Proc`).

compete.TheCompModel := misclib.Init_Comp

where procedure `Init_Comp` then calls this constructor, via the `New` statement:

```
case Comp_option of
    Comp_none :
        Init_Comp := New(comp_model_ptr, Init);
    Comp_AL : { ALMANAC }
        Init_Comp := New(AL_model_ptr, Init);
    else
        Init_Comp := New(comp_model_ptr, Init);
end { case }
```

The *Comp_option* variable is set to *Comp_none* when GAPS is loaded (with a **const** statement in `compete.pas`) and then set to this or another option when the sequence file (if any) is read (`fileio.load_sequence`).

The constructor initializes the object's variables to zero (no radiation or transpiration for any crop).

Destructor Done

This destructor is called by the simulation driver as part of its finalization (`Finalize`):

Dispose(TheCompModel, Done)

Its only purpose is to deallocate the heap space used by the object. Even though the Pascal procedure is empty, the compiler generates additional code to release the correct amount of memory from the heap.

Public procedure: BeginDay

This procedure is called by the simulation driver at the beginning of each day when there are any active crop models, after solar angles are computed but before any crop events, including planting and harvesting.

Its first purpose is to **partition the day's solar radiation** among the active crop models. In the default 'no competition' model, the first active crop gets all the radiation:

```
SolRadFrac[first_crop] := TheCrop[crop_i]^.GetTransFrac;
SolRad[first_crop] := clim.SolRad[real_day] * SolRadFrac[first_crop];
```

where *first_crop* is the first-listed crop in the cropping sequence file that is active. All other crops receive no radiation:

```
SolRad[crop_i] := 0;
SolRadFrac[crop_i] := 0
```

where *crop_i* are all the crops except *first_crop*.

These two arrays *SolRadFrac* and *SolRad* are stored in the competition object, and are provided to interested parties (i.e. the respective crop models) via the virtual access method *MySolRad*, which is defined in the base competition model.

Its second purpose is to **compute an overall potential transpiration fraction** for all crops taken together, to be reported with the access function `GetTransFrac`. In the default 'no competition' model, this is simply the transpiration fraction from the first active crop, which was computed immediately before as *SolRadFrac*[*first_crop*]:

```
TransFrac := SolRadFrac[first_crop]
```

Public procedure: TimeStep

This procedure is called by the simulation driver as part of its `TimeStep_Procedures`, each time step when there are any active crop models, after solar angles, precipitation, soil temperature, ETP computation and partitioning, but before any plant procedures.

The default 'no competition' model does nothing at this time scale.

Public function: MySolRad

This function supplies the daily solar radiation (MJ m^{-2} d^{-1}) available for one crop model. This is simply the value of *SolRad*[*crop_i*] computed by the competition model as part of its daily procedure and stored for later access via this function.

This function is called by crop models to determine their portion of the daily solar radiation.

Since each competiton model has its own `BeginDay` procedure, it can allow the default `MySolRad` to be called.

Public function: GetTransFrac

This function returns the fraction of total solar radiation that could be used for transpiration (i.e., the fraction that hits the leaves), for all crops taken together. It is called by the `PartitionETP` procedure (in atmoslib.pas), to partition total ETP into potential evaporation and potential transpiration. The function simply returns the value of the object field *TransFrac*, which is computed in procedure `BeginDay` of the various competition models.

Public function: GetPotTrans

This function returns the potential transpiraton rate for a single crop. It is called from the `SimpleWaterUptake` and `EPICWaterUptake` time-step procedures, at the point when these uptake procedures need to know this rate in order to determine how much water is extracted from each soil layer.

The default 'no competition' competition model gives the crop that portion of the potential transpiration rate that corresponds to the proportion of the solar radiation captured by the leaves of this plant, calculated in `BeginDay`:

```
GetPotTrans := sim.ETP * SolRadFrac[crop_i]
```

Since each competiton model has its own `BeginDay` procedure, it can allow the default `GetPotTrans` to be called.

5.5.3 Competition model: modified ALMANAC

```
source file: compete.pas
public procedures: Init, Done, BeginDay
```

The ALMANAC model was described by Kiniry *et al.* as a general, process-oriented model for two competing plant species. In its original form, it is a modication of the EPIC crop model, accounting for competition for light and water between a crop and a weedy competitior. The GAPS competition model called 'ALMANAC' follows the general structure of the original ALMANAC, with several major differences:

- only the competition aspects of ALMANAC are included; the atmospheric, soil, and plant water uptake processes modelled in ALMANAC (i.e. their EPIC components) have been removed, and the various GAPS methods of modelling these may be selected by the GAPS user. This allows the different crop models (e.g. SORKAM, Stockle-Riha, and Constant, as well as EPIC) to be used for the competing plant species.

- Competition for light follows the simpler and more tractable method of Wallace (1995), rather than the Spitters & Aerts (1983) method used in the original ALMANAC.

- Corrections to radiation-use efficiency due to vapor pressure deficit are not modelled (actually, these would be modelled in the EPIC crop model anyway, since they don't depend on competition).

- The competition for plant water uptake modelled in ALMANAC is included in the appropriate water uptake methods: Plant-available (a.k.a. 'Simple') and EPIC, because the competitive uptake is a simple extension of these methods. See these methods for details.

ALMANAC does not account for the actual horizontal and vertical geometry of plants in the field; instead the LAI and height are used to model spherical light interception.

This model object is descended from the generic competition model:

type AL_model = **object**(comp_model)…

It only over-rides the *BeginDay* procedure. It also is required to have a constructor and destructor.

Public procedure: BeginDay

This procedure is called by the simulation driver at the beginning of each day when there are any active crop models, after solar angles are computed but before any crop events, including planting and harvesting.

Its first purpose is to **partition the day's solar radiation** among the active crop models. To do this, it first collects information from each active crop: (1) the crop's **transpiration fraction**, and (2) the crop's **canopy height**.

The transpiration fraction must be supplied by an access procedure in each crop model[4]:

if (TheCrop[crop_i] <> nil) **then** TF[crop_i] := 1 - TheCrop[crop_i]^.GetTransFrac;

where *TheCrop[crop_i]* is a pointer to crop model *crop_i*, and *TF* is the complement of the transpiration fraction, i.e. it is the proportion of light that is *not* intercepted by the crop in monoculture. The model keeps a running sum of exponentials (= product of fractions) for total interception:

IntFact := IntFact * TF[crop_i]

where *IntFact* was initialized to unity. The model then queries each crop model for its current LAI (actually, it directly accesses a field in the object, which is not the preferred object-oriented approach; we should really use an access procedure):

LA[crop_i] := TheCrop[crop_i]^.LAI

The competition model assumes that each crop model expresses light interception by Beer's law, although in fact there are variations. For example, the Stockle model uses a polynomial function of LAI, and SORKAM uses a square-root function. However, at any given LAI, the same light interception could have been obtained from a Beer's Law equation with an appropriate *k*. So, to get all crop's interception on the same basis, we can calculate the presumed *k*. If we assume that *GetTransFrac* has the form:

1 - exp(-k*LAI), i.e.,
1 - GetTransFrac = exp(-k*LAI)

we can now back-calculate what *k* would have had to be to obtain the observed interception fraction:

K[crop_i] := -ln(TF[crop_i])/LA[crop_i]

The model queries each crop model for its current canopy height, using an access procedure, and keeps a running sum of the heights:

---

[4] The *GetTransFrac* access function is defined as a virtual object method in the generic crop model, which implements it as a simple Beer's Law function of LAI. So any crop model that can accept this function and which models LAI need not over-ride the virtual method.

```
H[crop_i] := TheCrop[crop_i]^.GetHeight;
sumH := sumH + H[crop_i]
```

Non-active crops receive the default values K = 0, TF = 1, LA = 0, H = 0, i.e. no light intercepted, no height, no leaves.

To partition the solar radiation to a crop using the method of Wallace (1995), we first determine *fs*, the fraction when the crop is completely shaded by the other crop, with Wallace's equation (6):

```
fs := (IntFact/TF[crop_i])*(1-TF[crop_i])
```

Recall that *IntFact* was the running product of the interception fractions, so that *IntFact/TF[crop_i]* is the running product of the interception fractions of all crops except crop *crop_i*. This is then multiplied by the percent captured by the crop in question. So *fs* is the fraction in complete shade: the lowest crop gets whatever light is left over after taller crops have intercepted what they can.

The other extreme is the case where the crop in question shades all other crops, then it would receive all the light it could intercept, i.e. its own transpiration fraction, here *1-TF[crop_i]*. Wallace proposes a simple linear interpolation between these two extremes, using the height ratio (his equation (10)):

```
f := H[crop_i] / sumH
```

Then the fraction of radiation going to the crop is found by interpolation:

```
SolRadFrac[crop_i] := (fs + (f * ((1-TF[crop_i]) - fs)));
```

and the actual radiation in MJ m^{-2} is:

```
SolRad[crop_i] := SolRadFrac[crop_i] * clim.SolRad[real_day]
```

These two arrays *SolRadFrac* and *SolRad* are stored in the competition object, and are provided to interested parties (i.e. the respective crop models) via the virtual access method *MySolRad*, which is defined in the base competition model.

The second purpose of this procedure is to **compute an overall potential transpiration fraction** for all crops taken together, to be reported with the access function `GetTransFrac`. This is simply the complement of the product of the various crop's interceptions (Wallace's equation (10)):

```
TransFrac := 1 - IntFact
```

As Wallace points out, this shows that "the total light intercepted by the mixture… is independent of the individual species heights, whereas the individual fractions… are not".

5.6. Soil water flow processes

GAPS provides three ways of modelling the state and fluxes of soil water in the one-dimensional soil profile.

The simplest in terms of both input parameter requirements, conceptual view of the soil profile and implementation is the 'tipping bucket', a capacity model based on the CERES maize model (Jones and Kiniry, 1986). This is an appropriate model if the modeller only has soil data on soil moisture contents at various field moisture states (saturation, field capacity, and permanent wilting point), and only needs to know the moisture content by depth.

A more fundamental point of view is taken by the 'Richards equation' model, in which water fluxes are driven by potential energy gradients of water. This approach keeps track of the soil water potential, soil water contents, and unsaturated hydraulic conductivities by depth over time. It requires as input parameters the soil moisture release curve (matric potential vs. volumetric water content), the saturated hydraulic conductivity, and the air-entry potential of each layer. These parameters are usually obtained through detailed physical measurements. This is an appropriate model if the modeller has these physical data, and needs to know the fluxes and potentials by depth. If plant water uptake is being simulated as being controlled by leaf and root water potentials (e.g. 'Critical_leaf_water_potential' and 'WaterUptake' procedures), the Richards equation must be used.

The 'Matric flux potential' method is a variant of the Richards equation method, and has identical parameter requirements and output.

5.5.1. Richards equation

```
source file: soillib.pas
procedures:: Richards_Equation_Init, Richards_Equation
```

The soil-water balance is obtained by a numerical solution of the Richards equation, which describes water flow and storage in soil. This equation is obtained by applying the continuity equation to Darcy's law (Campbell, 1985):

$$w*CP*\ dWP/dt = d/dz(K(WP) * dWP/dz - K(WP)*GR) + U \qquad (1)$$

where: w is the density of water (kg m^{-3}), CP is the specific water capacity (CP = dθ / dWP), θ is the volumetric soil water content ($m^3\ m^{-3}$), WP is the soil water potential (J kg^1), t is time (s), z is depth (m), K(WP) is the soil hydraulic conductivity (kg s m^{-3}), which is a function of the water potential, GR is the acceleration of gravity (9.8 m s^{-2}), and U is a source/sink term (kg $m^{-3}\ s^{-1}$). In this discussion, z is positive downward and WP is always negative.

This equations has to be solved numerically for realistic boundary conditions.

This model uses a network analysis approach to describe water transfer. The soil profile is divided into a number of layers whose properties are assumed to be concentrated at the nodes. The nodes are connected through conductors and associated with capacitors analogous to the electrical circuit problem (Campbell, 1985). The network problem can be solved numerically to determine the change in water potential and water content with time at each node. Steady flow occurs within each element. Storage is assumed to occur only at the nodes. Mass balance equations are written for each node of the profile and solved for the unknowns using an iterative process, the Newton-Raphson method (Campbell, 1985). Similar schemes have been used by Stoeckle (1985), Weaver (1984) and Bristow (1983) to describe different aspects of the soil-plant-atmosphere continuum.

This model simulates infiltration, redistribution, and soil evaporation. Input requirements for this procedure include soil physical parameters such as bulk density, particle density, saturated hydraulic conductivity, air entry potential, and soil b-value (slope of the moisture release curve). If these parameters have not been measured, they can be estimated from soil textural data (Campbell, 1985).

The 'Richards Equation' water flow method is suitable to use with the 'Potential-driven water uptake' method, both conceptualizing the soil water in terms of its energy.

At the beginning of simulation, the procedure `Richards_Equation_Init` is called to set up the module's private variables. In particular, this procedure computes the inter-node saturated hydraulic conductivities, given the layer saturated hydraulic conductivities in the soil input file. In the present

implementation, the inter-node conductivity is simply taken as the lesser (slower) of the layer conductivities of the layer containing the node, and the one below it.

At each time step, the procedure `Richards_Equation` is called to compute the soil water balance. This procedure is broken down into a series of sub-procedures:

• `Hydraulic_Conductivities`: calculate unsaturated hydraulic conductivities for each layer;, from the current water potentials
• `Soil_Evaporation`: compute actual evaporation from the top soil layer;
• `Jacobian`: set up simultaneous equations for determining new water potentials from the current water potentials and contents and hydraulic conductivities;
• `Thomas_Algorithm`: solve this system of equations;
• `New_Water_Contents`: calculate the new water contents for each layer from the updated water potentials

These are called in turn, as many times as necessary until the numerical solution of the Richards' equation converges, i.e., the mass-balance error is less than a pre-defined limit of 10^{-6} kg m^{-2} s^{-1}. Non-convergence is detected by counting the number of iterations, and halting the simulation if a pre-defined limit of 256 is exceeded.

Once the new water contents equilibrate by this iterative procedure, the sub-procedure `Calculate_Fluxes` is called to record the new water contents and related fluxes.

These sub-procedures are now discussed in turn.

5.5.1.1. Sub-procedure Hydraulic_Conductivities

This procedure calculates unsaturated hydraulic conductivities for each layer from the soil water potentials of the most recent time step, or the initial soil water potentials at the beginning of a simulation. The unsaturated hydraulic conductivity *sim.K* for each soil profile layer is calculated according to Campbell (1974):

$$sim.K_{layer} := soil.HydCond_{layer} * (soil.AirEntryPot_{layer} / sim.WP_{layer})^{N_{layer}} \quad (1)$$

where *soil.HydCond* is the saturated hydraulic conductivity (kg s m^{-3}), *soil.AirEntryPot* is the air entry potential (J kg^{-1}), *sim.WP* is the soil water potential (J kg^{-1}), and the exponent *N* is computed as:

$$N_{layer} := 2+ (3 / soil.BValue_{layer}) \quad (2)$$

where *soil.BValue* is the slope of the water release curve (moisture content vs. water potential), when plotted on log-log scale.

5.5.1.2. Sub-procedure Soil_Evaporation

This procedure computes the actual evaporation from the soil surface. First, the relative humidity *HA* of the soil surface layer is calculated from the soil water potential:

$$HA := \exp(MW*sim.WP_{MinLayer} / (R*(sim.SoilTemp_{MinLayer}+273))) \quad (1)$$

where: *MW* is the molecular weight of water (0.018 kg mole^{-1}), *sim.WP$_{MinLayer}$* is the simulated soil surface node water potential at the most recent time step (J kg^{-1}), *R* is the gas constant (8.3143 J mole^{-1} °K^{-1}), and *sim.Soiltemp* is the soil temperature this time step (°C). Once we have the relative humidity *HA*, the evaporative flux *sim.ActEva* (kg m^2 s$^{1)}$ of water from the surface layer is then calculated from this as:

```
sim.ActEva := sim.PotEva *
              (HA – (clim.RelHumid[day])/(1 – clim.RelHumid[day])))
```
(2)

where *sim.PotEva* is the potential soil evaporation rate (kg m^{-2} s^{-1}), and *clim.RelHumid* is the relative humidity of the air.

Evaporation during second and third stage drying is controlled by the humidity of the evaporative surface and the liquid flux to the soil surface from deeper soil layers. The derivative, with respect to time, of the evaporative flux *sim.DEva* (kg m^2 s$^{2)}$ is calculated for use in sub-procedure `Jacobian` as follows:

```
sim.DEva := sim.PotEva * MW * HA /
            (R * (sim.SoilTemp_MinLayer+273) * (1 – clim.RelHumid[day]))
```
(3)

5.5.1.3. Sub-procedure Jacobian

This procedure sets up simulataneous equations for determining new water potentials from the current water potentials and contents.

The water flux *WFlux* (kg m^{-2} s^{-1}, positive downward) in element *I* may be determined from the current water potentials *sim.WP* (J kg^{-1}) of this layer and the one above ('upstream from') it, the current hydraulic conductivity *K* (kg s m^{-3}) above the node, and a moisture release parameter *N* using the relation:

```
sim.WFlux[I] := – (K[I+1] * sim.WP[I+1]  –  K[I] * sim.WP[I]) /
                ((NodeDelta[I+1]) * (1 – N[I]))
```
(1)

where *N[I]* is related to the slope of the moisture release curve *BValue[I]* by:

```
N[I] := 2 + (3 / soil.BValue[I])
```
(2)

(after Campbell 1974) and *NodeDelta[I]* is the distance(m) from of soil node *I* to the node below it. This is computed before simulation starts, in procedure `InitSoil` (in source file 'misclib.pas') from the soil input data as:

```
NodeDelta[I] := NodeDepth[I+1] – NodeDepth[I]
```
(3)

where *NodeDepth[I]* is computed as the midpoint of soil layer *I* :

```
NodeDepth[I] := soil.LowBound[I–1] + (LayThick[I] /2)
```
(4)

where *soil.LowBound[I–1]* is the upper boundary of layer *I*, and *LayThick[I]* is the thickness of layer *I*, computed as:

```
LayThick[I] := soil.LowBound[I] – soil.LowBound[I–1]
```
(5)

Combining this equation with the mass balance equation for layer *I*:

$$FGauss[I] := WFlux[I-1] - WFlux[I] + U(I-1) - U(I) + KgFines[I] * (WN[I] - W[I]) * time\_step \quad (6)$$

where *KgFines[I]* (kg m^{-3}) converts from volume to mass in a layer, and letting the source term U[I] = GR*K[I], i.e., the flux into the node is the hydraulic conductivity times the driving force of gravity *GR* (m s^{-2}), yields the mass balance error *FGauss* for each node:

$$FGauss[I] := \frac{K[I] * sim.WP[I] - K[I-1] * sim.WP[I-1]}{(1-N[I]) * NodeDelta[I-1]} - \frac{K[I+1] * sim.WP[I+1] - K[I] * sim.WP[I]}{(1-N[I]) * NodeDelta[I]} - GR * (K[I-1] - K[I]) + \frac{(sim.WN[I] - W[I]) * KgFines[I]}{time\_step} \quad (7)$$

where *W* is the old water content (m^3 m^{-3}) and *sim.WN* is the new water content (m^3 m^{-3})

An appropriate mean water potential is calculated for each node using a weighted mean of the water potential at the i-th time step and the water potential at the (i+1)th time step:

$$WP[I] = n * WP[I]_{[J+1]} + (1-n) * WP[I]_{[J]} \quad (8)$$

where *J* denotes time, *I* denotes the node number, and *n* is a weighing factor between 0 (forward difference or 'explicit' method) and 1 (backward difference or 'implicit' method). When using the forward difference method, fluxes, conductivities, and capacities are calculated using the water potential gradient at the beginning of the time step. If n=1, fluxes are evaluated using the new water potentials only. If n=0.5 the method is called 'time-centered' or the Crank-Nicholson method and the arithmetic average of the water potential at the beginning and end of the time step is used to calculate fluxes of water. The choice of n depends on the time constant of the system and strongly affects the numerical stability and accuracy of the solution. When n=0 the solution can be unstable if the time steps chosen are too large or the fluxes of water entering the profile are great. For water flow problems, n=1 almost always gives the best results and will be used hereafter.

Other sources/sinks not represented in (6), such as root extraction, gravitational flux, precipitation, or irrigation, are added explicitly to the mass balance term in (7) and such become part of the solution. Root extraction, i.e. *sim.WUptake[]*, is added as a sink to each layer. Evaporation is assumed to be only from the surface, so *sim.ActEva* is added as a sink to the top soil layer only. Water input from precipitation or irrigation, *sim.WInput[MinLayer]*, is assumed to be only at the surface, so this term is added as a source to the top soil layer only. GAPS at present can not simulate ground water tables or lateral fluxes from soil layers; this could be done by calculating source or sink terms *WInput[]* for the lower layers.

Equation (7) can be written for each node in the simulated soil profile resulting in *LastLayer* equations in *LastLayer*+2 unknowns. Boundary conditions are used to reduce the number of unknowns by two so that the system of equations can be solved simultaneously. The three choices for boundary conditions are (1) measured values, (2) assumed constant flux and (3) assumed constant potential. For the profile's upper boundary, the soil water potential can be set to the air entry potential during infiltration in order to simulate saturation, or to a specified flux density in the case where the water input is less than the infiltration rate. At the bottom of the profile, the potential may be set to a known value, for example zero

for a water table. The potential could be set at some value that can be considered constant for the duration of the run. Or there could be some depth at which no water flux is assumed to occur.

