National Research FLAGSHIPS Sustainable Agriculture

CSIRO

AusFarm User Notes #2

Livestock Management Systems in AusFarm

May 2012

www.csiro.au

Enquiries should be addressed to:
Dr Andrew Moore

CSIRO Plant Industry
GPO Box 1600
Canberra ACT 2601
grazplan@csiro.au

Copyright and Disclaimer

Important Disclaimer

Contents

1. INTRODUCTION

AusFarm is a software tool that allows problems to be analysed with simulation models of physical and biological systems. AusFarm is highly generic, but it has been built primarily to assist decision-making in agricultural enterprises at scales ranging from paddocks to whole landscapes.

This document contains notes to assist users of AusFarm with the construction of simulation analyses that make use of the Stock component to model livestock production. The STOCK component encapsulates the GRAZPLAN animal biology model (Freer *et al.* 1997). This model predicts the intake, nutrition, growth, reproduction and mortality of sheep and/or cattle.

It should be read in conjunction with the AusFarm Help file and the document "AusFarm – a Tutorial" that is distributed with the AusFarm software.

2. CONCEPTS USED IN THE STOCK COMPONENT

Usually a single STOCK module is added to an AusFarm simulation, at the top level in the module hierarchy.

In a grazing system, however, there may be a variety of different classes of livestock. Animals may be of different genotypes (including both sheep and cattle); may be males, females or castrates; are likely to have a range of different ages; and females may be pregnant and/or lactating. The set of classes of livestock can change over time as animals enter or leave the system, are mated, give birth or are weaned. Further, animals that are otherwise similar may be placed in different paddocks, where their growth rates may differ.

Main Flock or Herd					
Index	1	2	3	4	5
Number of animals	2000	160	40	1500	300
Genotype	Merino	BL x Merino	BL x Merino	BL x Merino	BL x Merino
Sex	Wethers	Ewes	Ewes	Ewes	Ewes
Age	1.4 years	4.4 years	1.4 years	4.4 years	1.4 years
Base Weight	53.5 kg	51.4 kg	47.4 kg	48.6 kg	47.2 kg
Fleece Weight	2.23 kg	2.06 kg	1.78 kg	2.04 kg	1.74 kg
Number of offspring		1	1	1	1
Weight of foetus		2.7 kg	2.3 kg		
Paddock	"paddock_5"	"paddock_1"	"paddock_6"	"paddock_1"	"paddock_6"
Tag Value	2	1	1	1	1
Priority Score	3	2	1	2	1
Unweaned Offspring					
Number				1500	300
Genotype				(BL x M) x Dorset	(BL x M) x Dorset
Sex				Mixed Lambs	Mixed Lambs
Age				10 days	10 days
Base Weight				5.2 kg	4.7 kg
Fleece Weight				0.32 kg	0.30 kg

Figure 2.1. The list of animal groups at a particular time during a hypothetical simulation containing a STOCK module. Group 1 is distinct from the others because it has a different genotype and sex. Groups 2 and 3 are distinct because they are in different age classes (yearling vs mature). Groups 2 and 4 are distinct because they are in different reproductive states (pregnant vs lactating). Note how the unweaned lambs are associated with their mothers.

In the STOCK component, this complexity is handled by representing the set of animals in a simulated system as a list of *animal groups* (Figure 2.1). The members of each animal group have the same genotype and age class, but may have a range of ages (for example, an animal group containing mature animals may include four-year-old, five-year-old and six-year-old stock). The members of each animal group also have the same stage of pregnancy and/or lactation; the same number of suckling offspring; and occupy the same paddock.

The set of animal groups changes as animals enter and leave the simulation, and as physiological events such as maturation, mating, birth or weaning take place. Animal groups that become sufficiently similar are merged into a single group. The state of any unweaned lambs or calves is stored alongside that of their mothers; at weaning, the male and female weaners are transferred into two new animal groups within the main list.

In addition to the biological state variables that describe the animals, each animal group has four attributes that are of particular interest when writing management scripts:

Index Each animal group has a unique, internally-assigned integer index, starting at 1. Because the set of groups present in a component instance is dynamic, the index number associated with a particular group of animals can – and usually does – change over time. This dynamic numbering scheme has consequences for the way that animals of a particular kind must be located when writing management scripts.

