1 Report Model Documentation and Examples

Dean Holzworth (CSIRO, Australia) and Val Snow (AgResearch, New Zealand)

(Text last updated 24 October 2019)

Report is the model in APSIM that is used to generate columns of simulation outputs for further analysis.

The simulations in this example are to show examples of the different reporting capabilities and to provide a test that they continue to produce the correct results.

The intention is that this documentation is read alongside the simulation that produced it to see the detail of the examples. The simulation is to be found under the "Examples" button as Report.apsim.

2 Reporting Basics

Setting up outputs from a simulation requires describing **what** to output and **when** that should be done.

2.1 Properties - setting up what should be reported

Figure 1 shows a screen capture of a simple report. The upper box (Properties) holds descriptions of what should be reported. The general syntax is:

```
[Model].Variable
```

The first part, "[Model]" just above, gives the name of a model in the simulation. Models are almost any component in the simulation - e.g. Clock, Weather, Wheat, Irrigation - including Manager scripts (although these are a slightly special case and more on that below). The name of the model must be in square brackets. As with Manager scripts, Properties uses Intellisense to assist with constructing the output descriptors. For example, see row 1 of Figure 1. After typing "[Clock]" once a "." is typed a drop-down list of possible variables will appear as hints and in Figure 1 the choice from the list was "Today" which gives the date of the simulation output.

Note that model and variable names are case-sensitive. Capitalisation in the right places is important.

The component "ReportSimple" shows four output examples (see also below). The second row (see also below) shows an example of relabelling the output (the "as DairyRainfall" text) - this simply allows the user to give a more relevant label for the output in the output file.

```
[Weather].Rain as DailyRainfall
```

The third row shows that it may be necessary to work through several layers to get to the output wanted. In each case a "." after the variable name gives a list with hints for outputs.

```
[Wheat].Grain.Total.Wt
```

The fourth row (also below) shows the special case of reporting an output from a Manager script. The first part (" [SowingRule]") is the name of the Manager script (so the output text needs to say current with any name changes you might make). Outputs from Managers always need a ".Script" before Intellisense will show the possible outputs.

```
[SowingRule].Script.SowingDepth
```

2.2 Reporting frequency - setting up when reports should happen

Also needed is a specification of *when* an output should be made. The instruction for when to report is in the form of:

```
            [Model].Event or Date(s)
```

2.3 Event

An Event is something that happens, like a stage of the day or a management action. The most-used example of a reporting frequency is:

```
            [Clock].EndOfDay
```

and this creates an output for every "EndOfDay" event - "EndOfDay" is an event created by Clock every day after all the models have done their calculations. Any (almost) event can be used to control the frequency of reporting such as:

```
            [Wheat].Sowing
            [Wheat].Flowering
            [Wheat].Harvesting
            [Wheat].PlantEnding
            [Irrigation].Irrigated
            [Fertiliser].Fertilised
```

where “Wheat” might be any crop. This also shows that multiple triggers for reporting can be used.

It is worth noting that when an event is used to trigger reporting, the output happens immediately rather than at the end of the simulation day. If you are getting strange outputs then consider this as a possible reason. To understand more about what order models do their calculations see https://apsimnextgeneration.netlify.com/development.

2.4 Dates

In addition to specifying events, you can also specify one or more dates in the frequency window. Dates can be specified one per line at be in either dd-mmm or dd-mmm-yyyy form e.g.

```
      1-jul
      1-jul-1980
```

Note that there can be a mixture of events and dates.

2.5 Conditional reporting

Instead of specifying event names or a list of dates, you can specify a conditional function that when evaluated to true will cause a line of output to be written to the report table. For example:

```
      Clock.Today >= new DateTime(2011, 2, 21)
```

will do daily outputs after the 21st February 2011. Conditional functions always start with the if keyword. They are written in C# (like a manager script) and need to be wholely contained on a single line. Multiple conditional statments can be specified - one per line. Another example:

```
      Clock.Today >= new DateTime(2011, 2, 21) && Clock.Today <= new DateTime(2011, 12,
31)
```

This will do daily outputs between 21st February 2011 and the end of 2011.