Writing (7) for each node and expressing the set of simultaneous equations as a tridiagonal matrix results in the so-called 'Jacobian' matrix, shown below for a soil with four layers. The *BGauus[]* terms are the diagonal elements, the *AGauss[]* terms are the lower diagonal, and the *CGauss[]* are the upper diagonal.

$$\begin{vmatrix} BGauss(1) & CGauss(1) & 0 & 0 \\ AGauss(2) & BGauss(2) & CGauss(2) & 0 \\ 0 & AGauss(3) & BGauss(3) & CGAuss(3) \\ 0 & 0 & AGauss(4) & BGAuss(4) \end{vmatrix} * \begin{vmatrix} DP(1) \\ DP(2) \\ DP(3) \\ DP(4) \end{vmatrix} = \begin{vmatrix} FGauss(1) \\ FGauss(2) \\ FGauss(3) \\ FGauss(4) \end{vmatrix}$$

where: *FGauss[I]* is the mass balance error for node *I*, and the elements of the Jacobian are computed as the derivatives of the mass balance term by the the water potential at the appropriate node (i.e. above, at, and below the current node):

$$\begin{aligned} CGauss[I] &= d\ FGauss[I] / d\ sim.WP[I+1] \\ &= -K[I+1] / NodeDelta[I]) \end{aligned} \quad (10)$$

$$\begin{aligned} BGauss[I] &= d\ FGauss[I] / d\ sim.WP[I] \\ &= (K[I] / NodeDelta[I-1]) + (K[I] / NodeDelta[I]) \\ &+ CP[I]] \\ &– (GR * N[I] * K[I]/ sim.WP[I]) \end{aligned} \quad (11)$$

except at the top node, where the derivative of the water potential also includes the derivative of the rate of actual evaporation, and there is no higher layer:

$$\begin{aligned} BGauss[MinLayer] &= d\ FGauss[MinLayer] / d\ sim.WP[MinLayer] \\ &= (K[MinLayer] / NodeDelta[MinLayer]) \\ &+ CP[MinLayer]] \\ &– (GR * N[MinLayer] * K[MinLayer]/ sim.WP[MinLayer]) \\ &+ sim.DEva \end{aligned} \quad (11')$$

$$\begin{aligned} AGauss[I] &= d\ FGauss[I] / d\ WP[I-1] \\ &= -K[I-1]/(NodeDepth[I]-NodeDepth[I-1]) \\ &+ GR*N[I]*K[I-1]/sim.WP[I-1] \end{aligned} \quad (12)$$

except at the top node, where this is not in the matrix (to the left off the diagonal in the top row), so is not computed.

where *DP[I]* is the change in water potential over one iteration.

The node soil water capacity term *CP* (kg m^{-2}) is calculated as:

$$CP[I] = –KgFines[I] * sim.WN[I] / (soil.BValue[I] * sim.WP[I] * time\_step) \quad (13)$$

where *KgFines[I]* (kg m^{-3}) converts from volume to mass in a layer, *time_step* is the time step (s), *sim.WN* is the simulated water content (m^3 m^{-3}), and *soil.BValue* is the slope of the water release curve, which converts the simulated water potential *sim.WP* to a water content.

After each iteration, the mass balance error *SE* (kg m^{-2} s^{-1}) for the whole soil profile is determined by summing the mass balance errors from all the nodes:

SE=SE+ABS(FGauss[I]) (14)

Convergence is determined by checking whether the FGauss[I]'s are sufficiently close to zero. When the total mass balance error is less than the allowable mass balance error the procedure is completed.

5.5.1.4. Sub-procedure Thomas_Algorithm

In this procedure the set of equations set up by procedure `Jacobian` is solved by Gauss elimination, and a new set of soil water potential changes *DP[I]* is found and once again substituted into the Jacobian matrix. The soil water potential changes for each layer are calculated as:

DP[I] = FGauss[I] – CGauss * DP[I+1] (1)

except at the bottom boundary, where the back-substitution is started by:

DP[LastLayer] = FGauss[LastLayer] / BGauss[LastLayer] (1')

The water potential is updated for each node as follows:

sim.WP[I] = sim.WP[I] – DP[I] (2)

except at the bottom boundary, where the potential for the dummy bottom layer is set equal to that of the actual last layer of the soil, thereby re-establishing the bottom boundary condition.

The new water potentials are not allowed to exceed the air entry potential for the layer.

When the *DP[I]*'s for all the nodes in the profile become zero, the correct values for the soil water potentials have been found. Convergence is determined in procedure `Jacobian` by checking whether the FGauss[I]'s are sufficiently close to zero.

5.5.1.5. Sub-procedure New_Water_Contents

This procedure forms part of the iterative computation of the water status of the soil. It is called iteratively after the new water potentials have been determined in procedure `Thomas_Algorithm`, to determine the new water contents *sim.WN[]* for each layer, given the updated water potentials, according to the following empirical relation from Campbell (1985):

$$sim.WN[I] = WS[I] * (soil.AirEntryPot[I] / sim.WP[I])^{1/soil.BValue[I]} \quad (1)$$

where: *WS[]* is the saturated water content of the layer (m^3 m^{-3}), *soil.AirEntryPot[]* is the air entry potential at the node (J kg^{-1}), *sim.WP[]* is the water potential at the node (J kg^{-1}), and *soil.BValue[]* is the slope of the water release curve of the layer (moisture content vs. water potential), when plotted on log-log scale.

5.5.1.6. Sub-procedure Calculate_Fluxes

Once the solution to the Richards Equation satisfies mass balance criteria, this procedure is called to calculate the changes in amount of water for each layer and the whole profile. Fluxes between layers are also calculated and new water contents, which will be used as initial conditions at the next time step, are determined.

The water content change for each layer in the profile is calculated:

$$Node\_WC\_Chg[I] = KgFines[I] * (sim.WN[I] - W[I]) \quad (1)$$

where: *KgFines[I]* (kg m^{-3}) converts from volume to mass in a layer, *sim.WN[]* is the updated water content just computed during the last iteration of the Richards Equation solution (m^3 m^{-3}), and *W[]* is the water content at the beginning of the time step (m^3 m^{-3})

The changes for each layer are summed up to get the change in amount of water for the whole profile, and the rate of change *WC_Change* (kg m^{-2} s^{-1}) is calculated by dividing the total flux by the time step.

The fluxes for each node are calculated from the final values of the water potentials at the node and the one below it:

$$sim.WFlux[I] := ((sim.K[I] * sim.WP[I]) - (sim.K[I+1] * sim.WP[I+1]) / ((1-N) * NodeDelta[I])) + (GR * sim.K[I]) \quad (2)$$

Once we no longer need the difference between old and updated water contents, the 'old' water contents for the next time step are set equal to the 'new' ones from this step:

$$sim.WN[I] := W[I] \quad (3)$$

and the rate of drainage from the lower boundary of the soil profile *Drain_Rate* (kg m^{-2} s^{-1}) is calculated, assuming no matric potential induced flux at the bottom of the soil profile, as:

$$Drain\_Rate = GR * K[LastLayer] \quad (4)$$

where: *GR* is the acceleration of gravity (9.8 m s^{-2}), and *K[LastLayer]* is the hydraulic conductivity of the bottom layer of the profile (kg s m^{-3}). This equation shows that we assume that the only force on the water in the bottom layer is gravity downwards. Clearly this relation would have to be modified for soils within the influence of a water table.

5.5.2. Tipping bucket

```
source file: soillib.pas
procedures:: Tipping_Bucket_Init, Tipping_Bucket,
             Tipping_Bucket_Done
```

This module uses the soil water flow approach contained in the CERES Maize model (Jones and Kiniry, 1986). The code from the CERES-Maize was translated into Pascal without altering the general computational method. In addition, a soil evaporation routine was added by D. Rossiter to allow for evaporation from near-surface layers below the top soil layer.

As the name of this procedure suggests, the soil is conceptualized as consisting of a series of buckets, each having a specified capacity to hold water. Each 'bucket' corresponds to a soil layer, the dimensions and properties of which are specified in the soil input data file. Water is transferred from one layer to the next downward in the soil profile if the amount entering the layer exceeds the layer's water holding capacity. This is analogous to 'tipping' a bucket, thereby letting some water spill out of it to the one below. The capacity of each soil layer to hold water is calculated as the difference between the saturation water content *WS[]* and the current volumetric water content *sim.WN[]*, both expressed as (m^3 m^{-3}).

This method is not iterative; the capacities at the beginning of the time step are used to estimate the redistribution of water during that time step. Also, in this method water can not move upwards between layers (although water can evaporate from the upper layers).

Input requirements for this procedure include values for bulk density and particle density to calculate saturation water content, field capacity or 'Drained Upper Limit' *DUL[]* and wilting point or 'Lower Limit' volumetric water contents *LL[]*. Ideally, values for *DUL* and *LL* should be obtained from field measurements as described by Jones and Kiniry (1986). They can be estimated using the method presented in the 'Workability and Trafficability' section of this manual, using equation (1) of that section and its associated tables, using a water potential ψ = 100cm (= -9.8 MJ kg^{-1} = -0.1mbar) for the upper limit and ψ = 15 000cm (= -147.15 MJ kg^{-1} = -15mbar) for the lower limit These values can also be estimated from soil textural data and organic carbon content using empirical equations contained in CERES Maize (Ritchie et al., 1986) and later work along the same line, e.g. Ritchie & Crumb (1988).

The 'Tipping Bucket' water flow method is well-matched with the 'Plant available water uptake' method, both conceptualizing the soil layers as compartments which have a volumetric water capacity.

At the beginning of the simulation, the procedure `Tipping_Bucket_Init` is called to determine time-invariant coefficients for the module, including evaporation and drainage coefficients.

At each time step, the procedure `Tipping_Bucket` is called to compute the soil water balance. This procedure is broken down into a series of sub-procedures:

- `Null_Setting`: initialize inter-layer fluxes and drainage rate to zero;
- `Infiltration`: model the downward flow of water from 'bucket' to 'bucket';
- `Soil_Evaporation`: evaporate from the soil surface;
- `Calculate_Fluxes`: record the new water contents.

These sub-procedures, as well as the main (control) procedure, are now discussed in detail.

5.5.2.1. Procedure Tipping_Bucket_Init

This procedure is called at the beginning of the simulation, to estimate time-invariant coefficients used in the dynamic simulation. These deal with inter-layer drainage, and soil evaporation.

Drainage

The profile drainage coefficient *Profile_SWCON* (a fraction on [0..1]) is used in the sub-procedure `Infiltration` to limit the amount of water that can move out of a layer in one time step. This is to reflect the fact that water must move through pores at a finite speed. The drainage coefficient is thus a substitute for the saturated hydraulic conductivity.

The profile coefficient is estimated as the slowest (i.e. numerically smallest) of the layer drainage coefficients *SWCON*, which are estimated as the ratio of the pore space above the drained upper limit to to the total pore space:

$$SWCON := (WS_{layer} - DUL_{layer}) / WS_{layer} \quad (1)$$

Here, *WS[]* , which is the saturated water content ($m^3\ m^{-3}$), is taken as a direct estimate of the total pore space. This formula implies that drainage is more rapid with a greater difference between saturation and field capacity, which corresponds to more large, freely-draining pores.

Evaporation

Evaporation from the soil surface is modelled in sub-procedure `Soil_Evaporation`. That sub-procedure needs a static estimate of:

(1) how deep in the profile evaporation is effective, and

(2) how dry each near-surface layer can become solely due to evaporation.

These are determined as variables *EvapLastLayer* and *P^EvapFrac[]*, respectively. This latter variable is a dynamic Pascal variable, allocated on the heap at run-time, since the number of layers in which evaporation is effective can't be determined until that time.

In sub-procedure `Soil_Evaporation`, the term

soil.DUL[layer] * P_EvapFrac^[layer] (1')

represents a limiting water content, below field capacity, to which evaporation can dry the layer (n.b. the layer can dry to field capacity simply by drainage). The evaporation fractions *P_EvapFrac^[]* are the fractions (on [0..1]) of field capacity to which each layer can be dried by evaporation. Now we discuss how these are determined.

Both calculations begin with one input parameter, *loca.depth_of_evap* (m), which is the user-defined depth to which the soil can be dried to or below the drained upper limit by evaporation. This depth should be based on observations of the soil after it has drained and been subject to evaporation for several days. The default value is 0.05m (5 cm).

The evaporation fraction is a single value for the entire layer, however, its calculation is based on a continuous function of evaporative potential by depth. At the soil surface, we assume that evaporation could dry the soil completely, i.e. at a depth of 0 m, the evaporation fraction is assumed to be 0. At the maximum depth to which evaporation is effective, evaporation can dry the soil only to field capacity, i.e. at a depth of *loca.depth_of_evap* m, the evaporation fraction is assumed to be 1. We want to fit a continuous function between these two points, and we choose a logarithmic curve (i.e. inverse exponential) which passes through these points:

$$y = \ln ([(z / depth\_of\_evap) * (e-1)] + 1) \quad (1)$$

where *z* is the independent variable 'depth from the surface', *y* is the dependent variable, i.e. the evaporation fraction on [0..1], and *e* is the Euler constant. This has the effect of decreasing potential evaporation exponentially by depth. This function can be simplified by combining the constants in (1) by setting R = (e–1) / depth_of_evap:

$$y = \ln (Rz + 1) \quad (1')$$

To integrate this (e.g. over a layer), we change variables:

$$x = Rz + 1 \;\Rightarrow\; z = (x - 1) / R \tag{2}$$

leading to the definite integral

$$\frac{1}{R}\int_{Rz_1}^{Rz_2} \ln x \, dx = \frac{1}{R}\left[x \ln x - x \right] \Big|_{Rx_1}^{Rx_2} \tag{3}$$

To determine the average fraction for a discrete layer, we evaluate this definite integral over the layer, and divide by the layer thickness. We do this for layers from the surface downward, until the evaporation fraction exceeds 1, i.e. evaporation is no more effective than drainage.

The actual computation of the evaporation fractions is now described. The first step is to fit the curve through two (depth, fraction) points: (0,0) and (*loca.depth_of_evap*, 1), by computing the scale parameter *R* as:

$$R := (\exp(1) - 1) / \text{loca.depth\_of\_evaporation} \tag{4}$$

To initialize the by-layer calculation, we start the lower boundary of the 'previous' layer (which in this case is the surface) and compute its contribution to the definite integral in (3) (not including the scale divisor *R*) as:

$$z := R * 0 + 1 \Rightarrow z := 1 \; ; \; x_2 = z * (\ln(z) - 1) \Rightarrow x_2 := \ln(1) - 1 \tag{5}$$

For each layer, we set $x_1 := x_2$, i.e. the top boundary of the layer is the bottom boundary of the previous layer, so the corresponding contribution to the definite integral in (3) are the same (with a change of sign), and we then calculate the new contribution to the definite integral in (3) by:

$$z := R * \text{soil.LowBound[layer]} + 1 \tag{6}$$

$$x_2 := z * (\ln(z) - 1) \tag{7}$$

Here, equation (6) expresses the change of variables (2). We now complete the computation of the definite integral (3) to obtain the evaporation fraction.

$$\text{Integral} := (x_2 - x_1) / R \tag{8}$$

and divide by the layer thickness to obtain an average evaporation fraction over the layer:

$$\text{EvapFrac[layer]} := \text{Integral} / \text{LayThick[layer]} \tag{9}$$

where *LayThick[]* is the layer thickness (m), so that *EvapFrac[]* is a fraction.

The procedure of equations (5) – (9) is followed for each layer until *EvapFrac[]* becomes greater than 1 (i.e. the layer on average is too deep for evaporation to affect it) or the bottom of the profile, whichever comes first. The final layer in which evaporation is effective is recorded in module variable *EvapLastLayer*, which controls the evaporation loop in the time-step procedure.

If the surface layer as specified in the soil input file is too thick, even the first layer's fraction will be greater than 1. To allow evaporation in this special case, we arbitrarily set the fraction to 1 and evaporate only from this layer.

Finally, this procedure allocates storage for the evaporation fractions, using the module variable *P_EvapFrac^*, and copies the evaporation fractions *EvapFrac[]* to the newly-allocated array, for use in the time-step procedure.

Procedure Tipping_Bucket (main procedure)

This is the control procedure, called at each time step from the simulation driver. Dynamic input to this procedure are the current volumetric water contents by layer, *W[]* (m^3 m^{-3}). It first calls sub-procedure Null_Settings to reset all inter-layer fluxes *Flux[]* and the drainage rate *DrainRate* from the bottom of the profile to zero. Then the water flux into the soil surface *Flux[MinLayer]* (kg m^{-2}) is computed as:

Flux[MinLayer] := WInput[MinLayer] * time_step (1)

where *WInput[MinLayer]* (kg m^{-2} s^{-1}) is the rate of water addition to the surface for this time step of duration *time_step* (s), which was calculated in procedure `Distribute_Precipitation` of the 'atmospheric processes' library, prior to this water flow procedure being called.

Next, for each layer from the surface downwards, the sub-procedure `Infiltration` is called to move water into and out of the layer, recomputing *Flux[layer]* to be the flux out of the layer, and computing new water contents *sim.WN[]*... For all layers below the surface, the layer's flux in is initialized to the previous layer's flux out. The flux out of the bottom layer is the drainage rate *Drain_Rate* (kg m^{-2} s^1) from the profile:

Drain_Rate := Flux[LastLayer] / time_step (2)

Next, the sub-procedure `Soil_Evaporation` is called to remove water by evaporation from the near-surface layers, reducing the simulated new water contents *sim.WN[]*.. Finally, the sub-procedure `Calculate_Fluxes` is called to calculate the change in water content for the whole profile in this time step, and to update the simulated water contents for the next time step.

Procedure Infiltration

This sub-procedure is called for each layer, with the input variable *Flux[layer]* (kg m^{-2}) being the amount of water moving from the upstream layer (which, in the case of the surface layer is the atmosphere) into this layer.

First, the amount of additional water that the layer can hold *Hold* (kg m^{-2}) is calculated as the amount of water between saturation water content and the current water content:

Hold := (WS[I] – W[I]) * KgFines[I] (1)

where *KgFines[]* (kg m^{-2}) converts from volume to mass, *WS[]* is the layer's saturation water content (m^3 m^{-3}), and *W[]* is the layer's current volumetric water content (m^3 m^3).

If the flux into the layer is greater than the amount the layer can hold, flow is assumed to be saturated, and the sub-procedure `Saturated_Flow` is called to calculate flux out of the layer and the new water content. This sub-procedure first determines the amount which can drain from the layer during the time step *Drain* (kg m^{-2}) as:

Drain := Profile_SWCON * (WS[I] – DUL[I]) * KgFines[I] (2)

where *(WS[I] – DUL[I])* is the volume of water that can be contained in a layer above field capacity, *KgFines[]* (kg m^{-2}) converts from volume to mass, and *Profile_SWCON* is a fraction on [0..1] that controls the rate of saturated flow. Since the fraction is always less than 1, equation (2) implies that only a part of the water above field capacity drains in each time step. This static fraction was computed in the module initialization, `Tipping_Bucket_Init`. The higher this fraction, the quicker is the drainage. With this fraction, we can proceed to calculate the updated water content and the flux out of this layer by:

sim.WN[I] := WS[I] – ((Drain + (sim.WUptake[I] * time_step) / KgFines[I] (3)

Flux[I]:=Flux[I] – Hold + Drain (4)

where *sim.WUptake[]* (kg m^{-2} s^{-1}) is the rate of water uptake from this layer by plants, calculated by one of the plant water uptake models earlier in the time step. The previous water content *W[]* does not appear in equations (2) and (3), because it was already taken into account in the computation of *Hold* in (1).

Note that the new water content will be a bit below saturation, and the flux out of the layer will be the same as the flux in, less the additional amount the layer can hold, plus the same amount of drainage that caused the new water content to be below saturation.

If the flux into the layer is less than the amount the layer can hold, flow is assumed to be unsaturated. First, the layer's new water content is calculated by:

sim.WN[I] := W[I] + ((Flux[I] – (sim.WUptake[I] * time_step) / KgFines[I] (3')

where the current water content *W[]* replaces the saturation water content *WS[]* of equation (3), and there is no saturated flow (the *Drain* term in (3)). This new water content will be less than the saturated water content.

If this new water content is not above field capacity (i.e. *sim.WN[I] <= soil.DUL[]*), then the infiltration process is stopped by setting the flux out of this layer *Flux[I]* to zero.

Otherwise, some water is assumed to drain out of this layer to the next, and the sub-procedure `UnSaturated_Flow` is called to calculate this flux out of the layer and re-adjusted the water content, as follows. This sub-procedure first determines the amount which can drain from the layer during the time step *Drain* (kg m^{-2}) as:

Drain := Profile_SWCON * (sim.WN[I] – DUL[I]) * KgFines[I] (5)

where *(sim.WN[I] – DUL[I])* is the volume of water presently contained in this layer above field capacity, and *Profile_SWCON* controls the speed of drainage, as explained above for equation (2). This amount is then moved out of the layer by recomputing the layer water content and the flux into the next layer:

sim.WN[I] := sim.WN[I] – Drain (6)

Flux[I] := Drain * KgFines[I] (7)

At the end of this subprocedure, *Flux[I]* is the amount of water moving out of this layer.

Procedure Soil_Evaporation

This procedure is called at each time step to determine the actual evaporation from the soil surface, and to reduce the volumetric water content of the near-surface layers accordingly. It follows a simple model of

satisfying the total evaporative demand downward from the surface, by layer, very much as the tipping bucket infiltration fills each layer before water moves down.

The mass of water *water_evaporated* (kg m^{-2}) that can be evaporated from a layer during the time step is calculated as:

water_evaporated := KgFines[layer]
* (sim.WN[layer] – (soil.DUL[layer] * P_EvapFrac^[layer]) (1)

where *KgFines[]* (kg m^{-2}) converts from volume to weight, *sim.WN[]* is the current volumetric water content for the layer as computed in sub-procedure `Infiltration` (m^3 m^{-3}), *soil.DUL[]* is the field capacity water content (m^3 m^{-3}), and *P_EvapFrac^[]* is the fraction (on [0..1]) of field capacity to which the layer can be dried by evaporation. (These fractions are computed once for the entire simulation in procedure `Tipping_Bucket_Init`.) Thus the term

soil.DUL[layer] * P_EvapFrac^[layer] (1')

represents a limiting water content, below field capacity, to which evaporation can dry the layer (n.b. the layer can dry to field capacity simply by drainage).

Of course there must be evaporative demand to actually dry the soil, so the above calculation is contained in a loop which works down from the surface, attempting to satisfy the total evaporative demand *to_evaporate* (kg m^{-2}), expressed as an amount rather than a rate:

to_evaporate := sim.PotEva * time_step (2)

For each layer from the top (*MinLayer*) to *EvapLastLayer* (determined in procedure `Tipping_Bucket_Init`), formula (1) is used to determine the amount of water available for evaporation from the layer. If this exceeds the remaining evaporative demand, the amount *water_evaporated* is decreased to the total available for evaporation, i.e. *to_evaporate*. In either case, the water is removed from the layer and added to the running sum of actual evaporation:

sim.WN[layer] := sim.WN[layer] – (water_evaporated / KgFines[layer]) (3)

sim.ActEva := sim.ActEva + (water_evaporated / time_step) (4)

and the remaining evaporative demand is decreased accordingly:

to_evaporate := to_evaporate – water_evaporated (5)

This loop continues until the last layer in which evaporation can occur is reached, or until there is no more evaporative demand, whichever occurs first. Note that according to this model, the surface layer dries down to its lowest possible water content before the first sub-surface layer dries below field capacity, and so forth.

It is certainly possible that not all the evaporative demand can be met, because there is not sufficient evaporable water in the near-surface layers. It is also possible that not all the evaporable water can be evaporated, because of insufficient evaporative demand.

