

Daisy Tutorial

Abstract

Welcome to the Daisy tutorial. This document is intended to teach the use of Daisy by example. It is not by any stretch of the imagination a reference to all the features of Daisy, for that see [Abrahamsen, 1999]. It also does not attempt to explain the mathematical and physical models implemented by the Daisy software, for that see [Hansen et al., 1990] and [Hansen, 2002].

I hope that you will be able to get started with Daisy from these examples, and when you need to go beyond will be familiar enough with the system to make good use of the reference manual. Daisy has an amazing number of adjustable dials and wheels, only a tiny fraction of which are explained here. However, they are all at least mentioned in the reference manual, which is mostly automatically generated from the `daisy` executable itself.

If you have questions or suggestions to either the manual or the Daisy system itself, feel free to contact the authors at `<daisy@dina.kvl.dk>`.

Per Abrahamsen, March 29, 2007.

Contents

1	Retrieving and installing the files	3
2	Example	5
2.1	Input files	5
2.2	Weather	5
2.3	Horizons	6
2.4	Column	8
2.4.1	Soil	8
2.4.2	Organic Matter	9
2.4.3	Groundwater	10
2.4.4	Using it	10
2.5	Time	10
2.6	Management	10
2.7	Output	13
3	Weather	15
3.1	Scaled precipitation	15
3.2	Missing years	15

4	Column	16
4.1	Carbon Input	16
4.2	Background mineralization	17
4.3	Hydraulic properties	18
4.4	Groundwater and Drainage	19
4.4.1	Aquitard horizons	20
4.5	Macropores	21
4.5.1	Log models	22
4.6	Evapotranspiration	22
4.6.1	Potential and Reference Evapotranspiration	23
4.7	Discretization	23
4.8	Initialization of soil content	24
4.8.1	Soil water	24
4.8.2	Soil nitrogen	25
4.8.3	Soil temperature	26
5	Management	26
5.1	Crop Managements	26
5.1.1	Rotations	27
5.2	Tillage	28
5.2.1	Trafficability	28
5.3	Sowing and Harvesting	29
5.3.1	Crop Residuals	29
5.3.2	Multiple Cuts	29
5.4	Fertilization	31
5.4.1	Mineral Fertilizer	31
5.4.2	Organic Fertilizer	32
5.4.3	First and Second Year Utilization	33
5.4.4	Precision Fertilization	34
5.4.5	Incorporation	34
5.5	Irrigation	35
5.5.1	Conditions	36
5.5.2	Strategies	36
5.6	Pesticides	38
5.6.1	Defining New Pesticides	38
5.6.2	Canopy	38
5.6.3	Soil Surface and Root Uptake	39
5.6.4	Decomposition	39
5.6.5	Transportation	40
5.6.6	Log models	40

6	Log Files	41
6.1	Activating output	41
6.2	Log models	42
7	Plotting simulation results	43
7.1	Spreadsheets	43
7.2	ShowDaisyOutput	44
7.3	Generating gnuplot commands from Daisy	44
7.4	Simple example	44
7.5	Complex example	44
7.6	The 'arithmetic' source model	46
7.7	Daisy Data Files (*.ddf)	47
7.8	wgnuplot.exe	48
8	Organizing Your Parameterizations	48
8.1	Don't Copy, Derive	49
8.2	DAISYPATH	50
A	Experimental features	51
A.1	Ice	51
A.2	Soil Compaction	51
A.3	Phosphor	51
A.4	Dividing and Merging Fields	51
A.5	Ridging	51
A.6	Forced LAI	52
	A.6.1 Log models	53
B	test.dai	53
C	Bibliography	55

1 Retrieving and installing the files

We start by retrieving the files. With a web browser, go to the address <http://www.dina.kvl.dk/~daisy/>, the home page of the Daisy project. From there, you should download the following items:

Daisy Description For information about the mathematical and physical models in Daisy.

Daisy Tutorial You already got this, but if you are reading this from an old printout, you might want to get a newer version.

Daisy Program Reference Manual Sooner or later, you will need the reference manual. I suggest that you get the PDF version (see under the “Distribution” header), which is both properly formatted (unlike the HTML version), easy to browse, and easy to print.

command line executable Download the version for the platform you use. If you don’t know what I mean by this, you are most likely using “win32”.

parameter library This is a zip file containing containing standard parameterizations of e.g. crops. You will need this. You will also need a program to unzip the file, download it from a place like <http://www.tucows.com/> if you haven’t already got it. Search for unzip. Windows XP already knows how to unzip a file, just open it.

sample parameterizations This is another zip file containing parameterizations that are not intended to use as-is, but to serve as inspiration for your own setup files.

weather file The weather file contains observations from the KVL research station in Taastrup, Denmark. Most likely, you will eventually want weather data from closer to home, but this file will get you started.

log file viewer Daisy stores the results of its simulations in log files (with the extension `.dlf`). `ShowDaisyOutput.exe` is a win32 program that will show the content of these log files graphically, it can even animate some of the data.

We suggest you create a `daisy` directory near the root, for example (under MS Windows¹) “`C:\daisy`”, with the subdirectories “`bin`” (for the `.exe` files), “`doc`” (for the `.doc` and `.pdf` files), “`lib`” (for the files in `daisy-lib.zip` and “`sample`” (for the files in `daisy-sample.zip`).

MS Windows users should then install and configure TextPad as described on the Daisy home page, and open the `test.dai` file in the `sample` directory. You can now start Daisy from the `Tools` menu. If you do that, you will get a window with error messages, basically telling you that Daisy could not find the files it need. In section 8.2 you will find information about how to tell Daisy where the files are.

¹Unix users can use “`/daisy`”

2 Example

The `test.dai` file contains all the information about the simulation. We call such a file a Daisy setup file, or a `.dai` file for short. By convention, we give such files the `.dai` extension, even though Daisy doesn't care. Teaching you to write Daisy setup files is the main purpose of this tutorial.

First, let's examine line by line the content of the `test.dai` file (see appendix B for the full version). The first lines are

```
;;; test.dai -- Sample file using the Daisy libraries.  
  
(description "Simulation for use in tutorial.")
```

Semicolons are used for comments, everything from the semicolon to the end of the line is ignored by Daisy. This can (and should!) be used to put information about the simulation there for the person reading the file. That person may very well be you, who some month (or years) later have forgotten about the purpose of this particular simulation.

For the same reason, providing a description as above is also a good idea. The description will be written to the log files produced by Daisy, allowing you to later see what log files come from what simulation.

2.1 Input files

```
;;; Use standard parameterizations.  
(input file "tillage.dai")  
(input file "crop.dai")  
(input file "log.dai")
```

Here, we tell Daisy to read three additional files: `tillage.dai` containing descriptions of various standard tillage operations, such as plowing and seed bed preparation; `crop.dai` which contains descriptions of crops like spring barley or rye; and `log.dai` with directives for Daisy to produce files with information about various aspects of the simulation, such as the nitrogen or water balance. You can read more about input files in section 8.

2.2 Weather

Daisy need to know what the weather is during the simulation. This is usually read from a file.

```
;; Weather data.
(weather default "weather.dwf")
```

Here we specify where Daisy should find the weather data. Daisy can read weather data in several different formats, the `default` format is `.dwf`, short for *Daisy Weather Format*. The format is specified in an appendix of [Abrahamsen, 1999].

2.3 Horizons

Now we get to main part of the simulation, namely the soil profile, or *column*. In Daisy, the column is a one dimensional description of an agricultural system, with the weather at the top and the groundwater at the bottom. Before coming to that, we need to define two horizons. A horizon is a vertical layer of soil with similar chemical and physical properties. If you make a vertical cut in the soil, there is usually a clear visual distinction between the horizons.

```
;; We have some very sandy soil.
(defhorizon Ap FA03
  "Andeby top soil."
  (clay 8.0 [%])
  (silt 10.5 [%])
  (sand 81.5 [%])
  (humus 1.12 [%])
  (C_per_N 11.0 [g C/g N])
  (dry_bulk_density 1.5 [g/cm^3]))
```

Here, we define a horizon named `Ap` (`defhorizon` is short for “define horizon”). Like the place where we specified the weather, the `FA03` keyword means that we use that model for specifying horizons. The build-in horizon models are named after texture classification systems², `FA03` is the FAO approved classification system, see 27 in the reference manual for a full list of horizon models. At this point, we can only choose one of the build-in models, but that will soon change. The string `"Andeby top soil."` is a description of the horizon, and is a good place to put information about the source of data.

The first three parameters, `clay`, `silt`, `sand` together define the soil texture. The texture affects many aspects of the simulation, for example will

²A texture classification system specifies how to determine whether a particle is clay, silt or sand.

the ammonium adsorption depend on the clay content, and if not otherwise specified, the hydraulic parameters such as soil water retention characteristics are also estimated from the texture, a sandy soil is less capable of storing water than a clay soil.

The fourth soil constituent is the `humus`, which among other things creates the background mineralization, thus providing the soil with nitrogen. The numbers given for the five soil constituents are relative, that is they are weighted to the total. So you can specify them as fractions or percentage or whatever, as you wish.

The `C_per_N` parameter is the C/N ratio of the humus. This affects the background mineralization, a high C/N ratio will lead to low mineralization. If you do not have any idea what the C/N ratio for your soil is, leave out this parameter.

The `dry_bulk_density` is, as the name implies, the density of the dry soil. It affects the hydraulic properties of the soil and the total humus content. If you haven't measured this value, you can use a different function for estimating the hydraulic properties, by replacing it with

```
(hydraulic Cosby_at_al)
```

This pedotransfer function has been developed in America, and does not require knowledge of the dry bulk density. Even if you know the dry bulk density, using `Cosby_et_al` might be a better choice for simulation in that continent. The default pedotransfer function, `hypres`, has been developed based on European (especially North German) soil measurements.

The texture parameters all end with a `[]`. This is the dimension surrounded by square brackets. Since fractions are dimensionless, there is nothing in the brackets. The two last parameters have the dimension gC/gN and g/cm^3 . It is a good idea always to specify the dimension, as this will allow Daisy to check that dimension really is what you think it is. Misunderstandings about dimensions are a very common source of errors. The next horizon is simpler.

```
(defhorizon C Ap
  "Andeby C horizon."
  (humus 0.12 []))
```

Here, instead of `FA03`, we write `Ap`. This is the name of the horizon we just defined, and will cause Daisy to reuse the parameter values in the new horizon named `C` we define here. In particular, we give it the same texture as the `Ap` horizon. However, since the horizon is intended to describe lower soil, we specify less humus.

