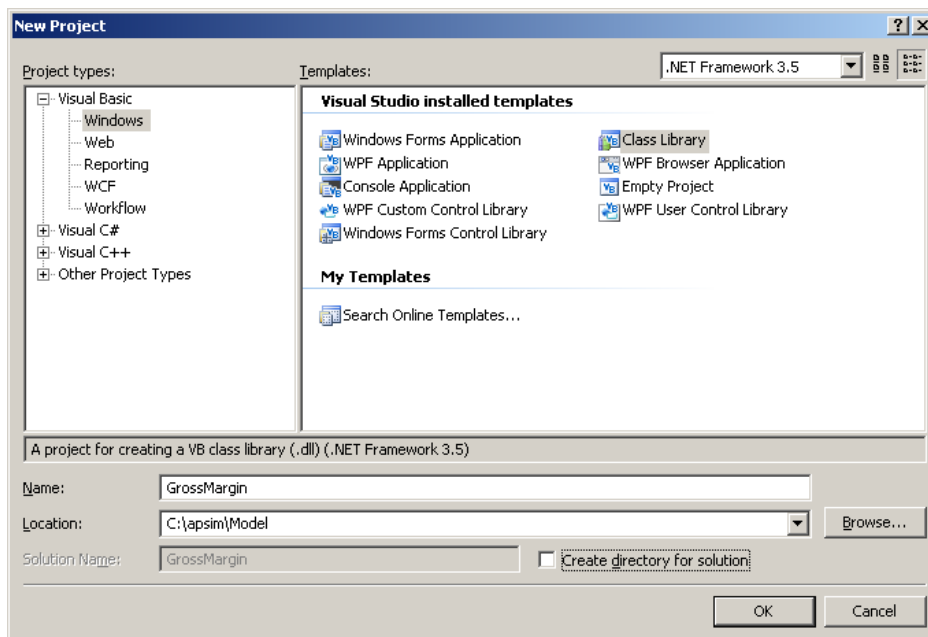


## Developing Models For APSIM

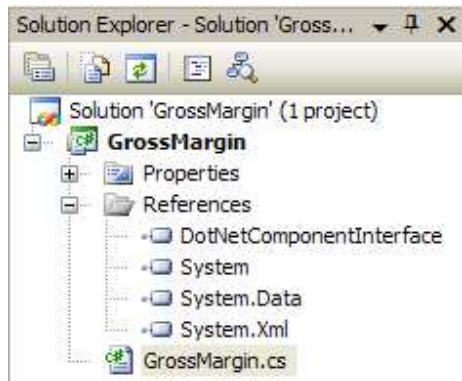
This guide demonstrates how to build a model using Microsoft Visual Studio 2005. While any of the .NET languages can be used, this guide uses C#.

### *Creating a simple gross margin example*

- In Visual Studio, create a new project of type “Class Library”. Uncheck the “create directory” box.



- Add a reference to DotNetComponentInterface.dll to the project (Right click on References in the Solution Explorer, click the *Browse* tab and navigate to the DLL.
- Save the solution, project and class file name as *GrossMargin*
- The Solution Explorer should look like this:



- o Enter the following code in GrossMargin.cs:

## GrossMargin.cs

---

```
using System;
using System.Collections.Generic;
using System.Text;
using ModelFramework;
public class GrossMargin : Instance
{
    private bool ResetBank;
    [Input] private double Fertiliser = 0;
    [Input] private double irrig_tot = 0;
    [Input] private double Yield = 0;
    [Input] private double grain_protein = 0;
    [Input] private string StageName = "";
    [Input] private string plant_status = "";
    [Param] private double NCost = 0;
    [Param] private double NApplicationCost = 0;
    [Param] private double WaterCost = 0;
    [Param] private double Price = 0;
    [Param] private double MinimumProtein = 0;
    [Param] private double ProteinIncrement = 0;
    [Output][Units("$")] double Bank;
    [Event] public event NullTypeDelegate NegativeBank;
    [EventHandler] public void OnPost()
    {
        // Reset the bank to zero the day after harvest.
        if (ResetBank)
        {
            ResetBank = false;
            Bank = 0;
        }

        // Subtract fertiliser cost from the bank
        if (Fertiliser > 0)
            Bank = Bank - (NCost * Fertiliser + NApplicationCost);

        // Subtract irrigation costs.
        if (irrig_tot > 0)
            Bank = Bank - WaterCost * (irrig_tot / 100);

        // Add harvest return to the bank
        if (StageName == "harvest_ripe" || plant_status == "dead")
        {
            double FullPrice = Price + (grain_protein - MinimumProtein)
                * 2 * ProteinIncrement;
            Bank = Bank + FullPrice * (Yield / 1000);
            ResetBank = true;
        }

        // Publish event if bank falls below zero.
        if (Bank < 0)
            NegativeBank.Invoke();
    }
}
```