Paddock Each animal group is also assigned a *paddock*. The forage and supplementary feed available to a group of animals are determined by the paddock it occupies. Paddocks are referred to by name in the STOCK component:

- To set the paddock occupied by an animal group, use the **`move`** event.
- To determine the paddock occupied by an animal group, use the **`paddock`** variable.

It is the user's responsibility to ensure that paddock names correspond to PADDOCK modules or other sources of necessary driving variables.

Tag Value Each animal group also has a user-assigned *tag value* that takes an integer value. Tag values have two purposes:

- They can be used to manage distinct groups of animals in a common fashion. For example, all lactating ewes might be assigned the same tag value, and then all animals with this tag value might undergo the same supplementary feeding regime.
- If tag values are assigned sequentially (starting at 1), they can be used to generate summary variables. For example, **`weight_tag[1]`** gives the average live weight of all animals in groups with a tag value of 1.

Note that animal groups with different tag values are never merged, even if they are otherwise similar.

- To set the tag value of an animal group, use the **`tag`** event.
- To determine the tag value of an animal group, use the **`tag_no`** variable.

Priority Score Finally, each animal group has a user-assigned *priority score* that takes an integer value. Priority scores are used to control the operation of the **`draft`** event. Positive values for the priority score denote the order in which animals should be moved to the available paddocks (with a score of 1 denoting that the animals should be moved to the highest-

quality pasture). Animal groups with the same priority score are placed in the same paddock by a **`draft`** event. Animals with a zero or negative priority score are not drafted.

- To set the priority score of an animal group, use the **`prioritise`** event.
- To determine the priority score of an animal group, use the **`priority`** variable.

3. ELEMENTS OF LIVESTOCK MANAGEMENT SYSTEMS

When constructing a livestock enterprise (or enterprises) in an AusFarm simulation, it is important to ensure that the rules in the management script answer the following questions:

Policy	Enterprise Type	Management Questions
Stocking rate & replacement	All enterprises	• How many animals should be in the system? • When should old or unproductive animals be removed from the system? How many should be removed? • When should replacement animals be added to the flock or herd? Where should they come from?
Shearing	Sheep	• Which sheep should be shorn, and at what times?
Reproductive management	Cows & ewes	• When should mating begin, and how long should it continue? • Should lambs or calves be castrated, and if so when? • When should suckling lambs or calves be weaned? • In cattle, should lactation continue after calves are weaned (i.e. is this a dairy system)?
Sales of young stock	Cows & ewes	• When should weaned lambs or calves be sold? • Should some be retained as replacements?
Supplementary feeding	All enterprises	• Under what conditions should supplementary feed be provided to animals? Which animals? Which supplement? How much?
Grazing management	Multi-paddock systems	• How should animals be moved from paddock to paddock?

In the following sections, some typical ways of answering these questions will be presented, along with fragments of management scripts that can be used to carry out each approach in an AusFarm simulation. These script fragments can be found in the AusFarm repository file **`livestock.afr`**; they can be copied into your own management scripts and adapted to circumstances.

The script fragments will assume that the STOCK module is called **animals**, and that the following variables have already been declared in the management script:

```
define integer no_paddocks  = 3     ! Number of paddocks
define text    padd_name[3] = ['paddock1','paddock2','paddock3']

define integer i    ! Index for looping over the animal groups in a STOCK module
define integer padd ! Index for looping over the paddocks in a simulation
```

4. STOCKING RATES AND LIVESTOCK REPLACEMENT POLICIES

When thinking about livestock numbers, it is usual to think in terms of *stocking rate*, the overall number of animals per unit of land area. Stocking rate is a property-level concept; it should be distinguished from *stocking density*, which is the number of animals per unit area in a specific paddock at a specific time.

It is important to be precise about the units in which stocking rate is expressed. For example, if the numerator of the stocking rate is the total number of animals, then unweaned lambs or calves will count the same as mature animals.

When part of the simulated land area is being used for non-livestock enterprises (cropping or forestry), it is also important to be clear about the denominator of the stocking rate: is it the total property area, or is the area devoted to pasture at a specific time of year?

If a fixed stocking rate is to be used, therefore, its value should be defined as a constant and the exact units given as a comment. It is also useful to compute the denominator of the stocking rate at the start of the simulation, e.g.:

```
define const stock_rate = 7.5 ! Ewes (including replacements)/property ha
define real  farm_area        ! Total property area

on start
{
  set farm_area = 0.0
  for padd = 1 to no_paddocks
    set farm_area = farm_area + @( padd_name[padd] & '.area' )
}
```

Note the way that the indirection function **@()** and the string concatenation operator **&** are used to obtain the paddock areas.