2.4 Column

Now we have described two horizons, we can describe a column.

```
;; We build the column from the horizons.
(defcolumn Andeby default
  "Data collected by F.Guf at the B.And farm, Andeby, 2002."
  (Soil (horizons (-20 [cm] Ap) (-2.5 [m] C))
        (border 1 [m])
        (MaxRootingDepth 60.0 [cm]))
  (OrganicMatter (init (input 1400 [kg C/ha/y])
                      (root 480 [kg C/ha/y])
                      (end -20 [cm])))
  (Groundwater deep))
```

Here, we name the column `Andeby`, and use the `default` model for Daisy columns³. The string following the `default` again serves as a description, and this time we use it to note who collected the data, from where, and when.

In Daisy, a column consists of many separate subsystems. Most of these have default values, and does thus not need to be specified. However, we need a description of the soil, we need an estimate of the quality (lability) of the humus, and we need to tell Daisy where the groundwater is. See section 4 for examples of other subsystems.

2.4.1 Soil

```
(Soil (horizons (-20 [cm] Ap) (-2.5 [m] C))
```

The first soil parameter, `horizons`, specify that the soil is divided into two horizons, the first span the top 20 cm of the soil, and the second span the soil from there down to 2.5 meters depth. There can be as many horizons as you wish, if you had a third it might be described as below.

```
(horizons (-20 [cm] Ap) (-1 [m] B) (-2.5 [m] C))
```

The B horizon would then span the depth from 20 cm to 1 meter.

You may note that we specify the depth in `[cm]` for the first depth, and `[m]` for the next. If you don't specify a dimension, Daisy will expect

³An alternative at this point would be to use the `inorganic` column model, which does not perform any calculations regarding organic matter, thus saving time.

numbers to be in centimeters, however if you specify the dimension as [m] knows how to convert.

We specify a numerical border at one meter.

```
(border 1 [m])
```

The `border` parameter is optional, but when you are interested in results for a particular interval, you should specify numerical borders at the beginning and end of the interval in order to get exact results. There are automatically numeric borders between the horizons.

The other mandatory soil parameter is `MaxRootingDepth`. This is a soil imposed barrier for the roots, they will not penetrate below that depth. To make writing Daisy setup files fun, as well as to keep you alert, this number is specified as positive depth, unlike the layer endpoints which were specified as negative heights. Daisy will tell you if you get it wrong.

2.4.2 Organic Matter

Knowing the humus content of the soil is not enough, Daisy need know the quality of the organic matter as well. The quality (or lability) of the organic matter affect how much background mineralization you can expect, which is an important part of the total nitrogen balance.

We can estimate the quality of the soil organic matter by comparing the humus content with the typical level of carbon input in the decades before the beginning of the simulation. If the carbon input in the period before the was high compared the general level of humus, we expect that the humus content has been growing, and much of the humus will then be fresh and labile. On the other hand, if the input level is low compared to the humus content, the humus level will be shrinking, and the most labile humus will already be gone.

It is important to understand that we are trying to estimate the quality of the humus that exists in the system at the beginning of the simulation period. Therefore we are interested in how the carbon input levels were before the simulation, not the levels during the period we are simulating. We specify the past input level with the following lines.

```
(OrganicMatter (init (input 1400 [kg C/ha/y])
                    (root 480 [kg C/ha/y])
                    (end -20 [cm])))
```

Here we specify that we want to initialize the organic matter subsystem from the carbon input. The first line tells Daisy that the total yearly carbon

input is 1400 kg/ha. The second line specify that out of these 1400 kg, the 480 kg are dead roots. The third line gives the depth of the plowing layer, here 20 cm. The reason we need the two later values (`root` and `end`) is that we need to know how the carbon input is distributed vertically. Daisy will assume most input is distributed evenly in the plowing layer, but that some of the roots will reach all the way down to the depth specified by `MaxRootingDepth`.

In section 4.1 we describe how to estimate the carbon input, and in section 4.2 how to improve the initialization of the organic matter further.

2.4.3 Groundwater

After specifying the soil and organic matter composition, we must tell Daisy the location of the groundwater. Here, we specify that the groundwater is located so deep that it does not directly affect the simulation. Other options include a constant groundwater table, field drainage, and a variable groundwater table specified in a file. See section 4.4 for examples.

2.4.4 Using it

We have now specified a column, and given it the name `Andeby`. Now we just need to tell Daisy to use this column for the simulation.

```
;; Use it.  
(column Andeby)
```

2.5 Time

After specifying the most complex Daisy parameter, the column, we relax with the simplest parameter.

```
;; Simulation start date.  
(time 1986 12 1 1)
```

As the comment say, that time we start the simulation. The four numbers given are year, month, day and hour in that sequence.

2.6 Management

Back to the complex stuff. Daisy now knows about the piece of land we want to simulate, the weather during the simulation, and when we start. However,

since this is an agricultural system, we also need to tell Daisy about the what management operations the farmer perform during the simulation.

How to approach this problem depends on whether we are simulating a real, already performed, experiment, or if we want to simulate a hypothetical situation. The difference is that in the first case, we (hopefully) *know* what the manager did, and when. In the second case, we need to build in some intelligence in the system, so the actions of the hypothetical manager depend on the state of the system. E.g. the manager is likely to wait for a crop to become ripe before harvesting it.

In this example, we do both.

```
(manager activity
  (wait (at 1987 3 20 1))
  (plowing)
  (wait (at 1987 4 4 1))
  (fertilize (mineral
              (weight 100.0 [kg N/ha])
              (NH4_fraction 0.5 [])))
  (wait (at 1987 4 5 1))
  (progn
    (sow "Grass")
    (sow "Spring Barley"))
  (wait (or (crop_ds_after "Spring Barley" 2.0)
            (at 1987 9 5 1)))
  (harvest "Spring Barley")
  (wait (at 1987 9 8 1))
  (fertilize (mineral
              (weight 80.0 [kg N/ha])
              (NH4_fraction 0.5 [])))
  (wait (at 1987 10 10 1))
  (harvest "Grass"
           (stub 8.0 [cm])           ;Leave 8 cm stub.
           (stem 1.00 []))         ;Harvest everything above stub.
  (wait (at 1988 4 1 1))
  (stop))
```

First, activity is an order to perform the specified actions in sequence. When one action is done, we start the next. Half of our actions consist of waiting for some condition to come true. The most common condition is that we have reached a specific point in time. For examples, the first line

```
(wait (at 1987 3 20 1))
```

mean wait until the simulation time is exactly 1:00 AM, March the 20th, 1987. Our specific action is `wait` and the condition is `at`.

The next action is `plowing`. This happens one hour later, so our poor farmer is out in the field plowing at 2:00 AM. This activity last (in the simulation) exactly one hour, after which he can go to the next activity, which involves waiting once again.

I'll describe the actions used for this sample manager here, more examples of management actions can be found in section 5.

```
(fertilize (mineral
            (weight 100.0 [kg N/ha])
            (NH4_fraction 0.5 [])))
```

This is an order to fertilize. The fertilization type is `mineral`, and the amount of nitrogen is specified with the `weight` parameter. The nitrogen can be in the form of ammonium (NH_4^+) or nitrate (NO_3^-). We specify with the `NH4_fraction` parameter that half the nitrogen is ammonium, and the rest is nitrate. Read more about fertilizer types in section 5.4.

```
(progn
  (sow "Grass")
  (sow "Spring Barley"))
```

Remember that `activity` was a directive to perform a list of actions in sequence? Well, `progn` is a directive to perform a list of actions in parallel. Here we sow two crops at once. The crops, `Grass` and `Spring Barley` are defined in the file `crops.dai`, which we included in section 2.1.

```
(wait (or (crop_ds_after "Spring Barley" 2.0)
          (at 1987 9 5 1)))
```

This is the only place we wait for a condition more complex than a specific point in time. Actually, there are three conditions here.

```
(at 1987 9 5 1)
```

is just a point in time, as you have seen before, however

```
(crop_ds_after "Spring Barley" 2.0)
```

is new. It is true when the development stage of a crop named “Spring Barley” is 2.0 or above. The development stage, or DS, is a key concept in describing the state of a crop. It starts at -1.0 when sowed, and increase with time. When it reached 0.0, the crop emerges. At 1.0 it flowers, and at 2.0 it is ripe. So this condition is true when there is a ripe spring barley on the field.

The third conditions is named `or`.

```
(or (crop_ds_after "Spring Barley" 2.0)
    (at 1987 9 5 1))
```

It is true when any of the specified conditions are true. I.e. it is true when there is a ripe spring barley on the field, or when the simulation has reached 1:00 AM, September the 5th, 1987.

In effect, we wait for the spring barley to get ripe, but if it isn’t ripe at the specified date, we stop waiting anyway

```
(harvest "Spring Barley")
```

and harvest it. This harvest is pretty brutal, we remove everything above the soil, and leave only the roots to rot in the soil. When we later cut the grass, we are more gentle.

```
(harvest "Grass"
  (stub 8.0 [cm])           ;Leave 8 cm stub.
  (stem 1.00 []))         ;Harvest everything above stub.
```

Here, we leave 8 cm stub. We still remove all the stem above that height, the parameter to `stem` is the fraction to remove. If we remove a smaller fraction, the remainder will stay on top of the soil as plant residuals, which will be incorporated into the soil either gradually by earth worms and other insects, or at once by a tillage operation.

The grass will most likely survive this cut, unlike the spring barley.

The last action, `stop`, is simply a command to stop the simulation.

2.7 Output

Now Daisy know everything it needs to know in order to run the simulation. We just need to tell it what information about the simulation is should save for later processing.

```

;; Create these log files.
(output harvest
  ("N Balance" (when monthly)
    (from 0 [m]) (to -1 [m]))
  ("Crop Production"
    (crop "Spring Barley")
    (where "sbarley.dlf"))
  (checkpoint (when (at 1987 8 7 6))))

;;; test.dai ends here.

```

The first entry in the list, `harvest`, produces a `harvest.dlf` file containing information about everything that has been harvested during the simulation. The *Daisy Log Format* (.dlf) is understood by the `ShowDaisyOutput.exe` program, but you can also view it with an editor or load it into a spreadsheet. The `harvest.dlf` file, in particular, is best viewed as plain text. See section 7 for more information about how to view the results.

Our next log model, `N Balance`, will write enough information about the nitrogen content of the soil to calculate a balance, in a file name `n_balance.dlf`. By default, it will write this information once a day, but by specifying `monthly` we reduce this to once a month. We also specify the soil layer to log to be the top meter of the soil. By default, it will log the nitrogen content of all the soil down to and including the last numeric layer.

We have two crops simultaneously on the soil. By default, `Crop Production` will add the content of both crops together, and log it in a `crop-prod.dlf` file. The weird looking

```
(crop "Spring Barley")
```

line tells Daisy to only log information about the spring barley, and by specifying a `where` parameter we tell it to store the information in a file named `sbarley.dlf` instead.

Note that writing these log files tend to be much slower than actually running the simulation, to the more information you tell Daisy to log, the slower the simulation will run.