Procedure Calculate_Fluxes

At the end of each time step's calculation, this procedure is called to calculates the change in profile water content this time step (which can be reported in the simulation output), as well as to reset the starting water contents for the next time step.

The rate of change in profile water content *WC_Change* (kg m^{-2} s^{-1}) is calculated in the obvious way:

$$WC\_Change := \frac{\left[\sum_{i=MinLayer}^{LastLayer} (sim.WN_i - W_i)\ KgFines_i\right] - DrainRate}{time\_step} \quad (1)$$

where the term *(sim.WN[i] – W[i])* represents the volumetric difference between the water contents at the end and beginning of the times step (m^3 m^{-3}), *KgFines[i]* (kg m^{-2}) converts from volume to mass per unit area, *DrainRate* (kg m^{-2}) is the flux out of the bottom of the profile, and *time_step* converts the mass to a rate.

The starting water contents for the next time step are reset by:

$$W[I] := sim.WN[I] \quad (2)$$

for each layer *I*.

5.5.3. Matric flux potential

```
source file: soillib.pas
procedures:: Matric_Flux_Potential_Init,
             Matric_Flux_Potential
```

This is a close variant of the Richards equation. The difference is that this procedure uses matric flux potentials instead of water potentials to compute the potential gradients that drive water flow.

In the initialization procedure, `Matric_Flux_Potential_Init`, reference values for the matric flux potentials, in the situation when the soil is equilibrated at the air entry potential, are computed as:

$$AirEntryMFP_{layer} := soil.HydCond_{layer} * soil.AirEntryPot_{layer} / (-1 - 3 / soil.BValue) \quad (1)$$

The procedure `Matric_Flux_Potential` is called at each time step to update the water status (potentials and contents) of the soil. The first step is to estimate the current matric flux potentials from the volumetric water contents *sim.WN*, the saturation water content *WS*, the air-entry matric flux potential, and the slope of the moisture release curve:

$$MFP_{layer} := soil.AirEntryMFP_{layer} * (sim.WN_{layer} - WS_{layer})^{soil.BValue_{layer}+3} \quad (2)$$

As in procedure `Richards_Equation`, this procedure then continues with series of sub-procedures, which are called iteratively until the mass balance is satisfied:

• `Hydraulic_Conductivities`: calculate unsaturated hydraulic conductivities for each layer;, from the current matric flux potentials
• `Soil_Evaporation`: compute actual evaporation from the top soil layer;
• `Jacobian`: set up simulataneous equations for determining new matric flux potentials from the current matric flux potentials, water contents and hydraulic conductivities;
• `Thomas_Algorithm`: solve this system of equations;
• `New_Water_Contents`: calculate the new water contents for each layer from the updated matric flux potentials

These are called in turn, as many times as necessary until the numerical solution of the modified Richards' equation converges, i.e., the mass-balance error is less than a pre-defined limit of 10^{-6} kg m^{-2} s^{-1}. Non-convergence is detected by counting the number of iterations, and halting the simulation if a pre-defined limit of 256 is exceeded.

Conductivities are calculated by:

$$sim.K_{layer} := soil.HydCond_{layer} * (MFP_{layer} / AirEntryMFP_{layer})^{B3_{layer}} \quad (3)$$

where the exponent *B3* is derived from the slope of the moisture release curve by:

$$B3_{layer} := (2 * soil.BValue_{layer} + 3) / (soil.BValue_{layer} + 3) \quad (4)$$

Soil evaporation is computed exactly as in the Richards Equation method. The mass balance equations are set up similarly, but the difference in matric flux potential approach are the gradients. The Thomas Algorithm is the same, but the values obtained by the back-substitution are the new matric flux potentials, not the water potentials. If any matric flux potential becomes too negative, it is set to a small positive potential (0.000001 J kg^{-1}). The the new water contents *sim.WN* are computed from the new matric flux potentials *MFP* and the constant saturated water content *WS* and the matric flux potential at saturation *AirEntryMFP* by:

$$sim.WN_{layer} := WS_{layer} * (MFP_{layer} / AirEntryMFP_{layer})^{(1/soil.BValue_{layer}+3)} \quad (5)$$

After the numerical solution converges, we can compute the fluxes in subprocedure `Calculate_Fluxes`, from the final matric flux potentials:

$$sim.WFlux[I] := ((sim.K[I] * sim.WP[I]) – (sim.K[I+1] * sim.WP[I+1]) / ((1–N) * NodeDelta[I])) + (GR * sim.K[I]) \quad (6)$$

5.5.4. Workability and Trafficability

source file: soillib.pas
procedures:: Is_Workable_Init, Is_Workable

Two important land qualities in agricultural operations are *workability*, i.e. whether the land can be tilled, and *trafficability*, i.e. whether agricultural machinery can drive onto the land. GAPS can simulate the number of hours per day when a given plant-soil-atmosphere environment is workable or trafficable.

GAPS simulates atmospheric and soil processes (e.g. precipitation, evaporation, water movement in the soil) on a less-than-daily time scale, selected by the user. This is typically an hour to four hours. At each time step, the moisture status of the soil, by depth, is known, either in terms of soil water potential (if the Richard's Equation or Matric Flux Potential is used to model water flow) or in terms of soil moisture

content (if the Tipping Bucket approach is used to model water flow). Also, in each time period GAPS simulates the solar elevation angle; from this it can estimate daylight and twilight hours.

At each time step, GAPS determines whether the soil is dry enough to a sufficient depth to permit field operations. If so, the length of the time step is added to a daily sum. These sums are accumulated and reported on a daily basis, in the GAPS '.sum' output file.

The determination of whether the soil can be worked or trafficked is based on two parameters: (1) the depth to which the soil must be dry enough, and (2) the critical soil water potential (Richards' equation, Matric Flux Potential) or soil moisture content (Tipping Bucket), below which the soil is considered dry enough. These parameters are supplied by the GAPS user, as entries in the 'location' ('.loc') input file.

The user must supply values for both tillage (e.g. plowing, disking) and traffic (e.g. sidedressing, harvesting). GAPS will apply the values for tillage when there is no active crop (i.e. before planting and after harvesting), under the assumption that the farmer's primary workability consideration in this period is tillage. When there is an active crop, GAPS will apply the values for traffic.

How does the model user determine values for the parameters?

Determination of the *depth to which the soil must be 'dry enough'* is fairly simple: it is the deeper of (1) the depth to which the soil will be worked by an implement (e.g. plow to 20cm), and (2) the depth to which the soil can be significantly compacted by the power source (tractors, draught animals etc.). The implement depth is set by the farmer, and of course is zero for traffic only (as opposed to tillage). The depth of compaction should ideally be determined by field observation with a instruments such as the penetrometer. This can be estimated from physical principles (see e.g. Hillel 1980), if the machine weight and design are known. . A reasonable approximation is to use the depth of the 'plow layer' or 'surface soil' as observed in the field, typically 15 to 20 cm, for tillage, and about 5cm for traffic only (n.b. this 5cm depth was used by Bouma & van Lanen in the studies cited below).

Determination of the *critical water potential* should ideally be done with a field study of infiltration rate or penetrometer resistance versus water potential (see e.g. Bouma & van Lanen 1987). Such studies reveal a critical, or threshold water potential, which corresponds to a penetrometer resistance of approximately 50 KPa. In studies reported by Bouma and van Lanen, this resistance corresponded to a pressure head of -90 cm water for a heavy clay, and to -40 cm for a sand. In GAPS, potentials are always expressed as MJ kg^{-1}; the conversion between the units is based on the acceleration due to gravity, 9.8m s^{-2}, so that to obtain potentials in MJ kg^{-1} for GAPS, one multiplies the pressure head expressed in cm by 0.098. For example, -90 cm of water is equivalent to -8.82 MJ kg^{-1}.

If the Tipping Bucket water flow model is used, the critical water potential must be converted to an equivalent *critical volumetric water content* (m^3 m^{-3}). Ideally this relation should be determined experimentally. A set of soil moisture vs. pressure head relations has been collected in the Netherlands (van Keulen & Wolf, 1986, p. 81), and an equation has been developed which allows prediction of water content (SM_ψ) from the suction head: (ψ):

$$SM_\psi = SM_o \, e^{-\gamma \, (\ln \psi)2} \qquad (1)$$

where ψ is expressed in cm of water (i.e. MJ kg^{-1} x 10.20), and SM_ψ is fractional water content (m^3 m^{-3}). This equation has two parameters: SM_o (m^3 m^{-3}) and γ (cm^{-2}), which are determined experimentally for a set of similar soils. The Dutch workers determined these values for 13 textural classes, more or less corresponding to the 12 USDA classes, as follows:

Texture class	SM_0	γ
coarse sand	0.395	0.1000
fine sand	0.364	0.0288
loamy sand	0.439	0.0330
fine sandy loam	0.504	0.0207
silt loam	0.509	0.0185
loam	0.503	0.0180
loess loam	0.455	0.0169
sandy clay loam	0.432	0.0096
silty clay loam	0.475	0.0105
clay loam	0.445	0.0058
light clay†	0.453	0.0085
silty clay	0.507	0.0065
heavy clay†	0.540	0.0042

† The dividing line between light and heavy clay is 65% clay.

This table should be used with caution in soils with dominance of shrink-swell clays, soils with high contents of sesquioxides, soils with high organic matter contents, slaking soils,.and soils with unusual packing density. P.M. Driessen (pp 217-221 in van Keulen & Wolf 1986) discusses some of the many difficulties in estimating SM_ψ - ψ relations in such soils.

In summary, **to determine a critical water content**:

(1) determine a critical water suction, in cm of water, based on the textural class and weight of machinery. If this is expressed in MJ kg^{-1} potential, multiply by –10.2 to obtain cm.

(2) Select parameters SM_0 and γ from Table 1, based on the textural class of the soil being modelled.

(3) Calculate according to equation (1).

For example, 90cm suction head of water in a clay loam corresponds to a moisture content of:

$$SM = 0.445\ e^{-0.0058\ (\ln 90)2} => 0.3957\ m^3\ m^{-3}$$

For a loamy sand, a more appropriate suction would be 40cm, and again applying equation (1) with parameters for this textural class we obtain a critical moisture content of:

$$SM = 0.439\ e^{-0.0330\ (\ln 40)2} => 0.2802\ \ m^3\ m^{-3}$$

Complications

If the Tipping Bucket water flow method is used to estimate soil water content, a problem may arise when estimating field hours if a plant is not actively growing. This is because in the Tipping Bucket method, water flow upward is not modelled. The only ways in which water can leave a profile are (1) drainage, (2) plant water uptake, and (3) near-surface evaporation. If a plant is growing, this is usually a good approximation to reality, as the plant pumps water from the profile at a much faster rate than could be achieved by evaporation from the surface and upward movement from lower layers. However, if the plant is not growing (e.g. in early spring), only evaporation can sufficiently dry the upper soil to allow workability, assuming that the critical limit for workability is higher than the drained upper limit. The complication is that the GAPS tipping bucket routine only evaporates in the top few cm, usually less than the depth of tillage (see the description of the Tipping Bucket method for a description of how to

determine the depth to which evaporation is effective). Thus, the layers between the depth of evaporation and tillage can't dry out. The solution is to limit the depth to which the soil must be 'dry enough' to the depth to which evaporation is effective. Thus, GAPS automatically adjusts the depth to which the soil must be workable to the evaporation depth if the Tipping Bucket water flow method has been selected.

GAPS vs. reality

When calibrating GAPS-predicted field hours against observed field hours (e.g. experiment station records), it is necessary to determine what kind of field operations could be performed, and set up the parameters (depth, moisture status) to correspond with this. For example, tillage hours are usually less than traffic hours for such purposes as sidedressing, with hours for planting in between. It may be necessary to simulate field hours for several classes of field operations, and then combine them according to the mix of operations in a time period.

5.7. Runoff

Some of the precipitation that reaches the soil surface does not infiltrate into the soil, but instead runs off the site. Without taking runoff into account, the simulated soil may be wetter than the actual soil. GAPS provides only one way to estimate runoff.

5.6.1. Curve-number [EPIC] runoff

```
source file: rainlib.pas
procedures:: Runoff_CN_Init, Runoff_CN_Daily,
             Runoff_CN_Done
access functions: Runoff_CN_GetThreshold
```

This method of estimating runoff closely follows the method which forms part of EPIC (Williams *et al*. 1989). It operates on a daily time step, using empirical formulas to estimate runoff from daily precipitation, the current water content of the soil, the site slope, and the runoff curve number developed by the USDA Soil Conservation Service (SCS 1972). The advantage of this method is that it uses readily-available data: daily rainfall and the curve number. This latter summarizes the runoff characteristics of the site with respect to soil type, land use, and management. Curve numbers range from 0 (no risk of runoff) to 100 (extreme risk of runoff). Runoff curves are easily obtained for all USA soils from the EPIC data set or SCS hydrology tables, which take into account the soil's hydrologic group (based on infiltration rate), hydrologic condition (e.g. surface sealing), land use, and management practices. For example, fallow on soils in group D results in a curve number of 94; close-seeded legumes on contoured and terraced soils in group A in good hydrologic condition results in a curve number of 51.

The basic idea of the curve number method is to estimate runoff from the daily rainfall and a *retention parameter*, which is adjusted according to the soil water content. The equation used to calculate runoff is:

$$\text{Runoff} := (\text{Precip} - 0.2*s)^2 / (\text{Precip} + 0.8s) \quad \text{if Precip} > 0.2*s$$
$$\text{Runoff} := 0 \quad \text{if Precip} \leq 0.2*s \quad (1)$$

where *Runoff* is the daily runoff, *Precip* is the daily precipitation, and *s* is the retention parameter, all expressed in kg m^{-2} (i.e. mm of water).

The value (0.2 * s) is a *threshold*, so that if the daily rainfall is sufficiently low, no runoff occurs. Above this value, this equation can be viewed either as a two-variable function, or a family of parameterized one-valued functions in either rainfall or retentivity.

The retention parameter *s*, in turn, is a daily adjustment of the curve number, based on the current water status of the soil:

$$s := s_1 * (1 - (FFC / (FFC + e^{(w_1 - w_2 * FFC)})) \quad (2)$$

where *FFC* is the day's fraction of field capacity, s_1 is a static amplitude parameter, and w_1, and w_2 are static shape parameters, computed at the beginning of the simulation (see below).

The daily fraction of field capacity is computed as a weighted average of the fractions of field capacity of each layer FFC_{layer} down to 1 meter or the bottom of the profile, whichever is shallower:

$$FFC := \frac{\sum_{layer=2}^{LastLayer} FFC_{layer} * \frac{LayThick_{layer}}{MidPoint_{layer}}}{\sum_{layer=2}^{LastLayer} \frac{LayThick_{layer}}{MidPoint_{layer}}} \quad (3)$$

where *LayThick* is the thickness of a layer and *MidPoint* is its midpoint, both expressed in m and computed in the obvious way from the soil layer lower boundaries *soil.LowBound[layer]*. In this expression, the denominator normalizes the result, and the multipliers of FFC in the numerator give more weight to thicker layers and those nearer the surface. The idea is to determine a composite single value representing the water status of the soil: the closer to field capacity, the more water will run off.

The fraction of field capacity of each layer, FFC_{layer} , is computed as:

$$FFC_{layer} := (sim.WN_{layer} - soil.DLL_{layer}) / (soil.DUL_{layer} - soil.DLL_{layer})$$

where *sim.WN* is the current volumetric water content, and *soil.DLL* and *soil.DUL* are the lower and upper limits of available water, as found in the soil input file. Thus if the water content is at the upper limit, FFC will be 1; if at the lower limit, FFC will be 0; if there is freely-draining water, FFC will be greater than 1. From equation (2), we see that if FFC is 0, i.e. the soil is essentially dry, the retention parameter *s* is equal to the parameter s_1, which represents the maximum retention and thus the least runoff. As FFC increases, the retention parameter decreases, and runoff increases.

Referring back to equation (2), we now discuss how the static parameters are derived. To do this, we first consider how the single curve number supplied as input, CN_2, is adjusted by a series of empirical equations for varying conditions. CN_2 is meant to represent average moisture conditions, i.e. moist soil. First, a curve number for wet soil conditions, CN_3, is derived from CN_2 by the following empirical equation:

$$CN_3 := CN_2 * e^{(0.00673\ (100-CN_2)} \quad (4)$$

At high curve numbers (i.e. high risk of runoff), the moist and wet curve numbers are essentially the same; at low curve numbers, the wet curve number diverges considerably from the moist curve; e.g. at $CN_2 = 50$, CN_3 is 70. At saturated conditions, the risk of runoff is much higher at such sites.

The curve number for moist conditions is then adjusted for site slope according to the empirical equation:

$$CN_{2S} := CN_2 + (1/3 * (CN_3 - CN_2)[\ 1 - 2 * e^{(-13.86 * soil.Slope)}\ ]) \quad (5)$$

where *soil.Slope* is the site slope, m m^{-1}. The effect of this equation is to increase the curve number exponentially (with a low coefficient) as slope increases. The factor ($CN_3 - CN_2$), along with equation (4), ensures that CN_{2S} does not increase past 100. Note that even at zero slope, there is some adjustment, in fact, CN_2 is decreased if CN_2 was not the maximum of 100. This is because CN_2 is considered to represent CN at sites with a 0.05 m m^{-1} slope.

Using the adjusted curve number for moist conditions, a curve number CN_1 for dry conditions is derived using the empirical equation:

$$CN_1 := CN_{2S} - ([20 * (100 - CN_{2S})] / [(100 - CN_{2S}) + e^{(0.0636 * (100-CN_{2S})}]) \quad (6)$$

I won't try to explain this equation. If the resulting value of CN_1 is less than 0.4th of CN_{2S}, it is set to this value. This sets a floor on CN_1, ensuring that there is still a reasonable chance of runoff even in dry conditions.

With $CN_{1,2,3}$ in hand, we can proceed to compute the shape parameters w_1 and w_2. These are obtained from a simulataneous solution of equation (2), with three known points: at FFC = 1.0, we want the retention parameter s to be its value for wet conditions, i.e. s_3 , at FFC = 0.5, we want s to be its value for moist conditions, i.e. s_2 , and at FFC = 0.0, s must be its value for dry conditions, i.e. s_1. These three values of s_x are computed directly from the corresponding curve number CN_x , using the basic relation between curve number and retention parameter:

$$s_x := 254 * ((100 / CN_x) - 1) \quad (7)$$

where the factor 254 converts from the [0..1] scale of curve numbers to kg m^{-2} (i.e. mm of water).

Once the three values $s_{1,2,3}$ have been computed, the shape parameters w_1 and w_2 are computed from the solution of the simultaneous equations:

$$w_2 := 2 * (-0.5 + \ln[0.5 / (1 - (s_2 / s_1))] - \ln[(1 / (1 - (s_3 / s_1))) - 1]) \quad (8a)$$
$$w_1 := w_2 + \ln ((1 / (1 - (s_3 / s_1)) - 1) \quad (8b)$$

There is one further adjustment to the daily retention parameter that is made if the soil temperature in the second layer of the profile is below freezing. The following empirical equation is used to decrease retention:

$$s := s * [1 - e^{(-0.00292 * s)}] \qquad \text{if sim.SoilTemp[3]} \le 0°C \quad (9)$$

This increases runoff for frozen soils at high values of *s*, i.e. under wet conditions, but has little effect on runoff for dry soils. This adjustment is still made even if soil temperature is not being simulated, in which case the initial value of soil temperature *soil.InitSoilTemp[3]* (default 20°C) is used throughout the simulation.

5.8. Plant water uptake (soil-plant interface)

GAPS provides three ways to simulate plant water uptake: 'potential-driven' and 'plant available' and the Epic-model. The first is intended to be matched with the Richard's Equation or Matric Flux Potential soil water flow models, since these consider water from the point of view of its potential energy in J kg^{-1} of water. The second is intended to be matched with the Tipping Bucket soil water flow model, since these

consider water from the point of view of its volumetric content in soil layers. The third matches the model described by Williams J.R., Jones C.A. and Dyke P.T (1989) as part of EPIC.

An interesting point is that these procedures do not actually remove water from the soil profile, i.e. the water content *sim.WN[]* is not affected. That step occurs during the water flow procedures, using the per-layer water uptake *sim.WUptake[]* calculated here.

5.7.1. Potential-driven water uptake

```
source file: soillib.pas
procedures:: WaterUptake_Init, WaterUptake,
             WaterUptake_Done
```

In this method, the amount of water flowing from soil to roots is considered to be directly dependent on how much more negative the root water potential *RootWP* is compared to the soil water potential *sim.WP[]* in a layer (Gardner, 1960). Water flow into roots has also been considered inversely dependent on the resistance to water flow in the soil and in the root (Gardner and Ehlig, 1962). If we combine these two concepts with the assumption that there is steady state water flow through the plant (i.e. there is no change in the amount of water stored in the plant), then we obtain the following way of calculating *sim.ActTrans*, the actual transpiration, i.e. the water uptake by the plant (under steady-state in the plant), in kg m^{-2} s^{-1}:

$$\text{sim.ActTrans} := \sum_{i=\text{plant.FRoot}}^{\text{sim.LRoot}} \frac{(\text{sim.RootWP}_i - \text{sim.WP}_i) + (\text{NodeDepth}_i * GR)}{\text{Res}_i} \qquad (1)$$

where *plant.FRoot* and *sim.LRoot* are the first and last soil layers containing roots, *sim.RootWP[I]* is the root water potential at a node (J kg^{-1}), *sim.WP[I]* is the soil water matric potential at the same node (J kg^{-1}), *NodeDepth[I]* is the depth of node *I* from the soil surface (m), *GR* is the acceleration of gravity downwards (9.8 m s^{-2}), and *Res[I]* is the root and soil resistance of layer *I* , which is considered to surround node *I* (kg s m^{-4})

When we begin this procedure at each time step, soil water potentials with depth *WP[I]* and the potential transpiration rate *sim.PotTrans* are known. If we can compute the resistances to water flow with depth, we can then calculate a single root water potential that will result in the sum of water flowing from soil to roots at each soil depth exactly equal to the potential transpiration. Consequently, this procedure is broken down into two main parts, which are called in sequence:

- `RootWaterPotential`: determine the root water potential;
- `RootWaterUptake`: determine the water taken up from each layer, knowing the root water potential.

However, this whole process is bypassed if, on entry to the procedure, it is determined that transpiration can not be occurring, i.e. *sim.PotTrans <= 0*. In this case, the actual transpiration *sim.ActTrans* and the uptake from each layer, *sim.WUptake[]*, are set to zero.

This module makes extensive use of dynamic variables, in order to save memory in those cases when this method of water uptake is not selected. The dynamic variables are allocated on the heap in procedure `WaterUptake_Init`, which is called before simulation begins, and deallocated in procedure `WaterUptake_Done`, which is called at the end of the simulation.