In all the following sample replacement policies, livestock numbers are reduced with the **sell** event and increased with the **buy** event. (Details of all events are given in the AusFarm tutorial.)

4.1 Selling a proportion of the mature animals and replacement to a fixed stocking rate

This policy is suitable for a wether system or a breeding ewe enterprise where the replacement process takes place after all lambs are sold. It assumes that all livestock occupy a single paddock.

```
define const real    stock_rate = 12.0    ! wethers/property ha
define const real    sale_propn = 0.25    ! Proportion of mature animals to sell

define       integer no_to_buy            ! Number of young stock to purchase

each 15 Jan
{
  for i = 1 to animals.no_groups
  {
    if animals.age_months[i] > 36          ! Are the i'th group older than 3 years?
    { animals.sell group=i, number=sale_propn*animals.number[i] } ! Then sell some
  }
                                           ! Acquire young replacement stock immediately
  set no_to_buy = stock_rate * farm_area - animals.number_all
  animals.buy genotype='Merino', number=no to buy, sex='wethers', age=15.0, ...[1]
              weight=48.0, fleece_wt=0.40
}
```

The part of this rule that handles the selling of stock is built using a structure that is highly useful when building management scripts for the STOCK component:

- a **`for`** loop that iterates through each animal group in turn;
- an **`if`** statement that selects only those animal groups that have specific attributes (in this case the animals that are older than 36 months); and
- an event that operates on the selected animal groups (in this case **`sell`**).

Once the sale animals have been removed, the variable **`number_all`** gives the total number of animals left in the system. The number of animals required to build the flock or herd up to the nominated stocking rate is calculated, and a **`buy`** event brings this many animals into the simulation.

There is no reason why the stocking rate value must be a constant. A flexible stocking rate policy can be implemented by calculating the desired stocking rate as a function of (e.g.) pasture conditions, immediately before computing the number to purchase. In this case, the **`stock_rate`** variable would have to be declared without the **`const`** keyword.

4.2 Culling old stock and replacement at a different time

This policy is suitable for a wether system or a simple ewe enterprise where the replacement process takes place after all lambs are sold. This time the replacement animals are placed in the same paddock as the first group of existing animals (it is assumed that all animals move together).

[1] While line continuations are necessary in this document, they are not permitted in management scripts. The scripts in the **`livestock.afr`** repository do not use them.

```
define const real    stock_rate = 12.0      ! wethers/property ha
define const integer cfa_years  = 6         ! Age in years at which old animals are sold

define       integer no_to_buy              ! Number of young stock to purchase
define       text    curr_paddock           ! Paddock currently occupied by the animals
define       integer group_count            ! number of animal groups
define       integer g                      ! loop counter

! Sell cast for age animals
each 15 Dec
{
  group count = animals.no groups
  !move old animals into a new group
  for g = 1 to group_count
    animals.split group=g , type='age' , value=365 * cfa_years

  for g = 1 to animals.no_groups
  {
    if age[g] >= 365 * cfa_years
      animals.sell group=g , number= animals.number[g]
  }
}

each 15 Mar
{
  if animals.no_groups > 0                  ! 2. Store the currently-occupied paddock
  { set curr_paddock = animals.paddock[1] }
  else
  { set curr_paddock = padd_name[1] }
                                            ! 3. Acquire young replacement stock
  set no_to_buy = stock_rate * farm_area - animals.number_all
  animals.buy genotype='Merino', number=no_to_buy, sex='wethers', age=9.0, ...
              weight=40.0, fleece wt=0.80
  animals.move group=animals.no groups, paddock=curr paddock
}
```

Animals of different ages can be combined by the STOCK module into a single animal group (the age structure of the group is followed through time). As a result, selecting and removing the animals that are older than a given age is a two-step process:

- A **`for`** loop is used to work through each animal group in turn. The **`split`** event is used to divide the groups into two, one containing the animals that are at least as old as the nominated age and the other the animals that are younger.
- The older sub-group of animals move into a new group following the **`split`** event. The **`sell`** event then removes them from the simulation.

The replacement process is the same as in the previous example. This time, however, the newly-purchased animals are placed in the same paddock as the rest of the flock or herd via a **`move`** event.

