

ACCUM	2
CLOCK	3
ECONOMICS	5
LOG	7
INPUT	8
MANAGER	10
MANAGER2 / SCRIPT	15
MET	26
OPERATIONS	30
PATCHINPUT	31
R LANGUAGE COMPONENT	33
REPORT (OUTPUT FILE)	35
SILOINPUT	36
SOI	37
TCLLINK	39
TRACKER	41
VENLINK	44

Accum

The APSIM accum module enables a user to accumulate values for variables over a period of days so that they may be used for management or reporting actions.

Any variable from a module included in a simulation can be accumulated over a defined period of time.

Examples:

Rain[5] – accumulates rainfall over the preceding 5 days.

Clock

Description

The APSIM Clock module is a mandatory module for having the simulation progress through time.

Module Output Variables

The APSIM Time Clock Module can provide the values of several state variables for reporting to an output file or use by other modules.

Name	Units	Description
Day		Day of Year (0-366)
Year		Year
Day_of_month		Day of Month (0-31)
Month		Month (1-12)
Start_week	True/False	Flag for start of week [†]
End_week	True/False	Flag for end of week [†]
Start_month	True/False	Flag for the start of each month
End_month	True/False	Flag for the end of each month
end_year	True/False	Flag for the end of year
today		Today's Julian Date
day_of_year		Today's day of year (same as 'day')
month_str		Today's month name (eg 'Jan')
dd/mm	dd/mm	Today's Date (eg '10/12')
dd/mm/yyyy	dd/mm/yyyy	Today's Date (eg '10/12/1990')
dd_mmm	dd_mmm	Today's Date (eg '10_dec')
dd_mmm_yyyy	dd_mmm_yyyy	Today's Date (eg '10_dec_1990')
mm/dd/yyyy		Today's Date (eg '12/15/1990')
mmm/dd/yyyy		Today's Date (eg 'Dec/15/1990')
time	hh:mm	Daily Time in 24 hour format
Simulation_start_day	Day of year	Day of year of simulation commencement
Simulation_start_year	Year	Year of simulation commencement
Simulation_end_day	Day of year	Day of year of simulation finish
Simulation_end_year	Year	Year of simulation finish

[†] This assumes a 52 week year with the duration of each week is adjusted so that the first week starts on 1 st of January and the last week ends on the 31st of December.

Using Sub-Daily Timesteps

It is possible to use timesteps to one minute in resolution within the current APSIM framework with the following constraints.

- The timestep is constant throughout the simulation
- The timestep is a factor of 1440 mins/day (rational fraction of one day)
- The met file contains data at the same timestep resolution for all days within the simulation period
- The simulation will start at the beginning of a day and finish at the end of a day
- It is the user's responsibility to ensure that all modules with the simulation are both capable and appropriately configured to operate on these timesteps.

```
start_date = 1/1/1988      ! simulation starting date
end_date = 31/12/1988     ! simulation ending date
time-step = 60 (min)      ! simulation timestep
```

The example above will specify the clock to step through the simulation essentially with 24 tick cycles per day. There is no thorough testing for synchrony of modules though some modules will give error messages if they perceive possible timestep errors. If the timestep parameter is not specified the clock module will default to a timestep of 1 day (1440 mins), that is, one tick cycle per day.

Events

The clock module produces several events that can be useful, particularly when using the TRACKER module.

Name	Description
Start_simulation	Published once at the start of every simulation
Tick	Published at the start of every day
End_day	Published at the end of every day
Start_week	Published at the start of every week
Start_month	Published at the start of every month
Start_year	Published at the start of every year
End_week	Published at the end of every week
End_month	Published at the end of every month
End_year	Published at the end of every year

Economics

Applicability

The economics module in APSIM was developed primarily to serve as another input to the management module - as an index of farm "stress" that modifies the risk aversion of the farm manager.

To this end, it simply maintains a cash balance through the simulation, which monitors all financial activity (eg. income, expenses, loan repayments).

The bottom line is that all events that occur on a farm have a cost, and these events **MUST** be captured by the economics module for a realistic simulation.

This includes not only crop income & expenses, but also annual operating expenses and capital replacement.

Operation

As with all apsim modules, the economics module responds to events. The main events of interest are "income" and "expenditure" events. The module will generate an "end_financial_year" event that other modules may have an interest in, internally it will calculate loan repayments on this date.

Typically, these events are generated from within the manager module, for instance when sowing or harvesting a crop, preparing a seedbed, spraying for weeds and so on.

The events contain parameters; typically most define a rate, category and area. The module will write a line to the summary file describing the outcome of the event, and also a line in the cash journal.

Expenditure Events

```
economics expenditure category = seedcost, name = wheat, rate = 120 (kg/ha), area = 200 (ha)
```

Here the module will look up the entry "wheat" in its "seedcost" section, multiply by rate by area and subtract the total from the current balance.

Note that while units are specified, no unit conversion is performed. As well the paddock area must be specified.

Income Events

```
crophyieldTONNES = wheat.yield / 1000.0
protein = wheat.grain_protein
economics income category= cropprice, name = wheat, yield = crophyieldTONNES, protein = protein, area = 200 (ha)
```

...

```
crophyieldTONNES = sorghum.yield / 1000.0
economics income category= cropprice, name = sorghum, yield = crophyieldTONNES, area =
200 (ha)
```

The first example shows the manager calculating a crop yield in tonnes and grain protein content, then telling the economics module (wheat price is dependent on quality). The second is a simpler version of the same.

Capital replacement

The simulation starts with an initial capital investment loan. All machinery items have an initial age, and are replaced when they exceeds thier useful life. Payments for these loans are made at the end of each financial year.

Additional machinery

Additional machinery is created by duplicating an existing node in the the ApsimUI tree. Simply rename the node, and check that the "Apsim Name" is unique.

Log

What is the Log Module?

The log module allows the user to track information and message flow through a simulation. The log file is formatted as a simple XML text document. These documents are often used to debug messaging problems within a simulation. Note that turning debug information on, and writing this to the log file, does slow down simulation execution.

Parameterisation

The Log module has one optional parameter:

logfile

which is the name of the log file

If not specified, the filename will be logfile.xml

Input

Introduction

Most simulation require data, other than module parameters to be available to all modules. This service is provided by the APSIM input module.

The input module is an instantiable module. This means that APSIM can run zero or more instances of the input module. The instance name **met** is special. If the input module is running as met , it will read its data from a single section called weather , otherwise it will read its data from a single section called data.

There are two types of data that the input module can be used to read, temporal data (eg. weather data), and data that is constant for a given simulation . This is why there are potentially two parts to the input section, an optional constants section followed by an optional temporal data section.

Input file format

```
alpha = one two three
beta = one two three four five six seven
Lumpia = -7.11

year day rug(1) rug(2) paxt
() () (MJ/m2) (MJ/m2) (oC)
1988 1 20 21 33.0
1988 2 23 24 33.8
1988 3 23 24 32.5
1988 4 19 20 30.8
1988 5 17 18 28.2
1988 6 22 23 29.0
1988 7 22 23 28.4
1988 8 25 26 31.2
1988 9 26 27 33.6
1988 10 25 26 34.4
1988 11 22 23 31.6
```

Constants part

Data in the constants section must be in the format “apsim_name = values ...”. Where apsim_name is the name of the APSIM variables and values is one value in the case of scalar data and one to 100 values separated by spaces in the case of array data. There is no limit to the number of variables that may be defined here.

Each value can be at most 50 characters long, with the added restriction that a given line may be at most 2000 characters long.

Temporal data part

Data in the temporal section is in tabular format, very similar to the output of the report module. This is deliberate, so that the input module can potentially use output from the report module.

Headings line

First there must be a line of space separated headings. In the case of scalar data, a heading is the APSIM name of the variable. In the case of array data is the heading is the APSIM name of the variable immediately followed by the array

index in parenthesis (no spaces allowed). For a given APSIM array variable, the array elements must appear in order beginning with element 1.