We now discuss the two sub-procedures.

5.7.1.1. RootWaterPotential

This procedure determines the current root water potential. Contained within it is the procedure `Resistances` which calculates the root resistances in each layer *Res[I]*.

The root xylem resistance is assumed to be negligible and therefore the root water potential is the same at all soil depths. Rearranging equation (1) to solve for *sim.RootWP* gives (Childs, 1977; Riha, 1984):

$$\text{RootWP} = \frac{\sum_{i=\text{plant.FRoot}}^{\text{sim.LRoot}} \left(\text{sim.PotTrans} + \frac{\text{sim.WP}_i - (\text{NodeDepth}_i * \text{GR})}{\text{Res}_i} \right)}{\sum_{i=\text{plant.FRoot}}^{\text{sim.LRoot}} \frac{1}{\text{Res}_i}} \qquad (2)$$

In this program we compute parts of this equation separately. In particular, *SSumRes* is the second term of the numerator of (2), and *PSumRes* is the denominatior of (2)

The most complex part of the entire water uptake procedure is the calculation of the resistances at each layer, *Res[I]*. The equation used is ultimately based on that of Gardner (1960):

$$q/A = -k * d\,WP/d\,r \qquad (3)$$

where q is water flux (kg s^{-1}), A is the cross sectional area for flow per unit length of root (m^2), k is hydraulic conductivity (kg s m^{-3}), r is the radial distance for flow (m), and WP is soil water potential of soil or root (J kg^{-1})

The cross sectional area for flow per unit length of root A is represented by:

$$A = 2\pi * \text{plant.RootRad} \qquad (4)$$

where *plant.RootRad* is the radius of the root (m). This is a model input.

The unsaturated hydraulic conductivity *sim.K* for each soil profile layer is calculated according to Campbell (1974):

$$\text{sim.K} := \text{soil.HydCond} * (\text{soil.AirEntryPot} / \text{sim.WP})^N \qquad (5)$$

where *soil.HydCond* is the saturated hydraulic conductivity (kg s m^{-3}), *soil.AirEntryPot* is the air entry potential (J kg^{-1}), *sim.WP* is the soil water potential (J kg^{-1}), and N is computed as:

$$N := 2 + (3 / \text{soil.BValue}) \qquad (6)$$

where *soil.BValue* is the slope of the water release curve (moisture content vs. water potential), when plotted on log-log scale.

Substituting equations (4) and (5) into (3) gives:

$$q / 2\pi * \text{plant.RootRad} = -\text{soil.HydCond} * \text{soil.AirEntryPot} / \text{sim.WP})^N * d\,\text{sim.WP} / d\,r) \qquad (7)$$

Separating variables and integrating this equations results in:

$$q/2\pi \ln (Res4 / plant.RootRad) = -soil.HydCond * soil.AirEntryPot^{N} * (1 / (1-N)) * (sim.WP^{1-N} - RootWP^{1-N}) \quad (8)$$

where *Res4* is the distance from the center of the root to the point where sim.WP is measured, which is calculated according to Gardner, 1960 by:

$$Res4 = \pi * plant.RootDens^{-0.5} \quad (9)$$

where *plant.RootDens* is the length of root per unit volume of soil (m m^{-3}).

The resistance form of equation (8) is:

$$q * plant.RootDens = (sim.WP - RootWP) / Res \quad (8')$$

where the combined resistance *Res* is:

$$Res = \frac{(1-N) \ln\left(\frac{Res4}{plant.RootRad}\right)}{2\pi\, plant.RootDens * soil.HydCond * soil.AirEntryPotential^{N}} * \frac{sim.WP - RootWP}{sim.WP^{1-N} - RootWP^{1-N}} + plant.RootRes \quad (10)$$

Substituting eq. (9) into eq. (10) gives:

$$Res = \frac{(1-N) \ln\left(\frac{1}{RootRad} * \sqrt{\pi\, RootDens}\right) * (sim.WP - RootWP)}{-2\pi\, RootDens * HydCond * AirEntryPotential^{N} * (sim.WP^{1-N} - RootWP^{1-N})} \quad (11)$$

The calculation of *Res* is straightforward for the soil resistance component of equation (11), when *RootDens*, *RootRad*, *B*, *sim.WP*, *HydCond*, and *AirEntryPotential* are specified with depth.

In subprocedure `Resistances`, the calculation of *Res1* through *Res4* solves the soil resistance component of eq. (11) as a function of depth. Root resistance with depth knowing the total root resistance *Res* (kg s m^{-4}) is calculated as follows (Campbell, 1986):

$$RootRes[I] = RootRes / (RootDens[I] * LayThick[I]) \quad (12)$$

where LayThick[I] is the vertical thickness of the layer (m).

5.7.1.2. RootWaterUptake

Knowing the root water potential, *RES[I*] and *sim.WP[I]* the root water uptake in each layer *sim.WUptake[]* (kg m^{-2} s^{-1}) is calculated from the equation:

sim.WUptake[I] = –((RootWP–sim.WP[I])+ soil.NodeDepth[I]*GR) / Res[I] (1)

This equation is applied to each layer with a positive root-soil resistance and a root water potential greater than –1500 J kg^{-1}. The actual transpiration *sim.ActTrans* for the time step (kg m^{-2} s^{-1}) is set equal to the sum of the uptakes from each layer.

5.7.2. Plant-available water uptake

```
source file: soillib.pas
procedures : SimpleWaterUptake_Daily, SimpleWaterUptake
```

Plant water uptake from the layers in the soil containing roots (i.e. *plant.FRoot* to *sim.LRoot*) is allowed to proceed until a lower limit of plant-extractable water (permanent wilting point, *soil.DLL[]*) is reached. The root density distribution in the soil profile is used to partition the transpirational demand between soil layers. Demand not met in any one layer is transferred to other layers as an additional demand. Root densities thus do not limit root water uptake in this simple representation, but serve solely to partition transpiration in the soil profile.

This method accounts for multiple crops. The first-listed active crop takes up all the water it requests from a layer, and so on down the profile, lowering the available water content accordingly. Then the next-listed active crop does the same, and so on. This is the method used in the ALMANAC competition model (Kiniry *et al.* 1992). It assumes that there is no competition *per se* for water, just that all crops extract what they can from the available-water in each layer with roots. The onset of drought stress may be delayed by one time step for the first-listed crop and advanced one time step for the other crops, but this is a minor effect. If there is only one crop, this method is equivalent to the method of GAPS V3 and earlier.

Procedure `SimpleWaterUptake_Daily` is called at the beginning of each day to calculate the total surface area of roots in the profile for each crop, *TheCrop[crop_i]^.TotalRoots* (m^2 m^{-3}), from the root length density *TheCrop[crop_i]^.crop.RootDens[]* (m m^{-3}) and the thickness *LayThick[]* (m) within the rooting zone. This can be done once a day, because the rooting depth only can change daily:

$$TotalRoots_{crop\_i} = \sum_{layer=plant.FRoot_{crop\_i}}^{LRoot_{crop\_i}} RootsDens_{crop\_i,layer} \cdot LayThick_{layer} \quad (1)$$

Procedure `SimpleWaterUptake` is called at each time step to compute the amount of rate of water uptake by the plant from each layer, *sim.WUptake[]* (kg m^{-2} s^{-1}) and the rate of actual transpiration *TheCrop[crop_i]^.ActTrans* for each crop(kg m^{-2} s^{-1}), which is the sum of the uptake rates from the layers, as well as the overall actual transpiration *sim.ActTrans*, which is the sum of the per-crop transpiration

This procedure works downwards from the top layer. First, the plant-available water *LayerAvail[]* (kg m^{-2}) in the layer is computed as the difference between the current water content *sim.WN[]* (m^3 m^{-3}) and the wilting point *soil.DLL[]* (m^3 m^{-3}), converted to a mass:

LayerAvail[i] := (sim.WN[i]–soil.DLL[i]) * KgFines[i] (2)

If the current water content is below the wilting point (e.g. in near-surface layers where evaporation has been effective), this estimate may be negative, in which case it is set to zero.

The following steps are performed for each active crop, in order of their listing in the cropping sequence file (not the order in which they were planted).

First, the procedure obtains and stores the crop's potential transpiration from the competition model, using an access method:

```
TheCrop[crop_i]^.PotTrans := TheCompModel^.GetPotTrans(crop_i)
```

Next, we compute the proportion of the total profile root area in the current layer as:

```
r_frac := (LayThick[layer]*TheCrop[crop_i]^.crop.RootDens[layer]) / TheCrop[crop_i]^.TotalRoots      (3)
```

where *RootDens[]* is the root length density (m m^{-3}) for this crop in this layer, *LayThick[]* is the layer thickness (m) which converts this to a surface area, and *TotalRoots* is the total profile root surface area (m^2 m^{-3}) for this crop, computed each day by (1). Assuming that uptake is proportional to root area, we estimate the layer's uptake for this crop *WUptake* from its root area fraction and the rate of transpirational demand *sim.PotTrans* (kg m^{-2} s^{-1}) as:

```
WUptake := (r_frac * TheCrop[crop_i]^.PotTrans)
           + (AdditionalDemand / time_step)                                    (4)
```

where the mass *AdditionalDemand*(kg m^{-2}) is unsatisfied demand from the overlying layers, as now explained. In the top soil layer, this is initialized to zero. Thereafter, if the estimate of water uptake from (4) exceeds the available water in the layer:

```
if (WUptake * time_step) > LayerAvail[i] then...                               (5)
```

In this case, the water uptake is limited to the plant-available water, and the portion of the transpiration demand that could not be satisfied by this layer is passed to the next layer down as the *AdditionalDemand* (kg m^{-2}):

```
WUptake := LayerAvail[i] / time_step
AdditionalDemand := sim.WUptake * time_step) – LayerAvail[i]                   (6)
```

The crop's active transpiration is accumulated, as well as the total uptake in the layer:

```
TheCrop[crop_i]^.ActTrans := TheCrop[crop_i]^.ActTrans + WUptake
sim.WUptake[layer] := sim.WUptake[layer] + WUptake
```

Finally, the available water in a layer is reduced by the amount just taken up:

```
LayerAvail[layer] := LayerAvail[layer] - (WUptake * time_step)
```

leaving less water for the next plant.

Note that after following this algorithm for the entire rooting profile, there may be some unsatisfied demand. This will be the difference between actual and potential transpiration for the crop.

The entire process is repeated for subsequent active crops.

5.7.3. EPIC model water uptake

```
source file: soillib.pas
procedures:: EPICWaterUptake
```

This procedure is based on the wateruse model as outlined in the EPIC-Erosion / Productivity Impact Calculator model documentation (Sharpley and Williams, 1990). It is similar to Plant-available water uptake, with two major differences:

• The way in which demand is apportioned to the different soil layers

• EPIC does not translate unsatisfied demand downwards

Instead of the explicit root density distribution used in the Plant-available water uptake method, the EPIC method uses an negative exponential distribution of roots with depth, described by one parameter, *soil.WUD*, which is an input parameter from the soil file. Demand is approportioned to depth in a curve that ranges from linear (for WUD approaching 0 from above) to increasingly curved (for increasing values of WUD).

The fraction of the total demand apportioned to the layers of soil from the surface down to the lower boundary low_bound_{layer} is:

$$(1 - (1 - e^{-WUD})) \times (1 - e^{-WUD \times (low\_bound_{layer} / low\_bound_{roots})})$$

At the surface, this function is 0, and at the lower boundary of the rooting zone it is 1. *WUD* controls the curvature: higher values of *WUD* imply that more of the demand is apportioned to near-surface layers. In this EPIC model, unmet demand is *not* transferred downwards, so that a high *WUD* models the situation where dry surface soils can not be compensated for by moist subsoils.

Here is a MathCAD worksheet showing the curve for some typical values of WUD:

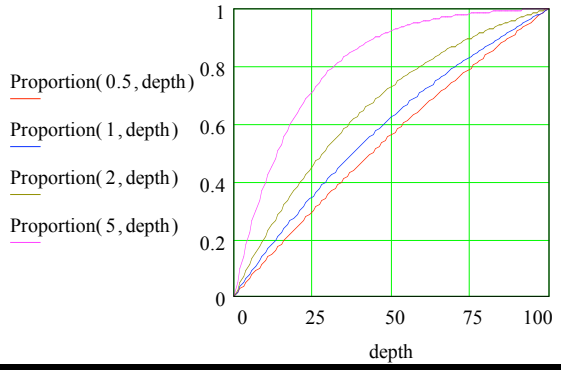
EPIC water uptake parameter WUD

rooting_depth := 100 Depth of the deepest roots

depth := 0.. rooting_depth Depths at which to evaluate the function

Cumulative proportion of total demand satisfied at a given depth

$$\text{Proportion}(\text{WUD}, \text{depth}) := \left\{\frac{1}{1 - e^{-\text{WUD}}}\right\} \cdot \left\{1 - e^{-\text{WUD} \cdot \frac{\text{depth}}{\text{rooting\_depth}}}\right\}$$



Cumulative proportion as a function of depth, for several values of the parameter WUD

This method accounts for multiple active crops in the same way as the Plant-Available Water Uptake method (see above).

The procedure works downwards from the top layer. First, the plant-available water *LayerAvail[]* (kg m^{-2}) in the layers in the soil from the surface to the last layer containing any roots (i.e. *MinRoot* to *sim.LRoot*) is computed as the difference between the current water content *sim.WN[]* (m^3 m^{-3}) and the wilting point *soil.DLL[]* (m^3 m^{-3}), converted to a mass:

LayerAvail[i] := (sim.WN[i]–soil.DLL[i]) * KgFines[i] (1)

If the current water content is below the wilting point (e.g. in near-surface layers where evaporation has been effective), this estimate may be negative, in which case it is set to zero.

The following steps are performed for each active crop, in order of their listing in the cropping sequence file (not the order in which they were planted).

First, the procedure obtains and stores the crop's potential transpiration from the competition model, using an access method:

TheCrop[crop_i]^.PotTrans := TheCompModel^.GetPotTrans(crop_i)

Next, the potential water uptake in the current layer is computed as:

WUptake = (TheCrop[crop_i]^.PotTrans / (1- e (-soil.WUD)))
* (1- e (-soil.WUD * soil.LowBound[i] /soil.LowBound[TheCrop[crop_i]^.LRoot]))
- aboveUptake; (2)

where *TheCrop[crop_i]^.PotTrans* is the rate of transpirational demand (kg m^{-2} s^{-1}) for the crop, *soil.WUD* is the water use distribution parameter, *soil.LowBound[i]* the depth for the current layer and *soil.LowBound[TheCrop[crop_i]^.LRoot]* the root zone depth for the crop. The *aboveUptake* is the water use rate for all the layers above the current layer (results from previous calculations as the procedure works downwards).

If the soil water storage is less than 25 % of the plant-available soil water:

if (soil.DUL[i] - soil.DLL[i]) /4 + soil.DLL[i] > sim.WN[i] **then**...

then the potential water uptake calculated for the current layer is reduced with the factor:

exp(20 * (sim.WN[i]-soil.DLL[i]) / (soil.DUL[i] - soil.DLL[i]) - 5)

where *sim.WN[i]* is the soil water content in layer i.

Thereafter, if the estimate of water uptake exceeds the available water in the layer:

if (WUptake * time_step) > LayerAvail[i] **then**...

the water uptake is limited to the plant-available water:

WUptake := LayerAvail[i] / time_step

The crop's active transpiration is accumulated, as well as the total uptake in the layer:

TheCrop[crop_i]^.ActTrans := TheCrop[crop_i]^.ActTrans + WUptake
sim.WUptake[layer] := sim.WUptake[layer] + WUptake

The procedure keeps track of the amount taken up so far, i.e., 'above' the next layer to be considered:

aboveUptake := aboveUptake + WUptake

And, the water available in this layer is reduced, so that there will be less for the next plant.

LayerAvail[layer] := LayerAvail[layer] - (WUptake * time_step)

Note that after following this algorithm for the entire rooting profile, there may be some unsatisfied demand. This will be the difference between actual and potential transpiration for the crop.

The entire process is repeated for subsequent active crops.

5.9. Water budget

This procedure is called by the simulation driver at the end of each time step, regardless of the soil water flow and plant water uptake models selected by the user, to update several variables reported by GAPS in the simulation summary ('*.sum) output file, using the time step's values of corresponding variables just computed.

One set of variables relates to the accumulated water budget, in kg m^{-2}:

SumPotTrans, SumPotEva : total potential transpiration and evaporation;
SumActTrans, SumEV : total actual transpiration and evaporation;

SumWIn : total water added to the profile (e.g. rainfall)
SumDrain : total drainage from the bottom of the profile;
SumWCChange : cumulative change in profile water

The current total profile water *ProfileWater* and the amount of this which is plant-available, *ProfileAvail* (both in $m^3\ m^{-3}$) are also computed, by summing the water contents in all layers; in the case of *ProfileAvail*, only the water above the lower limit is considered.

Despite the procedure name, it also computes two summary variables related to heat, both expressed in W m^{-2}:

NetHeatFlux : accumulated net heat flux
SumNetRad : accumulated net solar radiation

Finally, if the model user requested a simulation of field hours, and it is daytime, and the soil is workable (as determined by a call to the boolean function `Is_Workable`) the field hours for the day are incremented by the length of the time step.

5.10. Soil temperature

GAPS provides two ways to simulate the soil temperature by depth. The first is based on a numerical solution to differential equations representing the heat fluxes in response to thermal conductivities, very much analogous to the Richards Equation method of representing water flow. The second is a simple harmonic wave by depth, assuming that the soil is a uniform body with respect to thermal conductivity and water content. If neither of these methods is selected, the soil temperature remains at the initial value specified in the soil input file (or, if that is not specified, the default of 20°C) throughout the simulation.

5.9.1. Heat flow equations

```
source file: soillib.pas
procedures:: Soil_Temperature_Init, Soil_Temperature,
             Soil_Temperature_Done
```

The change in soil temperature over time depends on the ability of the soil to conduct heat, i.e. its thermal conductivity, and on the heat capacity of the soil. The thermal conductivity of the soil is to a large degree a function of its water content and bulk density.

The procedure `SoilTemperature_Init` is called at the beginning of the simulation to compute four empirical coefficients *Coeff1[] .. Coeff4[]* which are used to estimate the thermal properties of the soil in the numerical solution of heat flow equations. These coefficients are computed separately for each layer. *Coeff4* is the thermal conductivity when the soil is oven-dry (i.e. *soil.WN* = 0) and can be approximated from the moist bulk density *soil.BulkDensity* (Mg m^{-3}) following (Campbell, 1985):

$$\text{Coeff4[I]} := 0.3 + 0.1 * \text{soil.BulkDensity[I]}^2 \quad (1)$$

Coeff2 in part determines the differences in the thermal conductivity of saturated soils and is dependent on the total volume fraction of soils (Campbell, 1985):

$$\text{Coeff2[I]} := 1.06 * \text{soil.BulkDensity[I]} \quad (2)$$

where 1.06 includes a correction for particle density, assuming a particle density of 2.65 Mg m^{-3}. *Coeff1* also in part determines the differences in thermal conductivity of saturated soils and is based on work by

DeVries (1963). If the volume fraction of quartz is assumed to be zero and the value for particle density is assumed as above, the computation is:

$$Coeff1[I] := 0.65 - (0.78 * soil.BulkDensity[I]) + (0.6 * soil.BulkDensity[I]^2) \quad (3)$$

Coeff3 determines the water content at which thermal conductivity rapidly increases. This appears to be highly dependent on the % clay content of the soil, *soil.Clay[]*:

$$Coeff3[I]:= 1.0 + (2.6 / \sqrt{(soil.Clay[I] / 100)}) \quad (4)$$

The procedure `SoilTemperature` is called at each time step to compute the heat fluxes and the new soil temperatures by layer. This procedure is broken down into five subprocedures, which are called in order:

- `Boundary_Conditions` : establish temperatures at the upper and lower soil layers
- `Capacitance_Conductance` : determine the thermal capacity and conductances
- `Jacobian` : set up simultaneous solution to the heat-flow equations
- `Thomas_Algorthim` : solve this system of equations
- `Calculate_Fluxes` : determine the fluxes and new temperatures

These are now discussed in sequence.

The subprocedure `Boundary_Conditions` specifies the upper and lower boundary conditions used in the solution to the heat flow equations. Heat transfer from the surface of the soil to the air is assumed to be directly dependent on the difference between air temperature and surface soil temperature times a surface boundary conductance *loca.BLC* (W m^{-2} k^{-1}) provided as parameter in the 'location' input file. Thus the thermal conductance *K* (W m^2 k^{-1}) across the surface is set to this conductance, and the air temperature *AirTemp* (°C) is the temperature of the upper boundary:

$$K[MinLayer–1] := loca.BLC$$
$$sim.SoilTemp[MinLayer–1] := AirTemp \quad (1)$$

The temperature below the profile is kept constant at the initial soil temperature of the last layer, i.e. the soil temperature at this depth is considered to be constant, which is a reasonable estimate in soils deeper than about 1 meter:

$$sim.SoilTemp[soil.LastLayer+1] := soil.InitSoilTemp[soil.LastLayer] \quad (2)$$

The subprocedure `Capacitance_Conductance` uses the empirical coefficients calculated in the procedure `SoilTemperature_Init`, and the current water status of the soil, to determine the thermal conductance and heat capacitance for each layer. The heat capacitance *CP[]* (MJ m^{-2} k^{-1} s^{-1}) is calculated multiplying the volumetric specific heats of soil *SpecHeatSoil* (2.4 MJ m^{-3} K^{-1}) and water *SpecHeatWater* (4.18 MJ m^{-3} K^{-1}) by the fraction of the soil volume currently occupied by the mineral soil and soil water water, respectively, and normalizing by the layer thickness *LayThick[]* (m) and time step:

$$CP[I] := ((SpecHeatSoil * soil.BulkDensity[I] / soil.PartDensity[I]) + (SpecHeatWater * sim.WN[I])) * (LayThick[I] / time\_step) \quad (3)$$

where the term *soil.BulkDensity[I] / soil.PartDensity[I]* is the volume fraction of the soil taken up by mineral soil, and *sim.WN[]* is the current volumetric water content.

The equation used to estimate the soil thermal conductance *K[]* (W m^{-2} k^{-1}) from the empirical coefficients and water content is taken from McInnes (1981):

```
K[I] := (Coeff1[I]
        + (Coeff2[I] *sim.WN[I])
        – (Coeff1[I]–Coeff4[I]) * exp(–(Coeff3[I] * sim.WN[I])^4) )
        / NodeDelta[I])                                              (4)
```

where *NodeDelta[]* (m) is the distance from this node to the one above it. Dividing by this factor converts the conductivity into a conductance.