4.3 Culling empty ewes or cows

This management option would usually be used in conjunction with an age- or number-based replacement policy at a different point in the reproductive cycle.

```
define const integer cull_stage = 90  ! Days into pregnancy at which empty
                                      !   animals can be identified

if max(animals.pregnant) = cull_stage ! This will happen once per reproductive cycle
  for i = 1 to animals.no_groups
    if (animals.sex[i] = 'ewe') and not animals.pregnant[i] and not animals.lactating[i]
```

```
      animals.sell group=i, number=animals.number[i]
```

Note how this rule is built using the loop-select-event structure described above.

The use of the **pregnant** and **lactating** variables here is worth going into detail. If the animal group with index *i* is pregnant, then **pregnant[i]** gives the number of days since those animals conceived. If the animal is not pregnant, **pregnant[i]** is zero. The expression **max(animals.pregnant)** therefore evaluates to the length of time since the first mating took place, and the expression **not animals.pregnant[i]** will evaluate to TRUE if and only if the *i*'th animal group is not pregnant – and is subject to culling.

Note also that this script fragment will cull any ewe weaners remaining in the system. To avoid this, test the age of each animal group in the **if** statement.

4.4 A self-replacing cow system

In a self-replacing herd (or flock), the replacement animals appear in the simulation as part of the simulation of the biology of their mothers. What is required is to identify those animals that are to be set aside as replacements so that they are not sold. Tag values (see section 2.1) are designed to assist with tasks such as this.

In the following script, non-pregnant cows and those older than eight years are culled on 15 January. The replacement animals are "set aside" on 1 November, so that the remainder of the heifers (those with a tag value of **HEIFER_WEANERS**) can be sold prior to the culling date if desired:

```
define const integer BREEDER         = 1   ! Define constants to make the script more
define const integer MAIDEN          = 2   !   readable; each denotes a class of
livestock
define const integer HEIFER_WEANER   = 3
define const integer STEER_WEANER    = 4
define const integer STEER_YEARLING  = 5

define const real    stock_rate      = 1.5 ! breeding cows (incl. maiden
heifers)/property ha
define const integer cfa_years       = 8

! Cows are assumed to be mated in July-August and to give birth in April-May.

! Weaning takes place in September

each 15 Sep
{
  for i = 1 to animals.no_groups
    animals.wean group=i, number=animals.number yng[i]

  for i = 1 to animals.no_groups           ! Tag the new weaners
    if (animals.age[i] < 365) and (animals.sex[i] = 'steer')
    { animals.tag group=i, value=STEER WEANER }
    else if (animals.age[i] < 365) and (animals.sex[i] = 'heifer')
    { animals.tag group=i, value=HEIFER WEANER }
    else
    { animals.tag group=i, value=BREEDER } ! This re-labels last year's maiden cows as
```

```
}                                          !   part of the main breeding herd

! Prior to sale of excess heifers, identify the replacement animals

define integer days_to_replace             ! Days until culling
define integer cull_age_today              ! Animals older than this are sold at next
culling
define real    survival_to_replace         ! Survival rate from now to the next culling
define integer replacements_reqd           ! Number of heifers to set aside as
replacements
define real    keep_propn                  ! Proportion to set aside as replacements
define integer group_count                 ! number of animal groups
define integer g                           ! loop counter


each 1 Nov
{
  set days_to_replace = 76
  set cull_age_today  = 365 * cfa_years - days_to_replace
  set survival_to_replace = (1.0 - animals.genotype[1]:death_rate) ^
(days to replace/365)

 ! Separate out the breeding cows that will be culled for age next January,
 ! so that we can count the remainder
 group_count = animals.no_groups
 !move old animals into a new group
 for g = 1 to group_count
    if (animals.tag_no[g] = BREEDER)
      animals.split group=g, type='age', value=cull_age_today

  ! Calculate the number of replacements that will be required, as the difference
between
  ! the desired herd size and the number of pregnant cows expected to survive until
culling
  set replacements_reqd = stock_rate * farm_area
  for g = 1 to animals.no_groups
    if (animals.tag_no[g]=BREEDER) and (animals.age[g]<cull_age_today) and
animals.pregnant[g]
      set replacements_reqd = replacements_reqd - survival_to_replace *
animals.number[g]

  ! Select replacement animals uniformly from the groups of heifer weaners
  set keep_propn = replacements_reqd / animals.number_tag[HEIFER_WEANER]
  for g = 1 to animals.no_groups
    if (animals.tag_no[g] = HEIFER_WEANER)
    {
      animals.split group=g, type='number', value=keep propn*animals.number[g]
      animals.tag   group=g, value=MAIDEN
    }

! Sell cows that are empty or too old

each 15 Jan
{
  group_count = animals.no_groups
  for g = 1 to group_count
    if (animals.tag_no[g] = BREEDER)
    {
      if not animals.pregnant[g]
      { animals.sell  group=g, number=animals.number[g] }
      else
      {
        !move old animals into a new group
        animals.split group=g, type='age', value=365*cfa_years
      }
    }
  for g = 1 to animals.no_groups
  {
    if age[g] >= 365 * cfa_years
      animals.sell group=g , number= animals.number[g]
  }
}
```