2.6 Reporting at irregular intervals or specific dates

If reporting is needed for particular days (e.g. to compare against measurements in an experiment) a combination of a Manager and a Report component will do the trick. An example of such a Manager (**ReportHelper**) and two Report components (**ReportOnSpecificDaysEveryYear** and **ReportOnSpecificDates**) are included in this example. Note that when using a Manager to control reporting, the Report frequency in the Report component should be left blank.

[Note that an Operations component could also be used to trigger irregular reporting dates.]

3 Dealing with outputs that have layers

Several of the outputs from APSIM are arrays - water content in the soil layers is a good example of this - and there are some features that make reporting arrays easier in APSIM.

To report all elements of an array, the syntax is the same as reporting a single variable. See for example in Row 2 of "ReportArrays the text

```
            [Soil].SoilWater.SWmm
```

produces one column of output for each element (soil layer) in the array. When a particular layer is wanted then specify that in square brackets so the "[1]" is the top (closest to the soil surface) layer. For example,

```
            [Soil].SoilWater.SWmm[1] as TopLayerWater_mm
```

More often some sort of aggregation is in the array is wanted and for this it is necessary to specify both which elements of the array are to be aggregated and what type of aggregation is wanted.

There are four options for specifying how the array elements should be aggregated - Sum, Mean, Min and Max - the meaning of these is self-evident. These are applied to the array as, for example Mean(x) where the x is the output to be averaged. Use round brackets, capitalise the aggregation type and no spaces.

For specifying how the aggregation is to work there are several options. Giving no layer information at all includes all of the array. A range is specified as, for example [3:6] for the third to sixth (inclusive, here meaning four layers) layers. Giving the colon but no numerical value means from the first (e.g. [:5]) or to the last (e.g. [2:]) elements. Report will not indicate the depth of the layers but the user can either get this information from the Soil input or can output the data using [Soil].Thickness which is an array giving the thickness of each layer in mm.

Some examples of possible array outputs to show the syntax are:

```
            Sum([Soil].Thickness[1:3]) as DepthToBottomOfLayer3
            Sum([Soil].SoilWater.SWmm) as TotalWaterStored
            Sum([Soil].SoilWater.SWmm[1:2]) as WaterStoredTop2Layers
            Sum([Soil].SoilWater.SWmm[1:3]) as WaterStoredTop3Layers
            Sum([Soil].SoilWater.SWmm[4:]) as WaterStoredLayer4AndBelow
            Sum([Soil].SoilWater.SWmm[:6]) as WaterStoredDowntoLayer6
            Mean([Soil].Nutrient.NO3.ppm[1:4]) as MeanNO3ppmTop4Layers
            Min([Soil].Nutrient.NO3.ppm[1:4]) as MinNO3ppmTop4Layers
            Max([Soil].Nutrient.NO3.ppm[1:4]) as MaxNO3ppmTop4Layers
```

Simple, element-by-element, array operations can be included in the specification. For example

```
            [Soil].Nutrient.Urea.kgha + [Soil].Nutrient.NH4.kgha +
[Soil].Nutrient.NO3.kgha
```

will produce one column of data for each soil layer with the total of the amount of N as Urea, NH4 and NO3. Inserting

```
            Sum([Soil].Nutrient.Urea.kgha[1:3] + [Soil].Nutrient.NH4.kgha[1:3] +
[Soil].Nutrient.NO3.kgha[1:3]) as MineralN
```

will give a single column of data with the total of Urea-N, NH4-N and NO3-N dfrom the surface the the bottom of the third layer. These expressions can get more complex as in:

```
            Sum([Soil].Nutrient.Urea.kgha[1:3] + [Soil].Nutrient.NH4.kgha[1:3] +
[Soil].Nutrient.NO3.kgha[1:3]) * 1000 / Sum([Soil].Thickness[1:3] ) as
KgMinN_PerSoilMeter
```

which would give s single (somewhat nonsensical) output of the accumulation in mineral N with depth in the soil.