Heat fluxes are calculated by applying the equation of continuity to the Fourier law:

$$CP * d(SoilTemp)/dt = d (K d(SoilTemp)/dz) / dz \quad (5)$$

where *z* is the node depth.

The numerical solution to this equation is similar to the solution of the water flow problem by the Richards Equation. First a tridiagonal matrix representing the heat balance is formed in the procedure `Jacobian` and then solved by elimination and back-substitution. Since the conductances and capacitances are not a function of the driving force as is the case in the water flow problem (where the hydraulic conductivity is a function of the soil water potential and content), no iterations are needed to find the correct solution for the new soil temperatures. After a new set of soil temperatures is calculated, procedure `Calc_Fluxes` calculates the heat flux *sim.HeatFlux* (W m^{-2} s^{-1}) across the soil surface (positive downwards) from the thermal conductance of the surface node *K[MinLayer-1]* (W m^{-2} k^{-1}), and the gradient between the air and first soil layer at the end of the previous time step *(sim.SoilTemp[MinLayer-1]–sim.SoilTemp[MinLayer])* and at the end of this time step (*TN[MinLayer-1]–TN[MinLayer]*) , as:

```
sim.HeatFlux := K[MinLayer-1]
        * ( (1-loca.F) * (sim.SoilTemp[MinLayer-1]–sim.SoilTemp[MinLayer])
        +   (loca.F * (TN[1]-TN[2])) )                               (6)
```

where *loca.F* is the weighing factor for the finite difference solution, which is set as an input parameter in the 'location' file.

Finally, the soil temperatures for the next time step, *sim.SoilTemp[]*, are set equal to those at the end of this time step, *TN[]*.

5.9.2. Simple harmonic soil temperature

```
source file: soillib.pas
procedures:: Harmonic_Soil_Temp_Init,
             Harmonic_Soil_Temp_Daily, Harmonic_Soil_Temp
```

This method computes the soil temperature at the midpoint of each layer as a simple harmonic function of air temperature and depth. It is adapted from the discussion of this problem on pp.223-229 of Monteith & Unsworth (1990), which is similar to the approach found in Campbell (1977), pp. 14-19. The soil is modelled as a uniform 3-dimensional body with respect to its thermal conductivity, and the heat source is modelled as a diurnal sine wave of fixed amplitude and mean. This is only true for soils with uniform thermal conductances by depth and laterally, and uniform water content, and for uniform daily maximum and minimum temperatures. Obviously these conditions are very rarely met. However, the model provides a simple visualization of heat flow under these ideal conditions.

Procedure `Harmonic_Soil_Temp_Init` is called at the beginning of simulation to compute the time-invariant parts of the method. These are *Omega*, the frequency of the daily sine wave, which is $\Omega = 2\pi$ / 86 400 Hz, and the layer of the soil profile which will be considered to represent the water status of the entire profile for diffusivity calculations. Since most of the water and heat flux is near the surface, we choose a depth of 10cm, which is just below that where evaporation is typically effective, and determine which layer of the profile contains this depth; this layer is the representative layer *KappaLayer*.

Another approach would be to take the average water content of the profile, but this would typically give too much weight to deep layers where the temperature wave is attenuated. Either approach violates the assumptions on which the method is based anyway!

Procedure `Harmonic_Soil_Temp_Daily` is called at the beginning of each day to compute those parts of the method that are reasonably constant during the day.

The first of these are the surface amplitude and mean value of the temperature wave, computed from the air temperature as:

SurfaceAmplitude := (SmoothedMaxTemp – SmoothedMinTemp) / 2
SurfaceMean := (SmoothedMaxTemp + SmoothedMinTemp) / 2 (1)

where the *Smoothed...* temperatures are 3-day averages centered on the current day. Note that this method is really only valid if the driving function, i.e. air temperature, is a periodic function with constant amplitude and mean. In most climates this is not the case. So we approximate this by the 3-day values, and accept discontinuities between days.

Second, this procedure estimates the thermal diffusivity *Kappa* ($m^2\ s^{-1}$) of the profile, using the current water status of the representative layer *KappaLayer*, by linear interpolation in a piecewise linear function with the three known points being the diffusitivities at 0%, 20%, and 40% volumetric water content in a medium-textured soil. In the current implementation, these are fixed at the pairs (0, 0.2 x 10^{-6}), (20, 0.5 x 10^{-6}), and (0.4 x 10^{6}) (%, $m^2\ s^{-1}$), following empirical data from Campbell (1977, figure 2.7) and Monteith & Unsworth (1990, table 13.1). Knowing the diffusivity, the damping depth (m) (i.e. depth in the soil at which the wave is exactly π radians out of phase with the surface wave) is computed as:

$$\text{DampingDepth} := \sqrt{(2 * \text{Kappa} / \Omega)} \quad (2)$$

Note that by doing this computation only daily, we are assuming that the change during one day in the water content of the representative layer is not significant when compared to the other assumptions inherent in this method.

Procedure `Harmonic_Soil_Temp` is called at each time step to determine the soil temperature of each soil layer, using the *DampingDepth*, *SurfaceAmplitude*, and *SurfaceMean* computed daily, as well as the invariant wave frequency Ω. The temperature of the layer is taken to be the temperature at its midpoint. This is determined by first computing the proportion of the damping depth at the node:

$$\text{DampingFraction} := \text{NodeDepth[layer]} / \text{DampingDepth} \quad (3)$$

and from this, the amplitude of the wave at this depth:

$$\text{Amplitude} := \text{SurfaceAmplitude} * e^{-\text{DampingFraction}} \quad (4)$$

This equation shows that at the damping depth, the amplitude is reduced to 1/e or 37% of the surface amplitude. Finally, the temperature is computed as:

sim.SoilTemp[layer] := SurfaceMean
+ (Amplitude * sin(elapsed_secs * Ω – Damping Fraction)) (5)

where *elapsed_secs* is the number of seconds into the day, which converts the frequency of the wave into a fraction of a day. Note that *DampingFraction* has the effect of lagging the temperature wave as the depth increases.

6. GAPS for Programmers

GAPS was written in the Turbo Pascal language (Borland International), versions 4, 5, 6 and 7, and 9 (Delphi 2), over a period of seven years (and growing) by several programmers (see acknowledgements). Because of the 'multiple cooks', the design and coding style is not always as consistent as we would like. To compile the current version of GAPS, you need the command-line compiler ('tpc' or 'bpc') for Turbo (or Borland) Pascal 5.5, 6.0. or 7.0. and the Turbo Vision 2.0. application framework.

The DOS interface was built using the object-oriented application framework, Turbo Vision 2.0. from Borland International. The Turbo Vision source code and object files are copy protected. If you want to (re)compile the GAPS source code you need to buy the Turbo Vision library from Borland. Often Turbo Vision comes together with the Borland Pascal compiler. Some bugs in Turbo Vision had to be fixed. An 'un'official list of bugs in Turbo Vision 2.0 is put together by Brad Williams and can be downloaded via ftp at vtucs.cc.vt.edu in the turbo-vision/faq directory as TVBUGSx.ZIP.

Except for the Turbo Vision framework , the complete source code for GAPS is provided. A simple flowchart is given in appendix D. The compiler settings, memory sizes and directory settings used to compile the executables is given at the beginning of the files gaps.pas, gsb.pas and readme.pas.

6.1 Changing GAPS' limits

In a practical computer program, it is necessary to place limits on the size of data structures. In GAPS these are as follows. Variable names in source file 'global.pas' are given in parentheses, and the approximate additional memory in bytes needed for each increment in brackets.

1. the maximum number of time steps in a day (MaxTimeStep), fixed at 24 * 12, i.e. every 5 minutes [penalty: 0];

2. the maximum number of years that can be simulated in one run (MaxYears), fixed at 11 [penalty: 70];

3. the highest-numbered soil layer (MaxLayer), fixed at 20. This is one greater than the number of soil layers (since layer 1 represents the atmosphere), so that the maximum number of soil layers is 19 [penalty: 128 static, up to 48 dynamic, depending on procedures selected at run-time];

4. the maximum number of crops that can be simulated (MaxCrops), fixed at 6 [penalty: 70];

These could be all changed by a non-programmer, to allow more detailed or longer-running simulations. The penalty for increasing these limits is the extra space required to hold correspondingly-larger data structures, as well as longer running time for the additional time steps or soil layers.

To change these limits, you simply need to edit the source file 'global.pas' with any text editor, change the corresponding constant definition, and then rebuild GAPS and GSB. The resulting executables will incorporate the changed limits.

6.2. Changing GAPS' procedures

It is certainly possible to change GAPS to suit your particular modelling needs. There are three major classes of changes you might make:

(1) modifying an existing GAPS method,

(2) expanding the method repertoire with another way of doing something that GAPS can already do, and

(3) expanding GAPS to simulate a phenomenon not now simulated.

An example of the first type of change would be changing the computation of light transmitted through the canopy as a function of leaf area; an example of the second type of change would be adding another way of simulating soil temperature; an example of the third type of change would be adding a simulation of the soil solution (ionic equilibrium).

There are two problems with changing GAPS on your own. First is that your changes may be incompatible with enhancements which we may make, so that you would not be able to take advantage of future versions of GAPS. Second is that other users don't get the benefits of your insights. We encourage you to contact us before undertaking any changes, both to see if we or someone else might already be working on the same enhancement, and to coordinate such changes with us.

The file newcrop.pas presents a skeleton for a crop object that is descendent from the highest-level crop model . It explains the interfaces expected by the simulation driver.

7. References

Acevedo, E. 1975. The growth of maize (Zea mays L.) under field conditions as affected by its water relations. Ph.D. thesis, Univ. of California, Davis. 253 p.

Bouma, J., & van Lanen, H.A.J. 1987. Transfer functions and threshold values: from soil characteristics to land qualities. pp. 106-110 in: Beek, K.J., Burrough, P.A., & McCormack, D.E. (eds): *Quantified land evaluation procedures*, Enschede: ITC.

Bristow, K.L. 1983. Simulation of heat and moisture transfer through a surface residue-soil system. Ph.D. thesis. Washington State University, Pullman.

Brouwer, R., 1962. Distribution of dry matter in the plant. Netherlands Journal of Agricultural Science 10:361-76.

Buttler, I.W., & Riha, S.J. 1989. GAPS: a general purpose simulation model of the soil-plant-atmosphere system, Version 1.1 User's Manual. Ithaca, NY: Cornell University Department of Agronomy.

Campbell, G.S. 1974. A simple method for determining unsaturated hydraulic conductivity from moisture retention data. Soil Sci. 117:311-314.

Campbell, G.S. 1977. *An introduction to environmental biophysics*. New York: Springer Verlag. 159 pp.

Campbell, G.S. 1981. Fundamentals of radiation and temperature relations. Physiological Plant Ecology I. Encyclop. Plant Physiol., New Ser. 12A.

Campbell, G.S. 1985. *Soil physics with BASIC: transport models for soil-plant systems*. New York: Elsevier.

Childs, S. W., J. R. Gilley and W. E. Splinter. 1977. A simplified model of corn growth under moisture stress. Trans. ASAE 20:858-865.

Dale, R. F., Coelho, D. T., and K. P. Gallo. 1980. Prediction of daily green leaf area index for corn. Agron. J. 72:999-1005.

DeVries, D.A. 1963. Thermal properties of soils. p.210-235. in: van Wijk, W.R.(ed) *Physics of Plant Environment*. Amsterdam: North-Holland.

Fisher, M. J., Charles-Edwards, D. A., and M. M. Ludlow. 1981. An analysis of the effects of repeated short term soil water deficits on stomatal conductance to carbon dioxide and leaf photosynthesis by the legume Macroptilium atropurpureum cv. Siratro. Aust. J. Plant Physiol. 8:347-357.

Foth, H. D. 1962. Root and tap growth of corn. Agron J. 54:49-52.

Fuchs, M., Campbell, G.S. and R.I. Papendick. 1978. An analysis of sensible and latent heat flow in a partially frozen unsaturated soil. Soil Sci. Soc. Am. J. 42:379-385.

Gardner, W. R. 1960. Dynamic aspects of water availability to plants. Soil Sci. 89:63-73.

Gardner, W. R. and C. F. Ehlig. 1962. Some observtions on the movement of water to the plant. Agron. J. 54:453-456.

Gates, D.M. 1980. *Biophysical Ecology*. New York: Springer Verlag.

Gerik, T.J., Rosenthal, W.D., & Duncan. 1988. Field Crop Res 19:63-79.

Hesketh, J. and D. Baker. 1967. Light and carbon assimilation by plant communities. Crop Sci.7:285-293.

Hillel, D. 1980. *Introduction to soil physics.* Orlando: Academic Press.

Hillel, D. 1980. *Fundamentals of Soil Physics*. New York: Academic Press.

Hofstra, G. and J. D. Hesketh. 1969. Effect of temperature on the gas exchange of leaves in thelight and dark. Planta 85:228-237.

Jackson, J.E., and J.W. Palmer, 1979. A Simple Model of Light Transmission and Interception by Discontinous Canopies. Annals of Botany 44:381-83.

Jackson, J.E., and J.W. Palmer, 1981. Light Distribution in Discontinous Canopies: Calculation of Leaf Areas and Canopy Volumes Above Defined 'Irradiance Contours' for use in Productivity Modelling. Annals of Botany 47:561-65.

Johnson, I.R., 1990. Plant respiration in relation to growth, maintenance, ion uptake and nitrogen assimilation. Plant, Cell, and Environment 13:319-28.

Jones, C.A. and J.R. Kiniry (eds). 1986. *CERES-Maize: A simulation model of maize growth anddevelopement*. College Station, TX: Texas A&M University Press. 194 pp.

Jones, C.A., J.T Ritchie, J.R. Kining and D.C. Godwin. 1986. Subroutine Structure. In: Jones, C.A. and J.R. Kiniry (eds). 1986. *CERES-Maize: A simulation model of maize growth anddevelopement*. College Station, TX: Texas A&M University Press. 194 pp.

Jury, W.A. and C.B. Tanner. 1975. Advection modification of the Priestley-Taylor ET formula. Agron. J. 67:840-842.

van Keulen, H., & Wolf, J. (eds). 1986 *Modelling of agricultural production: weather, soils, and crops*. Wageningen: Pudoc.

Kiniry, J.R., Williams, J.R., Gassman, P.W., & Debaeke, P. 1992. A general, process-oriented model for two competing plant species. Transactions ASAE 35(3): 801-810

Lambers, H., A. van der Werf and H. Konings, 1991. Respiratory Patterns in Roots in Relation to Their Functioning. In: Y. Waisel, A. Eshel and U. Kafkafi (Eds) Plant Roots: the Hidden Half. Marcel Dekker, Inc. New York pp.925.

Linacre, E. T. 1977. A simple formula for estimating evaporation rates in various climates, using temperature data alone. Agric. Meteorol. 18:409-424.

McInnes, K.J. 1981. Thermal conductivities of soils from dryland wheat regions of Eastern Washington. M.S. thesis, Washington State University, Pullman.

Monteith, J. L. 1964. Evaportion and environment. In: the State and Movement of Water in Living Organisms. 19th Symp. Soc. Exp. Biol.

Monteith, J. L. 1981. Climatic variation and the growth of crops. Q.J.R. Meteorol. Soc. 107:749-774.

Monteith, J. L., & Unsworth, M.H. 1990. *Principles of environmental physics*, 2nd ed. New York: American Elsevier.

Nakayama, K. and A. Nakamura. 1982. Estimating potential evapotranspiration by the Priestley-Taylor model. J. Agr. Met. 37:297-302.

Norman, J. M. 1982. Simulation of microclimates. In: *Biometeorology in integrated pest management*. New York: Academic Press.

Parde, J., 1980. Forest Biomass. Forestry Abstracts 41:343-62.

Penman, H. L. 1948. Natural evaportion from open water bare soil and grass. Roy. Soc. London, Proc. Ser. A 193:120-146.

Pook, E.W., 1985. Canopy Dynamics of Eucalyptus marginata Hook. III. Effects of Drought. Australian Journal of Botany 33:65-79.

Priestley, C.H.B. and B.J. Taylor. 1972. On the assessment of surface heat flux evaporation using large-scale parameters. Mon. Weather Rev. 100:81-92.

Rauscher, H.M., J.G. Isebrands, G.E. Host, R.E. Dickson, D.I. Dickmann, T.R. Crow, and D.A. Michael, 1990. ECOPHYS: An ecophysiological growth process model for juvenile poplar. Tree Physiology 7: 255-81.

Riha, S. J. 1985. Estimating water fluxes in Douglas-fir plantations. Can. J. of For. Res. 5:701-707.

Ritchie, J.T., & Crum, J. 1989.. Converting soil survey characterization data into IBSNAT crop model input. pp. 155-167 in: Bouma, J. & Bregt, A.K. (eds.) *Land qualities in space and time*", Wageningen: PUDOC.

Ritchie, J.T., J.R. Kining, C.A. Jones and P.T. Dyke. 1986. Model Inputs. In: Jones, C.A. and J.R. Kiniry (eds). 1986. *CERES-Maize: A simulation model of maize growth anddevelopement*. College Station, TX: Texas A&M University Press. 194 pp.

Rosenthal, W.D., Vanderlip, R.L., Jackson, B.S., & Arkin, G.F. 1989. *SORKAM: a grain sorghum crop growth model*. TAES Computer Software Documentation Series MP1669. College Station, TX: Texas Agricultural Experiment Station.

Rossiter, D.G. 1991. *Modern computer programming techniques for environmental simulation modelling*. SCAS Teaching Series 1. Ithaca, NY: Department of Soil, Crop, & Atmospheric Sciences, Cornell University.

Sharma, M.L. 1985. Estimating Evapotranspiration. Advances in Irrigation 3:213-281.

Shaw, E.M. 1988. *Hydrology in practice*, 2nd ed. London: Van Nostrand Reinhold International.

Shinozaki, K., K. Yoka, K. Hozumi, and T. Kira, 1964. A quantitative analysis of plant form: the pipe model theory. II. Further evidence of the theory and its application in forest ecology. Japanese Journal of Ecology 14:133-39.

Spitters, C.J.T. & R. Aerts. 1983. Simulation of competition for light and water in crop-weed associations. Aspects of Applied Biology 4: 467-483.

Shuttleworth, W.J. and I.R. Calder. 1979. Has the Priestley-Taylor equation any relevance to forest evaporation? J. Appl. Meteorol. 18:639-646.

Stewart, R.B. and W.R. Rouse. 1977. Substantiation of the Priestley-Taylor parameter for potential evaporation in high latitude. J. Appl. Meteorol. 16:649-650.

Stoeckle, C.O. 1985. Simulation of the effect of water and nitrogen stress on growth and yield of spring wheat. Ph.D. thesis. Washington State University, Pullman.

Stockle, C. and G. S. Campbell. 1985. A simulation model for predicting effect of water stress on yield: an example using corn. Adv. in Irrigation 3:283-323.

Swift, L.W. 1976. Algorithm for solar radiation on mountain slopes. Water Res. 12:108-112.

Tanner, C.B. and W.A. Jury. 1976. Estimating evaporation and transpiration from a row crop during incomplete cover. Agron. J. 68:239-243.

Thornley, J.H.M., and I.R. Johnson, 1990. Plant and Crop Modelling. Oxford University Press, Oxford, UK.

USDA-Soil Conservation Service. 1972. National Engineering Handbook, Hydrology. Washington: USDA-SCS.

Wallace, J.S. 1995. Towards a coupled light partitioning and transpiration model for use in intercrops and agroforestry. pp. 153-162 *in*: Sinoquet, H. and Cruz, P. *Ecophysiology of tropical intercropping*. INRA, Paris.

Weaver, H.L. 1984. A mechanistic model of evapotranspiration from Saltcedar. Ph.D. thesis. Pullman: Washington State University.

Wilkerson, G.G., J.W. Jones, K.J. Boote, K.T. Ingram and J.W.Mishoe. 1983. Modeling soybean growth for crop management. Transactions of the Am. Soc. of Ag. Eng. 26:63-73.

Williams, J.R., Jones, C.A., and Dyke, P.T. 1989. EPIC--Erosion/Productivity Impact Calculator. 1. The EPIC model. Temple, TX: USDA-ARS.

Wilson, J.B., 1988. A Review of Evidence on the Control of Shoot:Root Ratio, in Relation to Models. Annals of Botany 61:433-49.

Appendix A: Common problems

Run-time errors

GAPS or GSB may 'crash' (i.e. stop with an internal program error), in which case you will see an error message at the top of the screen. This section discusses the most common of these, and which of them may be solved by the GAPS user.

Run-time errors other than those listed below, and also some of these, are programming errors. We would more than appreciate a note from you with the error message, program code address (like '09AB:137C'), and a brief description of what caused the crash. A printout or diskette with the input files and scenario would help.

Error 200: Division by zero

The denominator of an expression was numerically zero. This may result from non-physical inputs, which cause empirical relations to be evaluated outside their calibrated ranges.

Error 203: Heap overflow

This error is generated as GAPS attempts to allocate dynamic variables on the heap. Many of these are generated only for specific GAPS models, e.g. crop objects and soil layers. You can avoid this error by recompiling GAPS to allow more heap space.

Error 205: Floating point overflow

The result of a floating-point calculation could not be represented in a floating-point number. This is commonly a result of division by a very small number. It may result from non-physical inputs, which cause empirical relations to be evaluated outside their calibrated ranges.

Error 206: Invalid numeric format in Read

This is most likely called by an incorrect input data file. If you prepare your data files using correct formats, you should not get this error (if you do, it's a programming error). See the 'read' statements in the appropriate 'load_...' procedure in source file 'fileio.pas' for the formats of each data item.

Error 256: Non-convergence of iterative solution

See the section on the soils input file, §3.4.1.2 above, for a discussion of how soil files can be changed to avoid this error.

Error 259: Couldn't balance carbon

The Stockle-Riha wheat model uses an iterative Newton-Raphson root finding method to determine the daily partitioning of available C to roots, shoot, grain, and reserve. If this doesn't converge, error 259 is generated.

Appendix C: Input and output files

An example is given for most input and output files. Comments are printed in italic.