- When using tag values, it is usual to write a series of rules that ensures every group of animals has a non-zero tag value at all times.
- Numbering the tag values from 1 allows the tag-based output variables to be used effectively.
- In the script above, the replacement animals are selected uniformly from all the heifers. Other means of selecting the replacement animals are possible. For example, the following rule fragment chooses the heaviest heifers to be the replacement stock. Note how the condition in the **`while`** statement is defined to prevent the loop from executing indefinitely:

```
define real heifer_weights[2]  ! Array used to store the weight of each group of heifers

  ! Select replacement animals from the heaviest heifer weaners
  while (animals.number_tag[MAIDEN] < replacements_reqd) ...
        and (animals.number_tag[HEIFER_WEANER] > 0)
  {
    set heifer_weights = animals.weight    ! This changes the length of "heifer_weights"
    for i = 1 to animals.no_groups         ! Zero the weights of non-heifer groups
      if (animals.tag_no[i] /= HEIFER_WEANER)
        set heifer_weights[i] = 0.0

    for i = 1 to animals.no_groups         ! Find the heaviest remaining heifers
      if (heifer_weights[i] = max(heifer_weights)
      {
        animals.split group=i,type='number',value=replacements_reqd-
animals.number_tag[MAIDEN]
        animals.tag   group=i, value=MAIDEN
      }
  } ! end of "while" loop
```

5. SHEARING

In general, all animals in sheep enterprises should be shorn at least once a year. Shearing is simulated using the **`shear`** event of the STOCK component.

5.1 Shearing all weaned sheep on the same day of year

```
each 15 Jan
  animals.shear
```

This script fragment makes use of the notion of "default parameters". The **`shear`** event has two parameters: **`group`** denotes an animal group (by its index) and **`sub_group`** is a text string that denotes whether the main group of animals, suckling lambs, or both should be shorn. When a parameter is not given in the script, its value is taken to be zero for integer or real parameters, FALSE for logical parameters and the null string for text parameters. As a result, the statement above is equivalent to:

```
each 15 Jan
  animals.shear group=0, value=''
```

By convention, **`group=0`** denotes "act on all groups" for many of the events in the STOCK component, including **`shear`**; **`value=''`** denotes the main group of animals.

The script writer will often want to store the attributes of the fleeces for reporting purposes. These values must be stored before the **shear** event is executed and the values of the wool-related state variables alter. The following statements accomplish this for a system in which sheep are being classified using tag values:

```
define const integer WETHER          = 1   ! Define constants to make the script more
define const integer BREEDER         = 2   !   readable; each denotes a class of
livestock
define const integer MAIDEN          = 3
define const integer WETHER_WEANER   = 4
define const integer EWE_WEANER      = 5

! Shear all weaned stock on a fixed day of year, and store fleece attributes by tag
value

define integer number_shorn[5]        ! Numbers of anmals shorn
define real    clean_fleece_shorn[5]  ! Shorn clean fleece weights (kg/head)
define real    micron_shorn[5]        ! Fibre diameter of shorn fleeces (micron)

each 15 Dec
{
  set number_shorn       = animals.number_tag
  set clean_fleece_shorn = animals.c_fleece_wt_tag
  set micron_shorn       = animals.fibre_diam_tag

  animals.shear
}
```