4 Reporting at Intervals Beyond Every Day

***Note well** – reporting at aggregations other than every day can become complex. It is recommended that the results be critically evaluated to ensure that the reporting specified is as intended.*

APSIM allows reporting at various intervals. Several of these methods have been described above. In addition to the usual [Clock].EndOfDay for daily reporting, [Clock] also has events of EndOfWeek, EndOfMonth, EndOfYear and EndOfSimulation. When the reporting is not every day then it is necessary to consider what the aggregation of the output should be. Some variables would usually be reported as their value on the day – many state variables (e.g. plant biomass) are like this. Others are almost always wanted to be summed over the interval since the last report – drainage and evaporation are good examples here.

APSIM provides several ways to construct the aggregations. Not all the aggregations will be sensible for all outputs and it is up to the user to ensure the sensibility of the instructions to Report. For example, it makes no physical sense to sum the biomass of a plant in a monthly aggregation. It would make sense to report the value at the end of the month or to report the increase (difference) in biomass from the start to the end and it might make sense to report an average biomass. These issues must be considered when constructing more complex Report specifications. The general syntax of aggregated reporting is:

```
AggregationType of [Model].Variable from Start to End as Label
```

The new elements here are *AggregationType*, *Start* and *End*.

##AggregationType AggregationType can be any of:

- Sum of
- Mean of
- Min of
- Max of
- First of
- Last of
- Diff of

Most of these are self-evident. First and Last are chronological values. Diff is the increase in the output variable over the reporting interval – if the variable decreases then it will have a negative value.

##Start The most useful form of *Start* is *[ReportName].DateOfLastOutput* where the first part is the name of the current Report component. *DateOfLastOutput* is pretty much as stated. Other *Start* constructs might be a general, e.g. *1-jan*, or specific, e.g. *1-jan-1982*, date. *Start* can also be an event, e.g. *[Wheat].sowing* or *[Clock].StartOfYear*. When *Start* is a Clock or Report event the aggregation always starts at the start of the day. For events created by crop or other models the event can happen at any point of the day and it is not always clear without careful examination if the current-day calculations and updates will be included or excluded. This is user-beware.

##End *[Clock].Today* is the most useful *End* specification. This means that the end of the aggregation will be controlled by the Report frequency. *End* can also be an event, e.g. *[Wheat].Harvesting* or a general or specific date. As with *Start* and events, treat these with caution. Always consider the interaction between the aggregation interval and the report frequency. For example if the report frequency is *[Clock].EndOfMonth*:

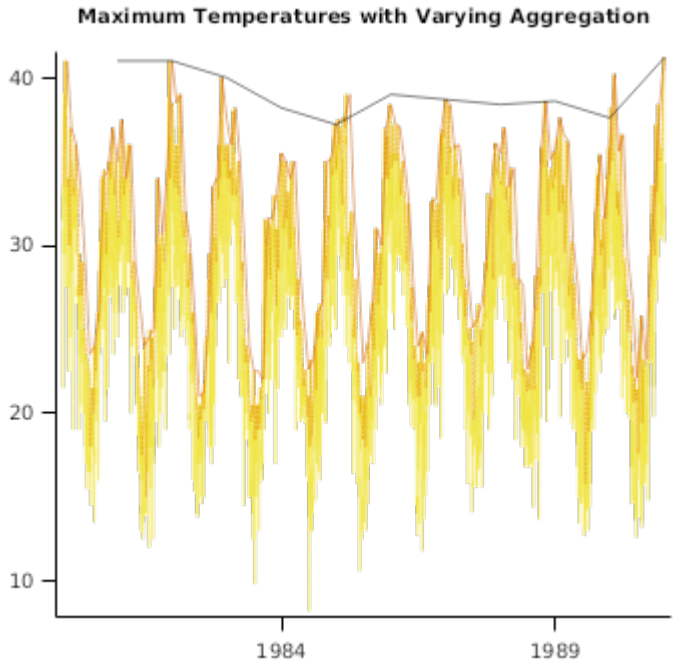
```
Sum of [Weather].Rain from [Report].DateOfLastOutput to [Clock].Today
```