Def files

Climate file (climate.def)

```
# GAPS 3.0  climate file with the default values
# The climate file must be saved with the extension .CLI
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single  (i)=integer  (b)=boolean  (s70)=string[70]
#
*** NO NAME ***          # ClimateName (s70)
1                # FirstDay (i) [day of year]
365              # LastDay  (i) [day of year]
# 1 input line for every day
# MinTemp       (s) [C]        : default 0.0
# MaxTemp       (s) [C]        : default 20.0
# SolRad        (s) [MJ/m2 d] : default -1 (≈ not known)(if -1 then sim.NetRad:= 0)
# Precip        (s) [mm/d]     : default 0.0
# RelHumid      (s) []         : default -1 (≈ not known)
# Windspeed     (s) [m/s]      : default -1 (≈ not known)(if -1 then RA:=crop.RA or 90 s/m)
# PanEV         (s) [mm/d]     : default 0.0
# VaporDensity (s) [g/m3]     : default -1 (≈ not known)
# SnowPack      (s) [cm]       : default 0.0
#MinT  MaxT  SolRad Precip RelHumid Windspeed PanEV VaporDnsity Snow
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  # Day 1  january 1
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  2
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  3
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  4
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  5
...
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  361
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  362
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  363
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  364
0.0    20.0   -1      0.0     -1       -1      0.0       -1     0.0  #  365
```

Location file (location.def)

```
# GAPS 3.0  location file with the default values
# The location file must be saved with the extension .LOC
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single  (i)=integer  (b)=boolean  (s70)=string[70]
#
NO NAME          # LocationName    (s70) []
0.0              # Latitude        (s) [°N or S]
43200            # TSN             (s) [s]
1.26              # Alpha           (s) [] (Priestly-Taylor coefficient)
1.0               # Kp              (s) [0-1] (Pan coefficient)
```

```
1.0              # Kc               (s) [0-1] (crop coefficient)
20               # BLC              (s) [W/m2 K]
0                # RainFirst        (i) [h]
24               # RainLast         (i) [h]
2.0              # WindHeight       (s) [m]
0                # Elevation        (s) [m]
0.15              # WorkMaxDepth[n] (s) [m] for n:= till (1) to traffic (2)
0.05             # WorkMaxDepth[n]  (s) [m]
0.37              # WorkLimitWN[n]  (s) [m3/m3]  for n:= till (1) to traffic (2)
0.37             # WorkLimitWN[n]   (s) [m3/m3]
-9.8              # WorkLimitWP[n]  (s) [J/kg] for n:= till (1) to traffic (2)
-9.8             # WorkLimitWP[n]   (s) [J/kg]
0.05             # Depth_of_evap    (s) [m]
0.54             # CO2ext           (s) [g/m3]
```

Soil file (soil.def)

```
# GAPS 3.0  soil file with the default values
# The soil file must be saved with the extension .SOL
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single   (i)=integer   (b)=boolean   (s70)=string[70]
#
#
*** NO NAME ***          # SoilName (s70)[]
0.78                          # SwAbs (s) [0-1.00]
78                            # CN2   (i) [0-100]
0.00                          # Slope (s) []
2                             # WUD   (s) [0-10]
5                             # Layer (i) [] LastLayer := Minimum of (Layer and MaxLayer-1)
#
# 1 layer inputline for n := FirstLayer to LastLayer;   all (s)
# LowBound      (s) [m] :
# BulkDensity   (s) [Mg/m3]    : default 1.2
# PartDensity   (s) [Mg/m3]    : default 2.6
# HydCon        (s) [kg s/m3] : default 0.003
# AirEntryPot   (s) [J/kg]     : default -0.3
# BValue        (s) []  : default 7.8
# InitWater     (s) []  : default -1 -> if -1, value is replaced by DUL
# DUL           (s) []  : default 0.3
# DLL           (s) []  : default 0.1
# InitSoilTemp (s) [C] : default 20
# Clay          (s) [cg/kg] : default 25
# Silt          (s) [cg/kg] : default 25
# SatWaterCon   (s) [] : default -1 -> if -1, value is replaced by (1-BulkDensity/PartDensity)
# CoarseFrag    (s) [] : default 0
#LowBound BulkDensity PartDensity HydCon AirEntryPot BValue InitWater DUL DLL InitSoilTemp Clay Silt
0.01          1.2         2.6      0.003    -0.3      7.8      -1     0.3 0.1      20        25   25
0.03          1.2         2.6      0.003    -0.3      7.8      -1     0.3 0.1      20        25   25
0.08          1.2         2.6      0.003    -0.3      7.8      -1     0.3 0.1      20        25   25
0.15          1.2         2.6      0.003    -0.3      7.8      -1     0.3 0.1      20        25   25
```

Plant file (plant.def)

```
# GAPS 3.0  plant file with the default values
# The plant file must be saved with the extension .PLT
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single   (i)=integer   (b)=boolean   (s70)=string[70]
```

```
#
# Global variables
1               # crop options (0     : None,
1               #               1 & 0 : Stockle-Riha
                #               1 & 1 : Stockle-Riha - maize
                #               1 & 2 :              - wheat
                #               1 & 3 :              - tree
                #               2 & 0 : Sorkam
                #               2 & 1 : Sorkam - temperate
                #               2 & 2 :        - tropical
                #               3 & 0 : Epic
                #               4 & 0 : Constant)
0               # proc_status[1]  --> Canopy water interception
0               # proc_status[2]  --> Critical leaf water potential
0               # proc_status[3]  --> -
0               # proc_status[4]  --> -
0               # proc_status[5]  --> -
0               # proc_status[6]  --> -
0               # proc_status[7]  --> -
0               # proc_status[8]  --> -
0               # proc_status[9]  --> -

# loaded in croplib.crop_model.Init
* GENERIC *     # crop.PlantName    (s70) []
1               # crop.SowingDate   (i) [d]
365             # crop.HarvestDate  (i) [d]
-1500           # crop.RootWP       (s) [J/kg]
0.0020          # crop.RootRad      (s) [m]
2.5E+0010       # crop.RootRes      (s) [m4/kg s]
2               # crop.FRoot        (i) []
19              # crop.NRoot        (i) []
20000           # crop.RootDens[2]  (s) [m/m3]; for n:= FRoot to NRoot
20000           # crop.RootDens[3]
20000           # crop.RootDens[4]
20000           # crop.RootDens[5]
20000           # crop.RootDens[6]
20000           # crop.RootDens[7]
20000           # crop.RootDens[8]
20000           # crop.RootDens[9]
20000           # crop.RootDens[10]
20000           # crop.RootDens[11]
20000           # crop.RootDens[12]
20000           # crop.RootDens[13]
20000           # crop.RootDens[14]
20000           # crop.RootDens[15]
20000           # crop.RootDens[16]
20000           # crop.RootDens[17]
20000           # crop.RootDens[18]
20000           # crop.RootDens[19]
1.50            # crop.CanopyHeight (s) [m]
30              # crop.RA           (s) [s/m]
20              # crop.RCopen       (s) [s/m]
0.75            # crop.AS           (s) []

# Main crop object variables loaded in croplib.crop_model.Init
0.00            # plant.LAI              (s) []
0.00            # plant.InitialDryMatter (s) [kg/ha]
1.20            # plant.MaxRootingDepth  (s) [m]
0.025           # plant.SowingDepth      (s) [m]

# variables only for Stockle-Riha crop model                                |
2.00            # plant_CanopyMax          (s) [kg/m2]                  |
-1500           # plant_CriticalLeafWP     (s) [J/kg]                       |
62500           # plant_PlantDensity       (s) [pl/ha]                  | only for
0.500           # plant_HarvestIndex       (s) []                       |
570             # plant_Kernels_per_plant  (i) []                       | Stockle-
8               # plant_Kernel_growth_rate (i) [mg/d]                       |
Riha
7.0             # plant_CLWP_Power         (s) []                       |
```

```
1.0             # plant_Pmax               (s) [mgCO2/m2Leaf.sec]              |
model
0.007           # plant_P_Effic            (s) [mgCo2/J]                |
0.75            # plant_P_Curve            (s) []                       |
0               # plant_PT_min             (i) [°C]                            |
25              # plant_PT_Opt             (i) [°C]                            |

# variables only for Stockle-Riha : corn model
3               # plant_Coeff[1]         (i)                     vegetative
49              # plant_Coeff[2]                                  pollination
56              # plant_Coeff[3]                                  late lag
58              # plant_Coeff[4]                                  early filling
65              # plant_Coeff[5]                                  senescing
# the following two ordered pairs are the (x,f) at two points on the
# height-vs-degree days curve, where:
#       x = degree days (AccDD) as a proportion of AccDD_to[pollination]
#       f = corresponding proportion of MaxHeight
0.3     0.2         # 1st calibration point on height curve (r, r) [0..1, 0..1]
0.5     0.8         # 2nd calibration point on height curve (r, r) [0..1, 0..1]

# variables only for Stockle-Riha : wheat model                               | only
for
0.20            # plant_RowSpacing (s) [m]                              |  wheat
100             # plant_Coeff[1]   (i)            emergencd
176             # plant_Coeff[2]                  1 leaf
328             # plant_Coeff[3]                  2 leaves
460             # plant_Coeff[4]                  3 leaves
602             # plant_Coeff[5]                  jointed
744             # plant_Coeff[6]                  5 leaves
886             # plant_Coeff[7]                  6 leaves
1028            # plant_Coeff[8]                  8 leaves
1170            # plant_Coeff[9]                  flag leaf extended
1312            # plant_Coeff[10]          booting completed
1454            # plant_Coeff[11]          heading completed
1514            # plant_Coeff[12]          flowering completed
1892            # plant_Coeff[13]          ripe
# the following two ordered pairs are the (x,f) at two points on the
# height-vs-degree days curve, where:
#       x = degree days (AccDD) as a proportion of AccDD_to[booted]
#       f = corresponding proportion of MaxHeight
0.3     0.2         # 1st calibration point on height curve (r, r) [0..1, 0..1]
0.5     0.8         # 2nd calibration point on height curve (r, r) [0..1, 0..1]

# variables only for Stockle-Riha : fast growing tree model                   |
0.0142          # plant_AccStemDM    (s) [kg/m2]                       |
0.0193          # plant_AccLeafDM    (s) [kg/m2]                       |
0.024           # plant_SinkGrowRate (s) [m/d]                             |
3               # plant_NSink        (i) []                            |
10000           # plant_LW_Sink      (i) [m/kg]                               |
13000           # plant_LW_Fine      (i) [m/kg]                               | only
for
0.18            # plant_SinkLength   (s) [m]                               |
fast
-4.358          # plant_aWood        (s) []                            |  growing
2.90            # plant_bWood        (s) []                            |   tree
-4.100          # plant_aHeight      (s) []                            |
1.949           # plant_bHeight      (s) []                            |
0.896           # plant_cHeight      (s) []                            |
25              # plant_RefTemp      (i) [°C]                                 |
6               # plant_SLA          (i) [m2/kg]                       |
365             # Plant_Init_Age     (i) [d]                                  |
1095            # plant_MaxLeafAge   (i) [d]                                  |
0.02            # OldRtBio[2]        (s) [kg/m3] for n:= FRoot to NRoot |
0.02            # OldRtBio[3]        (s) [kg/m3]                       |
0.015           # OldRtBio[4]        (s) [kg/m3]                       |
0.0017          # OldRtBio[5]        (s) [kg/m3]                       |

# variables only for Sorkam crop model                                       | only
for
```

```
0.90             # plant_RowSpacing   (s) [m]                                    |
sorkam
19               # plant_NLeaves      (i)                                  |
# the following two ordered pairs are the (x,f) at two points on the
# height-vs-degree days curve, where:
#       x = degree days (AccUnits1to3) as a proportion of GDD_to_3
#       f = corresponding proportion of MaxHeight
0.3     0.2         # 1st calibration point on height curve (r, r) [0..1, 0..1]
0.5     0.8         # 2nd calibration point on height curve (r, r) [0..1, 0..1]

# variables only for Epic crop model                                          | only
for
0.31             # plant_HarvestIndex       (s) []                         |   epic
10.0             # plant_BaseTemp           (s) [degrees C]                    |
25.0             # plant_OptimumTemp        (s) [degrees C]                    |
2000.0           # plant_PotUnits           (s) []                         |
25.0             # plant_BiomassEnergy      (s) [kg/Mj]                    |
0.60             # plant_DeclineFraction    (s) []                         |
4.0              # plant_MaxPotLAI          (s) []
# the following two ordered pairs are the (x,f) at two points on the
# LAI-vs-heat units curve, where:
#   x = proportion of heat units
#   f = corresponding proportion of LAI
0.29158  0.2         # 1st calibration point on LAI curve (r, r) [0..1, 0..1]
0.34282  0.8         # 2nd calibration point on LAI curve (r, r) [0..1, 0..1]1.00
# plant_LAIDeclineRate      (s) []                          |
0.01             # plant_DroughtSensitivity (s) []                         |
62500            # plant_PlantDensity       (s) [pl ha-1]
# the following two ordered pairs are the (x,f) at two points on the
# Max. LAI-vs-population density curve, where:
#   x = population density, pl ha-1 (cf. plant_PlantDensity)
#   f = corresponding proportion of maximum LAI attainable at that density
10000 0.2            # 1st calibration point on LAI curve (r, r) [pl ha-1, 0..1]
50000 0.8            # 2nd calibration point on LAI curve (r, r) [pl ha-1, 0..1]
13.8             # plant_MaxHeightPerWeight   (s) [m kg-1]

# variables only for Constant crop model                                      | only
for
4.00             # plant_MaxTopDM     (s) [kg/ha]                          | constant
0.00             # plant_CanopyMax    (s) [kg/m2]                          |  model
10.0               # plant_BaseTemp      (s) [degrees C]                        |
```

Save file (save.def)

```
# GAPS 3.0  save file with the default values
# The save file must be saved with the extension .SAV
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (i)=integer
#
# Number of hours (n) to save, followed by n lines with the hours (i)[0-23h]
1       # NumberOfHours (i)
12
# Number of days (n) to save, followed by n lines with the days (i)[d]
5       # NumberOfDays  (i)
100
150
200
250
300
```

Cropping Sequence file (sequence.def)

```
# GAPS 3.5 Crop Sequence file
# extension .SEQ
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single  (i)=integer  (b)=boolean  (s40)=string[40]
#
2                           # number of entries
#sowingYear           (i)    1 is start of simulation
#sowingDate           (i)    1..365
#harvestYear  (i)    must be >= sowingYear
#harvestDate  (i)    if harvestYear = sowingYear, must be > sowingDate
#plantFileName (s40)         *.plt
#sowingYear   sowingDate    harvestYear   HarvestDate   plantFileName
1      1      2      365    ..\tutor\first.plt          #constant
1      119    1      290    ..\tutor\second.plt         #stockle maize
#competition model
1                    # competition options loaded in fileio.load_sequence
                     # competition options (0 : None
                     #                     1 : ALMANAC
```

Climate file (example.cli)

```
# GAPS 3.0  climate file
# The climate file must be saved with the extension .CLI
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single   (i)=integer   (s70)=string[70]
#
CMN11000.cli                # ClimateName (s70)
1                           # FirstDay (i) [day of year]
365                         # LastDay (i) [day of year]
# 1 input line for every day
# MinTemp       (s) [C]        : default 0.0
# MaxTemp       (s) [C]        : default 0.0
# SolRad         (s) [Mj/m2 d] : default -1 (≈ not known)(if -1 then sim.NetRad:= 0)
# Precip        (s) [mm/d]     : default 0.0
# RelHumid      (s) []         : default -1 (≈ not known)
# WindSpeed     (s) [m/s]      : default -1 (≈ not known)(if -1 then RA:=crop.RA or 90s/m)
# PanEV         (s) [mm/d]     : default 0.0
# VaporDensity (s) [g/m3]      : default -1 (≈ not known)
# SnowPack      (s) [cm]       : default 0.0
#MinT   MaxT  SolRad  Precip  RelHumid  WindSpeed  PanEV  VaporDnsity  Snow
-12.2   -6.5    5.3    0.0      44.5       -1       0.0      -1         0.0   # 1   january 1
 -4.3    2.3    2.2    0.0      41.2       -1       0.0      -1         0.0   # 2   january 2
-14.6   -0.8    2.3    2.5      64.2       -1       0.0      -1         0.0   # 3   january 3
-18.8   -9.3    4.4    0.3      62.0       -1       0.0      -1         0.0   # 4   january 4
-17.9   -6.6    4.3    1.3      73.2       -1       0.0      -1         0.0   # 5   january 5
 -9.7   -6.1    4.6    3.7      67.2       -1       0.0      -1         0.0   # 6   january 6
-13.6   -2.0    6.0    0.1      62.1       -1       0.0      -1         0.0   # 7   january 7
-16.9   -4.3    5.8    0.0      50.2       -1       0.0      -1         0.0   # 8   january 8
 -9.8   -3.6    3.9    0.0      50.4       -1       0.0      -1         0.0   # 9   january 9
...
-15.5   -6.4    4.9    0.0      72.8       -1       0.0      -1         0.0   # 361 december 27
-13.8   -5.9    5.1    0.0      62.4       -1       0.0      -1         0.0   # 362 december 28
-22.3  -18.3    4.4    0.0      62.4       -1       0.0      -1         0.0   # 363 december 29
-14.7   -8.0    8.1    0.0      60.8       -1       0.0      -1         0.0   # 364 december 30
 -9.2   -2.7   10.5    0.0      60.2       -1       0.0      -1         0.0   # 365 december 31
```

Location file (example.loc)

```
# GAPS 3.0  location file
# The climate file must be saved with the extension .LOC
#
# Lines starting with the '#' sign are ignored at input.
# Comments added after the input data, preceded by a '#' are also ignored.
#
# (s)=single  (i)=integer  (s70)=string[70]
#
Redwood Co.,MN       # LocationName    (s70)
44.530               # Latitude        (s) [°N or S]
43200                # TSN             (s) [s]
1.26                 # Alpha           (s) [] (Priestly-Taylor coefficient)
1.0                  # Kp              (s) [0-1] (Pan coefficient)
1.0                  # Kc              (s) [0-1] (crop coefficient)
20                   # BLC             (s) [W/m2 K]
0                    # RainFirst       (i) [h]
24                   # RainLast        (i) [h]
5.0                  # WindHeight      (s) [m]
305                  # Elevation       (s) [m]
0.150                # WorkMaxDepth[n] (s) [m] for n:= till (1) to traffic (2)
0.050                # WorkMaxDepth[n] (s) [m]
0.370                # WorkLimitWN[n]  (s) [m3/m3] for n:= till (1) to traffic (2)
0.370                # WorkLimitWN[n]  (s) [m3/m3]
-9.8                 # WorkLimitWP[n]  (s) [J/kg] for n:= till (1) to traffic (2)
-9.8                 # WorkLimitWP[n]  (s) [J/kg]
0.050                # Depth_of_evap   (s) [m]
0.54                 # CO2Ext          (s) [g/m3]
```

Soil file (example.sol)

```
# GAPS 3.0  soil file
# The soil file must be saved with the extension .SOL
#
# Lines starting with the '#' sign are ignored at input.
# Comments added after the input data, preceded by a '#' are also ignored.
#
# (i)=integer (s)=single (s70)=string[70]
#
Dickman SL, 0-2%               # SoilName (s70)[]
1.00                           # SwAbs (s) [0-1.00]
78                             # CN2   (i) [0-100]
0.00                           # Slope (s) []
2                                   # WUD   (s) [0-10]
12                              # Layer (i) [] LastLayer := Minimum of (Layer and MaxLayer-1)
#
# 1 layer inputline for n := FirstLayer to LastLayer;   all (s)
# LowBound     (s) [m]
# BulkDensity  (s) [Mg/m3]    : default 1.2
# PartDensity  (s) [Mg/m3]    : default 2.6
# HydCon       (s) [kg s/m3] : default 0.003
# AirEntryPot  (s) [J/kg]     : default -0.3
# BValue       (s) []  : default 7.8
# InitWater    (s) []  : default -1 -> if -1, value is replaced by DUL
# DUL          (s) []  : default 0.3
# DLL          (s) []  : default 0.1
# InitSoilTemp (s) [C] : default 20
# Clay         (s) [cg/kg] : default 25
# Silt         (s) [cg/kg] : default 25
# SatWaterCon  (s) [] : default -1 -> if -1, value is replaced by (1-BulkDensity/PartDensity)
# CoarseFrag   (s) [] : default 0
#LowBound BulkDensity PartDensity HydCon AirEntryPot BValue InitWater DUL DLL InitSoilTemp Clay Silt
0.010   1.350          2.650    0.00300    -0.30      7.80   0.283  0.283 0.143   0.0       5.0 25.0
0.050   1.350          2.650    0.00300    -0.30      7.80   0.283  0.283 0.143   0.0      25.0  5.0
0.150   1.350          2.650    0.00300    -0.30      7.80   0.283  0.283 0.143   0.0      25.0 25.0
0.300   1.350          2.650    0.00300    -0.30      7.80   0.283  0.283 0.143   0.0      25.0 25.0
0.480   1.425          2.650    0.00300    -0.30      7.80   0.273  0.273 0.143   0.0       25.0 25.0
0.660   1.500          2.650    0.00300    -0.30      7.80   0.218  0.218 0.108   0.0      25.0 25.0
0.840   1.500          2.650    0.00300    -0.30      7.80   0.218  0.218 0.108   0.0      25.0 25.0
1.000   1.550          2.650    0.00300    -0.30      7.80   0.050  0.050 0.010   0.0      25.0 25.0
1.100   1.550          2.650    0.00300    -0.30      7.80   0.050  0.050 0.010   0.0      25.0 25.0
1.300   1.550          2.650    0.00300    -0.30      7.80   0.050  0.050 0.010   0.0      25.0 25.0
1.500   1.550          2.650    0.00300    -0.30      7.80   0.050  0.050 0.010   0.0      25.0 25.0
```