There may be up to 100 headings, each of which may be up to 35 characters long, with the added restriction that a given line may be at most 2000 characters long. The APSIM variable year must be present in the temporal data table. Either the APSIM variable day (meaning day of year) must be present or the APSIM variables month and day_of_moth must be present.

Units line

Then the units line must follow - a space separated units string for each heading. In the case of array data, the units string belonging to elements of the same APSIM variable must be identical.

Each units string may be up to 35 characters long, with the added restriction that a given line may be at most 2000 characters long.

Data lines

Then a line for each day must follow with the space separated data elements corresponding to each of the headings in order.

Each units string may be up to 50 characters long, with the added restriction that a given line may be at most 2000 characters long. Normally, each day for the APSIM simulation must be present in order.

Sparse Data

If the constants part of the section has a scalar APSIM variable allow_sparse_data is present, and its value is "true", then sparse data is permitted, and each day of the simulation need not be present. An attempt by other modules to get data on a day with no data will normally get the zero elements in its numvals field. Defaults for sparse data can be set in the constants section. In this case, if data is present in the temporal section on that day, the values specified in the temporal section will be returned. Otherwise the data specified in the constants section will be returned.

Met calculated data

If the input module is instantiated as met , then some calculated variables are available:

The real scalar day_length is calculated from latitude and made available.

Examples

Example input data can be found in the Apsim\Examples directory.

Manager

What is the manager module?

The manager module provides the capability to specify a set of rules using conditional logic during simulations to control the actions of modules within APSIM.

It does this by using “if” constructs created by the user. It also allows the user to create their own variables and define these as a function of other variables within APSIM.

This documentation only gives a brief insight into the possibilities achievable via the APSIM manager module.

How does it manage?

This module manages by issuing messages to modules in the system, many of which are conditional upon states or events within the modules during simulation.

For example:-

```
if (day = 100) then
  fertilise apply amount = 10 (kg/ha), type = urea (), depth = 50 (mm)
endif
```

Here the fertilise module will be sent a message containing a directive to apply (the action) fertiliser when the condition is satisfied. It receives a data string (the underlined text) which further describes the action. The manager module can broadcast a message to all modules by substituting the keyword 'act_mods' in the place of the module name. This capability is useful for multi-point simulations where a sow message needs to be sent to multiple points.

Mathematical operators

The following mathematical operators and reserved words are allowed in APSIM manager files.

Operator	Description
-	Subtraction
+	Addition
*	Multiplication
/	Division
^ or **	Exponent (eg. x**2 is the same as x 2)
=	Equality
<	Less than

Operator	Description
>	Greater than
<>	Not equal to
<=	Less than or equal to
>=	Greater than or equal to
()	Brackets
If	Logical IF
then	Logical THEN
elseif	Logical ELSEIF
else	Logical ELSE (for alternate logic)
endif	Logical ENDIF
or	Logical OR
and	Logical AND

Rules

- Names must begin with a letter.
- Numbers must begin with a digit.
- Literals must begin and end with an apostrophe.

Character set

The manager uses the following character set:

Character	Description
a to z	letters – case insensitive
0 to 9	Digits
_	Underscore
%	Percent sign

Character	Description
.	Period or decimal point
[]	Square brackets
()	Parentheses
-	Minus sign
+	Plus sign
*	Asterisk
/	Slash
'	Apostrophe
=	Equal sign
<	Less than
>	Greater than
	Blank
^	Caret

Manager Functions

Manager functions may not have any spaces. This applies from the first character of the function name to the terminating bracket. The manager has the following functions:-

Function name	Description
date	returns the julian day number of specified date. eg. date('1-oct'). The date must be a literal enclosed in single quotes.
date_within	returns 1 if "today's" date is within the range specified, otherwise returns 0. eg. date_within('1-oct,31-oct') The pair of dates must be two date literals separated by a comma, and the whole argument must be enclosed in quotes. Note the lack of quotes near the comma.

Function name	Description
nearest_int	Returns the nearest integer to the value specified. eg. nearest_int(var1). Here 'var1' must be a numeric variable. It cannot be a literal or an expression.
paddock_is_fallow	Returns 1 if there are no crops in the ground. e.g. if (paddock_is_fallow() = 1 and today = date('1-jun')) then wheat sow ... endif
add_months	Takes 2 parameters, a date and the number of months to add to the date. The new date is then returned. e.g. gsrDate = date('1/9/2005') gsrDate = add_months('gsrDate', 1) The whole argument to add_months must be enclosed in single quotes. The number of months can be positive or negative.

Dates may take the following forms:

- 30/6/95
- 30/6/1995
- Jun
- 30_Jun
- 30_Jun_1995
- 30-jun
- 30-jun-1995

For example:

- date('30/6/95') returns the julian day number for 30 jun 1995
- date('Jun') returns the julian day number for 1 jun for current year.
- date('30_jun') returns the julian day number for 30 jun for current year.
- date('30_jun_1995') returns the julian day number for 30 jun 1995

For related chronological or date variables which can be used by the manager module, see the documentation for the CLOCK module.

For example

day - returns the day of the month

dd/mm/yyyy - returns the day, month and year of the given day.

See the CLOCK module for further details.

Using the manager to send actions to other modules

The APSIM manager module can be used to invoke any action available by any module. Possible actions include:

- Resetting individual module values
- Reinitialising all data in modules to a given state
- Sowing, harvesting or killing crops.
- Applications of fertiliser, irrigation or tillage to soil.

Refer to the individual module's documentation for a list of available actions and examples of usage. Refer also to the module's sample files for further examples.

Manager2 / Script

1. Introduction

APSIM's 'Manager2' component is a powerful management tool for controlling APSIM simulations. It differs significantly from the original manager which had its own programming language by giving the user the option of coding their scripts in either VB.NET or C#, both of which are standardized languages developed by Microsoft for the .NET platform on Windows (also available on Linux via Mono). Whereas the initial manager had somewhat limited functionality yet was simpler to code, 'Manager2' has vastly increased functionality with a small increase in complexity and the advantage of utilizing a well-known language that new-comers will hopefully be familiar with.

Coding a new Manager2 script is identical to coding a new APSIM module in VB.NET or C#, however the environment you use is somewhat different. Where one would usually use an IDE such as Visual Studio or MonoDevelop to create a new APSIM module the Manager2 component provides an editor within APSIM that automatically imports the required DLLs and provides a code 'skeleton' as a starting point. This tutorial will introduce some basics of the VisualBasic.NET programming language with the purpose of getting you into programming APSIM Manager2 modules as fast as possible. This is by no means a comprehensive introduction to VB.NET or object oriented programming, there are plenty of those available on the internet (a good starting place – and community – for learning to program in any language is www.dreamincode.net). If you are already well-versed in either VB.NET or C# then it is still advisable to briefly skim this section to pick up a few APSIM-specific pieces.

2. Programming Basics – VisualBasic.NET

2.1 Imports

As far as coding for APSIM Manager2 scripts goes most people will never need to modify the Imports section of the script as all the required Imports are already set. Imports are used to add references to external libraries you use within your code.

2.2 Classes

For the purpose of this tutorial a Class may be thought of as a 'container' of sorts for your code. Defined using the following terminology it is compulsory to put everything except 'Imports' statements inside a Class:

VB.NET

```
Class Script
End Class
```

2.3 Variables

A Variable is a container used to store information your code uses. Variables have names which can consist of any number of letters, underscores and numerals (but cannot begin with a numeral). In VB.NET all variables must be 'declared' before they may be used. The general format for doing so is as follows:

```
Dim *var_name* As *VarType*
```

Where **var_name** is the name of your variable and **VarType** is the type (see ‘Types’ - 3.3.1). Optionally you may also set the value of this variable at the same time as you declare, example:

```
Dim limit As Integer = 10      'declare a variable called 'limit' and set it to 10
```

Once the variable has been declared it may then be used just like any other programming language.