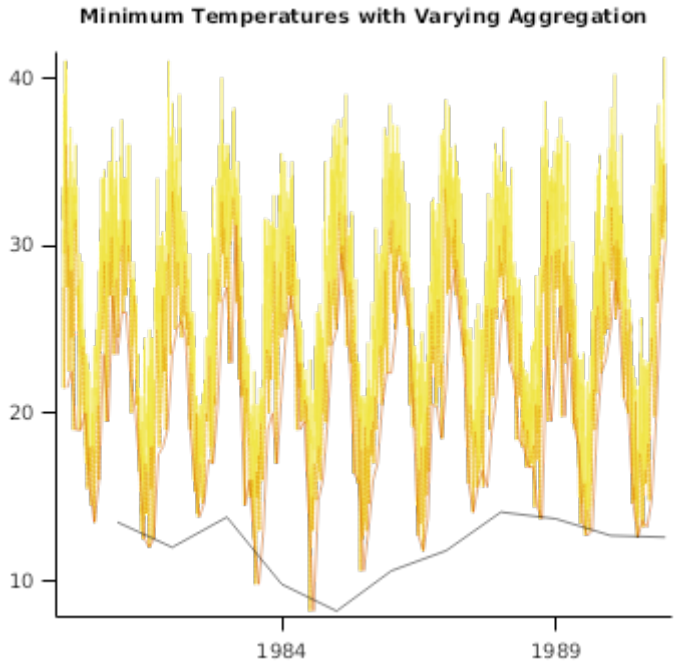
will produce a monthly rainfall total while

```
Sum of [Weather].Rain from 1-jan to [Clock].Today
```

will produce a cumulative rainfall as the months of the year progress and the accumulation will reset again on the next 1-Jan.

##Some examples The figures below show some aggregation examples of daily, weekly, monthly and annual aggregation reporting maximum temperature in various ways. Note that an aggregation to the end of the simulation is also possible. See the accompanying example *Report.apsimx* for details of these aggregations.





5 More Reporting Examples

5.1 Perennial Crop Example

The simulation in this section (titled "Annual Reporting In June") is a multi-year pasture cutting trial simulation. The Report components provide some examples of ways to get useful outputs from a simulation. See the generated documentation but also look at the output specifications in the Reports components.

Note that in the documentation, many of the output specifications are broken over two or more lines. This is only to show then text in the generated PDF. In a Report component, all the text would be on a single line.

Annual Reporting In June

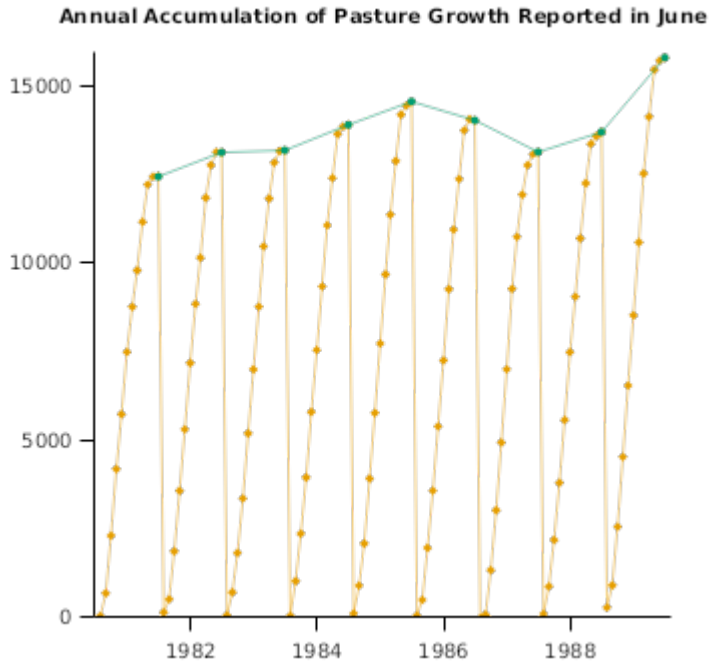
5.1.1 Getting Annual Patterns of Herbage Accumulation