5.2 Shearing young stock at a separate time of year

Separate shearing dates are scripted using the loop-select-event pattern discussed in section 4.1:

```
define integer t

! Shear weaners in November and the main flock in January

each 1 Jul                              ! Reset fleece variables
 for t = WETHER to EWE_WEANER
 {
   set number_shorn[t]       = 0
   set clean_fleece_shorn[t] = 0.0
   set micron_shorn[t]       = 0.0
 }

each 10 Nov
{
  for t = WETHER_WEANER to EWE_WEANER
    if t <= length(animals.number_tag)  ! Check that there are animals with this tag
    {
      set number_shorn[t]       = animals.number_tag[t]
      set clean_fleece_shorn[t] = animals.c_fleece_wt_tag[t]
      set micron_shorn[t]       = animals.fibre_diam_tag[t]
    }

  for i = 1 to animals.no_groups
    if (animals.tag_no[i] >= WETHER_WEANER) and (animals.tag_no[i] <= EWE_WEANER)
      animals.shear group=i
}
```

```
each 15 Jan
{
  for t = WETHER to MAIDEN
    if t <= length(animals.number_tag)
    {
      set number_shorn[t]       = animals.number_tag[t]
      set clean_fleece_shorn[t] = animals.c_fleece_wt_tag[t]
      set micron_shorn[t]       = animals.fibre_diam_tag[t]
    }

  for i = 1 to animals.no_groups
    if (animals.tag_no[i] >= WETHER) and (animals.tag_no[i] <= MAIDEN)
      animals.shear group=i
}
```

5.3 Shearing all unweaned lambs

```
each 15 Sep
  animals.shear sub_group='lambs'
```

6. MANAGEMENT OF THE REPRODUCTIVE CYCLE

The following events need to be considered when setting up a script for management of the reproductive cycle:

Reproductive Event	Manager Event	Questions to consider
Mating	`join`	• At what time(s) of year should ewes or cows be joined? How long should the joining period be? • Are young ewes or heifers to be mated? If so, are they joined at the same time as mature animals?
Births		• Births will happen without management intervention
Castration	`castrate`	• Should male lambs or calves be castrated (or made cryptorchid), and if so when?
Weaning	`wean` `dryoff`	• When should suckling lambs or calves be weaned? • In cattle, should lactation continue after calves are weaned (i.e. is this a dairy system)?

6.1 A simple reproductive cycle for ewes

This set of rules mates all ewes on the same day of year (excluding animals less than a year old, which avoids unnecessarily mating ewe weaners) and castrates all male lambs shortly after the last lambs are born.

```
! Start of mating
each 1 Feb
{
  for i = 1 to animals.no_groups
    if (animals.age[i] >= 365)
      animals.join group=i, mate_to='Poll Dorset', mate_days=42
}

! After 8 days, and then at 17-day intervals, a proportion of the ewes is mated,
! resulting in multiple groups of animals at different stages of pregnancy. Births take
! place without further intervention during July and early August.

! Castrate male lambs
each 25 Aug
  animals.castrate number=animals.number_yng_all

! Wean all lambs
each 15 Oct
  animals.wean number=animals.number all
```

6.2 Joining maiden ewes separately

In this set of rules, tag values are used to identify ewes that are too young to be joined, those that are being mated for the first time, and mature ewes. The maiden ewes are then mated two weeks after the mature ewes:

```
define integer join_age_years  = 1

define integer BREEDER     = 2
define integer MAIDEN      = 3
define integer REPLACEMENT = 6

! Start of mating
each 1 Feb
{
  for i = 1 to animals.no_groups                    ! Set tag values according to
current age
    if (animals.sex[i] = 'ewe')
    {
      if (animals.age[i] >= 365*(join_age_years+1))
      { animals.tag group=i, value= BREEDER }
      else if (animals.age[i] >= 365*join_age_years)
      { animals.tag group=i, value= MAIDEN }
      else
      { animals.tag group=i, value= REPLACEMENT }
    }

  for i = 1 to animals.no_groups                    ! Start mating of mature ewes
    if (animals.tag_no[i] = BREEDER)
      animals.join group=i, mate_to='Poll Dorset', mate_days=42
}

each 15 Feb                                         ! Start mating of maiden ewes
{
  for i = 1 to animals.no_groups
    if (animals.tag_no[i] = MAIDEN)
      animals.join group=i, mate_to='Poll Dorset', mate_days=42
}
```