Plant file (example.plt)

```
# GAPS 3.0  plant file
# The plant file must be saved with the extension .PLT
#
# Lines starting with the '#' sign are ignored at input.
# Comments added after the input data, preceded by a '#' are also ignored.
#
# (s)=single  (i)=integer  (s70)=string[70]
#
# Global variables
1                    # crop options loaded in croplib.load_crop_option
1                    # crop options 1 & 1 : Stockle-Riha - maize
0                    # proc_status[1]  --> Canopy water interception
0                    # proc_status[2]  --> Critical leaf water potential
0                    # proc_status[3] --> -
0                    # proc_status[4] --> -
0                    # proc_status[5] --> -
0                    # proc_status[6] --> -
0                    # proc_status[7] --> -
0                    # proc_status[8] --> -
0                    # proc_status[9] --> -
# loaded in croplib.crop_model.init
corn - late          # crop.PlantName    (s70) []
123                  # crop.SowingDate   (i) [d]
296                  # crop.HarvestDate  (i) [d]
-1500                # crop.RootWP       (s) [J/kg]
0.0020               # crop.RootRad      (s) [m]
2.5E+0010            # crop.RootRes      (s) [m4/kg s]
3                    # crop.FRoot        (i) []
10                   # crop.NRoot        (i) []
50000.00             # crop.RootDens[3]     (s) [m/m3]; for n:=FRoot to NRoot
40000.00             # crop.RootDens[4]     (s) [m/m3]; for n:=FRoot to NRoot
27000.00             # crop.RootDens[5]     (s) [m/m3]; for n:=FRoot to NRoot
18000.00             # crop.RootDens[6]     (s) [m/m3]; for n:=FRoot to NRoot
13000.00             # crop.RootDens[7]     (s) [m/m3]; for n:=FRoot to NRoot
6700.00              # crop.RootDens[8]     (s) [m/m3]; for n:=FRoot to NRoot
2300.00              # crop.RootDens[9]     (s) [m/m3]; for n:=FRoot to NRoot
690.00               # crop.RootDens[10]     (s) [m/m3]; for n:=FRoot to NRoot
1.50                 # crop.CanopyHeight (s) [m]
40                   # crop.RA           (s) [s/m]
20                   # crop.RCopen       (s) [s/m]
0.60                 # crop.AS           (s) []

# Main crop object variables loaded in croplib.crop_model.Init
0.01                 # plant.LAI              (s) []
0.01                 # plant.InitialDryMatter (s) [kg/ha]
1.60                 # plant.MaxRootingDepth  (s) [m]
0.0250               # plant.SowingDepth      (s) [m]

# variables only for Stockle-Riha crop model
2.00                 # plant_CanopyMax           (s) [kg/m2]
-1500                # plant_CriticalLeafWP      (s) [J/kg]
62500                # plant_PlantDensity        (s) [pl/ha]
0.500                # plant_HarvestIndex        (s) []
540                  # plant_Kernels_per_plant   (i) []
8                    # plant_Kernel_Growth_Rate  (i) [mg/d]
7.0                  # plant_CLWP_Power          (s) []
0.0                  # plant_Pmax                (s) [mgCO2/m2Leaf.sec]
0.0                  # plant_P_Effic             (s) [mgCO2/J]
0.0                  # plant_P_Curve             (s) []
0                    # plant_PT_min              (s) [°C]
0                    # plant_PT_Opt              (s) [°C]

# variables only for Stockle-Riha corn model
```

```
3                   # plant_Coeff[1]          (i)
49                  # plant_Coeff[2]
56                  # plant_Coeff[3]
58                  # plant_Coeff[4]
65                  # plant_Coeff[5]
```

Save file (example.sav)

```
# GAPS 3.0  save file with the default values
# The save file must be saved with the extension .SAV
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
#  (i)=integer
#
# Number of hours (n) to save, followed by n lines with the hours (i)[0-23h]
1       # NumberOfHours (i)
12
# Number of days (n) to save, followed by n lines with the days (i)[d]
25      # NumberOfDays  (i)
1
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
```

Cropping Sequence file (example.seq)

```
# GAPS 3.5 Crop Sequence file
# extension .SEQ
#
# Lines starting with the '#' sign are ignored at input
# Comments added after the input data, preceded by a '#' are also ignored
#
# (s)=single   (i)=integer   (b)=boolean   (s40)=string[40]
#
2                           # number of entries
#sowingYear          (i)    1 is start of simulation
#sowingDate          (i)    1..365
#harvestYear   (i)    must be >= sowingYear
#harvestDate   (i)    if harvestYear = sowingYear, must be > sowingDate
```

```
#plantFileName (s40)         *.plt
#sowingYear    sowingDate    harvestYear    HarvestDate    plantFileName
1       1      2       365    ..\tutor\first.plt           #constant
1       119    1       290    ..\tutor\second.plt          #stockle maize
#competition model
1                     # competition options loaded in fileio.load_sequence
                     # competition options (0 : None
                     #                      1 : ALMANAC
```

Scenario file (example.set)

```
# GAPS 3.0 scenario file (extension ".set")
#
# Lines starting with the "#" sign are ignored at input.
# Comments added after the data, preceded by a "#" are also ignored.
# "-" = no file selected.
#
# (s)=single (i)=integer (s70)=string[70]
#
GAPS V 3.0                                  # version
2                                           # (i) number of simulation_years
C:\GAPS3\EXAMPLE.CLI                        # (s70) climateFileNames[n]
C:\GAPS3\EXAMPLE.CLI                        # (s70) climateFileNames[n]
C:\GAPS3\EXAMPLE.LOC                        # (s70) locationFileName
C:\GAPS3\EXAMPLE.SOL                        # (s70) soilFileName
C:\GAPS3\EXAMPLE.PLT                        # (s70) plantFileNames[n]
-                                           # (s70) plantFileNames[n]
C:\GAPS3\EXAMPLE.SAV                        # (s70) saveFileName
C:\GAPS3\EXAMPLE.SUM                        # (s70) sumFileName
C:\GAPS3\EXAMPLE.DET                        # (s70) detFileName
C:\GAPS3\EXAMPLE.LAY                        # (s70) layFileName
C:\GAPS3\EXAMPLE.SOF                        # (s70) sofFileName
C:\GAPS3\EXAMPLE.PLF                        # (s70) plfFileName
C:\GAPS3\EXAMPLE.CLF                        # (s70) clfFileName
1         # (0 or 1) proc_status[10] --> ETP-Priestley Taylor
0         # (0 or 1) proc_status[11] --> ETP-Penman
0         # (0 or 1) proc_status[12] --> ETP-Linacre
0         # (0 or 1) proc_status[13] --> ETP-Pan
0         # (0 or 1) proc_status[14] --> -
1         # (0 or 1) proc_status[15] --> Flow-Richards Eq.
0         # (0 or 1) proc_status[16] --> Flow-Tipping bucket
0         # (0 or 1) proc_status[17] --> Flow-Matrix flux pot.
1         # (0 or 1) proc_status[18] --> Uptake-Pot. driven
0         # (0 or 1) proc_status[19] --> Uptake-Plant avail.
0         # (0 or 1) proc_status[20] --> Field hours
0         # (0 or 1) proc_status[21] --> Input-Runoff
0         # (0 or 1) proc_status[22] --> EPIC-water uptake
0         # (0 or 1) proc_status[23] --> -
0         # (0 or 1) proc_status[24] --> -
0         # (0 or 1) proc_status[25] --> -
0         # (0 or 1) proc_status[26] --> -
0         # (0 or 1) proc_status[27] --> -
0         # (0 or 1) proc_status[28] --> Soil-Soil temp.
1         # (0 or 1) proc_status[29] --> Soil-Harmonic temp.
0         # (0 or 1) proc_status[30] --> -
3600      # (s) [900-92400 s] timestep
90        # (i) req_first_day
365       # (i) req_last_day
4         # (i) [1-4] run_views
14        # (i) [1-40] initviews[1]
27        # (i) [1-40] initviews[2]
30        # (i) [1-40] initviews[3]
35        # (i) [1-40] initviews[4]
```

Printout of input files

Climate file printout (climate.txt)

```
GAPS Climate Input Data

------------------------------------------------------------------------

Climate Data File Name : C:\GAPS3\EXAMPLE.CLI                          ClimateFileNam
es[]
Climate Name           : CMN11000.cli                                  clim.climateNa
me
Report Date            : 5/24/1994                                     date
Report Time            : 10:42: 5                                      time

------------------------------------------------------------------------
       Air   Air    Solar
Day    Temp  Temp   Rad    Rain     Rel.           Pan   Vapor  Snow
of    (max) (min)   MJ/    fall   Humidity  Wind Evap  Dens    cm
Year   (Celsius)  (m2 day)(mm/d) (fraction) (m/s) (mm/d) (g/m3)
------------------------------------------------------------------------

  1    -6.5 -12.2    5.4    0.0     44.5     ----   0.0   ----   0.0
  2     2.3  -4.3    2.2    0.0     41.2     ----   0.0   ----   0.0
  3    -0.8 -14.6    2.3    2.5     64.2     ----   0.0   ----   0.0
  4    -9.3 -18.8    4.4    0.3     62.0     ----   0.0   ----   0.0
  5    -6.6 -17.9    4.3    1.3     73.2     ----   0.0   ----   0.0
  6    -6.1  -9.7    4.6    3.7     67.2     ----   0.0   ----   0.0
  7    -2.0 -13.6    6.0    0.1     62.1     ----   0.0   ----   0.0
  8    -4.3 -16.9    5.8    0.0     50.2     ----   0.0   ----   0.0
  9    -3.6  -9.8    3.9    0.0     50.4     ----   0.0   ----   0.0
 10     1.0 -14.3    7.2    0.0     49.5     ----   0.0   ----   0.0
...
361    -6.4 -15.5    4.9    0.0     72.8     ----   0.0   ----   0.0
362    -5.9 -13.8    5.1    0.0     62.4     ----   0.0   ----   0.0
363   -18.3 -22.3    4.4    0.0     62.4     ----   0.0   ----   0.0
364    -8.0 -14.7    8.1    0.0     60.8     ----   0.0   ----   0.0
365    -2.7  -9.2   10.5    0.0     60.2     ----   0.0   ----   0.0

------------------------------------------------------------------------

1       2     3       4      5       6        7      8     9     10
```

1 day
2 clim.MaxTemp[day]
3 clim.MinTemp[day]
4 clim.SolRad[day]
5 clim.Precip[day]
6 clim.RelHumid[day]
7 clim.WindSpeed[day]
8 clim.PanEV[day]
9 clim.VaporDensity[day]
10 SnowPack[day]

Location file printout (location.txt)

```
GAPS Location Input Data

-------------------------------------------------------

Data File Name : C:\GAPS3\EXAMPLE.LOC
                                                                locationFileNa
me
Site Name      : Redwood Co.,MN
                                                                loca.locationN
ame
Report Date    : 5/24/1994                                      date
Report Time    : 10:42:15                                       time

-------------------------------------------------------

Latitude                                  :      44.53          loca.Latitude
Time of Solar Noon                        :      43200          loca.TSN
Priestley-Taylor Alpha                    :       1.26          loca.Alpha
Pan coefficient (Pan_ETP)                 :       1.00          loca.Kp
Crop coefficient (Pan_ETP)                :       1.00          loca.Kc
Boundary layer conductance                :         20          loca.BLC
First Hour of rain                        :          0          loca.RainFirst
Last  Hour of rain                        :         24          loca.RainLast
Wind height                               :        5.0
                                                                loca.WindHeigh
t
Elevation (m)                             :        305          loca.Elevation
Workability requirements for tillage:
  Depth of workability (m)                :      0.150          loca.WorkMaxDepth[till]
  Limiting water content (m3/m3)          :      0.370          loca.WorkLimitWN[till]
  Limiting water potential (J/kg)         :       -9.8          loca.WorkLimitWP[till]
Workability requirements for traffic:
  Depth of workability (m)                :      0.050
                                                                loca.WorkMaxDepth[traffic
]
  Limiting water content (m3/m3)          :      0.370          loca.WorkLimitWN[traffic]
  Limiting water potential (J/kg)         :       -9.8          loca.WorkLimitWP[traffic]
Depth to which evap. can dry to DUL (m)   :      0.050          loca.depth_of_evap
Atmospheric CO2 concentration (g/m3)      :       0.54          loca.CO2ext

-------------------------------------------------------
```

Soil file printout (soil.txt)

```
GAPS Soil Input Data

-------------------------------------------------------------------------

Soil File Name          : C:\GAPS3\EXAMPLE.SOL                  soilFileName
Soil Name               : Dickman SL, 0-2%                      soil.soilName
Shortwave absorptivity : 1.00                                   soil.SwAbs
Runoff Curve number     : 78                                    soil.CN
Site slope, m/m         : 0.00                                  soil.slope
Water use distribution : 2.0                                    soil.WUD
Last Layer              : 12                                    soil.LastLayer
Report Date & Time      : 5/24/1994 at 10:42:25                 date   time

-------------------------------------------------------------------------------------------------------
```

Layer no.	Lower Boundary (m)	Bulk Density ---(Mg/m3)----	Particle Density	Saturated Hydr. Conductivity (kg s/m3)	Air Entry Potential (J/kg)	B Value	Initial Water Content ------(m3/m3)------	DUL	DLL	Initial soil Temp (deg C)	Cl -
2	0.01	1.35	2.65	0.00300	-0.30	7.80	0.28	0.28	0.14	20.0	25
3	0.05	1.35	2.65	0.00300	-0.30	7.80	0.28	0.28	0.14	20.0	25
4	0.15	1.35	2.65	0.00300	-0.30	7.80	0.28	0.28	0.14	20.0	25
5	0.30	1.35	2.65	0.00300	-0.30	7.80	0.28	0.28	0.14	20.0	25
6	0.48	1.42	2.65	0.00300	-0.30	7.80	0.27	0.27	0.14	20.0	25
7	0.66	1.50	2.65	0.00300	-0.30	7.80	0.22	0.22	0.11	20.0	25
8	0.84	1.50	2.65	0.00300	-0.30	7.80	0.22	0.22	0.11	20.0	25
9	1.00	1.55	2.65	0.00300	-0.30	7.80	0.05	0.05	0.01	20.0	25
10	1.10	1.55	2.65	0.00300	-0.30	7.80	0.05	0.05	0.01	20.0	25
11	1.30	1.55	2.65	0.00300	-0.30	7.80	0.05	0.05	0.01	20.0	25
12	1.50	1.55	2.65	0.00300	-0.30	7.80	0.05	0.05	0.01	20.0	25

1 2 3 4 5 6 7 8 9

1 layer
2 LowBound[layer]
3 soil.bulkDensity[layer]
4 soil.PartDensity[layer]
5 soil.HydCond[layer]
6 soil.AirEntryPot[layer]
7 soil.BValue[layer]
8 soil.InitWater[]
9 soil.DUL[layer]
10 soil.DLL[layer]
11 soil.InitSoilTemp[
12 soil.Clay[layer]
13 soil.Silt[layer]
14 soil.SatWt[]
15 CoarseFrag[]

Plant file printout (id. example.plt)

Save file printout (save.txt)

```
GAPS Save Format File

-----------------------------------------------------------

Data File Name : C:\GAPS3\EXAMPLE.SAV
Report Date    : 5/24/1994
Report Time    : 10:41:53

-----------------------------------------------------------

Number of hours to save : 1                                  NumberOfHours

Hour  1: 12

Number of days  to save : 25                                 NumberOfDays

Day   1:   1   Day   2:  10   Day   3:  20   Day   4:  30
Day   5:  40   Day   6:  50   Day   7:  60   Day   8:  70
Day   9:  80   Day  10:  90   Day  11: 100   Day  12: 110
Day  13: 120   Day  14: 130   Day  15: 140   Day  16: 150
Day  17: 160   Day  18: 170   Day  19: 180   Day  20: 190
Day  11: 200   Day  22: 210   Day  23: 220   Day  24: 230
Day  25: 240
```

Model summary (example.det)

```
GAPS Simulator

Starting simulation on  5/24/1994 at 10:40:10

***** Input file names **********************
Site      :  C:\GAPS3\EXAMPLE.LOC                                    LocationFileNa
me
Soil      :  C:\GAPS3\EXAMPLE.SOL                                    SoilFileName
Climate   :  C:\GAPS3\EXAMPLE.CLI                                    ClimateFileNam
es[]
Plant 1   :  C:\GAPS3\EXAMPLE.PLT                                    PlantFileNames
[]
Save      :  C:\GAPS3\EXAMPLE.SAV                                    SaveFileName

***** Output file names **********************
Summary      : C:\GAPS3\EXAMPLE.SUM                                  DetFileName
Model        : C:\GAPS3\EXAMPLE.DET                                  SumFileName
Layer flux   : C:\GAPS3\EXAMPLE.LAY                                  LayFileName
Soil flux    : C:\GAPS3\EXAMPLE.SOF                                  SofFileName
Plant flux   : C:\GAPS3\EXAMPLE.PLF                                  PlfFileName
Climate flux: C:\GAPS3\EXAMPLE.CLF                                   ClfFileName

***** Simulation procedures ******************

Soil temperature     : not simulated
                                                                     SoilTem
p_method
Evapo-transpiration : Priestley-Taylor (Priestley_Taylor_ETP)
                                                                     ETP_met
hod
Surface runoff       : not simulated
                                                                     Runoff_
method
Soil water flow      : Tipping Bucket (Tipping_Bucket)
                                                                     Flow_me
thod
Plant water uptake   : Plant-available water (SimpleWaterUptake)
                                                                     Uptake_
method
Field hours          : not simulated
                                                                     Do_Fiel
dHours

Maize variation of Stockle-Riha crop model
Stockle-Riha crop model
Model options: > Yield


***** Beginning simulation year  1 *************                     TheYear

First day               :      1                                     FirstDay
Last day                :    365                                     LastDay

Maize variation of Stockle-Riha crop model
Stockle-Riha crop model
Model options: > Yield
>> Crop planted, beginning growth on real day 123 (= elapsed day 0)
>> Crop emerged on real day 138 (= elapsed day 15)
```

```
>> Crop silked on real day 221 (= elapsed day 98)
>> Crop began late lag on real day 234 (= elapsed day 111)
>> Crop began grain fill on real day 237 (= elapsed day 1)
>> Crop senesced on real day 251 (= elapsed day 128)
>> Crop matured on real day 268 (= elapsed day 145)
>> Crop harvested on real day 296 (= elapsed day 173)

***** generic crop model summary *************
Accumulated top dry matter (kg/m2)    :     2.05
                                                                        AccTopDryMatte
r
Accumulated root dry matter (kg/m2)   :     0.25
                                                                        AccRootDryMatt
er
Yield (kg/m2)                         :     0.84                        AccGrainYield
Grain moisture ( % )                  :    22.91                        GrainMoisture
Accumulated stress index              :    19.81                        AccStressIndex

Acc. Degree Days from Emergence to Maturity:    62.14                   AccDDM

Maize variation of Stockle-Riha crop model
Stockle-Riha crop model
Model options: > Yield

Acc. degree days from emergence to maturity :    62.14

****** summary water budget (kg/m2 or mm) *****
Total Precipitation             :  586                                  AccPrecip
Total Runoff                    :    0                                  AccRunoff
Total Water Input to Surface    :  586                                  AccWI
Total Potential ET              : 1319                                  AccETP
Total Potential Transpiration   :  420                                  AccPotTrans
Total Actual Transpiration      :  296                                  AccActTrans
Total Potential Evaporation     :  899                                  AccPotEva
Total Actual Soil Evaporation   :  262                                  AccActEva
Accumulated Deep Drainage       :   64                                  AccDrain
Initially in profile            :  245                                  InitialWater
Finally in profile              :  184                                  FinalWater
Change in Storage               :  -61                                  FinalWater -
InitialWater

**********************************************
Ending simulation on  5/24/1994 at 10:40:49
```

Daily summary (example.sum)

1	0.64	0.64	0.64	0.00	0.00	0.00	-0.64	0.00	245	131	0.00	50.01	0.010	0.010	0.000	0.000
10	0.98	0.98	0.98	0.00	0.00	0.00	-0.98	0.00	242	129	0.00	59.12	0.010	0.010	0.000	0.000
20	0.81	0.81	0.81	0.00	0.00	0.00	-0.81	0.00	241	128	0.00	68.94	0.010	0.010	0.000	0.000
30	0.86	0.86	0.86	0.00	0.00	3.62	-4.48	0.00	246	132	0.00	67.33	0.010	0.010	0.000	0.000
40	1.24	1.24	0.00	0.00	0.00	0.00	-0.00	0.00	240	126	0.00	81.05	0.010	0.010	0.000	0.000
50	1.68	1.68	0.00	0.00	0.00	0.00	-0.00	0.00	240	126	0.00	106.38	0.010	0.010	0.000	0.000
60	1.32	1.32	0.00	0.00	0.00	0.00	-0.00	0.00	240	126	0.00	108.07	0.010	0.010	0.000	0.000
70	2.67	2.67	2.67	0.00	0.00	0.12	-1.04	1.75	243	130	0.00	119.23	0.010	0.010	0.000	0.000
80	4.10	4.10	0.00	0.00	0.00	0.00	-0.00	0.00	240	126	0.00	177.89	0.010	0.010	0.000	0.000
90	4.66	4.66	2.18	0.00	0.00	3.36	-5.55	0.00	240	127	0.00	195.00	0.010	0.010	0.000	0.000
100	4.38	4.38	1.87	0.00	0.00	0.00	0.62	2.49	240	126	0.00	186.65	0.010	0.010	0.000	0.000
110	2.30	2.30	2.30	0.00	0.00	0.00	4.25	6.55	244	130	0.00	102.20	0.010	0.010	0.000	0.000
120	10.96	10.96	0.00	0.00	0.00	0.00	-0.00	0.00	240	126	0.00	329.74	0.010	0.010	0.000	0.000
130	6.35	6.35	5.96	0.00	0.00	0.00	-0.82	5.14	241	127	0.00	237.72	0.010	0.010	0.000	0.000
140	9.70	9.06	0.00	0.64	0.64	0.10	-0.74	0.00	238	125	0.00	291.89	0.013	0.012	0.001	0.000
150	7.97	7.26	0.00	0.71	0.71	0.00	-0.71	0.00	237	124	0.00	268.66	0.019	0.016	0.003	0.000
160	6.46	5.55	0.32	0.91	0.91	0.00	-0.84	0.38	230	117	0.00	214.91	0.034	0.026	0.007	0.000
170	12.62	8.58	0.00	4.04	4.04	0.00	-4.04	0.00	211	98	0.00	363.09	0.092	0.072	0.020	0.000
180	5.16	1.92	0.00	3.24	3.24	0.00	-3.24	0.00	181	68	0.00	181.52	0.217	0.175	0.042	0.000

```
190   6.37   0.67   0.67   5.70   5.70   0.00 -6.37   0.00   184    71    0.00   202.43 0.521 0.436 0.085 0.000
200   5.28   0.28   0.28   5.01   3.64   0.00 -3.92   0.00   219   105    0.00   167.72 0.864 0.739 0.124 0.000
210   4.20   0.19   0.19   4.01   2.91   0.00 -3.11   0.00   176    63    0.00   132.43 1.353 1.180 0.173 0.000
220   4.55   0.19   0.19   4.36   4.36   0.00 -4.55   0.00   154    40    0.00   140.00 1.781 1.570 0.210 0.000
230   6.05   0.31   0.31   5.74   2.89   0.00 -3.20   0.00   170    56    0.00   200.34 1.991 1.764 0.227 0.000
240   3.86   0.31   0.31   3.55   1.79   0.00 -2.10   0.00   175    61    0.00   117.49 2.177 1.937 0.240 0.108

 1     2      3      4      5      6      7      8      9     10    11     12     13     14     15
```

```
1  real_day              7  sumDrain             13  SumNetRad            19  GDD
2  sim.PoTET             8  Sumwcchange          14  AccTotalDryMatter    20  LAI
3  sumPotEV              9  SumWI                15  AccTopDryMatter      21  RootingD
4  sumev                10  ProfileWater         16  AccRootDryMatter     22  AccStres
5  sumPotTrans          11  ProfileAvail         17  AccYield             23  SumPS
6  sumActTrans          12  NetHeatFlux          18  AccDD                24  SumField
```