When working with perennial crops that are harvested or grazed frequently, often an output that plots the accumulation of growth during a year is wanted. That output should be zeroed between years. If simulating a site in the Southern Hemisphere, usually the end of the year will be in June or July (winter). Here this is achieved as "CumulativeAnnualNetGrowth" using reporting at the end of the month (see in MonthlyReporting) and:

```
Sum of ([Ryegrass].NetGrowthWt + [WhiteClover].NetGrowthWt)
      from 1-Jul to [Clock].Today as CumulativeAnnualNetGrowth
```

with the output shown as the orange line below. Note that here it is necessary to calculate the output summing ryegrass and white clover. For comparison, the pattern of accumulation is plotted against the annual total (green line) and that was specified in AnnualReporting using:

```
Sum of ([Ryegrass].NetGrowthWt + [WhiteClover].NetGrowthWt)
      from [AnnualReporting].DayAfterLastOutput to [Clock].Today
      as SumYearlyNetGrowth
```



5.1.2 Working with Soil Carbon

When working with soil carbon (or organic nitrogen), gradual changes can hide systematic changes because the totals are so large and while the changes are small, they become important. In this example, using annual reporting only, there are three examples of reporting soil carbon.

The green line is the total soil carbon and is the most basic output. The specification is:

```
      Sum([Soil].SoilNitrogen.TotalC)/1000 as SoilCarbonToday_tonnesPerHa
```

where the "/1000" converts the standard output from kg /ha to tonnes /ha. The output is plotted against the right-hand axis and shows little change.

The orange line is the annual **change** in soil carbon (plotted against the left-hand axis) was created using:

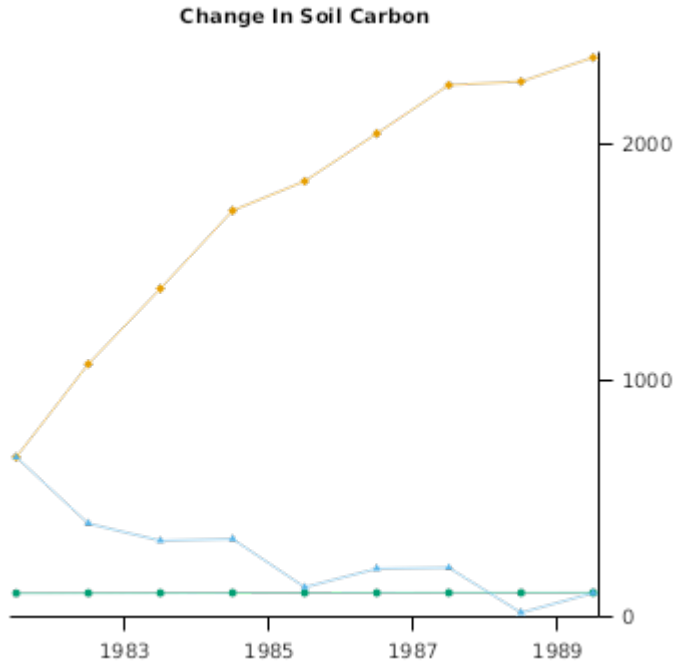
```
Diff of Sum([Soil].SoilNitrogen.TotalC) from [Clock].StartOfSimulation
                  to [Clock].Today as ChangeInSoilCarbon
```

This clearly shows that soil carbon is increasing most years but that the increase is diminishing towards the end of the simulation. Note that for this simulation the line could also have been created using:

```
Diff of Sum([Soil].SoilNitrogen.TotalC)
      from 01-jul-1980 to [Clock].Today as ChangeInSoilCarbon
```

but the first version is more generic in that if the start date of the simulation is changed it will still be valid. The second version can be more useful where there is an initial spin-up period of several years before changes in soil carbon are of interest. Another form of soil carbon output that can be useful is the change within any year. That (the blue line) shows even more clearly the slow stabilisation of the total carbon and is specified using:

```
Diff of Sum([Soil].SoilNitrogen.TotalC)
      from 01-jul to [Clock].Today as AnnualChangeInSoilCarbon
```



5.1.3 Daily and Monthly Leaching

The example below shows daily, monthly, and cumulative annual leaching. The daily output is from "DailyReporting" and is simply:

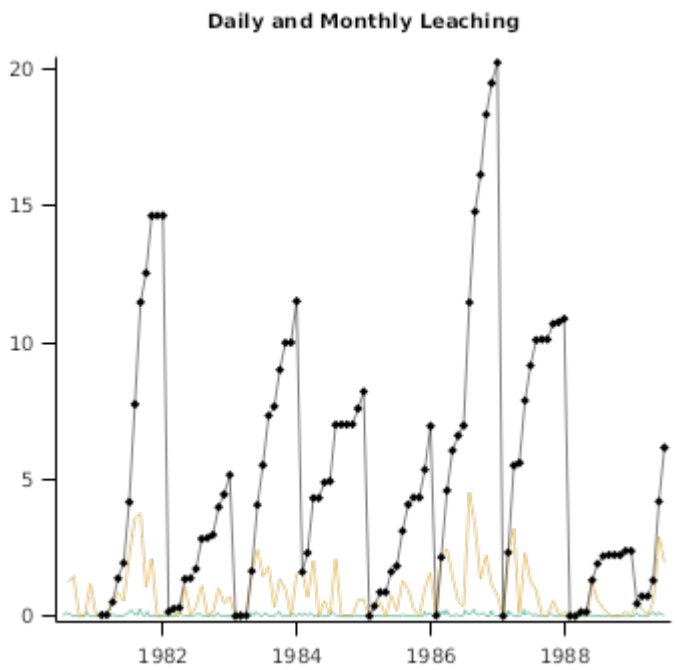
```
[Soil].SoilWater.LeachNO3
```

The other two outputs are both calculated in "MonthlyReporting" as:

```
Sum of [Soil].SoilWater.LeachNO3 from [MonthlyReporting].DayAfterLastOutput
      to [Clock].Today as SumLeaching_kgPerHa
```

and

```
Sum of [Soil].SoilWater.LeachNO3
      from 1-Jan to [Clock].Today as CumLeaching_kgPerHa
```



5.2 Annual Crop Example

In annual cropping simulations, users often want to know the values of outputs only during the period that the crop is in the ground with the outputs summarised over the interval between sowing and harvesting. The examples in this section show how to do this and also show some results that might be unexpected to look out for.

Note this example also shows and example of specifying dates in the report frequency e.g.

```
1-jan
1-feb
1-mar
2-jan-1980
```

See ReportSpecificDates report model.

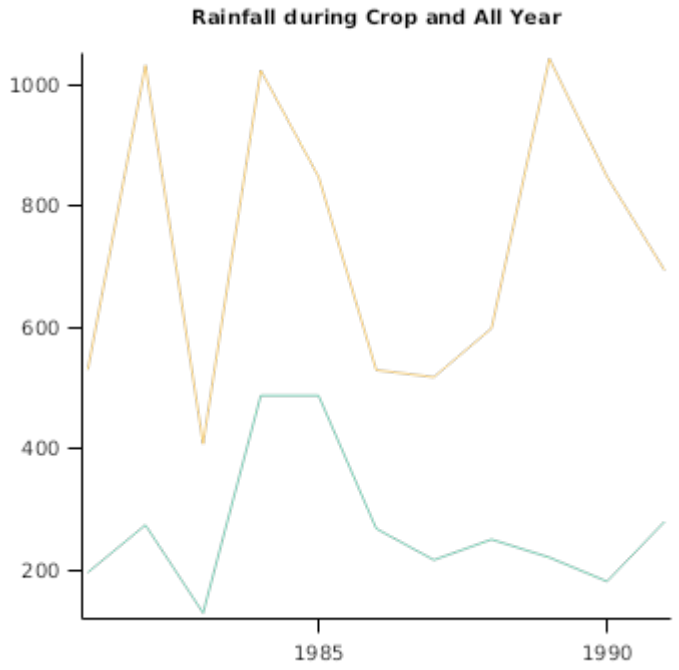
ContinuousWheatExample

5.2.1 Comparing Rainfall during the Crop and All Year

The example below shows, for each calendar year, the amount of rain that fell when the crop was in the ground as compared to the annual rainfall. The Report frequency in "ReportInCropAnnually" is {Clock].EndOfYear and the output specifications are:

```
Sum of [Weather].Rain from [Wheat].Sowing
      to [Wheat].Harvesting as InCropRainfall

Sum of [Weather].Rain from 1-Jan to
      [Clock].Today as AllYearRainfall
```