6.3 Varying the timing of events with the mating date

In this set of rules the days of year on which reproductive events take place are set relative to the start of mating. (The mating date can then be made the subject of a factor in an analysis.)

```
define text    bull_breed      = 'Angus'
define integer join_day        = 197 ! 15 July
define integer join_age_years  = 2
define integer join_length     = 42  ! days
define integer wean_age        = 120 ! days
define logical castrate_calves = FALSE

define integer gestation       = 281 ! Length of cow gestation, in days
define integer castrate_day

on start
{ set castrate_day = 1 + (join_day + join_length + gestation) mod 365 }

! Start of mating
if day = join_day
{
  for i = 1 to animals.no_groups
    if (animals.sex[i] = 'cow') and (animals.age[i] >= 365*join_age_years)
      animals.join group =i, mate to=bull breed, mate days=join length
}

! Castrate male calves
if castrate_calves and (day = castrate_day)
  animals.castrate number=animals.number yng all

! Wean all calves once the average calf age reaches the nominated value
if animals.age_yng_all >= wean_age
{
  animals.wean number=animals.number_all
  animals.dryoff group=0
}
```

Note:

- The use of a known gestation length and the mod operator to identify the day-of-year on which castration should take place.
- The same technique could have been used for weaning, but instead the age_yng_all variable (which returns the weighted average age of suckling offspring) has been used. When there are no suckling offspring, the age_yng_all variable will return zero.
- In cattle enterprises, the end of lactation must be explicitly marked with a dryoff event.

7. SALES OF YOUNG STOCK

7.1 Sell at a fixed age

7.2 Sell at a target weight

7.3 Sell when lucerne forage runs out

7.4 Retain replacement ewes

8. SUPPLEMENTARY FEEDING

A simple method:

```
! Supplementary feeding

if stock.cond_score_all < 1.0
  supplement.feed supplement='wheat', amount=0.40*stock.number_all, paddock='paddock1'
```

A more detailed method for managing the condition scores:

```
!================================================================================
! Supplementary feeding
!================================================================================

for i = 2 to length(silo.stores)                    ! Ignore store #1 (the hay store)
{ if silo.stores[i]:stored < 20000.0                ! Replenish each supplement store
  { silo.buy supplement=silo.stores[i]:name, amount= 100000.0-silo.stores[i]:stored }
}

!================================================================================

define integer feeding_stage = 1
define logical feeding[11]
define real    feed_rate[11]
define real    low_cs
define text    feed_paddock
define real    crit_dwt
define real    target_wt
define integer days_left

from start
{ if day = wean_day                              ! Work out which set of condition score
  { set feeding_stage = 1 }                      ! thresholds we should use
  if day = flush_day
  { set feeding_stage = 2 }
  if day = join_start
  { set feeding_stage = 3 }
  if day = late_preg_day
  { set feeding_stage = 4 }
  if day = births_day
  { set feeding_stage = 5 }
}
```

```
for t = REPLACEMENT to LAMB_XBRED_F
{ set low_cs = 5.0                                  ! Find the lowest CS in each mob
  for i = 1 to animals.no_groups
  { if animals.tag_no[i] = t
    { set low_cs = min( low_cs, animals.cond_score[i] ) }
  }

  set feed_paddock = padd_name[1]                   ! Find the paddock occupied by this mob
  for i = 1 to animals.no_groups
  { if animals.tag_no[i] = t
    { set feed_paddock = animals.paddock[i] }
  }

  if feed_paddock = FEEDLOT                 ! This ensures that confinement feeding happens
  { set feeding[t] = TRUE }
  if not feeding[t] and (low_cs < critical_cs[feeding_stage,t])
  { set feeding[t] = TRUE }
  else if feeding[t] and (low_cs > critical_cs[feeding_stage,t]+0.1)
  { set feeding[t] = FALSE }

  if feeding[t]                                     ! Vary the feeding rate
  ! Feed ewes up to target condition prior to joining
  { if ((t = MAIDEN_DRY) or (t = EWE_DRY)) and (feeding_stage = 1)
    { if (day <= join_start)
      { set days_left = join_start - day +   1 }
      else
      { set days_left = join_start - day + 366 }
      set target_wt   = animals.genotypes[1]:srw * (1.0 + 0.15 *
(critical_cs[feeding_stage,t] - 3.0))
      set crit_dwt    = (target_wt - animals.base_wt_tag[t]) / days_left
    }
                                        ! Sale weaners are fed to reach target weight
    else if (t = LAMB_MERINO_M) or (t = LAMB_MERINO_F) or (t = LAMB_XBRED_M) or (t =
LAMB_XBRED_F)
    { if lamb sale first day <= lamb sale last day
      { set sale_date_range = (lamb_sale_first_day <= day) and (day <= lamb_sale_last_day)
}
      else
      { set sale_date_range = (lamb_sale_first_day <= day) or  (day <= lamb_sale_last_day)
}

      if (lamb_sale_policy /= 1) or not sale_date_range
      { set crit_dwt = 0.05 }
      else
      { if (day <= lamb_sale_last_day)
        { set days_left = lamb_sale_last_day - day +   1 }
        else
        { set days_left = lamb_sale_last_day - day + 366 }
        set crit_dwt    = (lamb_sale_wt - animals.weight_tag[t]) / days_left
      }
    }

    else
    { set crit_dwt = 0.00 }
     ! see notes for explanation of this calculation
    set feed_rate[t] = max( 0.0, min( max_feed_rate, feed_rate[t] - 0.5 *
(animals.wt change tag[t]-crit dwt) ) )
    set suppt_fed_kg[t] = animals.number_tag[t]*feed_rate[t]

    silo.feed supplement=maint_feed, amount=suppt_fed_kg[t], paddock=feed_paddock
  }
  else
  { set feed_rate[t]    = 0.0
    set suppt_fed_kg[t] = 0.0
  }
}
```