Finally, we tell Daisy To create a checkpoint file at 6:00 AM, August the 7th, 1987. This file will be named `checkpoint-1987-8-7+6.dai`. As the extension hints, this is not a log file, but a setup file containing all information about the current state of the simulation. In fact, you can restart the simulation by typing

```
daisy checkpoint-1987-8-7+6.dai
```

You can also browse it with a text editor to get detailed information about the state of the simulation at that point in time.

See section 6 for more information about log models.

3 Weather

It is common not to have sufficient weather data, either because there is no access to data from a nearby weather station, or because you are running a predictive simulating, and the data isn't available yet. Daisy has two features for handling such cases with available data.

3.1 Scaled precipitation

Precipitation is very locale dependent, it may vary even within a region with similar temperature and radiation. If you know the general monthly difference between a specific locale, and a regional weather station, you can use the `PrecipScale` attribute:

```
(weather default "weather.dwf"  
  (PrecipScale 1.1 1.1 1.1 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2))
```

The `PrecipScale` needs 12 numbers, which will be multiplied to all precipitation the corresponding month. In this example, we use the data from “weather.dwf”, but with 10% more precipitation from January to March, and 20% more the remaining 9 months.

3.2 Missing years

The `missing_years` attribute allows you to reuse the weather data for additional years. For example, if you have weather data for 1991 to 2000 in a file named “weather.dlf”, but want to run a simulation from 1987 to 2001, you can specify the weather model like this.

```
(weather default "weather.dwf"  
  (missing_years ((1987 1990) (1993 1996))  
    ((2001 2001) (1991 1991))))
```

This should be read as an order to Daisy to, in the period from 1987 to 1990, use the weather data from 1993 to 1996, and for 2001 use the weather data from 1991.

This is particularly useful for running Daisy with a “warming up” period where we don’t have actual measurements, and for predictive simulations, where you can run a crop rotation with many different weather combinations.

In the example above we map two periods (1987–1990 and 2001), but you can have as many periods as you like, so you can e.g. run a 100 year simulation with just 10 years worth of climate data.

Note that Daisy will always use the actual data when available, so if you in the example above have some data from 2001 but not all, Daisy will not switch to 1991 data before it have used up the real data.

4 Column

In this section we give examples of common customization of the various subsystems of the column.

4.1 Carbon Input

You can use Daisy to estimate the carbon input used daisy for organic matter initialization (see section 2.4.2) with the following steps.

1. Create a crop rotation you believe is typical for the period before the simulation start.
2. You use the same `Soil` and `Groundwater` parameters as you will for the real simulation, but omit the `OrganicMatter` initialization.
3. Add "`Carbon Balance`" to the output.
4. Run a simulation with that rotation repeated a couple of times, using meteorological data from the location.

The `daisy.log` file will now contain lines that looks like these near the end.

```
Fertilizer =    0 [kg C/ha/y]
Residuals  = 1454 [kg C/ha/y]
  Roots    =  651 [kg C/ha/y]
Bioinc_CO2 = -727 [kg C/ha/y]
-----
Total C input = 1378 [kg C/ha/y]
=====
```

The two values you need are “Total C input”, which should be used for the `input` parameter, and “Roots”, which should be used for the `root` parameter. You can now add an `OrganicMatter` initialization section, as in section 2.4.2, and run the setup again. The total C input should now be slightly different in the second run. Use this number for the real simulation.

4.2 Background mineralization

Having Daisy estimate the organic matter quality is a poor substitution for measuring it. While measuring it directly is hard, it is possible to measure the background mineralization indirectly.

The background mineralization is here defined as the difference between the netto mineralization and the amount of added organically bound nitrogen. The number can be both positive and negative. A positive number means that we get more nitrogen from mineralization than we add to the system. This nitrogen must come from the humus content, which must then be decreasing for the balance to add up. Similarly, a negative background mineralization means that some surplus nitrogen is stored in a growing humus content.

If you look in the `daisy.log` file, you will find a line like this

```
Background mineralization: 4.8127 [kg N/ha/y].
```

near the beginning. This specifies that Daisy expects 5 kg N/ha to be released from the humus every year for the current setup. By adding the parameter `background_mineralization` to the organic matter initialization, we tell Daisy to attempt to initialize the system differently, to produce the specified level of background mineralization.

```
(OrganicMatter (init (input 2400 [kg C/ha/y])
                    (root 800 [kg C/ha/y])
                    (background_mineralization 30 [kg N/ha/y])
                    (end -20.0 [cm])))
```

Here, we specify that with the given input levels, we want 30 kg N/ha to be released from the humus every year. It is not certain Daisy will be able to fulfil that though, there are a number of sanity checks build in which will limit the background mineralization can be generated. You can find the expected background mineralization in the beginning of the `daisy.log` file as before. And the *actual* background mineralization may be very different from this, for two reasons.

1. The actual input levels in the simulation period can be different from the input levels used for the organic matter initialization.
2. The expected background mineralization is calculated by assuming the soil moisture and temperature are constant, while the simulated background mineralization depends on the variable simulated soil moisture and temperature.

The simulated background mineralization can be found by looking at the change in the organic nitrogen content (the SOM column in the "N Balance" log).

If you measure how much nitrogen is removed by harvest, you can adjust the `background_mineralization` parameter until the simulation gives you the same results. However, this only works if the background mineralization really is the only unknown. Specifically, you must be sure that the water balance is correct, otherwise the amount of nitrogen lost in leaching will also be an unknown. And you must be sure that the crop is never nitrogen saturated. Preferably, you should have an unfertilized field, but if you do fertilize it must be less than the crop is capable of taking up.

It is possible to calibrate the background mineralization from dry matter harvest numbers, but only if you are sure the crop growth has been nitrogen limited.

4.3 Hydraulic properties

In section 2.3, we let Daisy guess the hydraulic properties from the texture. This kind of estimates (wild guessing) are unreliable at best, and since the hydraulic properties are the key most other aspects of the simulation depend on, getting them right should be a priority. Daisy supports a number of models for hydraulic properties, the most important are listed here, more can be found in the reference manual [Abrahamsen, 1999].

`B_BaC` Brooks and Corey retention curve model with Burdine theory.

`B_C` Campbell retention curve model with Burdine theory.

`B_vG` van Genuchten retention curve model with Burdine theory.

`M_BaC` Brooks and Corey retention curve model with Mualem theory.

`M_C` Campbell retention curve model with Mualem theory.

`M_vG` van Genuchten retention curve model with Mualem theory.

Here is an example where we specify the van Genuchten retention curve model with Mualem theory

```
(defhorizon B ISSS4
  (clay 8.0) (silt 10.5) (coarse_sand 65) (fine_sand 16.5)
  (humus 1.12)
  (hydraulic M_vG
    (K_sat 10 [cm/h])
    (Theta_res 5 [%])
    (Theta_sat 42.4 [%])
    (alpha 0.069 [cm-1])
    (n 1.527)))
```

The three parameters

K_sat Water conductivity of saturated soil.

Theta_sat Saturation point (a.k.a. soil porosity).

Theta_res Soil residual water.

are shared by all the models, except that **Theta_res** must be zero for Campbell. The other parameters are model specific.

4.4 Groundwater and Drainage

Obviously, the groundwater is not always located “far below” the root zone, as specified by

```
(defcolumn Andeby default
  (Groundwater deep)
  ;; Other parameters...
```

in the example. We can therefore also specify a fixed groundwater table as in

```
(Groundwater fixed -100 [cm])
```

where we specify groundwater in one meters depth. Note that if we specify a fixed groundwater table, it must be higher than our lowest node. A third common case is soil drainage

```
(Groundwater pipe)
```

which is often enough for drained soil. A full specification would be

```
(Groundwater pipe
  (L 18 [m])
  (x 9 [m])
  (pipe_position -1.1 [m])
  (K_aquitard 1e-4 [cm/h])
  (Z_aquitard 2 [m])
  (h_aquifer 2 [m]))
```

where

L Distance between pipes.

x Horizontal distance to nearest pipe. By default, this is $1/2L$, i.e. in the middle between the pipes.

pipe_position Height pipes are placed in the soil.

K_aquitard Conductivity of the aquitard.

Z_aquitard Thickness of the aquitard.

h_aquifer Pressure potential in the aquifer below the aquitard. By default this is equal to **Z_aquitard**.

The above numbers are the default values, i.e. what you get if you don't specify anything. **L** and **pipe_position** are standard values for Danish fields. The aquitard is water blocking layer explained in [Hansen, 2002], the **K_aquitard** parameter is what you usually want to calibrate.

Note: When using drained soil, all numerical nodes will contribute to the drain sink term, not just the node containing the pipe. This is because the pipe is assumed to be at some distance from our node (as specified by the **x** or **L** parameter), and the sink term is actually the simulated horizontal movement of water towards the pipe.

4.4.1 Aquitard horizons

For some soils the groundwater table can fall into the aquitard on dry years. For this reason, Daisy will automatically add an extra horizon below the bottom horizon you specified. This will affect logging, if you don't want to log the aquitard horizon you must explicitly set the **to** parameter.

4.5 Macropores

By default, Daisy will only calculate water flow through the matrix, not through macropores. This is usually adequate, since the vast amount of water travel through the matrix. However, when calculating leaching of nitrogen and especially pesticides, water flow through macropores become important, because it provides a must faster route to the groundwater.

Here is an example where we specify macropores:

```
(defcolumn Andeby default
  (SoilWater (macro default
    (height_start 0 [cm])
    (height_end -200 [cm])
    (distribution (-200 1) (-80 0.1) (0 0))
    (pressure_initiate -5 [cm])
    (pressure_end -30 [cm])
    (pond_max 0.5 [mm])))
  ;; Other parameters...
```

We specify that the macropores start at the soil surface, and end in 2 meters depth with `height_start` and `height_end`. All macropores are assumed to start at `height_start`, but their endpoints are distributed as specified by the `distribution` parameter. Here, we specify that all the macropores have ended at 2 meters depth, 10% of the macropores end above 80 cm depth, and none end above the surface (doh!). We use linear interpolation between these points, so we can conclude that 5% of the macropores end above 40 cm depth. You do not have to specify `height_start` and `height_end` explicitly, if you don't, the last and first value point of `distribution` will be used.

The macropores can be activated in two different ways. Macropores that, like here, reach the surface, can be activated by ponding. The parameter `pond_max` specifies that the macropores will be activated when we have half a millimeter water on the soil surface. Once activated, the macropores will empty the pond. The other way macropores can be activated is if the water pressure in any node within the region with macropores reaches above `pressure_initiate`. In that case, it will be drained immediately down to `pressure_end`.

The water, and any solutes in it will be distributed according to where the macropores end. I.e. if the macropores was activated by ponding, 10% of the ponded water would have entered the matrix at 80 cm depth, and all the ponded water would have entered the matrix at 2 meters depth.

4.5.1 Log models

The following log models are useful when dealing with macropores:

Preferential Water Flux The preferential water flux between all numeric soil layers. File: `pref_flux.dlf`.

Preferential N03 Transport The preferential nitrogen transport between all numeric soil layers. File: `pref_N03.dlf`.