5.2.2 Soil Water storage during the Cropping Phase

The next example also uses outputs from "ReportInCropAnnually" and shows how to get information about soil conditions during the crop. The output specification:

```
Mean of Sum([Soil].SoilWater.SWmm) from [Wheat].Sowing
      to [Wheat].Harvesting as InCropMeanSoilWater
```

gives the whole-soil profile soil water storage as a mean while the crop is in the ground. The output is reported at the end of the year but is for the period from sowing to harvesting only. If the interest is only in the water storage in the top three layers then the specification would be:

```
Mean of Sum([Soil].SoilWater.SWmm[:3]) from [Wheat].Sowing
      to [Wheat].Harvesting as InCropMeanSoilWaterTopThreeLayers
```

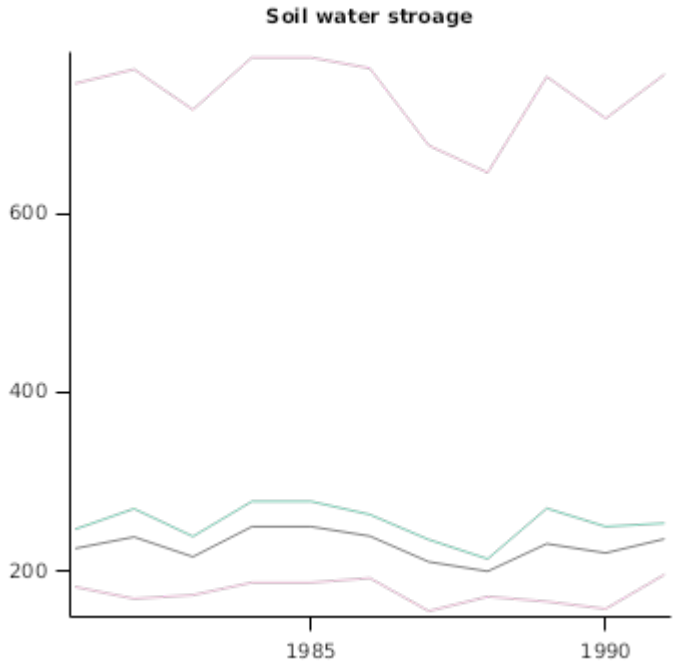
and that produces the black line in the figure below. If the interest was in the water storage during the vegetative stages compared to the reproductive stages then:

```
Mean of Sum([Soil].SoilWater.SWmm[:3]) from [Wheat].Sowing
      to [Wheat].Flowering as VegetativeMeanSoilWaterTopThreeLayers
```

and

```
Mean of Sum([Soil].SoilWater.SWmm[:3]) from [Wheat].Flowering
      to [Wheat].Harvesting as ReproductiveMeanSoilWaterTopThreeLayers
```

produce the green and red lines to show that there was less water storage during the reproductive stages. Of course other types of aggregation could be reported such as the difference durign the phases, min/max etc.



5.2.3 Reporting Yield and When Things Can Seem to Go Wrong

The Report model is a powerful and very useful component to get the information needed from the simulation - but there are some traps that should be noted.

The example below is primarily based on "ReportInCropAnnually" which has a reporting frequency of [Clock].EndOfYear. That Report uses an output specification of:

```
Max of ([Wheat].Grain.Wt * 10000 / 1000) from [Wheat].Sowing
      to [Wheat].Harvesting as FinalYield_kg_Ha
      // * 10000 / 1000 to convert from g/m2 to kg/ha
```

(note that the double slash, //, denotes a comment which is shown in green in the user interface). This produces the green line in the figure below. It is immediately noticeable that the yeild in late 1984 was exactly the same as the previous year - this should raise red flags. Alternative outputs that might be expected to give the same values are:

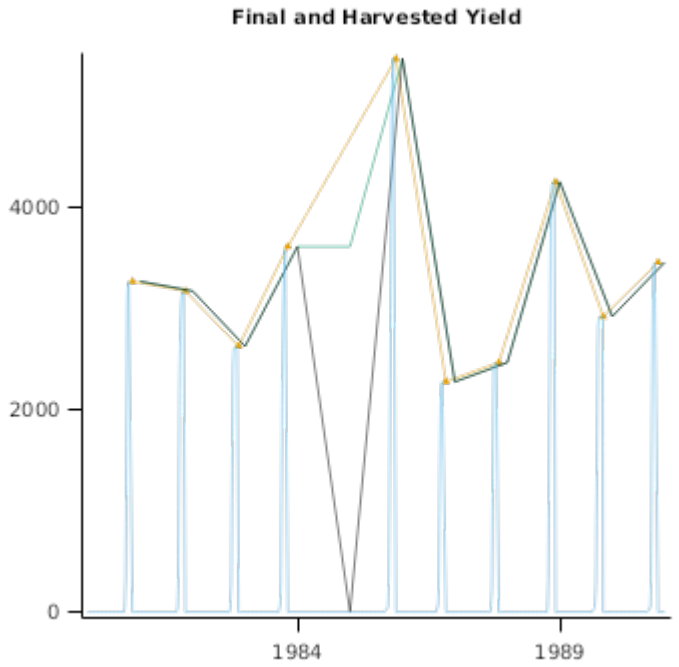
```
Max of ([Wheat].Grain.Wt * 10000 / 1000)
      from [ReportInCropAnnually].DayAfterLastOutput
      to [Wheat].Harvesting as FinalYield_kg_Ha_v1
```

also at a reporting frequency of [Clock].EndOfYear (the black line), or

```
([Wheat].Grain.Wt * 10000 / 1000) as HarvestedYield_kg_Ha
```

with a reporting frequency of [Wheat].Harvesting which produced the orange line (see ReportGrainOnHarvesting), or for more detail report the same output on a daily basis (see ReportGrainDaily) which is the blue line.

The additional reporting clearly shows that there was no grain yield in 1984 and further investigation (the summary file) shows that there was no crop sown because the sowing conditions were not met. The apparent yield in the green line is not an error in Report but is a result of the specification in combination with the possibility that there will not be a crop every year. When Report does its work at the end of 1984 it outputs the maximum grain yield from the last sowing event to the last harvesting event from the Wheat model - and these were the ones from 1983. This has also affected the green line in Section 5.2.1 and all the lines in 5.2.2 but they were not so immediately obvious. The salient message is to consider the output specifications within the context of the management conditions in the simulation.



6 Grouping

Report has the ability to produce temporally aggregated variables grouped by another variable. The example in this section has two contrasting reports that summerise variables seaasonally in 2 different ways.

1. The first report (SeasonalOverall) temporally aggregates from start to end of simulation and has a reporting frequency of end of simulation. It also has a 'Group By' of [Weather].Season which, at the end of the simulation, will split each variable into values for each season resulting in multpile rows of output, one for each season (4 rows of output).
2. The second report (SeasonalByYear) has multiple reporting frequencies, one for each end of season. This produces a year x season output table.

The group by keyword

To use the 'Group By' capability, variables need to be aggregated like this:

```
from [Clock].StartOfSimulation to [Clock].EndOfSimulation
```

The reporting frequency must be:

```
[Clock].EndOfSimulation
```

The 'Group By' variable can be any APSIM variable e.g.

```
[Clock].Today.Month
[Weather].Season
```

The on keyword

This simulation also introduces the *on* keyword. This keyword changes the timing of variable collection. By default (as in SeasonalByYear report), the values of variables are collected at the end of each day. When the variable is *[AGPWhiteClover].HarvestedWt* this results in a lot of zeros being collected because HarvestedWt is zero every day except when there is a harvest. This makes the mean calculation very low as it has a lot of zero values for HarvestedWt.

In the *SeasonalByYearWithOnKeyword* report the *on* keyword is used to specify that the values for HarvestedWt should be collected whenever there is a *[SimpleGrazing].Grazed* event. In other words, a value for HarvestedWt is only collected whenever there is a harvest. Note how the mean values are much higher than the previous report.

Seasonal