`dW = (stock.wt_change_tag[t]-crit_dwt)`	This is the difference between the current rate of weight gain and the rate that we want to see. If the difference is positive, then we can afford to feed the animals less; if it is negative, we need to feed them more
`dF = -0.5 * dW`	Take a guess at the change in the rate of feeding required to bring the difference down to zero. The units of the multiplier are (kg supplement required)/(kg weight change required). It is a negative value, in order to produce less feeding if the animals are doing too well and *vice versa.* The value -0.5 is only approximately correct, but it's near enough to allow the calculation to converge over a small number of days.
`max(0.0, min(max_feed_rate, feed_rate[t]+ dF))`	The last step is to put lower & upper bounds on the feeding rate. 0.0 is obvious; the upper bound prevents stupid things happening in the case where the animals can't eat enough MJ to meet their needs

9. GRAZING MANAGEMENT

10. INITIALIZING FLOCKS AND HERDS

```
! A simplified example of starting with a flock of different ages in the same paddock

define const integer EWES  = 1
define const integer LAMBS = 2

define flock_size
define real    farm_area    = 750.0
define real    pasture_padd_area  = 400.0
define real stock_rate

on start
{
  !set these from the Generic component
  set stock_rate = SRATE
  set join_start = JOINSTART

  set flock_size=stock_rate * pasture_padd_area
  if flock_size > 0
  { animals.buy genotype='Medium Merino', number=flock size*0.22,sex='ewe', age=30,
weight =50
    animals.tag group=1, value=EWES
    animals.move group=1, paddock='paddock02'

    animals.buy genotype='Medium Merino', number=flock_size*0.21,sex='ewe', age=42,
weight =50
    animals.tag group=2, value=EWES
    animals.move group=2, paddock='paddock02'

    animals.buy genotype='Medium Merino', number=flock_size*0.19,sex='ewe', age=54,
weight =50
    animals.tag group=3, value=EWES
    animals.move group=3, paddock='paddock02'

    animals.buy genotype='Medium Merino', number=flock size*0.18,sex='ewe', age=66,
weight =50
    animals.tag group=4, value=EWES
    animals.move group=4, paddock='paddock02'
  }
}
```

REFERENCES

Freer M, Moore AD & Donnelly JR (1997). GRAZPLAN: decision support systems for Australian grazing enterprises. II. The animal biology model for feed intake, production and reproduction and the GrazFeed DSS. *Agricultural Systems* **54**, 77-126

Contact Us
Phone: 1300 363 400
+61 3 9545 2176
Email: enquiries@csiro.au
Web: www.csiro.au/flagships

CSIRO and the Flagships program

Australia is founding its future on science and innovation. Its national science agency, CSIRO, is a powerhouse of ideas, technologies and skills. CSIRO initiated the National Research Flagships to address Australia's major research challenges and opportunities. They apply large scale, long term, multidisciplinary science and aim for widespread adoption of solutions.