- The APSIM .NET component interface relies heavily on reflection tags to analyse the model source code and locate variables and methods.
  - The `[Input]` tag denotes that a value for this variable needs to be supplied by APSIM. In the example, APSIM will locate a variable called *Fertiliser* in another model, retrieve its value and *push* the value to the *Fertiliser* variable in this *GrossMargin* model. APSIM will do this every day.
  - The `[Param]` tag denotes that the variable is a parameter. Like the *Input* tag, APSIM will supply a value but only at the beginning of the simulation. APSIM only looks for parameters in the XML configuration for this model.
  - The `[Output]` tag denotes a variable that APSIM can supply to other APSIM models when requested.
  - The `[Units("$")]` tag supplies metadata to APSIM. The REPORT module writes the units to the output file.
  - The `[EventHandler]` tag signals to APSIM that the following method 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.
  - The `[Event]` tag signals that the following .NET event should be published to the wider APSIM simulation. Other modules in APSIM will be able to subscribe to this event. The name of the event (*NegativeBank* in this example) will be the name of the event that APSIM sees. Events must be declared using one of the APSIM types. See the description of APSIM types below.

### ***VB.NET specifics***

For VB components, the syntax of the metadata tags is different. Rather than square brackets surrounding the tags, use angle brackets instead e.g. `<Input()>`. Visual Studio also inserts round brackets after the tag.

The VB source code becomes:

## GrossMargin.vb

---

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports ModelFramework
Public Class GrossMargin
    Inherits Instance
    Private ResetBank As Boolean
    <Input()> Private Fertiliser As Double = 0
    <Input()> Private irrig_tot As Double = 0
    <Input()> Private Yield As Double = 0
    <Input()> Private grain_protein As Double = 0
    <Input()> Private StageName As String = ""
    <Input()> Private plant_status As String = ""
    <Param()> Private NCost As Double = 0
    <Param()> Private NApplicationCost As Double = 0
    <Param()> Private WaterCost As Double = 0
    <Param()> Private Price As Double = 0
    <Param()> Private MinimumProtein As Double = 0
    <Param()> Private ProteinIncrement As Double = 0
    <Output()> <Units("$")> Private Bank As Double
    <[Event]()> Public Event NegativeBank As NullTypeDelegate
    <EventHandler()> Public Sub OnPost()
        ' Reset the bank to zero the day after harvest.
        If ResetBank Then
            ResetBank = False
            Bank = 0
        End If

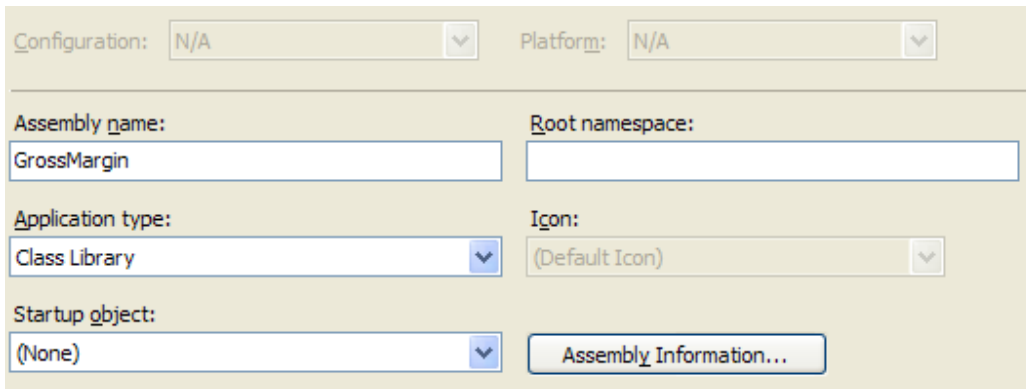
        ' Subtract fertiliser cost from the bank
        If Fertiliser > 0 Then
            Bank = Bank - (NCost * Fertiliser + NApplicationCost)
        End If

