1 Manager Tutorial

2 Manager Model Documentation and Examples

Manager is the model in APSIM that is used to script the on-farm management of a simulation. The examples in this document increase in complexity. Read this to better understand the structure of a manager script

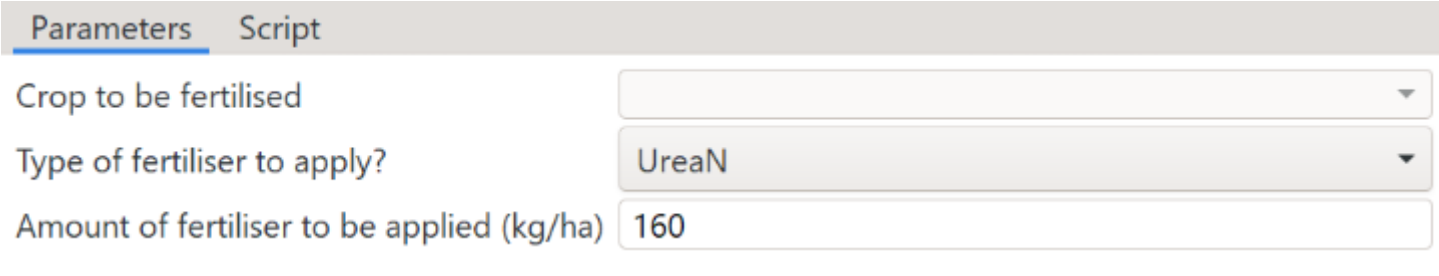
3 NTopupSimulation

3.1 Change sowing fertiliser to a topup rule

For this tutorial, we are going to take the *Fertilise at sowing* rule from the management toolbox and turn it into an N topup rule. The original code looks like this:

```
using Models.Soils;
using System;
using Models.Core;
using Models.PMF;
namespace Models
{
[Serializable]
public class Script : Model
{
[Link] Clock Clock;
[Link] Fertiliser Fertiliser;

[Description("Crop to be fertilised")]
public IPlant Crop { get; set; }

[Description("Type of fertiliser to apply? ")]
public Fertiliser.Types FertiliserType { get; set; }

[Description("Amount of fertiliser to be applied (kg/ha)")]
public double Amount { get; set; }

[EventSubscribe("Sowing")]
private void OnSowing(object sender, EventArgs e)
{
Model crop = sender as Model;
if (Crop != null && crop.Name.ToLower() == (Crop as IModel).Name.ToLower())
Fertiliser.Apply(Amount: Amount, Type: FertiliserType);
}
}
}
```

This script has 3 user-input properties which looks like this to the user:



The script that relates to these parameters looks like this:

```
[Description("Crop to be fertilised")]
public IPlant Crop { get; set; }

[Description("Type of fertiliser to apply? ")]
public Fertiliser.Types FertiliserType { get; set; }

[Description("Amount of fertiliser to be applied (kg/ha)")]
public double Amount { get; set; }
```

For a simple N topup rule we don't need the crop or the fertiliser type and instead of specifying an amount to apply, we'll change this to a threshold amount of N that the script will top up the NO3 to. We'll rename this property to *NThreshold*:

```
Description("Threshold amount of NO3 below which fertiliser will be added (kg/ha)")]
public double NThreshold { get; set;}
```

Next we need to add a link to the NO3 solute as we'll need this to determine how much to apply. The easiest way to do this is to find the NO3 solute model in the simulation tree, right click on it and select *Copy manager snippet for model*. This will copy a snippet of script code to the clipboard that you can then paste into your manager script. The pasted code looks like this:

```
using Models.Soils;
[Link(ByName=true)] private Solute NO3;
```

The snippet gives you the correct using statement which you should move to the using section of your script. The link statement should be moved to the link section of your manager script.

This link looks different to the others because there are multiple solutes in the simulation and we want to make sure we are linking to the NO3 solute and not NH4 or Urea. A regular *[Link]* looks for a model of the correct type (Solute) and doesn't use the name to perform the match. If we used a regular link for this solute then we will get a link to the first solute found which may not be NO3. To fix this, the snippet above links using the name and type, meaning that the link will be resolved by looking at the name of the variable (NO3 in this case) to do the match.