2.3.1 Types

In VB.NET and C# all variables have a Type and can only store values that are of the same Type (there are exceptions but these are beyond the scope of this tutorial). While there are a large number of types available to use, most of the time you will only be using a handful:

```
Dim a As Integer      'used for whole numbers between -2147483648 and 2147483647
Dim b As Long         'used for very large whole numbers between +/- ~9.2e18
Dim c As String       'used for holding string values (text)
Dim d As Single       'used for holding numbers with decimal point
Dim e As Double       'as with single, but with greater precision
Dim f As DateTime     'used for holding/processing dates and times, very useful
```

There are also a number of ‘APSIM’ types available for use, one for every APSIM model in fact. These are very useful as they allow the programmer easy access to that model’s output variables as well as events that it subscribes to and publishes.

2.3.2 Arrays

Arrays are data structures used to hold any number of values of the same type that are usually all related, such as the soil water contents of multiple soil layers in the profile. Arrays can be ‘n-dimensional’, that is may consist of any number of dimension, however for the sake of simplicity we will only refer to 1-dimensional arrays here (numerous examples of arrays with more dimensions may be found online). When using arrays (1-D) it can be helpful to think of them as a collection of ‘cells’, similar to a row or column of data in a spreadsheet. Each ‘cell’ can hold a single value and is accessed using the name of the array variable as well as an ‘index’. Arrays are declared and ‘typed’ just like any other kind of variable, several examples of declaration and usage are below, of particular note is the ‘.Length’ property that every array has which is very useful in ‘For’ loops (see 3.5).

2.3.3 Scope

‘Scope’ is a concept that affects the availability of a given variable for use in different parts of code. A variable may be declared within a Class or Sub/Function (see 3.4), the exact location defines the ‘scope’. When a variable is declared within a Class (a ‘global’ variable) it is accessible from any Sub or Function within that class, however when a variable is declared within a Sub or Function it is only accessible inside that Sub or Function. Global variables are usually used for model parameters, inputs and outputs. If it is at all possible to declare a variable inside a Sub or Function instead of globally it is best to do so, having too many global variables can lead to confusion.

2.4 Subs and Functions

Sub is short for ‘subroutine’ and is used to contain a piece of code within a class that performs a specific task. Subs are generally ‘called’ (or ‘run’) from within other Subs; however some of the Subs in your manager script will be made visible to APSIM which will call them as part of the simulation (see section 4) – thus forming an ‘entry point’ (a point where some external program calls your code). Depending on the intended function of the code within the sub you may want to pass some parameters into it when it is called, see the code example in section 3.5 to see how this is done. A ‘Function’ is like a ‘Sub’ except the code within it must return a value before it finishes. All code statements, barring variable declarations and ‘Properties’ (beyond the scope of this tutorial), must be contained within either a ‘Sub’ or a ‘Function’.

2.5 For and While loops

‘For’ and ‘While’ loops can be used for any number of repetitive tasks such as cycling through the elements of an array or the paddocks in a simulation. While technically they may be used interchangeably, ‘For’ loops are usually used for rigidly defined tasks (such as processing every element in an array) whereas ‘While’ loops are usually used for more open-ended tasks, such performing a given action until the required result is achieved. Examples:

```
'sum an array
Function CalcArray(ByVal a As Double()) As Double
    Dim result As Double = 0
    For i As Integer = 0 To a.Length - 1
        result += a(i)
    Next
    result = Math.Sqrt(result)

    Return result
End Function

Function CalcWhile(ByVal a As Double, ByVal max_err As Double) As Double
    Dim prev_y As Double = Double.MaxValue
    Dim y As Double = 0
    Dim err As Double = max_err + 1
    Dim x As Double = 0

    'solve y = a / x, increasing x until (y - prev_y) < max_err
    While err > max_err
        x += 1
        y = a / x
        err = prev_y - y
        prev_y = y
    End While

    Return x
End Function
```

2.6 If Statements and Select/Case Blocks

‘If’ statements are simple conditional ‘if something is true then do this’ with the option of saying ‘else do something else’ if the given condition is false. It is possible to chain several ‘If/Else’ statements together in order to define different actions depending on the value of a given variable. Sometimes if a lot of ‘If/Else’ statements are chained together for this purpose it is more appropriate to use a ‘Select/Case’ block. Examples (contained in 2 functions to give context):

```

'ByVal input As Double" specifies that whatever calls this function must give it a
'double (or 'real' number) as a parameter, example (assigning result to 'x'):
'x = ExampleIf(3.2)
Function ExampleIf(ByVal input As Double) As Double
    'If
    If input > 2 Then
        Return input * 0.5
    End If

    'If/Else
    If input < 1 Then
        Return input * 2
    Else
        Return input
    End If

    'NOTE: Usually you would chain these two 'if' statements together into an
    'If/ElseIf/Else' statement
End Function

Function Calc (ByVal a As Double, ByVal b As Double, ByVal mthd As String) As Double
    If mthd = "method_A" Then
        Return a * b
    ElseIf mthd = "method_B" Then
        Return ExampleIf(a * b)
    ElseIf mthd = "method_C" Then
        Dim x As Double = 2
        x = x - a
        Return x + b / a
    Else
        Return 0
    End If

    'does the same as above
    Select Case mthd
        Case "method_A"
            Return a * b
        Case "method_B"
            Return ExampleIf(a * b)
        Case "method_C"
            Dim x As Double = 2
            x = x - a
            Return x + b / a
        Case Else
            Return 0
    End Select
End Function

```

3. Reflection Tags - Exposing Variables and Subs to APSIM

While all the concepts introduced so far are important for programming APSIM Models/Manager2 scripts we have not yet covered how you let APSIM know what data you need from the simulation, what events you want to subscribe to or what data and events you want to make available to other models. The way we do this is by inserting 'reflection tags' above the variable/sub we want to expose. A reflection tag is simply a word surrounded by '<' '>' characters and there

are several different tags that APSIM uses, each with its own meaning. While reading this section it is recommended you refer to section 5.1

3.1 Input

The <Input()> tag denotes that a value for this variable needs to be supplied by APSIM. APSIM will use the name of the variable you tag with 'Input' to try and find a corresponding 'Output' (3.1.3) from another model. If APSIM cannot find the specified variable, then it will throw a fatal error. This error can be prevented by specifying the Input as optional by setting the 'IsOptional' parameter to 'true' e.g.

```
<Input(IsOptional:=True)>
```

3.2 Param

The <Param()> tag denotes that the variable is a parameter. APSIM looks for parameters in the XML configuration for this model with the same name as the variable with the <Param()> tag. When the code is part of a script component the parameter values will be on the "Properties" tab. If APSIM cannot find the specified parameter then it will throw a fatal error. As with 'Input', this error can be prevented by specifying the Input as optional by setting the 'IsOptional' parameter to 'true'. In addition to 'IsOptional' there are two other named parameters that can be used to 'bound' the incoming data and another that allows us to use an alias for the expected Param so our internal variable can have a different name. Perhaps the best explanation for this is an example, the following example defines an optional parameter bounded between 1 and 10 that is called 'Param_1' in the user interface but 'x' in our code:

```
<Param(IsOptional:=True, MaxVal:=10, MinVal:=1, Name:"Param_1")> Public x As Double
```

3.3 Output

The <Output()> tag denotes a variable that APSIM can supply to other APSIM models when requested. Like Param, an alias can be specified for the output if you want to use a different name for your variable e.g.

```
<Output(Name:"Output1")> Public x As double
```

3.4 Units

The <Units("")> tag supplies metadata to APSIM for an <Output()> and should be placed in between the <Output()> tag and the variable declaration. The REPORT module writes the units to the output file. Some examples:

```
<Units("")>:  
<Units("mm")>  
<Units("g")>  
<Units("kg/ha")>
```

3.5 EventHandler

The <EventHandler()> tag signals to APSIM that the following 'Sub' is an event handler that needs to be called whenever the APSIM event is published. The convention is that all handlers will have an On prefix. The name of the method (minus

the On prefix) denotes the APSIM event name that will be trapped. For example to handle the 'prepare' event, name your Sub 'OnPrepare' and insert an <EventHandler()> tag before it.