Soil layer (example.lay)

```
3                 soil.LastLayer
 97.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
 98.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
 99.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
100.50   0.400 0.373 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
101.50   0.445 0.403 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
102.50   0.426 0.417 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
103.50   0.163 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
104.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
105.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
106.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
107.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
108.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
109.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.
110.50   0.050 0.324 -17 -17 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 0.0E+0000 20.0 20.0   0.

  1     <    2    ><   3  ><        4         ><        5         ><         6        ><   7     ><
```

```
1  decimal_day                          6  sim.WUptake[MinLayer to LastLayer]
2  sim.WN[MinLayer to LastLayer]        7  sim.SoilTemp[MinLayer to LastLayer]
3  sim.WP[MinLayer to LastLayer]        8  sim.AccLayerRtBio[MinLayer to LastLayer]
4  sim.K[MinLayer to LastLayer]
5  sim.WFlux[MinLayer to LastLayer]
```

Soil flux (example.sof)

```
  1.50 0.0000344 0.0000344 0.0000344 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000344  245  131
 10.50 0.0000545 0.0000545 0.0000545 0.0000545 0.0000000 0.0000000 0.0000000 -0.0000545  243  130
 20.50 0.0000434 0.0000434 0.0000434 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000434  242  128
 30.50 0.0000409 0.0000409 0.0000409 0.0000000 0.0000000 0.0000434 0.0000000 -0.0000844  248  134
 40.50 0.0000571 0.0000571 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000000  240  126
 50.50 0.0000773 0.0000773 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000000  240  126
 60.50 0.0000563 0.0000563 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000000  240  126
 70.50 0.0001052 0.0001052 0.0001052 0.0000000 0.0000000 0.0000008 0.0000212 -0.0000849  244  130
 80.50 0.0001621 0.0001621 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000000  240  126
 90.50 0.0001797 0.0001797 0.0001029 0.0000000 0.0000000 0.0000338 0.0000000 -0.0001366  241  127
100.50 0.0001593 0.0001593 0.0000301 0.0000000 0.0000000 0.0000000 0.0000301 -0.0000000  240  126
110.50 0.0000783 0.0000783 0.0000783 0.0000000 0.0000000 0.0000000 0.0000792  0.0000009  242  128
```

```
120.50 0.0003694 0.0003694 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 -0.0000000  240  126
130.50 0.0002115 0.0002115 0.0002115 0.0000000 0.0000000 0.0000000 0.0000620 -0.0001495  241  127
140.50 0.0003149 0.0002942 0.0000000 0.0000207 0.0000207 0.0000006 0.0000000 -0.0000213  239  126
150.50 0.0002550 0.0002323 0.0000000 0.0000227 0.0000227 0.0000000 0.0000000 -0.0000227  238  124
160.50 0.0001969 0.0001691 0.0000046 0.0000278 0.0000278 0.0000000 0.0000046 -0.0000278  230  117
170.50 0.0003917 0.0002662 0.0000000 0.0001255 0.0001255 0.0000000 0.0000000 -0.0001255  213  100
180.50 0.0001633 0.0000607 0.0000000 0.0001027 0.0001027 0.0000000 0.0000000 -0.0001027  182   69
190.50 0.0002077 0.0000217 0.0000217 0.0001860 0.0001860 0.0000000 0.0000000 -0.0002077  187   74
200.50 0.0001697 0.0000089 0.0000089 0.0001609 0.0001170 0.0000000 0.0000000 -0.0001258  221  107
210.50 0.0001311 0.0000060 0.0000060 0.0001251 0.0000910 0.0000000 0.0000000 -0.0000970  178   65
220.50 0.0001482 0.0000061 0.0000061 0.0001421 0.0001421 0.0000000 0.0000000 -0.0001482  156   42
230.50 0.0002116 0.0000109 0.0000109 0.0002007 0.0001010 0.0000000 0.0000000 -0.0001119  171   57
240.50 0.0001311 0.0000104 0.0000104 0.0001207 0.0000607 0.0000000 0.0000000 -0.0000712  175   62

   1        2         3         4         5         6         7         8          9      10   11

1  decimal_day                              7  Drain_rate
2  sim.ETP                                  8  WIn
3  sim.PotEva                               9  WC_change
4  sim.ActEva                              10  ProfileWater
5  sim.PotTrans                            11  ProfileAvail
6  sim.ActTrans
```

Plant flux (example.plf)

```
  1.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 10.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 20.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 30.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 40.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 50.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 60.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 70.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 80.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
 90.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
100.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
110.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
120.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
130.50 0.0000000 0.000000 -1500     0   0.00     0 1.00 0.00 0.00
140.50 0.0000207 0.000021 -1500     0   0.39  3200 1.00 0.00 0.00
150.50 0.0000227 0.000023 -1500     0   0.35  3200 1.00 0.00 0.00
160.50 0.0000278 0.000028 -1500     0   0.56  3200 1.00 0.00 0.00
170.50 0.0001255 0.000126 -1500     0   2.33  3200 1.00 0.00 0.00
180.50 0.0001027 0.000103 -1500     0   2.72  3200 1.00 0.00 0.00
190.50 0.0001860 0.000186 -1500     0  10.00  3200 1.00 0.00 0.00
200.50 0.0001609 0.000117 -1500     0  11.11  3200 0.73 0.27 0.00
210.50 0.0001251 0.000091 -1500     0  11.61  3200 0.73 0.27 0.00
220.50 0.0001421 0.000142 -1500     0  17.97  3200 1.00 0.00 0.00
230.50 0.0002007 0.000101 -1500     0   3.40  3200 0.50 0.49 0.83
240.50 0.0001207 0.000061 -1500     0   7.04  3200 0.50 0.49 0.83

   1       2        3      4      5    6    7    8    9    10

1  decimal_day                              6  PS
2  sim.PotTrans                             7  NonstressRes
3  sim.ActTrans                             8  PSStressFac
4  sim.RootWP                               9  1-PSStressFac
5  leafWP                                  10  LAIStressFac
```

Climate flux (example.clf)

```
  1.50 0.0000344 0.0000344 0.0000000    0.00 -7.3 230.1
 10.50 0.0000545 0.0000545 0.0000000    0.00 -1.2 280.0
 20.50 0.0000434 0.0000434 0.0000000    0.00 -9.0 314.6
 30.50 0.0000409 0.0000409 0.0000000    0.00 -7.2 272.1
 40.50 0.0000571 0.0000000 0.0000000    0.00 -3.1 317.3
 50.50 0.0000773 0.0000000 0.0000000    0.00 -2.6 420.9
 60.50 0.0000563 0.0000000 0.0000000    0.00 -7.9 385.8
 70.50 0.0001052 0.0001052 0.0000000    0.00  7.8 399.6
 80.50 0.0001621 0.0000000 0.0000000    0.00  9.0 595.6
 90.50 0.0001797 0.0001029 0.0000000    0.00 10.4 636.7
100.50 0.0001593 0.0000301 0.0000000    0.00 10.0 569.5
110.50 0.0000783 0.0000783 0.0000000    0.00  8.5 291.4
120.50 0.0003694 0.0000000 0.0000000    0.00 28.9 940.4
130.50 0.0002115 0.0002115 0.0000000    0.00 15.2 669.7
140.50 0.0003149 0.0000000 0.0000207    0.00 28.7 802.8
150.50 0.0002550 0.0000000 0.0000227    0.00 21.1 723.0
160.50 0.0001969 0.0000046 0.0000278    0.00 21.4 555.2
170.50 0.0003917 0.0000000 0.0001255    0.00 32.9 955.0
180.50 0.0001633 0.0000000 0.0001027    0.00 18.0 489.1
190.50 0.0002077 0.0000217 0.0001860    0.00 24.8 556.7
200.50 0.0001697 0.0000089 0.0001170    0.00 24.7 455.5
210.50 0.0001311 0.0000060 0.0000910    0.00 25.0 350.8
220.50 0.0001482 0.0000061 0.0001421    0.00 26.2 389.9
230.50 0.0002116 0.0000109 0.0001010    0.00 22.0 591.5
240.50 0.0001311 0.0000104 0.0000607    0.00 27.5 339.1

   1        2          3          4        5    6   7

1  decimal_day
2  sim.ETP
3  sim.ActEva
4  sim.ActTrans
5  sim.Heatflux
6  AirTemp
7  sim.NetRad
```

Printout of output files

Model summary (id. example.det)

Daily summary (daily.txt)

```
SUM Data File Name : C:\GAPS3\EXAMPLE.SUM
Date               : 5/24/1994
Time               : 10:42:59
```

Day	ETP	PotEV	AcEV	PTran	ATran	Drain	WCC	WIn	Wat_P	AvailP	HeatF	NetRad	TotDM	TopDM	RootDM	Yield	ACC
	(---------- mm/day ----------)							(--- mm ---)			(-- W/m2 --)		(---- kg/m2 -----)				
1	0.6	0.6	0.6	0.0	0.0	0.0	-0.6	0.0	245	131	0.0	50.0	0.01	0.01	0.00	0.00	0.
10	1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	242	129	0.0	59.1	0.01	0.01	0.00	0.00	0.
20	0.8	0.8	0.8	0.0	0.0	0.0	-0.8	0.0	241	128	0.0	68.9	0.01	0.01	0.00	0.00	0.
30	0.9	0.9	0.9	0.0	0.0	3.6	-4.5	0.0	246	132	0.0	67.3	0.01	0.01	0.00	0.00	0.
40	1.2	1.2	0.0	0.0	0.0	0.0	-0.0	0.0	240	126	0.0	81.1	0.01	0.01	0.00	0.00	0.
50	1.7	1.7	0.0	0.0	0.0	0.0	-0.0	0.0	240	126	0.0	106.4	0.01	0.01	0.00	0.00	0.
60	1.3	1.3	0.0	0.0	0.0	0.0	-0.0	0.0	240	126	0.0	108.1	0.01	0.01	0.00	0.00	0.
70	2.7	2.7	2.7	0.0	0.0	0.1	-1.0	1.8	243	130	0.0	119.2	0.01	0.01	0.00	0.00	0.
80	4.1	4.1	0.0	0.0	0.0	0.0	-0.0	0.0	240	126	0.0	177.9	0.01	0.01	0.00	0.00	0.
90	4.7	4.7	2.2	0.0	0.0	3.4	-5.6	0.0	240	127	0.0	195.0	0.01	0.01	0.00	0.00	0.
100	4.4	4.4	1.9	0.0	0.0	0.0	0.6	2.5	240	126	0.0	186.6	0.01	0.01	0.00	0.00	0.
110	2.3	2.3	2.3	0.0	0.0	0.0	4.3	6.6	244	130	0.0	102.2	0.01	0.01	0.00	0.00	0.
120	11.0	11.0	0.0	0.0	0.0	0.0	-0.0	0.0	240	126	0.0	329.7	0.01	0.01	0.00	0.00	0.
130	6.3	6.3	6.0	0.0	0.0	0.0	-0.8	5.1	241	127	0.0	237.7	0.01	0.01	0.00	0.00	1.
140	9.7	9.1	0.0	0.6	0.6	0.1	-0.7	0.0	238	125	0.0	291.9	0.01	0.01	0.00	0.00	4.
150	8.0	7.3	0.0	0.7	0.7	0.0	-0.7	0.0	237	124	0.0	268.7	0.02	0.02	0.00	0.00	6.
160	6.5	5.6	0.3	0.9	0.9	0.0	-0.8	0.4	230	117	0.0	214.9	0.03	0.03	0.01	0.00	8.
170	12.6	8.6	0.0	4.0	4.0	0.0	-4.0	0.0	211	98	0.0	363.1	0.09	0.07	0.02	0.00	14.
180	5.2	1.9	0.0	3.2	3.2	0.0	-3.2	0.0	181	68	0.0	181.5	0.22	0.17	0.04	0.00	19.
190	6.4	0.7	0.7	5.7	5.7	0.0	-6.4	0.0	184	71	0.0	202.4	0.52	0.44	0.09	0.00	23.
200	5.3	0.3	0.3	5.0	3.6	0.0	-3.9	0.0	219	105	0.0	167.7	0.86	0.74	0.12	0.00	29.
210	4.2	0.2	0.2	4.0	2.9	0.0	-3.1	0.0	176	63	0.0	132.4	1.35	1.18	0.17	0.00	35.
220	4.6	0.2	0.2	4.4	4.4	0.0	-4.6	0.0	154	40	0.0	140.0	1.78	1.57	0.21	0.00	40.
230	6.1	0.3	0.3	5.7	2.9	0.0	-3.2	0.0	170	56	0.0	200.3	1.99	1.76	0.23	0.00	46.
240	3.9	0.3	0.3	3.5	1.8	0.0	-2.1	0.0	175	61	0.0	117.5	2.18	1.94	0.24	0.11	52.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

1	day	7	sumDrain	13	SumNetRad	19	GDD
2	sim.PoTET	8	Sumwcchange	14	AccTotalDryMatter	20	LAI
3	sumPotEV	9	SumWI	15	AccTopDryMatter	21	RootingD
4	sumev	10	ProfileWater	16	AccRootDryMatter	22	AccStres
5	sumPotTrans	11	ProfileAvail	17	AccYield	23	SumPS
6	sumActTrans	12	NetHeatFlux	18	AccDD	24	SumField

Soil layer (layer.txt)

```
LAY Data File Name : C:\GAPS3\EXAMPLE.LAY
Date               : 5/24/1994
Time               : 10:43:8
```

Day	Hour	Layer	WN	WP	K	WFlux	WUptake	SoilTemp	Root Biomass
			m3/m3	J/kg	kg s/m3	(----- kg/m2 hr -----)		C	kg/m3
1	12	2	0.245	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	3	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	4	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	5	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	6	0.273	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	7	0.218	-11	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
1	12	8	0.218	-11	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	2	0.043	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	3	0.277	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	4	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	5	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	6	0.273	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	7	0.218	-11	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
10	12	8	0.218	-11	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	2	0.045	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	3	0.244	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	4	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	5	0.283	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	6	0.273	-3	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000
20	12	7	0.218	-11	0.0E+0000	0.0E+0000	0.0E+0000	0.0	0.000

...

(Note that WFlux and WUptake are in units of kg/m2 s in the LAY file.)

1 2 3 4 5 6 7 8 9 10

1 day
2 hour
3 layer
4 sim.WN[layer]
5 sim.WP[layer]
6 sim.K[layer]
*7 sim.WFlux[layer]*3600*
*8 sim.WUptake[layer]*3600*
9 sim.SoilTemp[layer]
10 sim.AccLayerRtBio[layer]

Soil flux (soilfl.txt)

```
SOF Data File Name : C:\GAPS3\EXAMPLE.SOF
Date               : 5/24/1994
Time               : 10:43:25
```

Day	Hour	ETP	PotEv	ActEv	PotTrans	ActTrans	Drain	WIn	WCChg	WaterP	P
		(----------	----------	----------	kg/m2 hr	----------	----------	----------	----------)	mm3	
1	12	0.123	0.123	0.123	0.000	0.000	0.000	0.000	-0.123	245	
10	12	0.196	0.196	0.196	0.000	0.000	0.000	0.000	-0.196	243	
20	12	0.156	0.156	0.156	0.000	0.000	0.000	0.000	-0.156	242	
30	12	0.147	0.147	0.147	0.000	0.000	0.156	0.000	-0.303	248	
40	12	0.205	0.205	0.000	0.000	0.000	0.000	0.000	-0.000	240	
50	12	0.278	0.278	0.000	0.000	0.000	0.000	0.000	-0.000	240	
60	12	0.202	0.202	0.000	0.000	0.000	0.000	0.000	-0.000	240	
70	12	0.378	0.378	0.378	0.000	0.000	0.002	0.076	-0.305	244	
80	12	0.583	0.583	0.000	0.000	0.000	0.000	0.000	-0.000	240	
90	12	0.646	0.646	0.370	0.000	0.000	0.121	0.000	-0.491	241	
100	12	0.573	0.573	0.108	0.000	0.000	0.000	0.108	-0.000	240	
110	12	0.281	0.281	0.281	0.000	0.000	0.000	0.285	0.003	242	
120	12	1.329	1.329	0.000	0.000	0.000	0.000	0.000	-0.000	240	
130	12	0.761	0.761	0.761	0.000	0.000	0.000	0.223	-0.538	241	
140	12	1.133	1.059	0.000	0.074	0.074	0.002	0.000	-0.076	239	
150	12	0.918	0.836	0.000	0.081	0.081	0.000	0.000	-0.081	238	

```
160   12     0.708     0.608     0.016     0.100     0.100     0.000     0.016    -0.100     230
170   12     1.410     0.958     0.000     0.451     0.451     0.000     0.000    -0.451     213
180   12     0.587     0.218     0.000     0.369     0.369     0.000     0.000    -0.369     182
190   12     0.747     0.078     0.078     0.669     0.669     0.000     0.000    -0.747     187
200   12     0.610     0.032     0.032     0.579     0.421     0.000     0.000    -0.452     221
210   12     0.472     0.021     0.021     0.450     0.327     0.000     0.000    -0.349     178
220   12     0.533     0.022     0.022     0.511     0.511     0.000     0.000    -0.533     156
230   12     0.761     0.039     0.039     0.722     0.363     0.000     0.000    -0.402     171
240   12     0.472     0.037     0.037     0.434     0.218     0.000     0.000    -0.256     175

(Note that all water flux densities are in units of kg/m2 s in the SOF file.)

 1     2       3         4         5         6         7         8         9         10       11

1  day                                    7  sim.ActTrans*3600
2  hour                                   8  Drain_rate*3600
3  sim.ETP*3600                           9  WIn*3600
4  sim.PotEva*3600                       10  WC_change*3600
5  sim.ActEv*3600                        11  ProfileWater
6  sim.PotTrans*3600                     12  ProfileAvail
```

Plant flux (plantfl.txt)

```
PLF Data File Name : C:\GAPS3\EXAMPLE.PLF
Date               : 5/24/1994
Time               : 10:43:58

Day  Hour  PotTrans  ActTrans    RootWP    LeafWP       PS      NonStressRes PSStressFac Stress LAIStr
          (---- kg/m2 hr ----)  (----- J/kg -----)   g/m2 hr       s/m

  1   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 10   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 20   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 30   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 40   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 50   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 60   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 70   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 80   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
 90   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
100   12    0.000      0.000      -1500         0       0.00            0        1.00     0.00    0.00
110   12    0.0000     0.0000     -1500         0       0.00            0        1.00     0.00    0.00
120   12    0.0000     0.0000     -1500         0       0.00            0        1.00     0.00    0.00
130   12    0.0000     0.0000     -1500         0       0.00            0        1.00     0.00    0.00
140   12    0.0745     0.0756     -1500         0       0.39         3200        1.00     0.00    0.00
150   12    0.0817     0.0828     -1500         0       0.35         3200        1.00     0.00    0.00
160   12    0.1001     0.1008     -1500         0       0.56         3200        1.00     0.00    0.00
170   12    0.4518     0.4536     -1500         0       2.33         3200        1.00     0.00    0.00
180   12    0.3697     0.3708     -1500         0       2.72         3200        1.00     0.00    0.00
190   12    0.6696     0.6696     -1500         0      10.00         3200        1.00     0.00    0.00
200   12    0.5792     0.4212     -1500         0      11.11         3200        0.73     0.27    0.00
210   12    0.4504     0.3276     -1500         0      11.61         3200        0.73     0.27    0.00
220   12    0.5116     0.5112     -1500         0      17.97         3200        1.00     0.00    0.00
230   12    0.7225     0.3636     -1500         0       3.40         3200        0.50     0.50    0.83
240   12    0.4345     0.2196     -1500         0       7.04         3200        0.50     0.50    0.83

(Note that PotTrans and ActTrans are in units of kg/m2 s in the PLF file.)

 1     2       3         4         5          6         7           8           9       10       11

1  day                                    7  PS * 3600/time_step
2  hour                                   8  NonstressRes
3  sim.PotTrans*3600                      9  PSStressFac
```

*4 sim.ActTrans*3600*
5 sim.RootWP
6 leafWP
10 1-PSStressFac
11 LAIStressFac

Climateflux (climfl.txt)

```
CLF Data File Name : C:\GAPS3\EXAMPLE.CLF
Date               : 5/24/1994
Time               : 10:44:8
```

Day	Hour	ETP	ActEV	ActTrans	HeatFlux	SimNetRad	Airtemp
		(----------	----------	W/m2 ----------	----------	----------)	C
1	12	84.4	84.4	0.0	0.0	230.1	-7.3
10	12	133.7	133.7	0.0	0.0	280.1	-1.2
20	12	106.5	106.5	0.0	0.0	314.7	-9.0
30	12	100.3	100.3	0.0	0.0	272.2	-7.2
40	12	140.1	0.0	0.0	0.0	317.3	-3.1
50	12	189.6	0.0	0.0	0.0	421.0	-2.6
60	12	138.1	0.0	0.0	0.0	385.9	-7.9
70	12	258.1	258.1	0.0	0.0	399.6	7.8
80	12	397.6	0.0	0.0	0.0	595.6	9.0
90	12	440.8	252.4	0.0	0.0	636.8	10.4
100	12	390.8	73.8	0.0	0.0	569.5	10.0
110	12	192.1	192.1	0.0	0.0	291.4	8.5
120	12	906.1	0.0	0.0	0.0	940.5	28.9
130	12	518.8	518.8	0.0	0.0	669.7	15.2
140	12	772.4	0.0	50.8	0.0	802.8	28.7
150	12	625.5	0.0	55.7	0.0	723.0	21.1
160	12	483.0	11.3	68.2	0.0	555.2	21.4
170	12	960.8	0.0	307.9	0.0	955.1	32.9
180	12	400.6	0.0	251.9	0.0	489.2	18.0
190	12	509.5	53.2	456.3	0.0	556.8	24.8
200	12	416.3	21.8	287.0	0.0	455.6	24.7
210	12	321.6	14.7	223.2	0.0	350.8	25.0
220	12	363.5	15.0	348.6	0.0	389.9	26.2
230	12	519.1	26.7	247.8	0.0	591.6	22.0
240	12	321.6	25.5	148.9	0.0	339.2	27.5

(Note that ETP, ActEV, and ActTRans are in units of kg/m2 s in the CLF file.)

```
 1     2      3         4          5           6          7         8
```

1 day
2 hour
*3 sim.ETP*ktw*
*4 sim.ActEva*ktw*
*5 sim.ActTrans*ktw*
6 sim.HeatFlux
7 sim.NetRad
8 AirTemp

(ktw = 2,453,000)