Preferential NH4 Transport The preferential ammonium transport between all numeric soil layers. File: `pref_NH4.dlf`.

See section 6 for more information about log models.

4.6 Evapotranspiration

By default, Makkink is used for finding the reference evapotranspiration. It is a simple model, but adequate for simple simulations. It is specified as

```
(defcolumn Andeby default
  (Bioclimate default (pet makkink))
  ;; Other parameters...
```

except that you don't have to, since it is default.

Sometimes you have access to reference evapotranspiration data in the weather file, in order to make Daisy use these you should specify pet like

```
(Bioclimate default (pet weather))
```

Important: If you specify reference evapotranspiration in the weather file, but leave out the above line in the Daisy setup file, Daisy will ignore the data in the weather file, and continue to use Makkink for reference evapotranspiration.

Daisy also support the more advanced model Penman-Monteith

```
(Bioclimate default (pet PM))
```

If you have reliable information about vapor pressure and wind, it is much preferable to use Penman-Monteith over Makkink. If the data are missing of poor quality, the simple Makkink are more robust.

It is possible to use Penman-Monteith together with Shuttleworth-Wallace for the most accurate model of the bioclimate supported by Daisy. Enable it with

```
(Bioclimate default (pet PM) (svat PMSW))
```

This requires hourly weather data to be useful.

4.6.1 Potential and Reference Evapotranspiration

You may also want to change how potential evapotranspiration is calculated from reference evapotranspiration. Both the soil surface and crops influence this, depending on how much of the soil is covered by crops. To change the soil surface factor, specify `EpFactor` like this:

```
(defcolumn Foulum default
  (Surface (EpFactor 1.0))
```

The default value is 0.8.

For crops, derive new crop types like

```
(defcrop "Andeby Spring Barley" "Spring Barley"
  "Spring Barley with modified EpFac as measured
  in Andeby, 2002, for the SuperGrow(TM) cooperation."
  (Canopy (EpFac 1.2)))
```

overwriting the default `EpFac` of 1.0. Remember that you need to sow and harvest `Andeby Spring Barley` instead of `Spring Barley`. Or alternatively, overwrite the parameter at the time of sowing, like this:

```
(sow ("Spring Barley"
      (Canopy (EpFac 1.2))))
```

If you want `EpFac` to depend on the development stage, specify it like

```
(sow ("Spring Barley"
      (Canopy (EpFacDS (0.0 1.0) (1.0 1.2) (2.0 1.0)))))
```

Here, `EpFac` will be 1.0 at emergence (DS 0.0), gradually rise to 1.2 at flowering (DS 1.0), and then fall towards 1.0 at ripeness (DS 2.0).

4.7 Discretization

Internally, Daisy divides the soil into a number of discrete layers. During the simulation, it will keep track of the content (water, heat, nitrogen, etc.) of each layer, and the flows between the layers. A fine discretization causes more exact results, but slows down the simulation, and very fine discretization may cause numeric problems. The default discretion represents a balance between these concerns, but sometimes it will be advantageous to specify it manually.

We specify the a numeric discretization of the soil with the `zplus` parameter.

```

(defcolumn Andeby default
  "Data collected by F.Guf at the B.And farm, Andeby, 2002."
  (Soil (horizons (-20 [cm] Ap) (-2.5 [m] C))
    (MaxRootingDepth 60.0 [cm]))
  (Movement vertical
    (Geometry (zplus -2.5 -5 -10 -15 -20
      -25 -30 -40 -50 -60 -70
      -80 -90 -100 -125 -150 -175
      -200 -225 -250 [cm]))))
  (Groundwater deep))

```

Like for the horizons, we list the endpoints of each numeric layer (or *node*) from the top to the bottom. The [cm] at the end of the list indicates that all the previous numbers are given in cm.

A good rule of thumb is to use small intervals at the top (one or two cm) where most of the action is, and use larger steps (dozens of cm) near the bottom, where things are more quiet, at least with a fixed groundwater level.

A hard rule is that numeric layers may not cross horizon boundaries, which mean the end points in the horizon list must also be somewhere in the zplus list.

The default discretization obey the `border` parameter, when you specify `zplus` explicitly, `border` no longer applies.

4.8 Initialization of soil content

Daisy will by default initialize the content of the soil with “reasonable” values, which are most likely wrong. So if you have actual measurements you should use those instead.

4.8.1 Soil water

You can initialize the soil water content or the soil water potential, but not both. The “other one” will be automatically initialized from the values you provided, and the soils hydraulic properties (section 4.3). To specify initial water content, use this:

```

(defcolumn Andeby default
  (SoilWater (initial_Theta (-100 [cm] 20 [%])
    (-150 [cm] 0.1 [])))
  ;; More parameters...

```

Here we specify that there are 20% water in the first meter, and 10% water in the next 50 cm. The empty dimension [] means “fraction”. Here we divided the soil in two layers, but you can divide it into as many layers as you want.

Soil water pressure is initialized like this:

```
(SoilWater (initial_h ( -50 [cm]  -10 [kPa])
              (-100 [cm] -100 [cm])
              (-1.5 [m]    2 [pF])))
```

Here we specify field capacity all the way, using three different units.

4.8.2 Soil nitrogen

There are three different ways to specify mineral nitrogen in the soil. The first is per dry soil matter.

```
(defcolumn Andeby default
  (SoilN03 (initial_Ms (-30 [cm] 10 [ppm])
            (-70 [cm] 5 [ppm])))
;; More parameters...
```

Here we specify the weight of nitrate nitrogen (i.e. we ignore the weight of the oxygen atoms) in the system (both soil and water), as a fraction of the dry soil weight. The idea is that you first extract the nitrate from a measured soil sample, then weight it, compensate for the oxygen, dry the soil sample, and weight that. That will give you the nitrate nitrogen fraction of the soil.

The second method is per soil water.

```
(defcolumn Andeby default
  (SoilNH4 (initial_C (-100 [cm] 1 [ppm])))
;; More parameters...
```

Here the idea is that we extract a sample of water from the soil, and measure the ammonium nitrogen concentration in that sample. This will *not* give you the total amount of ammonium nitrate in the soil, as some of it is adsorbed to the soil. That fraction will be found by assuming the adsorbed and solved phases are in equilibrium.

The third method is to specify the total content per volume, counting both soil, water and air. This number gives a content that is independent of the dry bulk matter density and the water content.

```
(SoilN03 (initial_M (-100 [cm] 50 [g/cm3])))
```

4.8.3 Soil temperature

You can also initialize the initial temperature of the soil.

```
(defcolumn Andeby default
  (SoilHeat (initial_T (-100 [cm] 1 [dg C])))
  ;; More parameters...
```

5 Management

Writing management specifications that behave both semi-intelligently and robustly to the simulation state is a challenge. However, specifying each and every little management operation for a 30 year run can get both tedious and error prone.

In this section, we will therefore start by introducing you to a way to organize your manager in “crops” and “rotations”, which will save you much work, and then continue to give examples of various common management operations.

5.1 Crop Managements

Our first hint is to collect all management information about a specific crop, as in the example below.

```
;; Spring Barley management.
(defaction sbarley activity
  (wait_mm_dd 3 20)
  (plowing)
  (wait_mm_dd 4 10)
  (fertilize (pig_slurry (weight 100.0 [Mg w.w./ha])))
  (wait_mm_dd 4 15)
  (seed_bed_preparation)
  (sow "Spring Barley")
  (wait_mm_dd 7 1)
  (wait (or (crop_ds_after "Spring Barley" 2.0 []) ;Ripe
            (mm_dd 10 1)))
  (harvest "Spring Barley"))
```

Here, we define a new management operation (or *action*) named `sbarley`, which can be used like any other management operation, e.g. plowing, it just last a bit longer. Specifically, it last the good part of a year.

We define `sbarley` as an `activity`, just like in the example in section 2.6. The trick is to give it a name (`sbarley`), and specify it in a way so it can be used any year. The key to the later is to use

```
(wait_mm_dd 3 20)
```

which, unlike (from the example)

```
(wait (at 1987 3 20 1))
```

does not specify a year⁴. It matches the 20th of March every year, allowing us to use in any year. Later, when waiting for the crop to become ripe, we similarly use an `mm_dd` condition, which become true at 8:00 AM October the 1st every year, instead of waiting testing for a more specific date as in the example.

5.1.1 Rotations

If we have defined crop management actions as in the previous section for spring barley, rye, and grass, we can use them in a manager specification like this

```
(manager activity
  sbarley sbarley grass rye)
```

As you see, we only have to specify the management operations associated with growing spring barley once, even though we grow spring barley twice in the simulation. We can take this one step further, by naming and reusing specific rotations.

```
(defaction rotation_bbgr activity
  sbarley sbarley grass rye)
```

```
(manager activity
  rotation_bbgr rotation_bbgr rotation_bbgr rotation_bbgr rotation_bbgr)
```

Here, we define and name a whole rotation containing four seasons, and repeat that rotation five times. All in all, we have specified management for 20 years in two lines.

⁴The observant reader will notice that the hour isn't specified either. You can specify the hour, see [Abrahamsen, 1999], but usually the default of 8:00 AM is fine. It is certainly less stressful to the farmer than the 1:00 AM we send him plowing earlier.

This is how I in general recommend writing management specifications. Break it down into management specifications for individual crops, then combine the crops into rotations. To make sure the initial state of Daisy is reasonable, it is a good idea to run at least one rotation before the period you are interested in simulating.

In the next sections I will present more specific examples of common management operations.

5.2 Tillage

The tillage operations defined in `tillage.dai` are `plowing`, `rotavation`, `disk_harrowing`, `stubble_cultivation` and `seed_bed_preparation`. You use them all like `plowing` in the example.

All the tillage operations kill any crops on the soil, and homogenize the top soil content (be it water, organic or inorganic matter, pesticides, or heat). Plowing will additionally swap the content of the top and bottom of the plowing layer. The content on the soil surface will be incorporated into the soil, for `disk_harrowing` and `stubble_cultivation` this incorporation will only be partial.

5.2.1 Trafficability

A real farmer might want to delay some operations if the soil is too wet, or frozen. The `trafficable` condition will test for this. You can use it like

```
(wait trafficable)
(plowing)
```

but it might be a good idea to add a timeout for robustness.

```
(wait (or trafficable (mm_dd 04 01)))
(plowing)
```

Here we wait until the soil is trafficable, or we have reached the first of April. This is important if we use it in a crop management description like in section 5.1. Imagine we have the following piece of management activity:

```
(wait trafficable)
(plowing)
(wait_mm_dd 4 10)
```

If the conditions doesn't get trafficable before the 10'th of April, we will plow after the 10'th of April and then continue to wait for the 10'th of April

next year. If we only run a short simulation, this might be OK, we can then discard the results and investigate what went wrong. However, if we are simulating a 30 year period, it will be highly annoying to have to rerun because a single year had an unusually later winter or wet spring. For the sake of such long simulations, it is a good idea to get in the habit of adding timeouts to such conditions.