3.6 Event

The <Event[]> tag signals that the following .NET event should be published to the whole APSIM simulation. Other components in APSIM will be able to subscribe to this event. The name of the event will be the name of the event that APSIM sees. Events must be declared using one of the APSIM types. The reason for the square brackets around 'Event' is that 'Event' is a reserved word in the VB.NET language and the square brackets 'escape' it (this does not apply to C#)

3.7 Link

The <Link()> tag tells APSIM to establish a permanent link between the variable tagged and a corresponding model in the simulation. Unlike inputs and parameters, APSIM doesn't care about the name of the variable and instead looks at its 'type' to determine which model it should link into. Because each APSIM model has its own .NET 'type' (2.3.1) this task is easy and the resulting variable allows access to all outputs, events and event handlers as well as some inputs of the model. The most common link used is that to the 'paddock' in which the model/Manager2 script resides, however you may create a link to any other component that resides at the same level as yours, ie:

```
<Link()> Dim mypaddock As Paddock
<Link()> Dim fert As Fertiliser
```

The 'Paddock' you get from this link is the parent paddock that contains your model or Manager2 script. From there you will be able to access other components at the same level as your component or cycle through sub-paddocks to retrieve the components within them – provided sub paddocks exist within the paddock you are in. If your component is at the 'Simulation' level (above all paddocks) APSIM will treat the entire simulation as one 'Paddock' and give you that.

```
For Each subpaddock As Paddock In mypaddock.ChildPaddocks
    'do something with subpaddocks
Next

For Each crop As Component In mypaddock.Crops
    'do something with each crop
Next
```

Each paddock also has a Parent method that you can call to get the parent paddock. This is particular useful in multi paddock simulations.

```
Dim ParentPaddock as Paddock = mypaddock.Parent;
```

'apply fertiliser by getting the component via the paddock

```
Dim fert As Fertiliser = mypaddock.LinkByType("fertiliser")
fert.Apply(50, 10, "NO3_N")
```

APSIM will look for these models within the current 'scope' of the component that is running. Currently, this will be in the current paddock or the parent paddock. Once you have a link or variable that represents another model in the simulation you can then query that model for the values of its variables. You can also set the values of some APSIM variables, i.e.

```

<Link()> Dim SoilWaterModel As SoilWat

'...
    Dim sw As Single() = SoilWaterModel.sw
    sw(1) = 0.29
    sw(2) = 0.28
    '...
    SoilWaterModel.sw = sw
'...

```

4. Hints and Helper Libraries

To retrieve a list of APSIM variables and events, use the "Variables" and "Events" components under the "Output file" in the simulation tree in the APSIM User Interface – pretend as though you are adding a variable to your output file (or changing the reporting frequency) and browse the lists of outputs and events.

To investigate what accessible data or functions a variable has, simply type the name of the variable followed by ‘.’ then hit ‘CTRL-SPACE’ to bring up a box with more information.

There are two main ‘helper’ libraries available that contain several useful functions (for some examples see section 7)

DateUtility

Contains functions for turning Julian Dates and day/month strings into ‘DateTime’ or ‘Date’ variables – essential for turning user inputs into usable variables. For a list of all functions type ‘DateUtility.’ into the script editor and hit ‘CTRL-SPACE’ to bring up a box with more information. Ensure that the line Imports CSGeneral for VB.NET or using CSGeneral; for C# is included at the top of the script.

ManagerUtility

Contains a function that will take any delimited string (comma, space or tab) and turn it into an array of a given type. Very handy if the user-interface component of your manager script makes use of these strings. Also contains a function that will change the type of an array as well as a new ‘Type’ that can be used to track the last ‘x’ values of a given variable and return the average, sum or raw values. For a list of all functions type ‘ManagerUtility.’ into the script editor and hit ‘CTRL-SPACE’ to bring up a box with more information. Ensure that the line Imports CSGeneral for VB.NET or using CSGeneral; for C# is included at the top of the script.

5. A Basic Shell for a Manager2 Script/APSIM Model

5.1 VB.NET

```

Imports System
Imports ModelFramework
Imports CSGeneral

Public Class Script

    <Link()> Dim MyPaddock As Paddock
    <Param> Private A As String      ' The value for this will come from the Properties
page.
    <Output> Public B As Double     ' An example of how to make a variable available to
other APSIM modules

```

```

    <Input> Dim Today As DateTime    ' Equates to the value of the current simulation
date - value comes from CLOCK

    <EventHandler()> Public Sub OnInitialised()
        ' INITIALISATION
        ' delete this Sub if not required
    End Sub

    <EventHandler()> Public Sub OnPrepare()
        ' START OF DAY
        ' delete this Sub if not required
    End Sub

    <EventHandler()> Public Sub OnProcess()
        ' MAIN DAILY PROCESSING STEP
        ' delete this Sub if not required
    End Sub

    <EventHandler()> Public Sub OnPost()
        ' END OF DAY
        ' delete this Sub if not required
    End Sub

End Class

```

5.2 C#

```

using System;
using ModelFramework;
using CSGeneral;

public class Script
{
    [Link]    Paddock MyPaddock;
    [Input]   DateTime Today;    // Equates to the value of the current simulation date -
value comes from CLOCK
    [Param]   string A;          // The value for this will come from the Properties page.
    [Output]  double B;          // An example of how to make a variable available to
other APSIM modules

    [EventHandler]
    public void OnInitialised()
    {
        // INITIALISATION
        // can delete if not required
    }

    [EventHandler]
    public void OnPrepare()
    {
        // START OF DAY
        // can delete if not required
    }

    [EventHandler]
    public void OnProcess()
    {

```

```

        // MAIN DAILY PROCESSING STEP
        // can delete if not required
    }

    [EventHandler]
    public void OnPost()
    {
        // END OF DAY
        // can delete if not required
    }
}

```

6. Conversions from Original Manager in ‘Continuous Wheat’ Simulation

Note that the ‘Imports’ and ‘using’s have been removed from the top of the VB.NET and C# script examples to preserve space. They are the same as the ‘Imports’ and ‘using’s in section 5.

6.1 Sowing Rule

Original Code – Script Name = start_of_day

```

if (paddock_is_fallow() = 1 and FallowIn <> 'yes' and (NextCrop = 0 or NextCrop =
'[crop]')) then
    if (date_within('[date1]', [date2]') = 1) then
        if (rain[[rainnumdays]] >= [raincrit] AND esw >= [esw_amount]) OR
            ('[must_sow]' = 'yes' AND today = date('[date2]')) THEN
            ChooseNextCrop = 'yes' ! for rotations
            [crop] sow plants = [density], sowing_depth = [depth], cultivar =
[cultivar], row_spacing = [row_spacing], crop_class = [class]
        endif
        if today = date('[date2]') then
            ChooseNextCrop = 'yes'
        endif
    endif
endif
endif

```

VB.NET

```

Public Class SowingRule
    <Param()> Private date1 As String
    <Param()> Private date2 As String
    <Param()> Private must_sow As String
    <Param()> Private raincrit As Double
    <Param()> Private rainnumdays As Integer
    <Param()> Private esw_amount As Double
    <Param()> Private crop As String
    <Param()> <Output()> Private density As Double
    <Param()> Private depth As Double
    <Param()> Private cultivar As String
    <Param()> Private [class] As String
    <Param()> Private row_spacing As Double

    <Link()> Private mypaddock As Paddock
    <Input()> Private Today As System.DateTime
    <Input()> Private rain As Double
    <Input()> Private esw As Double