        ' Subtract irrigation costs.
        If irrig_tot > 0 Then
            Bank = Bank - WaterCost * (irrig_tot / 100)
        End If

        ' Add harvest return to the bank
        If StageName = "harvest_ripe" OrElse plant_status = "dead" Then
            Dim FullPrice As Double = Price + (grain_protein - MinimumProtein) *
                2 * ProteinIncrement
            Bank = Bank + FullPrice * (Yield / 1000)
            ResetBank = True
        End If

        ' Publish event if bank falls below zero.
        If Bank < 0 Then
            RaiseEvent NegativeBank()
        End If
    End Sub
End Class
```

By default Visual Studio also specifies a “Root Namespace”. This needs to be removed otherwise APSIM cannot find the class called GrossMargin. Go to the project properties and under the Application tab set the Root Namespace to blank:



Configuration: N/A Platform: N/A

Assembly name: GrossMargin Root namespace:

Application type: Class Library Icon: (Default Icon)

Startup object: (None) Assembly Information...

### Model configuration XML

The APSIM DotNetComponentInterface will construct a model, at runtime, from an XML specification. This XML describes what objects get created at runtime and what parameters are pushed to the model at startup. The XML for our example model above would look like this:

```
GrossMargin.xml  
  
<?xml version="1.0" encoding="utf-8" ?>  
<Model>  
  <GrossMargin>  
    <NCost>1.2</NCost>  
    <NApplicationCost>15</NApplicationCost>  
    <WaterCost>0</WaterCost>  
    <Price>140</Price>  
    <MinimumProtein>10.5</MinimumProtein>  
    <ProteinIncrement>2</ProteinIncrement>  
  </GrossMargin>  
</Model>
```

- The top level XML element must be <Model>. The infrastructure will look for this and start instantiating classes below this element.
- The DotNetComponentInterface will see the <GrossMargin> element and look for a class with that name, using reflection. It will then create an instance of it and pass all the specified parameters to it.

## Telling the APSIM User Interface about our new model.

The APSIM User Interface needs to be told about our new model.

- In the Solution Explorer in Visual Studio, create a new *UserInterfaceType* directory and create an XML configuration file that tells the APSIM user interface about our model and how to create a .sim file to pass to APSIM. Create the following file:

### GrossMarginType.xml

---

```
<GrossMargin>
  <UItype>VBUserInterface.GenericUI</UItype>
  <IsCrop>No</IsCrop>
  <ShowInMainTree>Yes</ShowInMainTree>
  <Image>%apsim%\UserInterface\Images\banner2.jpg</Image>
  <Documentation></Documentation>
  <LargeIcon>%apsim%\UserInterface\Images\cubes32.png</LargeIcon>
  <SmallIcon>%apsim%\UserInterface\Images\cubes16.png</SmallIcon>
  <drops>
    <drop name="area" />
    <drop name="folder" />
  </drops>
  <variables>
    <variable name="bank" description="Gross margin ($)"/>
  </variables>
  <events>
    <event name="NegativeBank"
      description="Fired when bank balance falls negative"/>
  </events>
  <ApsimToSim>
    <type>component</type>
    <dll>c:\hol353\Misc\GrossMargin\bin\Debug\GrossMargin.dll</dll>
    <ini>c:\hol353\Misc\GrossMargin\GrossMargin.xml</ini>
    <componentinterface>%apsim%\Model\dotnetcomponentinterface.dll
    </componentinterface>
  </ApsimToSim>
</GrossMargin>
```

node types it can be dropped on by the user, the output variables to display in the *outputfile* user interface component and how to write the .sim script to pass to APSIM.

- Note the <ini> and <dll> elements need to point to our model configuration file and dll respectively.

- Point the user interface to our new *UserInterfaceType* directory by editing the *Apsim.xml* file and adding the following line:

```
Apsim.xml  


---

  
<UIConfigurationFolder>c:\hol353\Misc\GrossMargin\UserInterfaceType  
</UIConfigurationFolder>
```

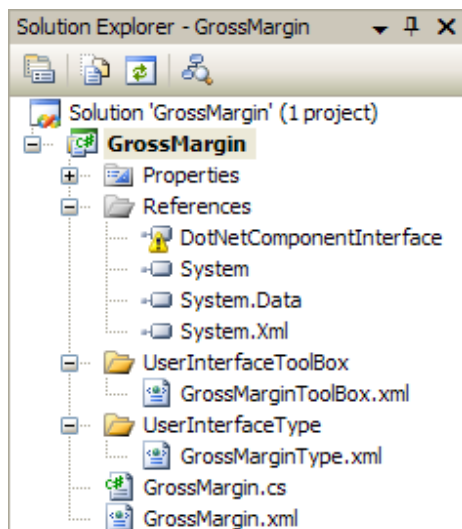
- In the Solution Explorer, create a new folder called *UserInterfaceToolBox*, and in there create a new toolbox file for our model so that we have something to drag to our simulation.

```
GrossMarginToolBox.xml  


---

  
<folder name="GrossMarginToolBox">  
  <GrossMargin>  
    <NCost>1.2</NCost>  
    <NApplicationCost>15</NApplicationCost>  
    <WaterCost>0</WaterCost>  
    <Price>140</Price>  
    <MinimumProtein>10.5</MinimumProtein>  
    <ProteinIncrement>2</ProteinIncrement>  
  </GrossMargin>  
</folder>
```

- Your Solution Explorer should look like:



- Start the user interface, and add the toolbox folder we just created (Click *Manage* at the bottom of the screen).
- You should also see your toolbox with a *GrossMargin* object. This can now be dragged onto your simulation. We'll create a *Continuous Wheat Simulation* and drop the *GrossMargin* object onto the paddock. We'll then drag an irrigation component to our paddock. This is because our model has an input called *irrig\_tot*.
- The simulation should now run. You should also be able to output *Bank* or use it in Manager expressions. The event *NegativeBank* will also be available as an output frequency.
- The APSIM User interface will look for variables and events for the component in the type XML file (*GrossMarginType.xml*). The variables and events can be listed as in the example above. The alternative is that the variables and events can be generated automatically. A tool called *ProbeDLL* is released with APSIM that calls the DLL and generates a description XML file. This tool is run over all APSIM components prior to creating the APSIM release installation. The description files are all put in the `c:\program files\apsim7\documentation\modelinfo` directory. The type XML files then point to these files eg.

```

GrossMarginType.xml


---


...
<variables link="%apsim%\Documentation\ModelInfo\grossmargin.xml" />
<events link="%apsim%\Documentation\ModelInfo\grossmargin.xml" />
...

```

The *ProbeDLL* is a command line tool that takes 3 arguments. The first is the name of the DLL to probe, the second is the name of the XML configuration file to pass to it and the third is the name of the XML file to generate. The APSIM build system uses *ProbeDLL* in a batch file like this:

```

ProbeAll.bat


---


ProbeDLL %MODEL%\Plant.dll %MODEL%\Barley.xml %MODELINFO%\Barley.xml
ProbeDLL %MODEL%\Plant.dll %MODEL%\Oats.xml %MODELINFO%\Oats.xml
...

```

`%MODEL%` is an environment variable that points to the model directory (e.g. `c:\Apsim\Model`). `%MODELINFO%` is another environment variable that points to the modelinfo directory (e.g. `c:\Apsim\Documentation\ModelInfo`)

## The APSIM type system

All types in APSIM are defined in this XML file:

%apsim%\Model\DataTypes\DataTypes.xml

### DataTypes.xml

---

```
<type name="FomAdded">
  <field name="layers" unit="mm" kind="double" array="T" />
  <field name="fom" array="T">
    <field name="weight" unit="kg/ha" kind="double"/>
    <field name="n" unit="kg/ha" kind="double"/>
    <field name="p" unit="kg/ha" kind="double"/>
    <field name="s" unit="kg/ha" kind="double"/>
    <field name="ash_alk" unit="mol/ha" kind="double"/>
    <field name="dmd" unit="-" kind="double"/>
  </field>
</type>
```

To create a new data type, simply edit this file and rebuild APSIM by running %apsim%\Model\Build\BuildAll.bat. This XML is then used to auto-create source code for all supported languages (including .NET). The new type should then be available in your model. The suffix *Type* is added to all types e.g. in this example the .NET type would be *FomAddedType*.