5.3 Sowing and Harvesting

By including the `crops.dai` file, you get access to the following crop types: Fodder Beet, Grass, Maize, Pea, Potato, Rye, Spring Barley, Spring Rape, Sugar Beet, Spring Wheat, Winter Barley, Winter Rape and Winter Wheat.

In general, creating parameterizations for new crops requires field experiments where you carefully measure the nitrogen content during the development with different fertilization levels. However, making small modifications for specific purposes is feasible, as demonstrated in section 4.6.

5.3.1 Crop Residuals

In general, we harvest when the crop is ripe, and kill the crops in the process. The main decision is whether to remove the crop residuals, or leave them as litter on the field. Here is an example where we remove the crop residuals

```
(harvest "Spring Barley"  
  (stub 8 [cm]))
```

and one where we leave them as litter on the field.

```
(harvest "Spring Barley"  
  (stub 8 [cm])  
  (stem 0.0 [])  
  (leaf 0.0 []))
```

In both cases, we leave the stub and roots on the field, and remove the storage organ (that is, the grains).

5.3.2 Multiple Cuts

For some crops, like grass, we will want to harvest or cut them multiple times. In such cases, the stub is essential, as that determine how much leaf area there will be left to start regrowth. If you don't leave any stub (the default!), the plant will be unlikely to recover.

Writing the generic management specification for grass is a lot more complex than for barley, because we do not know in advance how many times we are going to cut it. It depends on how fast it grows. Therefore, we start by defining two simpler actions.

```
(defaction cut_grass activity
  (wait (or (crop_ds_after "Grass" 0.7 [])
            (crop_dm_over "Grass" 4000 [kg DM/ha] (height 10.0 [cm]))))
  (harvest "Grass" (stub 10.0 [cm])))
```

Here, we define a `cut_grass` operation which wait for the grass to grow old or big enough, and then cut it.

```
(defaction cut_grass_fertilize activity
  (cut_grass)
  (wait_days 1)
  (fertilize (N25S (weight 100.0 [kg N/ha]))))
```

Here, we extend the `cut_grass` operation by fertilizing after the cut.

Now we can define our grass management action:

```
(defaction grass activity
  (wait_mm_dd 3 20)
  (wait trafficable)
  (plowing)
  (wait_mm_dd 4 10)
  (fertilize (N25S (weight 100.0 [kg N/ha])))
  (wait_mm_dd 4 15)
  (seed_bed_preparation)
  (sow "Grass")
  (while (wait_mm_dd 9 1)
    (activity
      ;; Max 5 fertilized cuts.
      cut_grass_fertilize cut_grass_fertilize cut_grass_fertilize
      cut_grass_fertilize cut_grass_fertilize
      ;; As many unfertilized cuts as we want.
      (repeat cut_grass)))
  (while (wait_mm_dd 11 1)
    ;; Additional unfertilized cuts.
    (repeat cut_grass)))
```

The initial part is quite conventional, as seen in sections 2.6 and 5.1, until the two `while` actions. A `while` action take two other actions, and perform them both in parallel, until the first action is done. It does not matter whether the second action finish or not.

In the example above, the first action is (in both cases), a `wait_mm_dd` operation. This means that the `while` will simply continue until we reach the specified date. While we are waiting for that date, it will do as specified by the second action. This means that until the 1'st of September, we may see up to 5 cuts followed by fertilization. We might see less that that, if the grass does not grow fast enough, but never more. After the 5 fertilized cuts, assuming it is not yet September, we may see any number of unfertilized cuts. The `repeat` action will perform the specified operation over and over forever, which means it is only useful inside a `while`, otherwise it will never break out of the loop.

After that, we start a new round of unfertilized cuts, which last until November. You may wonder why we didn't just used a single `while`, which waited until November. What is the difference? I'm glad you asked. With the description above, we get a manager that will apply fertilizer at most six times (including the initial fertilization the 10'th of April), thus staying below a possibly politically induced upper limit. *And*, this is the trick, never apply fertilizer after August. Even if we have only had, say, three fertilized cuts before September, and later cuts will be unfertilized. The idea being that we do not believe the grass will grow enough that late in the year to justify the additional fertilization.

Read more about fertilization in the next section.

5.4 Fertilization

We have two basic types of fertilizer, *mineral* and *organic*. These will be described in the two next sections, followed by examples of some more advanced types of fertilization.

5.4.1 Mineral Fertilizer

In section 2.6 we used mineral fertilizer, as below:

```
(fertilize (mineral
           (weight 100.0 [kg N/ha])
           (NH4_fraction 0.5 [])))
```

If we used this fertilizer type often, it might be nice to give it a name. We could do this with the following definition:

```
(defam SuperGrow mineral
  (NH4_fraction 0.5 []))
```

To use it, we would write:

```
(fertilize (SuperGrow (weight 100.0 [kg N/ha])))
```

It only saves one line, but the producers of SuperGrow™ will be eternally grateful. The `fertilizer.dai` file already contain the following mineral fertilizers:

Ammonia Pure NH_4^+ .

AmmoniumNitrate A 50-50 mix of NH_4^+ and NO_3^- .

Nitrate Pure NO_3^- .

N25S 50.8% NH_4^+ .

5.4.2 Organic Fertilizer

We used organic fertilizer, specifically pig slurry, in the example in section 5.1. The obvious difference is that the weight is given in metric tons of wet weight, rather than kg nitrogen. The `fertilizer.dai` file contains the following ready to use definitions of pig slurry (named `pig_slurry`), cattle slurry (`cattle_slurry`), pig manure (`pig_manure`), cattle manure (`cattle_manure`), and horse manure (`horse_manure`).

For purely hypothetical scenarios, using the predefined organic fertilizer specifications are adequate, however, if you know more about the system being simulated, getting a better description of the slurry or manure being applied is important, as the quality varies a lot. You can define a new slurry like this:

```
(defam Andeby_pig_slurry slurry
  "Data collected by F. Guf, 2002 at the B. And farm, Andeby."
  (dry_matter_fraction 0.073)
  (total_C_fraction 0.323)
  (total_N_fraction 0.0863)
  (NH4_fraction 0.7)
  (NO3_fraction 0.0)
  (volatilization 0.15))
```

where

`dry_matter_fraction` Dry matter fraction of total (wet) weight.

`total_C_fraction` Carbon fraction of dry matter.

`total_N_fraction` Nitrogen fraction of dry matter

`NH4_fraction` Ammonium fraction of total N in fertilizer.

`NO3_fraction` Nitrate fraction of total N in fertilizer, by default zero.

`volatilization` Fraction of NH_4^+ that evaporates on application.

The organic nitrogen is what is left having removed the two inorganic fractions.

It is possible to specify other parameters such as the turnover rate and efficiency for the organic fertilizer, but these require specialized knowledge of both the fertilizer itself, and the organic matter model in Daisy (see [Hansen, 2002]).

5.4.3 First and Second Year Utilization

Sometimes organic fertilization plans are made from estimated first and second year utilization of the nitrogen. In Denmark, this even have legal force. None of the fertilizers in `fertilizer.dai` contain such estimates, as they varies both in time and jurisdiction, but you can specify them yourself, as below.

```
(defam "Andeby Pig Slurry" pig_slurry
  (first_year_utilization 0.40)
  (second_year_utilization 0.15))
```

Here, we estimate that 40% of the nitrogen (organic or mineral) is utilized the first year, and 15% the second year. The rest is presumably lost. Note again, that these estimates have absolutely no effect on how much will actually be utilized during the simulation.

To use this information, we can write the following

```
(fertilize ("Andeby Pig Slurry")
  (equivalent_weight 100.0 [kg N/ha]))
```

in our management specification. We let Daisy calculate the amount of wet weight to apply, to get a first year utilization of 100 kg N/ha. Note that `equivalent_weight` is outside the parentheses around the "Andeby Pig Slurry", unlike `weight`. The `weight` parameter is a property of the

fertilizer itself, while `equivalent_weight` is a property of the fertilize action. Thus, the difference.

To use the `second_year_utilization` parameter, we should let one harvest operation (usually the first) the next year have the form

```
(fertilize ("Andeby Pig Slurry")
  (second_year_compensation true)
  (minimum_weight 50.0 [kg N/ha])
  (equivalent_weight 150.0 [kg N/ha]))
```

The `second_year_compensation` directive means that Daisy should subtract any second year effect accumulated from fertilizations the previous year from `equivalent_weight` before calculating the amount of fertilizer to apply. By specifying `minimum_weight` we put a minimum on the amount of fertilizer we will bother applying. That is, if the accumulated second year effect is more than 100 kg N/ha, we won't fertilize at all.

5.4.4 Precision Fertilization

Precision fertilization allows you to let the amount of fertilizer you apply depend on the nitrogen content of the soil in a specified zone.

```
(fertilize (N25S)
  (precision 150.0 [kg N/ha] (from 0.0 [cm]) (to -100.0 [cm]))
  (minimum_weight 50.0 [kg N/ha]))
```

Daisy will measure the amount of fertilizer in the specified zone (from the soil surface to 1 m depth, which is also the default), and supply enough extra nitrogen to reach the specified target (150 kg N/ha). If less than the specified minimum (50 kg N/ha) nitrogen is needed in order to reach the target, no fertilizer will be applied.

You can use the `precision` parameter with organic fertilizer as well, in that case you must also specify the `first_year_utilization` fertilizer parameter.

5.4.5 Incorporation

By default, fertilizer will be applied directly on the soil surface. There, organic fertilizer will be incorporated gradually by biological means (earth worms and the like), or suddenly by mechanical means, i.e. by a tillage operation. Inorganic fertilizer will penetrate the soil along with the rain or irrigation water.

However, if a farmer has the equipment for it, he will prefer incorporating the fertilizer directly into the soil. It in general increase utilization, and decrease complaints about pig slurry smell from neighbors.

Use the `from` and `to` fertilize parameters to specify incorporation, as below:

```
(fertilize (pig_slurry (weight 100.0 [Mg w.w./ha]))
           (from -5 [cm]) (to -20 [cm]))
```

Again, note the extra set of parentheses that binds `weight` to `pig_slurry`. Unlike `from` and `to`, `weight` is a property of the `pig_slurry`, not of the `fertilize` operation.

5.5 Irrigation

Irrigation can be applied overhead the canopy (e.g. in the form of sprinklers), on the soil surface, or incorporated directly into the soil.

To specify overhead irrigation, write

```
(irrigate_overhead 30 [mm/h])
```

This will irrigate with 30 mm/h for one hour, giving a total of 30 mm. The water will first hit the canopy, where some will stay until it eventually evaporate, and some may wash down any pesticides on the canopy.