Currently in the script there is an OnSowing method that gets called whenever a sowing event occurs. We need to change this to a method that gets called every day of the simulation. Most management scripts use DoManagement for this purpose.

```
[EventSubscribe("DoManagement")]
private void OnDoManagement(object sender, EventArgs e)
```

The event name is *DoManagement* and this goes in the *EventSubscribe* attribute. The convention then is to name the method the same as the event but with *On* prefixed to the front of the event name.

Now we can perform the calculation of the amount of NO3 to apply.

```
double NAmount = NThreshold - NO3.kgha.Sum();
```

NO3.kgha is an array variable with one value per layer. We need to sum this over the whole array so we use a *Sum* function to do this. For this to compile though we need to include another namespace at the top of the script.

```
using System.Linq;
```

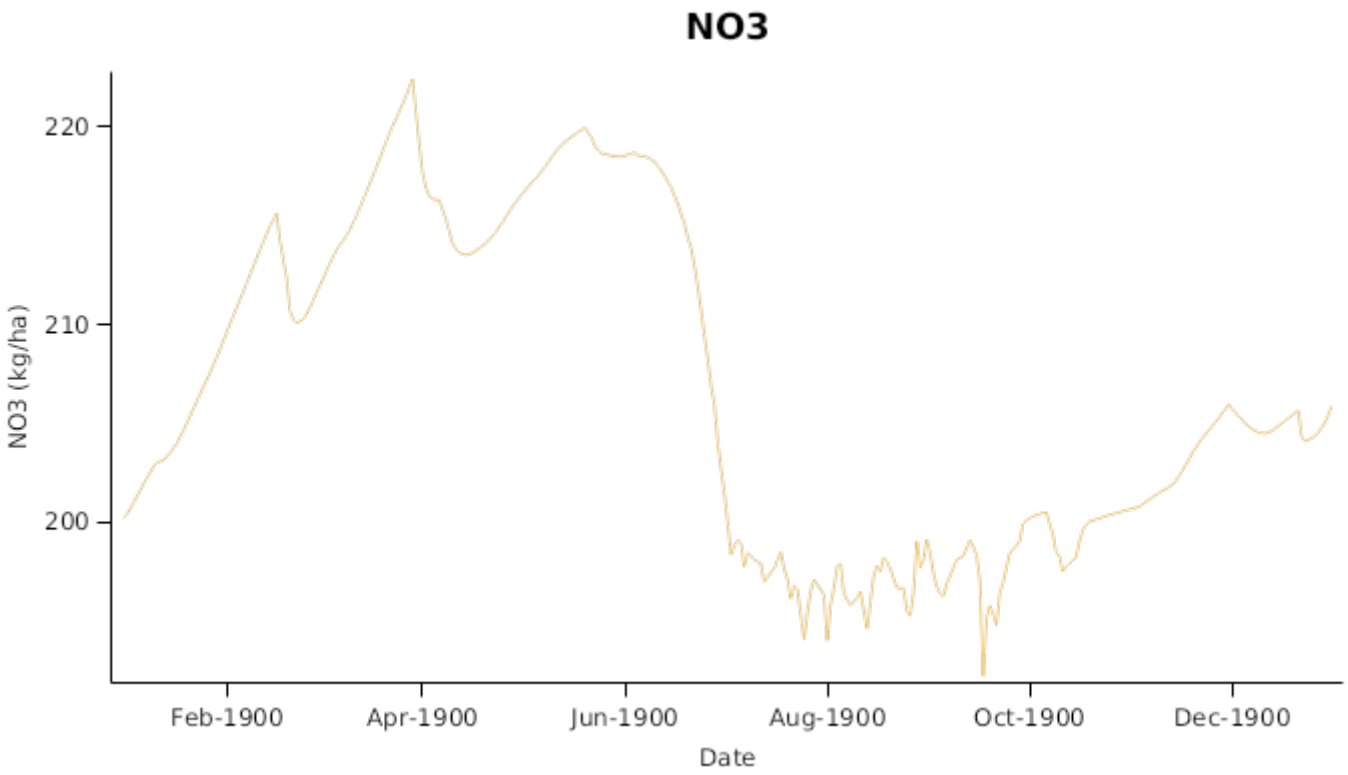
We can remove the existing references to *Crop* as this rule will apply all year around, not just within crop. We also hard code the fertiliser type on the apply line.

```
Fertiliser.Apply(Amount: NAmount, Type: Fertiliser.Types.NO3N);
```