```

```

<Output()> Private ChooseNextCrop As String = "no"

Private rain_tracker As ManagerUtility.Tracker(Of Double)

Private default_sowtype As SowType

<EventHandler()> _
Public Sub OnInitialised()
    rain_tracker = New ManagerUtility.Tracker(Of Double)(rainnumdays)
End Sub

<EventHandler()> _
Public Sub OnPost()
    rain_tracker.Add(rain)

    If ManagerUtility.PaddockIsFallow(mypaddock) Then
        If DateUtility.WithinDates(date1, Today, date2) Then
            If rain_tracker.Sum() > raincrit And esw > esw_amount Or must_sow =
"yes" And DateUtility.DatesEqual(date2, Today) Then
                ChooseNextCrop = "yes"

                Dim data As New SowType()
                data.plants = density
                data.sowing_depth = depth
                data.Cultivar = cultivar
                data.row_spacing = row_spacing
                data.crop_class = [class]

                mypaddock.LinkByName(crop).Publish("sow", data)
            End If

            If DateUtility.DatesEqual(date2, Today) Then
                ChooseNextCrop = "yes"
            End If
        End If
    End If
End Sub
End Class

```

6.2 Sowing Fertiliser

Original Code – Script Name = [modulename.eventname](#)

```

[fertmodule] apply amount = [fert_amount_sow] (kg/ha), depth = 50 (mm), type =
[fert_type_sow]

```

VB.NET

```

Public Class Script
    <Param()> Private modulename As String
    <Param()> Private eventname As String
    <Param()> Private fertmodule As String
    <Param()> Private fert_amount_sow As Single
    <Param()> Private fert_type_sow As String

    <Link()> Private mypaddock As Paddock

```



```

Private fert As Fertiliser

<EventHandler()> _
Private Sub OnInitialised()
    mypaddock.LinkByName(modulename).Subscribe(eventname, AddressOf
ApplyFertEventHandler)

    'grab a reference To the fertiliser Module specified so we can use it later
    fert = DirectCast(mypaddock.LinkByName(fertmodule), Fertiliser)
End Sub

Private Sub ApplyFertEventHandler()
    Dim fert_type As New FertiliserApplicationType()
    fert_type.Amount = fert_amount_sow
    fert_type.Depth = 50
    fert_type.Type = fert_type_sow

    fert.Apply(fert_type)
End Sub
End Class

```

6.3 Harvesting rule

Original Code – Script Name = end_of_day

```

if [crop].StageName = 'harvest_ripe' or [crop].plant_status = 'dead' then
    [crop] harvest
    [crop] end_crop
endif

```

VB.NET

```

Public Class Script
    <Param()> Private crop As String
    <Input(IsOptional:=True)> Private density As Double = 100
    <Link()> Private mypaddock As Paddock

    Private mycrop As Wheat

    <EventHandler()> Private Sub OnInit2()
        mycrop = DirectCast(mypaddock.LinkByName(crop), Wheat)
    End Sub

    <EventHandler()> Private Sub OnPost()

        If mycrop.Variable("StageName").ToString() = "harvest_ripe" OrElse
mycrop.Variable("plant_status").ToString() = "dead" Then
            Dim ht As New HarvestType()
            ht.Height = 50
            ht.Plants = density
            ht.Remove = 0
            ht.Report = "yes"
            mycrop.Harvest(ht)

            mycrop.EndCrop()
        End If
    End Sub
End Class

```

Met

Description

The APSIM Met module provided daily meteorological information to all modules within an APSIM simulation.

Operation

The APSIM Met Module requires parameters to specify the climate of the site for each APSIM time step. This information is included in a 'weather', or 'met', file.

Climate data can exist in the met file in two ways :

As **Constants** or as **Daily (Column)** values.

Constant Values

Climate information for the simulation site that is independent of time can be specified at the top of the data section. Examples of how constant values can be specified are the values:

site = toowoomba

latitude = -26.8 (degrees)

Daily (Column)Values

Information that needs to be specified for each day can be arranged in space-delimited columns. The data columns can be arranged in any order and contain any data of any type. The line following the list of column names must be followed by a line containing the units, in brackets '()', for the information in each column.

The only constraints are as follows:

column headers must use the standard APSIM state variable names so that the data can be recognised by the module communications. The list of columns must contain adequate information for identifying time for each row. This means that the list of columns must contain either:

'day' and 'year'

OR

'day_of_month', 'month' and 'year'.

AN EXAMPLE

```
site = toowoomba
latitude = -26.8 (degrees)
```

year	day	radn	maxt	mint	rain
()	()	(MJ/m2)	(oC)	(oC)	(mm)
1996	65	20	29	20.5	0
1996	66	20	29	20	0
1996	67	20	29	21	0
1996	68	20	22.5	20	0
1996	69	20	27.5	18.5	0
.
.
.

Setting a Daily(Column) Value to a Constant Value

Sometimes you may want to switch off a column in your met file and replace it with a constant value.

For example you may want to instead of using the rain column in your met file, you may want to set the rain to always be a constant value of 0mm (so no rainfall).

Of course you can always just the met file itself and change the column values to be zero, but this is time consuming. A faster way is to just put the constant value,

rain=0 (mm)

at the top of your met file, and then rename the column heading for the rain column to something that won't be recognised as an APSIM variable,

eg.
rain
(mm)

to

xrain
(mm)

Resetting Met Variables (eg Climate Change scenario)

The APSIM Met Module can reset the values of any variables specified in the met file. Both constants and column values (apart from the time specification column values) for the current timestep can be reset using standard APSIM communication techniques.

However, changing met data dynamically during a simulation must only occur at a predefined stage during daily simulation execution. See example below.

Resetting met variables at 'start_of_day' or 'end_of_day' could result in critical modules (eg Soilwat2) missing the information due to process ordering. Setting met variables at the 'preNewMet' stage (see example) will avoid this risk.

AN EXAMPLE

Most modules in APSIM get their weather data from a newmet event that is produced by the MET module, so we need to change the weather variables before this event gets sent out. The MET module produces an event that makes this easy. It is called prenewmet. This event is fired just before a newmet event. So all we have to do is to trap this event and change the weather variables.

From the manager\sample\manager.par file:

```
modify_met.manager.preNewmet
maxt = maxt + 2
mint = mint + 2
```

When the prenewmet event fires, we increase the maximum and minimum temperature by 2 degrees, which then gets propagated to all other APSIM modules when the newmet event is sent by the MET module.

Module Output Variables

The APSIM Met Module can provide the values of several state variables for reporting to an output file or use by other modules.

Name	Units	Description
keyword	as specified	Each keyword climate constant is available for output by the user.
Column name	as specified	Each column data field is available for output by the user. The value for the current simulation timestep will be returned. The time specifiers (ie day, year, day_of_month, month) are not available for output.
Day_length	hours	Day length (hours of plant photosensitive light) for the current simulation day. This value is calculated using day of year and latitude and includes civil twilight with a sun angle of 6 degrees

Name	Units	Description
------	-------	-------------

below the horizon.

Easy way to calculate the tav and amp constants

The software team provides a tool called TAV_AMP that will calculate these 2 constants and insert them into a met file.