To give the same amount of water over a timespan of 10 hours, you can instead write

```
(irrigate_overhead 3 [mm/h] (hours 10))
```

To specify surface irrigation, write

```
(irrigate_surface 30 [mm/h] (temperature 10.0 [dg C]))
```

Again, this will give a total of 30 mm. The water will bypass the canopy, and directly hit the soil surface where it might create a pond that slowly evaporates or enter the soil.

If you don't specify the temperature parameter, as in the `irrigate_overhead` case, the air temperature will be used.

The syntax for incorporating the water directly into the soil is similar, here is an example:

```
(irrigate_subsoil 1 [mm/h] (days 1) (hours 6)
                  (from -5.0 [cm]) (to -10.0 [cm])
                  (solute (NH4 1.0 [mg N/l]) (NO3 7.0 [mg N/l])))
```

Here we specify subsoil irrigation with 1 mm/h for 30 hours, giving us a total of 30 mm. It will be incorporated into the soil between 5 and 10 cm's depth. We also specify a content of inorganic nitrogen in the water, something we can also do for overhead and surface irrigation. We do not and cannot specify the water temperature for subsoil irrigation, it is assumed to be the same temperature as the soil itself.

5.5.1 Conditions

If we know exactly when the irrigation is to be done, we can simply wait for the date, as here

```
(wait_mm_dd 07 15)
(irrigate_overhead 30 [mm/h])
```

However, when writing generic crop managements as in section 5.1, we can't use that technique. It might rain the first two weeks of July for all we know, which will make irrigating rather wasteful. Daisy therefore provides ways to test for the soil water content.

```
(wait (not (soil_water_pressure_above (height -30.0 [cm])
                                         (potential -1000 [cm])))
(irrigate_overhead 30 [mm/h])
```

Here, we emulate a tensiometer placed in the field at 30 cm depth, triggering irrigation when the potential gets below -1000 cm.

An alternative condition would be

```
(wait (not (soil_water_content_above 200.0 [mm]
                                         (from 0 [m]) (to 1 [m])))
```

where we wait until there is less than 200 mm water in the top meter of the soil.

5.5.2 Strategies

However, conditions like these are not enough, since we cannot know in advance how many times it gets necessary to irrigate, if any. This problem is similar to the problem of multiple cuts we had in section 5.3.2, and the solution also build on our “parallel processing” directive, `while`. Imagine we already have a crop management description, like `sbarley` in section 5.1. We can then add an irrigation strategy like this:

```

(defaction irrigate_30 activity
  (wait (not (soil_water_pressure_above (height -30.0 [cm])
                                         (potential -1000 [cm]))))
  (irrigate_overhead 30 [mm/h]))

(manager activity
  (while sbarley
    (repeat irrigate_30))
  (stop))

```

Here, we define a management action named `irrigate_30`, which consists of waiting for the the soil to dry out, and then irrigating with 30 mm. Our management specification says that as long as we are managing our spring barley (as defined in `sbarley`), we will repeatedly run `irrigate_30`, that is, wait for the soil to dry, and then irrigate.

An advantage of implementing our irrigation strategy like this is that we can apply the strategy on any crop. There are, however, two problems we need to solve before it gets useful. First, we should only irrigate in the growth season, even if the soil somehow dries out during the winter. The second is that irrigation takes one hour, but the water may very well take longer to reach the tensiometer. This means that the tensiometer will keep triggering irrigation several time, until the water finally reaches it.

We solve the first problem by adding an “in season” test to the wait conditions, and the second problem by adding a delay after irrigation.

```

(defaction irrigate_30 activity
  (wait (and (after_mm_dd 5 1)
            (before_mm_dd 9 1)
            (not (soil_water_pressure_above (height -30.0 [cm])
                                             (potential -1000 [cm])))))
  (irrigate_overhead 30 [mm/h])
  (wait_days 2))

```

If there are limits on the total amount of water we are allowed to use for irrigation, we can limit the macimal number of times we irrigate, like below.

```

(manager activity
  (while sbarley
    (activity irrigate_30 irrigate_30 irrigate_30 irrigate_30))
  (stop))

```

Here, we will at most irrigate four times.

5.6 Pesticides

Pesticides have no effect in Daisy (i.e. Daisy crops are immune to everything, and weeds are something that happens to other models). Nonetheless, Daisy allows you to apply them to the field. Daisy will then keep track on them, as the are transported with the water through the system, or are removed from the system by either decomposition, root uptake, or leaching to the groundwater.

The command to apply pesticides is

```
(spray Bentazon 1000 [g/ha])
```

with `Bentazon` being the name of the pesticide. The currently available pesticide parameterizations includes `Atrazine`, `Bentazon`, `IPU`, `MCPPP`, `Pendimethalin`. To use them, you must add

```
(input file "chemistry.dai")
```

to the start of the setup file.

5.6.1 Defining New Pesticides

You can easily introduce new pesticides by specifying a few parameters. The parameters may not be that easy to obtain, though. Here is an example:

```
(defchemical Atrazine default
  "From a FAO sponsored field experiment near Andeby, 2002."
  (canopy_dissipation_rate 8e-3 [h^-1])
  (canopy_washoff_coefficient 1.0 [])
  (crop_uptake_reflection_factor 0.8 [])
  (decompose_rate 7e-5 [h^-1])
  (diffusion_coefficient 8e-6 [cm^2/s])
  (solute (adsorption linear (K_clay 7 [cm^3/g])))
```

As we have seen before, in the first line we *define* a new *chemical* name *Atrazine*, using the *default* (and currently only) model for chemicals in Daisy. In the second line, we put some information about this chemical so we will know later where these numbers come from.

5.6.2 Canopy

The pesticide will be introduced into the system with the `spray` command. Some of the pesticide will land directly on the soil surface, while other will

hit the canopy, depending on how large a fraction of the soil is covered by the crops. The fraction that hits the canopy is assumed to stay there, until it either dissipate entirely out of the system, or is washed down to the soil surface by rain or irrigation.

The parameter `canopy_dissipation_rate` indicates how large a fraction of the pesticide dissipate from the canopy each hour, while `canopy_washoff_coefficient` is a measure of how the pesticide dissolve in the intercepted water. If you set it 0.0 it does not dissolve at all, and will never be washed off. If you set it to 1.0 it will be fully dissolved, and thus follow any water that fall off the canopy.

5.6.3 Soil Surface and Root Uptake

Down on the soil surface, Daisy will pretend that the pesticide is immediately fully dissolved in any water that hits the soil surface, and follow the water when it enters the soil. Nothing else will happen to the pesticides when on the soil surface. Pesticides in the top of the soil can also move up to the soil surface in case of ponding. This effect can be significant when there are macropores activated by the ponding, as pesticides that have already entered the soil can later be transported with macropores from the surface.

In the soil, the pesticides can be adsorbed, transported, decomposed, or taken up by the roots by transpiration. The later is controlled by `crop_uptake_reflection_factor`. In the example, we specify that 80% of the pesticides dissolved in the water extracted by the roots for transpiration will be reflected, that is will stay in the soil water. The 20% that is uptaken will disappear out of the system.

5.6.4 Decomposition

We set `decompose_rate` to the fraction of pesticides assumed to be decomposed each hour. This rate is affected by a number of factors, listed here:

`decompose_heat_factor`, `decompose_water_factor` Biological decomposition tend to accelerate with increasing heat and moisture. Unless you specify otherwise, the factors used for mineralization will be reused here.

`decompose_CO2_factor` CO₂ production is an indicator of an active soil biomass. By specifying this parameter, you can let the general biological activity influence the decomposition.

decompose_conc_factor Some pesticides will be hard to decompose at high or low concentrations, which can be specified with this parameter.

decompose_depth_factor Sometimes the decompose rate depend on how deep in the soil we are, which is what this parameter is for.

decompose_lag_increment Some pesticides require specialized organisms to decompose, which may take some time to appear. Daisy therefore keep track of a time lag variable for each pesticide, which starts at zero. When it reaches one, decomposition begins. You can set this parameter to specify the hourly increment, as a function of the concentration in the soil. By default, the increment is 1, which means decompositions starts immediately, with no time lag.

See [Abrahamsen, 1999] for more specific information about each of these factors.

5.6.5 Transportation

The pesticide in the soil can be transported with convection and diffusion. Set **diffusion_coefficient** to specify the later. Finally, the pesticide may adsorb to the soil. Daisy have a number of advanced models for adsorption, but for pesticides it is usually adequate to either disable adsorption (**adsorbtion none**), or use a simple linear correspondence between adsorbed and dissolved matter, $A = K\rho C$, where A is the amount of adsorbed matter, K is a pesticide and soil specific parameter, ρ is the bulk density of the soil, and C is the amount of dissolved matter. In Daisy, we have split K into a **K_clay** parameter which is multiplied with the clay fraction of the soil, and a **K_OC** parameter which is multiplied with the humus fraction of the soil.

5.6.6 Log models

Here are some log models that are useful when dealing with pesticides:

Surface Chemicals Summary information about chemicals on the soil surface. File: **surface_chemicals.dlf**.

Soil Chemical Summary information about chemicals in the soil. File: **soil_chemical.dlf**.

Chemical Amount of chemicals in each numeric soil layer. Note that nothing will appear in the log file until the chemichals have entered the soil. Not even zeros. File: **chemical.dlf**.

In general, you will get the sum of all chemicals. To log the content of a specific pesticide, such as 'Atrazine', specify the *chemical* parameter, as in:

```
("Soil Chemical" (chemical Atrazine))
```

6 Log Files

In this section I list some useful log models not listed elsewhere, but first lets refresh the general useful parameters. The following two parameters can be used for almost all log models:

when Indicates how often to log. In theory, you can use any condition, like e.g. `crop_dm_over`, but in practice `hourly`, `daily`, `weekly`, `monthly` and `yearly` are the most useful ones. The log models will automatically accumulate flux values between time steps. When not otherwise indicated, log models log every hour.

where Select the file in which to log the results.

As explained in 2.4, the soil is split into a large number of numeric layers for computational purposes. Some log models, such as `Soil Temperature`, will log the content of a single state variable for each of these layers. Other log models will log many state variables by integrating the content of the variable in for all layers in some interval, or for flux variables, use the values at the top or bottom of the interval. `N Balance` is an example of such a log model. You can control what interval to summarize by setting the `from` and `to` parameters.

We use all of them in this example:

```
(output ("N Balance" (when monthly)
          (where "nitrogen-balance.dlf")
          (from 0 [m]) (to -1 [m])))
```

6.1 Activating output

Since writing a log file tend to be as slow as running the simulation itself, and if you specify many log files Daisy will become very slow indeed. It also depend on the log file, as a general rule, large log files are much more time consuming than small log files. So don't ask for hourly values of daily or even monthly values will do. But the 'harvest' log model is nearly free. And if you are using Daisy interactively, don't specify a log file just because

you *may* need the information. If you later discover that you need the information in that log file, you can rerun the simulation specifying log file only.