Finally, we rename the manager model to *NTopup*. The final script looks like:

```
using Models.Soils;
using APSIM.Shared.Utilities;
using Models.Soils.Nutrients;
using Models.PMF;
using Models.Core;
using System;
using System.Linq;
using Models.Soils.Nutrients;
namespace Models
{
[Serializable]
public class Script : Model
{
[Link] Clock Clock;
[Link] Fertiliser Fertiliser;
[Link(ByName=true)] private Solute NO3;
[Description("Threshold amount of NO3 below which fertiliser will be added (kg/ha)")]
public double NThreshold { get; set;}

[EventSubscribe("DoManagement")]
private void OnDoManagement(object sender, EventArgs e)
{
double NAmount = NThreshold - NO3.kgha.Sum();
Fertiliser.Apply(Amount: NAmount, Type: Fertiliser.Types.NO3N);
}
}
}
```

3.2 NO3



4 WaterAndNTopupSimulation

4.1 Change N topup rule to N and water topup rule

This simulation extends the previous simulation by adding a water topup rule to the topup rule. This is useful when a potential crop/pasture yield is required.

This script has a single user-input property, the N threshold below which fertiliser will be applied.

```
[Description("Threshold amount of NO3 below which fertiliser will be added (kg/ha)")]
public double NThreshold { get; set;}
```

This is still needed but we need to add a similar property for a water threshold.

```
[Description("Threshold amount of water below which irrigation will be added (mm)")]
public double WaterThreshold { get; set;}
```

Next we need to add a link to the SoilWater model as we'll need this to get the current value of soil water. You can do this using the same manager snippet technique outlined above.

We then need to add an irrigation model to our field (use the manager snippet). Right click on the Field, select *Add Model* and add an irrigation model.

In the *OnDoManagement* script, we can calculate the amount of water to add then apply the irrigation if the amount is greater than zero.

```
double SWAmount = WaterThreshold - SoilWater.SWmm.Sum();
if (SWAmount > 0)
Irrigation.Apply(SWAmount);
```

Finally, the manager model should be renamed to something more meaningful e.g. *NAndWaterTopup*. The final script looks like:

```
using Models.Interfaces;
using Models.Soils;
using APSIM.Shared.Utilities;
using Models.PMF;
using Models.Core;
using System;
using System.Linq;
using Models.Soils.Nutrients;
using Models.WaterModel;

namespace Models
{
[Serializable]
public class Script : Model
{
[Link] private Clock Clock;
[Link] private Fertiliser Fertiliser;
[Link(ByName=true)] private Solute NO3;
[Link] private Soil Soil;
[Link] private Irrigation Irrigation;
[Link(ByName=true)] private WaterBalance SoilWater;

[Description("Threshold amount of NO3 below which fertiliser will be added (kg/ha)")]
public double NThreshold { get; set;}

[Description("Threshold amount of water below which irrigation will be added (mm)")]
public double WaterThreshold { get; set;}


[EventSubscribe("DoManagement")]
private void OnDoManagement(object sender, EventArgs e)
{
double NAmount = NThreshold - NO3.kgha.Sum();
Fertiliser.Apply(Amount: NAmount, Type: Fertiliser.Types.NO3N);

double SWAmount = WaterThreshold - SoilWater.SWmm.Sum();
if (SWAmount > 0)
```

```
Irrigation.Apply(SWAmount);
}
}
}
```

4.2 Soil water