To download the tool goto: http://www.apsim.info/apsim/Downloads/tav_amp.exe

Full instructions for using the tool can be found here: http://www.apsim.info/apsim/Products/tav_amp.pdf

Operations

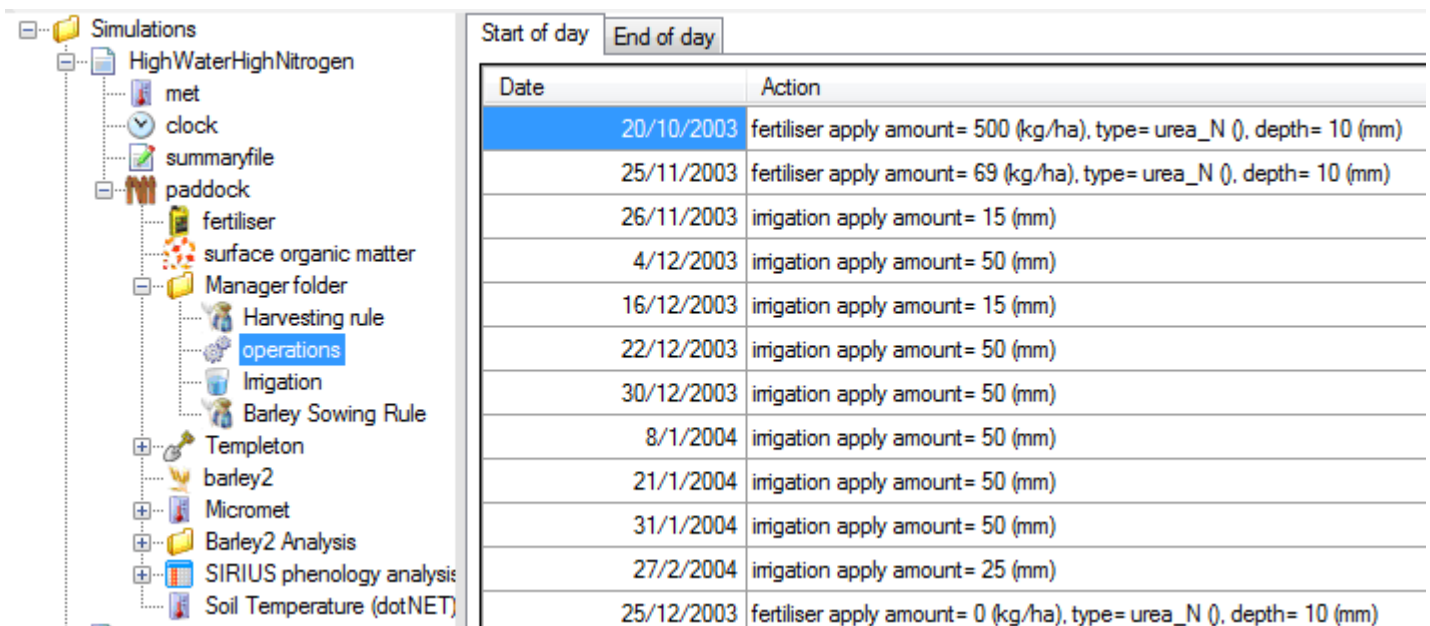
What is the Operations Schedule Module?

The operations module is similar to the APSIM manager module in that it allows the user to specify set day and year timing of simulation actions for modules. It is a fixed schedule manager.

The benefit it has over the standard manager in these conditions is speed of execution.

Because of its fixed style of specification, criteria can be checked and acted upon much more quickly than parsing of user-defined logic statements.

Large fixed schedule management data can be quickly specified by the user and brought into APSIM simulation runs.



Date	Action
20/10/2003	fertiliser apply amount= 500 (kg/ha), type= urea_N (), depth= 10 (mm)
25/11/2003	fertiliser apply amount= 69 (kg/ha), type= urea_N (), depth= 10 (mm)
26/11/2003	irrigation apply amount= 15 (mm)
4/12/2003	irrigation apply amount= 50 (mm)
16/12/2003	irrigation apply amount= 15 (mm)
22/12/2003	irrigation apply amount= 50 (mm)
30/12/2003	irrigation apply amount= 50 (mm)
8/1/2004	irrigation apply amount= 50 (mm)
21/1/2004	irrigation apply amount= 50 (mm)
31/1/2004	irrigation apply amount= 50 (mm)
27/2/2004	irrigation apply amount= 25 (mm)
25/12/2003	fertiliser apply amount= 0 (kg/ha), type= urea_N (), depth= 10 (mm)

PatchInput

Introduction

Some simulations require weather data (or other types of data) to be patched on certain days. For example, to simulate a grower's paddock for a particular season, it is necessary to patch the nearby weather stations rainfall data with the actual grower's rainfall and temperature data.

The PatchInput module does this patching automatically, without the need for running external patching tools. An example of where this functionality is used is in the Yield Prophet project.

In this project, a grower does a soil sample on a pre-sowing date e.g. 1 April 2004 . During the wheat season, the grower enters their rainfall (from their rain guage) from the 1 April 2004 to the present day.

The Yield Prophet project then:

- Runs APSIM for the last 100 years. They reset the soil status to the measured value on 1 April every year.
- They then use the PatchInput module to patch the current year (2004) rainfall (from rain guage) and maxt, mint and radn (from the nearest weather station) values over the last 100 years for dates between 1 April and the present date.
- All climate data after the present day are not patched but left at the historical values.
- This then gives provides a yield distribution for what could happen on their paddock for the remainder of the season.

Details

The PatchInput module is a derivative of the Input module and thus reads its own data file in exactly the same way as Input. It is specified in the control file like any other module:

```
module = PatchInput(patch) patch.dat [test]
```

The data file looks similar to any other APSIM data file:

```
[test.patch.data]
allow_sparse_data = true
patch_all_years = false
date patch_rain
() (mm)
1988/1/10 55
1988/1/15 66
1988/1/20 77
1988/1/25 88
```

All variables in the data file (except year, day and date) should be prefixed with 'patch_'. This indicates that the corresponding variable in the Input module will be overwritten with the value in the patch file. In the above example the Input module's variable called 'rain' will be overwritten with the values from the patch data file.

If a patch data file has a constant of patch_all_years = true, then the data will be overwritten on the specified days for all years. If patch_all_years = false, then the data will only be patched in the specific year mentioned. Sparse data, like in the above example, is supported.

Another optional parameter exists to patch other met variables for all years. Eg

```
[test.patch.data]
allow_sparse_data = true
patch_all_years = true
patch_variables_long_term = maxt mint radn

date patch_rain
() (mm)
1988-12-30 55
1988-12-31 66
1989-1-1 77
1989-1-2 88
```

In this example rainfall will be overwritten on the days 30 Dec through to 2 Jan for all years. In addition, because of the 'patch_variables_long_term' parameter, the values of maxt, mint and radn will be read from the dates 30/12/1988 through to 2/1/1989 and patched over all other years of the climate record.

R Language Component

[r2369](#) introduces a component that embeds the [R Language](#) in apsim simulations: users are presented with UI tab similar to the existing manager tabs; though the language is R.

R program fragments are associated with apsim events in a familiar fashion - via the tab name.

Users should install a recent (2.14+) version of R, and install two R packages: Rcpp and RInside.

The APSIM component will find the latest installed version of R from the system registry, load it, and call it when the nominated events occur. Unlike other apsim modules, the R interpreter is "shared" between apsim modules - there is only one R namespace.

A sample simulation that demonstrates basic functionality is in the "Examples" directory.

The module can display R Graphics via the usual windows graphics device, but does not support interactive (console) evaluation of R expressions. Debugging R code is limited to print statements (that are recorded in the summary file), and calls to `save.image()`.

The APSIM component implements a R language module "apsim", that implements 7 functions to interact with a simulation:

```
value <- apsimGet(name)
```

Returns the value of an apsim variable called "name".

```
apsimSet(name, value)
```

Set an apsim variable called "name" to "value". Value may be any simple (numeric, string, vector, scalar) type.

```
apsimPublish(name, type, structure)
```

Publish an apsim event called "name". For non-null events, a "type" structure is sent.

```
apsimSubscribe(name, type, handler)
```

Subscribe to an apsim event called "name". When the event occurs, the R function called "handler" is invoked with a structure of the nominated type as its argument.

```
apsimExpose(name)
```

Exposes an R variable called "name" to the apsim system.

```
apsimQuery(string)
```

Query the system for:

"*" will return a list of all components.

"wheat.*" will return a list of all variables for the wheat module

"*.lai" will return a list of all lai variables for all modules.

```
apsimFatal(message)
```

Cause a fatal error in the simulation.