You can also limit the period where log file is kept, both to speed up the simulation, and to avoid uninteresting data. This is especially useful in conjunction with a warmup period. You do this by setting `activate_output` at the top level, like this:

```
(activate_output (after 1997 3 31 23))
```

This will activate the log files from April 1st, 1997 and to the end of the simulation. You can specify all the same kind of conditions as in management. To limit the period in both ends, you can use

```
(activate_output (and (after 1997 3 31 23)
                      (before 1998 4 1 0)))
```

which will give you log files containing data from the hydraulic year 1997.

6.2 Log models

Here are some useful log models, you can find more in the `log.dai` file, and scattered around this manual.

Total Soil Content Information about the water, carbon and nitrogen content of the soil. File: `soil_content.dlf`.

Soil NO3 Concentration Amount of nitrate in all numeric soil layers. This include both the nitrogen and oxygen atoms of the nitrate. File: `soil_NO3.dlf`.

Soil NO3-N Concentration Amount of nitrate nitrogen in all numeric soil layers. This include only nitrogen atoms of the nitrate. File: `soil_NO3-N.dlf`.

Soil Temperature Temperature in all numeric soil layers. File: `soil_temperature.dlf`.

Soil Water Potential Water potential in all numeric soil layers. File: `soil_water_potential.dlf`.

Soil Water Sink The water sink in all numeric soil layers. The water sink is an aggregate of root uptake, preferential flow, flow to drain pipes, and subsoil irrigation. File: `soil_water_sink.dlf`.

Matrix Water Flux The matrix water flux between all numeric soil layers.
File: `matrix_water_flux.dlf`.

Surface Water Balance Enough information to create a water balance for the soil surface. File: `surface_water_balance.dlf`.

Root Zone Water Balance Enough information to create a water balance for the root zone. File: `root_zone_water_balance.dlf`.

N Balance Enough information to create a balance for the soil nitrogen.
File: `N_balance.dlf`.

Carbon Balance Enough information to create a balance for the soil carbon.
File: `C_bal.dlf`.

Soil Heat Balance All available information about the soil heat subsystem. File: `soil_heat_balance.dlf`.

Snow All available information about the snow subsystem. File: `snow.dlf`.

7 Plotting simulation results

The `.dlf` files produced by Daisy are text files with (by default) tab separated columns, and can be viewed by any text editor, or by most spreadsheets. There also exist a specialiced program, `ShowDaisyOutput`, to display the results, and Daisy has a build in fascility to covert the `.dlf` files (as well as the related weather (`.dwf`) files and data files `.ddf` with measurements) into commands to the general plot program `gnuplot`.

7.1 Spreadsheets

Note that Daisy uses US style numbers, that is decimal point rather than decimal comma. If the numbers looks weird in your spreadsheet, check how they look in a text editor like notepad. If the numbers are different, you may need to change the settings of the program to accept american style numbers. In MS Windows, you can change the number style from the control panel (Start → Control panel → International).

For Excel, the easiest way to open Daisy log files are either to drag them into a shortcut to Excel, or to associate the `.dlf` files with Excel (right click on the file, and use “Open with” to open the file with Excel).

7.2 ShowDaisyOutput

The ShowDaisyOutput can be fetched from the Daisy homepage. It only works under MS Windows. It is very good for a fast overview of the results, you simply drag the `.dlf` file to the ShowDaisyOutput icon, and the program will display the first (non-time) column. You then have the option to display another column from a menu, or in some versions to zoom into the data by dragging the mouse on the plot area you want to view. For the `crop_prod.dlf` file, ShowDaisyOutput is able to display all DM at once.

For some log files, ShowDaisyOutput can display the data as a soil profile with depth at the y-axis and value at the x-axis. You can then animate the data to see changes in time.

There is no printing facilities in ShowDaisyOutput, use one of the other options for that.

7.3 Generating gnuplot commands from Daisy

The *gnuplot* program build into Daisy allows you to create command files for external `gnuplot` or `wgnuplot.exe` to create plots in encapsulated postscript (`*.eps`), Acrobat Reader PDF (`*.pdf`), LaTeX (`*.tex`) format or be shown directly in a window. The data to be plotted can be read from Daisy log files (`*.dlf`), Daisy weather files (`*.dwf`) or the Daisy data files (`*.ddf`).

7.4 Simple example

```
(defgnuplot test time
  (source (column (file "harvest.dlf") (tag "leaf_DM"))
          (column (file "harvest.dlf") (tag "sorg_DM"))
          (column (file "crop_prod.dlf") (tag "Leaf AI"))))

(defprogram "Andeby Graphs" gnuplot
  (graph test))

(run "Andeby Graphs" )
```

This will create a `daisy.gnuplot` file that generates a window with a plot of leaf and storage organ yields taken from the `harvest.dlf` file, and LAI from the `crop_prod.dlf` file.

7.5 Complex example

```
(defgnuplot another_test time
```

```

"Test of 'gnuplot' program."
(ymin -1) (ymax 6) (y2min 0) (y2max 10)
(begin 1986 9 1 0) (end 1987 11 1 0)
(title "gnuplot demonstration")
(size 1.0 0.5)
(extra "set xtics rotate")
(legend se)
(source (column (file "harvest.dlf")
              (tag "leaf_DM")
              (title "Leaf Harvest")
              (with points)
              (dimension "Mg DM/ha")))
(column (file "harvest.dlf")
        (tag "sorg_DM")
        (title "Grain Harvest")
        (with points)
        (dimension "Mg DM/ha")))
(column (file "crop_prod.dlf")
        (filter ("day_length" "10" "8")
              ("month" "9" "10")))
        (style 1)
        (tag "WLeaf"))
(column (file "crop_prod.dlf")
        (style 2)
        (tag "WSOrg"))
(column (file "crop_prod.dlf")
        (tag "Leaf AI"))))

(defprogram "Andeby Graphs" gnuplot
  (command_file "kurt.gnuplot")
  (graph (another_test)
    (test (where "test.eps"))
    (another_test (where "another-test.eps"))))

(run "Andeby Graphs")

```

This example defines a gnuplot graph called “another test”.

- This graph selects the range for the x-axis (begin and end) and both y-axes (ymin, ymax, y2min, y2max) manually, instead of letting gnuplot choose its own.

- We give the graph a title, “gnuplot demonstration”.
- We ask for a graph with the default width, but only half the default height.
- An uninterpreted gnuplot order is send via the 'extra' keyword. In this case in order to rotate the labels of the tics on the x axis 90 degrees. You can also set fonts and font sizes, ask for monochrome plots, and many other things.
- We ask to get the legend in the South-East corner, instead of letting Daisy find the corner farthest away from an data.
- We use different titles and different names for the dimensions for the 'harvest.dlf' data, and we plot them with points instead of lines. By default Daisy use points for '*.ddf' files, and lines for '*.dlf' and '*.dwf' files.
- We only get WLeaf data from 'crop-prod.dlf' that are from days with length 10 or 8, and from September or October.
- We use the first and second style for plotting WLeaf and WSOrg from 'crop_prod.dlf', instead of letting Daisy find the next available style. This means they have the same style as the corresponding data from 'harvest.dlf', except that they are plotted as lines instead of points.

In order to actually plot the graphs we have defined, we define a gnuplot program called “Andeby Graphs”. This program will generate command in the “kurt.gnuplot” file to plot three graphs. First the graph we just defined named “another_test” with no output file sepcified, meaning the graph will be shown on the screen. Then the program generates commands for plotting the original test graph in a file called “test.eps”, and then the new graph in the file “another-test.eps”.

7.6 The 'arithmetic' source model

The `arithmetic` source mode allows simple arithmetics with columns in a single file.

```
(defprogram crop gnuplot
  "Plot crop dry matter partitions."
  (source (arithmetic (file "crop_prod.dlf")
    (expr WSOrg))
```

```

(arithmetic (file "crop_prod.dlf")
            (expr WRoot))
(arithmetic (file "crop_prod.dlf")
            (title "WShoot")
            (expr (+ WStem WLeaf WDead))))

```

Here we plot storage organ and roots as usual, but add the stem, leaves and dead leaves pools into a single graph named “*WShoot*”. When you use expressions like this, you usually will want to set the legend title explicitly, otherwise it would have been called “+”.

The format of arithmetic expressions are always prefix, the name of the operator followed by the operands. Daisy currently understands the following functions +, -, *, /, min, max, pow, exp, sqrt, sqr, ln, and log10.

Numbers are written as (const *val dim*), like

```
(expr (+ WSOrg (const 1.0 [Mg DM/ha])))
```

if we want to lie about our yield.

You can convert dimensions with (convert *expr dim*) like

```
(expr (convert WSOrg [g DM/m^2]))
```

Daisy is in general unable to calculate the dimensions of the result of some arithmetic expression, except in the most simple cases (like +). You can tell Daisy the dimension with (dim *expr dim*), like

```
(expr (dim (* WSOrg (const 2.0 [])) [Mg DM/ha]))
```

The difference from ‘convert’ is that dim will not change the value, only the dimension.

7.7 Daisy Data Files (*.ddf)

Th ‘*.ddf’ files are intended for measured data. The data are by default plotted as points, and with the same styles as data sources in the same sequence from ‘*.dlf’ and ‘*.dwf’ files. The idea is that you want to compare measured and simulated numbers.

The format is:

```

ddf-0.0 You can insert a description after the seven first letters.
# Here you can add comments, here.
# As long as they begin with a '#'.
# After a line of hyphens, tab separated data should be listed.

```

```
# The two first lines are tags and dimensions, like for '*.dlf' files.
```

```
-----  
Date           Height  
                cm  
2005-09-24T01  20  
2005-09-24T07  21
```

You can either separate year/month/mday/hour columns, as in the dlf and dwf files, or a single Date column, with the format 'yyyy-mm-ddThh' (which happens to be an ISO standard).

If you don't have a dimension line, you must specify the dimension with the `original` keyword for each source.

7.8 wgnuplot.exe

The Daisy home page at <http://www.dina.kvl.dk/~daisy/> has a link to a MS Windows binary for gnuplot. You can execute the gnuplot commands by dragging the `daisy.gnuplot` file to the `wgnuplot.exe` binary. If you open the application before dragging, you will be able to see any error messages that may be produced. The gnuplot homepage are at <http://www.gnuplot.info/> where you can find Unix and Mac versions of the program. You need at least version 4.0 of gnuplot.

8 Organizing Your Parameterizations

If you follow the advice in this tutorial, you will create a number of parameterizations for things like horizons, columns, fertilizer types, and crop management strategies. Most likely, many of these parameterizations will be generically useful, that is, useful in many different simulations. My advice is that you from the start create a *library* of such parameterizations, for later use. This library should be kept separate from both the standard library that comes with Daisy, and from the files that are specific to your current simulation. By separate, I mean that you should keep the files in three or more different directories.