Examples

Sow, harvest and end a wheat crop.

```
apsimPublish("wheat.sow", "Sow", list(plants=100, sowing_depth=25, Cultivar="hartog"))  
...  
apsimPublish("wheat.harvest", "Harvest", list(Remove=1.0))  
apsimPublish("wheat.end_crop", "Null", list())
```

Apply fertiliser

```
apsimPublish("fertiliser.apply", "FertiliserApplication", list(Amount = 100, Depth =  
50, Type = "NO3_N"))
```

Report (Output File)

Operation

The APSIM report module creates a columnar output file to record data from an APSIM simulation.

Output files contain data in columns with headers specifying variable names and units. The format is called space separated where each value or word is separated by one or more spaces (blanks). These files are the same format as used for input module files.

There is an option to create the files in CSV (comma delimited) format, for direct spreadsheet application.

Parameter file settings

The APSIM report module can report the state of any variable available to the system, from any module.

The user can specify various reporting parameters and the method for doing this is similar to the initialisation of parameters in any other module.

Instantiation of the Report module

Like all other APSIM modules, the report module can be instantiated to allow any number of output files to be created. You can specify more than one output file in any given simulation. This is useful for grouping certain types of variables together for analysis or for different reporting frequencies. So for example you might group all your soil nitrogen variables into one .out file, and all your soil water variables into another. Or you might have one output file containing daily data and another that contains harvest data.

You can specify a comma separated output file instead of a space delimited output file (the default). This is helpful when using statistics packages such as R and S+ which prefer comma separated files for inputting. In the User Interface under the "Variables" sub component (where you specify what variables to output) of the outputfile component, there is a section at the top called "Constants to put in the top of the output file:". In this section type in:

```
format = csv
```

This will then turn the output file into a comma separated file.

Optionally you set the precision (number of decimal places) outputted for all the variables by also typing in here:

```
precision = 5
```

This will make all the variables output values to 5 decimal places. To just alter the precision for an individual variable, then next to the variable in the variable list you just need to add "format 5". eg. to make the yield output to 5 decimal places you would type into the variable list:

```
yield format 5
```

SiloInput

Introduction

Some simulations require up to date weather data from a SILO weather station. In the past this meant extracting the required weather file(s) from SILO prior to running the simulation.

This new module automates this process. Now the user can specify the SILOInput module in the control file instead of the INPUT module. In the parameter file the user specifies the SILO station number to use. When APSIM is run, the SILOInput module will then extract the required weather file from SILO and provide the weather data to the APSIM simulation, all without any user intervention.

```
station_number = 15027 ! ROCKHAMPTON DOWNS
```

See the SILOInput Example for a complete working example.

SOI

Introduction

Many farming decisions are influenced by the long-range climate outlook and specifically by the phases of the SOI (Stone et al., 1996).

Examples are, for instance, the amount of nitrogen fertilizer applied to a crop, deciding whether to double crop or to fallow, investigating if it would be better to grow wheat now or sorghum next spring, investigating if earlier or later sown crops more economical.

There are many more of these decisions. In order to simulate the tactical and strategic responses to SOI conditions, the SOI module was developed that allows conditional systems simulations based on the SOI phases.

Implementation

The SOI Phases module requires an up to date list of soi phases. This is usually comes with your apsim installation as %apsuite\apsim\soi\sample\phases.soi.

This source of this file is <\\thor\public\met\phases.soi>.

To make the module available to APSIM the following line should be added to the configuration (.con) file:

```
module = soi phases.soi [soi]
```

Manager Rules

The basic format of the SOI module syntax is:

```
Soi[<month> or <lag>] = <phase>
```

Where <month> is a month in either numeric format (1,2,3,4...12) or 3-letter month abbreviation (Jan,Feb,Mar...Dec).

The full Date will also work, but remember the SOI Phases are monthly values.

Where <lag> is a negative or zero numerical value that indicates the number of months prior to the current Date, that we are comparing the SOI Phase to.

Eg: -1 is one month prior, -2 is two months prior...

Where <phase> is the SOI Phase we are comparing to. There are five Phases, 1 to 5:

- 1 - Consistently Negative SOI
- 2 - Consistently Positive SOI
- 3 - Rapidly Falling SOI
- 4 - Rapidly Rising SOI
- 5 - Consistently Near Zero SOI

Examples

```
! Sow Sorghum if the SOI Phase in February is Consistently Negative </p>
If soi[Feb] = 1 then
    Sorghum sow ....
Else
    Cotton sow ...
Endif
```

```
! Set the Soilwater if the current SOI Phase is Rapidly Rising
if soi[0] = 4 then
    soilwat2 init
    soilwat2 set sw = 0.344 0.347 0.369 0.33 0.34 0.33 0.345 (mm/mm)
endif
```

```
if soi['-1'] = 1 then ! Notice the inverted commas around the negative value
```

```
if soi['15-Oct']= 5 then
```

```
if (today = date('15-Oct') AND soi['-2'] = 3) then
```

Report Rules

The SOI Phases module is used in the Report module as it is in the Manager, with one exception.

Because the Report module doesn't use the Manager to parse its variables, lag values (negative numbers) should not be enclosed in inverted commas.

Soi[Apr]

Soi[-2]

TclLink

Description

This module provides a simple link between apsim and TCL interpreters. Variables are exchanged through a Get/Set mechanism.

Settings

Rules are defined just like the manager: the rules in [xyz.tcllink.init] will be evaluated at the simulation's initialisation. The other rules of interest are "init", "prepare", "start_of_day", "process", "post", "end_of_day", and "exit".

Apsim messages are generated with the TCL procedure `apsimSendMessage`.

It has two mandatory arguments, the destination module, the message name, and optional message name/value pairs are encoded as list elements.

For example,

```
apsimSendMessage wheat sow {cultivar hartog} {plants 120} {sowing_depth 30}
```

Apsim variables are set/get with `apsimGet`, `apsimSet`.

The procedure `apsimWriteToSummaryFile` sends a log message to the current summary file

Command Line Debugger

A simple "command line" debugger is provided in the sample directory.

It demonstrates the ability to examine apsim variables, step through simulations and set breakpoints.

nb. In the following samples,

1. On the first line, the "(Sample) # %" (where # is just a number) part is just the command prompt. It is what comes after that is important. It is the command.
2. Also the next line underneath (if it exists) is the result or output(or the echo) of doing the command above.
3. The final line (if it exists) is an explanation of the result or output(or echo).

```
(Sample) 1 % link clock.day day
```

Link the apsim variable "clock.day" to the tcl variable "day".

```
(Sample) 2 % b {$day == 182}  
{$day == 182}
```

Set a breakpoint for day 182. Equivalent to

```
" b {[apsimGet clock.day] == 182} "
```

```
(Sample) 3 % c  
-1
```

Continue. Control returns after a crop is planted

```
(Sample) 4 % p wheat.biomass  
12.874
```

Print an apsim variable

```
(Sample) 5 % s  
1
```

Single step

```
(Sample) 6 % p wheat.biomass  
14.095  
etc..
```

```
(Sample) 7 %
```

Missing features

Cannot uncrack messageData structures (eg newmet)

Tracker

What is the Data Tracker Module?

The tracker module allows the user to track information through a simulation and perform simple statistics to that data prior to recording it in an APSIM output file.

For example, tracker can record and accumulate flows of matter (e.g. rainfall, irrigation, drainage, fertiliser) between simulation outputs.

2 Definition of statistical types

Statistics can be provided on either events or the states of variables within simulation execution.

In both cases, the statistic can be based upon a single “spot” sample at some defined point or it can be based upon an accumulated sample of values for a defined timeframe within model execution.

In all cases the statistical variable is defined in the parameter file via a specification of the form, “specification”

where specification is a description of the statistic via a predefined grammar.

2.1 Event Statistics

2.1.1 Spot Sample

Statistic of Event as Alias

where:

Statistic : is the type of statistic to be applied. Currently the only option here is “date”.

Event : is the event on which the statistic is to be applied. This can be any event within the APSIM simulation.