- A directory containing the parameterization files distributed by KVL, i.e. the content of `daisy-lib.zip`. You should never change these files, and never add new files yourself. If you want to add anything there, send your additions to `<daisy@dina.kvl.dk>`, and let me do it. The advantage of this is that it will be easy to upgrade your Daisy

installation when a new and improved version with new and improved features, and new and improved bugs, arrives. You just delete the old content of the directory, and install the new files instead. Or better, rename the old directory so you can backup in case anything is horrible wrong with the new version.

- A directory containing files with those parameterizations that are not specific to a single simulation. In that directory you can organize the parameterizations by locations (e.g. put all horizon and column parameterizations containing data measured in Andeby in a single `andeby.dai` file) or by type (e.g. put all crop management specifications in a single `crop-man.dai` file) or both, depending on taste.
- A directory for each simulation, containing a setup file, extra files with simulation specific parameterizations or other data, and all the output (`.dlf`) files produced by the simulation run.

From experience, such an organization saves time in the long run.

8.1 Don't Copy, Derive

A related hint for your own parameterizations: Don't copy, derive. For example, if you want a slightly different grass than the one provided by the standard Daisy library, don't just copy the definition from `grass.dai` and edit the parameter you want to change. And certainly don't change the `grass.dai` directly. Instead, create a derived parameterization as we did in section 4.6:

```
(defcrop "Andeby Spring Barley" "Spring Barley"
  "Spring Barley with modified EpFac as measured
  in Andeby, 2002, for the SuperGrow(TM) cooperation."
  (Canopy (EpFac 1.2)))
```

The advantage of doing it this way twofold. First, if we fix or improve the grass parameterization in `grass.dai`, your derived version will automatically get the improvements when you install the updated Daisy version. Furthermore, we may make changes to the default crop model that requires corresponding changes to the parameterizations. Of course, we do that for the standard parameterizations right away. But if you have made a copy of the parameterizations, it will break when you install the new version.

8.2 Daisypath

Now that you have followed my advice, and spread your files over three directories, how do we tell Daisy where to find it? Well, Daisy uses two sources to find the files:

- A *path* containing information about which directories to look, and
- the *current directory*, which is also the place it will put any log files.

The preferred way to tell Daisy what path to use is through the DAISYPATH environment variable. Unfortunately, how to set this depend on what operating system you are using. Under Unix (including Linux), it has the same syntax as the PATH environment variable, and is set the same place, i.e. your `.bashrc` or `.cshrc` file, depending on your shell.

On MS Windows, it still have the same syntax as PATH⁵, and is set the same way. If you are one of the rare MS Windows users who know how to do that, go ahead and do it. It does depend on which exact version of MS Windows you use, though.

In [Abrahamsen, 1999] there are some examples of how to set DAISYPATH on various systems. However, you may prefer the alternative of putting this information in the setup files themselves. Insert

```
;; Search file files these places.  
(path "." "c:/My Files/daisy/lib/" "c:/Program Files/daisy/lib/")
```

assuming you installed the Daisy standard parameter files in “c:\Program Files\daisy\lib” and your own parameter files in “c:\My Files\daisy\lib”. Note that “.” means “look in the current directory” and that you should use forward slashes (/) instead of backslashes (\) in the path names.

If you start Daisy from the command line, as in section 1, the current directory will be the directory where you run the command. However, you can also specify it explicitly in the setup file, like this:

```
;; Run program here.  
(directory "c:/My Files/Andeby/sim01")
```

An advantage of doing it this way, is that you can start the simulation by dragging the setup file to the `daisy.exe` icon in the file manager. If you don't specify `directory`, it will run the program somewhere silly.

⁵With the disclaimer that the syntax for PATH is different on Unix and MS Windows.

The reason I recommend the DAISYPATH route, is that the setup files become location independent. That is, if you send it to someone else who have installed Daisy in another location, he can still run the setup file unchanged. If you specify `path` in the setup file, he has to change one line. If you specify `directory`, he has to change two lines. What I don't recommend is neither specifying DAISYPATH, nor setting `path`, but using absolute file names. E.g.

```
(input file "c:/Program Files/daisy/lib/fertilizer.dai")
```

It works, but gives a lot of lines that must be changed when moving the files.

A Experimental features

These features are still experimental, and should not be used for “normal” simulations. Use them for simulations where the main purpose is to examine the effect of each feature.

A.1 Ice

A.2 Soil Compaction

A.3 Phosphor

A.4 Dividing and Merging Fields

A.5 Ridging

A special tillage operation is ridging. Ridging is common in tropical agriculture, where it has various beneficial effects, such as preventing erosion, controlling the soil water, and increasing the effective rooting depth, depending on the situation. It is also used under temperate conditions, for example for potato farming in Denmark, where ridging provide better growing conditions in the soil, especially with regard to heat.

In Daisy, ridges only affects the flow of soil and surface water, and a hard-to-penetrate crust is assumed, making it less relevant to Danish potato farming.

You can specify that the field should contain ridges with the `ridge` operation.

```
(ridge (z (0.0 0.0) (0.4 0.0) (0.6 50) (1.0 50.0))  
(R_crust 50 [h]))
```

where

z The ridge height, formulated as $z(x)$, where x is the relative distance from the middle of the ridge. $x = 0.0$ is in the middle of a ridge, while $x = 1.0$ is at the maximal distance. The ridge is assumed to be symmetric. $z(x)$ is measured in centimeter above the unridged soil surface, which means it is in the same reference system as the rest of the model.

R_crust Resistance in crust. Adjust this to control the rate in which water infiltrate.

The ridge must not be taller than the top horizon, as there should be uniform hydraulic properties throughout the ridge.

To remove the ridges, specify any other tillage operation.

Technically, the ridge operation creates a special subsystem within Daisy where water flow sideways into the ridge wall or down through the ridge bottom. You can log this subsystem with the following log parameterization:

Ridge Information about the Ridge subsystem. File: `ridge.dlf`.

A.6 Forced LAI

You can force the CAI to have a specific value for part of the year. This is useful if you have measured the total CAI and want Daisy to use those values for photosyntheses and transpiration (and interception capacity). Here is an example:

```
(defcolumn Andeby default
  "The B.And farm, Andeby, 2003."
  (Vegetation crops
    (ForcedLAI (1987 (100 2.0) (200 4.0))
              (1988 (100 0.0) (150 1.5) (200 5.0))))
  (Soil (horizons (-20 [cm] Ap) (-2.5 [m] C))
        (MaxRootingDepth 60.0 [cm]))
  (Groundwater deep))
```

The code above should be read: In 1987 we start using forced CAI day 100 where it is 2.0 and end forced CAI in day 200 where it is 5.0. We use linear interpolation between these two points, so in day 150 it will be 4.0. Before day 100 and after day 200 we use the simulated CAI values. In 1988 we start day 100 at CAI 0.0, increase linearly to 1.5 day 150, and from there

again linearly to 5.0 day 200, where we switch back to using simulated CAI. All other years we use simulated LAI.

Only parameterizations of the `default` crop model will be affected by the `ForcedLAI` attribute.

If there are multiple crops, the forced CAI will be distributed among them corresponding to the relative size of their simulated CAI. When forced CAI is active, this is the only use of the simulated CAI.

A.6.1 Log models

The simulated CAI will be calculated anyway and can be logged. Most log files will only log the CAI actually used which, when Forced LAI is in effect, will not be the sum of Leaf, Stem and SOrg AI. The new "Forced LAI" log model will log both the used, forced and simulated CAI. By default, it will log the sum of all crops, set `crop` to log a particular crop alone as below:

```
(output "Forced LAI"
  ("Forced LAI"
    (crop "Spring Barley")
    (where "sbarley.dlf"))
  ("Forced LAI"
    (crop "Grass to grain")
    (where "grass.dlf")))
```

B test.dai

```
;;; test.dai -- Sample file using the Daisy libraries.
```

```
(description "Simulation for use in tutorial.")
```

```
;; Use standard parameterizations.
```

```
(input file "tillage.dai")
```

```
(input file "crop.dai")
```

```
(input file "log.dai")
```

```
;; Weather data.
```

```
(weather default "weather.dwf")
```

```
;; We have some very sandy soil.
```

```
(defhorizon Ap FA03
```

```

"Andeby top soil."
(clay 8.0 [%])
(silt 10.5 [%])
(sand 81.5 [%])
(humus 1.12 [%])
(C_per_N 11.0 [g C/g N])
(dry_bulk_density 1.5 [g/cm^3]))

(defhorizon C Ap
  "Andeby C horizon."
  (humus 0.12 [%]))

;; We build the column from the horizons.
(defcolumn Andeby default
  "The B.And farm, Andeby, 2002."
  (Soil (horizons (-20 [cm] Ap) (-2.5 [m] C))
    (border 1 [m])
    (MaxRootingDepth 60.0 [cm]))
  (OrganicMatter (init (input 1400 [kg C/ha/y])
    (root 480 [kg C/ha/y])
    (end -20 [cm]))))
  ;; Measured numbers for groundwater table can be read from a file, like this:
  ;; (Groundwater file "example.gwt")
  ;; But here we assume free drainage.
  (Groundwater deep))

;; Use it.
(column Andeby)

;; Simulation start date.
(time 1986 12 1 1)

(manager activity
  (wait (at 1987 3 20 1))
  (plowing)
  (wait (at 1987 4 4 1))
  (fertilize (mineral (weight 100.0 [kg N/ha])
    (NH4_fraction 0.5 [])))
  (wait (at 1987 4 5 1))
  (progn

```

```

(sow "Grass")
(sow "Spring Barley")
(wait (or (crop_ds_after "Spring Barley" 2.0)
          (at 1987 9 5 1)))
(harvest "Spring Barley")
(wait (at 1987 9 8 1))
(fertilize (mineral (weight 80.0 [kg N/ha])
              (NH4_fraction 0.5 [])))
(wait (at 1987 10 10 1))
(harvest "Grass"
         (stub 8.0 [cm])                ;Leave 8 cm stub.
         (stem 1.00 []))                ;Harvest everything above stub.
(wait (at 1988 4 1 1))
(stop))

;; Create these log files.
(output harvest
  ("N Balance" (when monthly)
               (from 0 [m]) (to -1 [m]))
  ("Crop Production"
   (crop "Spring Barley")
   (where "sbarley.dlf"))
  (checkpoint (when (at 1987 8 7 6))))

;;; test.dai ends here.

```

C Bibliography

References

- [Abrahamsen, 1999] Abrahamsen, P. (1999). Daisy program reference manual. Technical Report 81, Dina KVL.
- [Hansen, 2002] Hansen, S. (2002). Daisy, a flexible soil-plant-atmosphere system model. <http://www.dina.kvl.dk/~daisy/ftp/DaisyDescription.doc>.
- [Hansen et al., 1990] Hansen, S., Jensen, H. E., Nielsen, N. E., and Svendsen, H. (1990). Daisy — soil plant atmosphere system model. Technical Report A10, Miljøstyrelsen.