Alias : is the tracker output variable alias to be applied to this statistic. This is the name by which other modules can access the value of this statistic.

Sample 1 – Report the sowing date of the last wheat crop

```
date of wheat.sowing as wheat_sowing_date
```

Sample 2 – Report the harvesting date of the last wheat crop

```
date of wheat.harvesting as wheat_harvest_date
```

2.1.2 Accumulated Samples

Statistic of Event from **StartEvent** to **EndEvent** as **Alias**

where

Statistic : is the type of statistic to be applied. Currently the only option here is “count”.

Event : is the event on which the statistic is to be applied. This can be any event within the APSIM simulation.

StartEvent : is the APSIM event which signals the start of the sampling window.

EndEvent : is the APSIM event which signals the start of the sampling window.

Alias : is the tracker output variable alias to be applied to this statistic. This is the name by which other modules can access the value of this statistic.

Sample 1 – Count the number of lucerne cuts since the last report to the output file.

```
count of lucerne.harvesting from reported to now as number_of_cuts
```

Sample 2 – Count the number of days since chickpea was last sown

```
count of tick from chickpea.sowing to now as days_since_chickpea_sowing
```

2.2 State Variable Statistics

2.2.1 Spot Sample

Statistic of Variable on **Event** as **Alias**

where

Statistic : is the type of statistic to be applied. Currently the only option for the user is “value”.

Variable : is the APSIM state variable upon which statistics are calculated. This can be any variable within the APSIM simulation.

Event : is the event on which the statistic is to be applied. This can be any event within the APSIM simulation.

Alias : is the tracker output variable alias to be applied to this statistic. This is the name by which other modules can access the value of this statistic.

Sample 1 – Report the yield of the last wheat crop

```
value of wheat.yield on wheat.harvesting as wheat_yield
```

Sample 2 – Report the total soil water at the sowing of the last wheat crop

```
value of sw_dep() on wheat.sowing as wheat_sowing_sw
```

2.2.2 Accumulated Samples

Statistic of Variable on [**last nnn**] **Event** [from **StartEvent** to **EndEvent**] as **Alias**

where

Statistic : is the type of statistic to be applied. Currently the user can choose from “sum”, “average”, “maximum” and “minimum”.

Variable : is the APSIM state variable upon which statistics are calculated. This can be any variable within the APSIM simulation.

last nnn : is an optional specification to allow the user to specify a moving sample set.

Event : is the event on which the statistic is to be applied. This can be any event within the APSIM simulation.

StartEvent : is the APSIM event which signals the start of the sampling window.

EndEvent : is the APSIM event which signals the start of the sampling window.

Alias : is the tracker output variable alias to be applied to this statistic. This is the name by which other modules can access the value of this statistic.

Note: the “from StartEvent to EndEvent” is optional only if the “last nnn” is specified.

Sample 1 – Report the accumulated rainfall since the last report to the output file

```
sum of rain on start_of_day from reported to now as rainfall
```

Sample 2 – Report the accumulated rainfall for the last 3 days

```
sum of rain on last 3 start_of_day as rain3
```

Sample 3 – Report the accumulated lucerne harvest since the last report to the output file

```
sum of lucerne.biomass on lucerne.harvesting from reported to now ...  
as cut_biomass
```

Sample 4 – Report the in-crop rainfall for the last wheat crop

```
sum of met.rain on met.newmet from wheat.sowing to wheat.harvesting as  
wheat_incrop_rain
```

Sample 5 – Report the 7 day moving average of the wheat crop water stress factor

```
average of wheat.swdef_photo on last 7 end_of_day from wheat.sowing ...  
to wheat.harvesting as wheat_weekly_stress
```

VenLink

Linking APSIM and VENSIM using the Venlink module

Patrick Smith and Dean Holzworth

Overview

The ability to integrate diverse modules through the so-called “plug-in-pull-out” approach is one of APSIM's key strengths. This guide describes an enhancement of this functionality through the linkage of APSIM with VENSIM, an ‘off the shelf’ modelling package produced by Ventana Systems Inc. Enabling APSIM to call and execute VENSIM models allows researchers to quickly and easily develop new modules for APSIM, making full use of the functionality and intuitive icon-based interface of VENSIM. New VENSIM based modules fully interact with the APSIM Manager in the same manner as existing modules, but may be developed by the user without the need for significant interaction with the APSIM software engineers.

Procedure for producing a VENSIM-based module

To produce a new VENSIM-based module for APSIM the following three steps are recommended:

- Construct and test a fully functional stand-alone model in VENSIM
- Convert the ‘shared’ variables within the VENSIM model to ‘APSIM setttable’ types
- Add the necessary sections and commands to the APSIM files.

Step 1. Construct and test a fully functional stand-alone model in VENSIM

VENSIM is quite intuitive to use and previous users of icon-type modelling software should have little difficulty getting up to speed with it. New users are recommended to do the tutorials provided under VENSIM Help. As you design and build your model distinguish between two types of variables – those variables that will be used exclusively within the VENSIM module, and those variable that will be passed from VENSIM to APSIM and vice versa. It may be helpful to visually identify the latter “shared” variables in your VENSIM model diagram by using a special font, colour or outline.

Before attempting to link a VENSIM model to APSIM it is important to ensure that the model is working properly in isolation. This means that all variables that eventually will be shared between the programs will initially need to have values allocated within VENSIM. For constants this is simply a matter of specifying a value in the VENSIM equation editor. For variables that require sequential data it may be useful to dynamically link VENSIM to a spreadsheet using the GET XLS DATA or GET 123 DATA functions in VENSIM. Doing a dummy APSIM run may be a convenient way to generate the data you need to test your VENSIM model, and in many cases it may be the same data that is eventually passed to VENSIM by APSIM (e.g. daily rainfall, soil moisture or crop biomass).

The only rule that must be followed when constructing your VENSIM model is that:

The initial value of all ‘levels’ (= ‘stocks’ or ‘state variables’) in VENSIM must be set indirectly by means of a constant linked to the level with a connecting arrow.

This is because APSIM cannot set VENSIM levels directly.

In addition to the rule concerning levels, the following naming convention is encouraged:

Names of variables in VENSIM models should be as descriptive as possible to assist other users and to help 'self document' the model. Spaces are permitted in VENSIM and these will be automatically converted within APSIM to underscores (eg. initial htt in VENSIM will become initial_htt in APSIM..

Step 2. Convert the 'shared' variables within the VENSIM model to 'APSIM settable' types

Having established that your VENSIM model is working properly it is a simple matter to enable communication with APSIM. Variables that need to be set by APSIM during the course of a run should be converted to one of two variable types:

Variables that require only the initial value to be set by APSIM (namely constants, including those that specify the initial values of levels (as described in Step 1)) should be set to "Constant" and "Normal" under the variable type selectors in the equation editor

Variables that are required to be set by APSIM on a daily basis should be set to "Auxiliary" and "Gaming" under the variable type selectors in the equation editor.

Note that variables owned exclusively by VENSIM do not need to be changed from Auxiliary/Normal or Constant/Normal types. Such variables can still be retrieved by APSIM for calculations and reporting.

Step 3. Add necessary sections to the APSIM files

Firstly you need to drag and drop a VenLink component from the standard toolbox and drop it on your paddock. Next you need to specify the model file name and the VENSIM variable names that you want to access from APSIM. This can be done by clicking on VenLink and editing the parameters on the right hand side.

Next you need to give the VENSIM "constants" an initial value. Click on the ini node under VenLink and browse to a .ini file. The contents of the .ini file should look something like the following example:

```
[standard.venlink.constants]

initial_sum_htt = 0
soil_moisture_threshold = -0.9
l115 = 0.5
soil_temp_threshold = 20
```

Finally, VENSIM variables that need to be set every day must be done through a logic component.

After dropping a Logic component from the standard toolbox onto your paddock enter script in the "Start_of_day" tab that looks like this example:

```
venlink.st1 = st(1)
venlink.sw1 = sw(1)
```