pCOLAD-for-Dynamo development

part 2

Contents

[Introduction 4](#_Toc444012867)

[For and While loops do work in Dynamo 4](#_Toc444012868)

[C# Zero Touch example code: 5](#_Toc444012869)

[Translated Zero Touch example code to VB.net: 6](#_Toc444012870)

[The basic functionality of pCOLADdynamo is working 10](#_Toc444012871)

[The code of HelloDynamo.cs: 12](#_Toc444012872)

[Need to go to C# and XAML 17](#_Toc444012873)

[What finally works is the following. 20](#_Toc444012874)

[A closer look to Model-View-ViewModel pattern 22](#_Toc444012875)

[A simple WPF example 24](#_Toc444012876)

[Simple solution for image changing button 34](#_Toc444012877)

[The most simple MVVM CSV LOADER 38](#_Toc444012878)

[Basic script for pSHARE for Dynamo 39](#_Toc444012879)

[Add a checkbox for obstruction of parameter value 41](#_Toc444012880)

[Get the output of pSHARE in right format 50](#_Toc444012881)

[Save to csv file and History file 54](#_Toc444012882)

[How to put your solution on GitHub 61](#_Toc444012883)

[Make a variable input node 71](#_Toc444012884)

[Add the Histroy button that hides the others if clicked and shows the history of csv changes. 77](#_Toc444012885)

[Add CheckAll and UncheckAll buttons 80](#_Toc444012886)

[Changes in red 80](#_Toc444012887)

[Thumbnails and full screen images 91](#_Toc444012888)

[Use WPF Profiling Tools to Profile a WPF Applicaton 104](#_Toc444012889)

[Updating Windows, Visual Studio and Dynamo 105](#_Toc444012890)

[Test pCOLAD in Grasshoper and Dynamo together 105](#_Toc444012891)

[Reminder for working with the code after some time. 110](#_Toc444012892)

Figures

[Figure 1 Search for things you don’t understand 4](#_Toc444012893)

[Figure 2 Find the right branch in the GitHub repository of Dynamo 4](#_Toc444012894)

[Figure 3 For and While loops do work now in Dynamo 5](#_Toc444012895)

[Figure 4 My first C# custom node for Dynamo 5](#_Toc444012896)

[Figure 5 Zero Touch approach in VB.net works OK. 6](#_Toc444012897)

[Figure 6 A form can be displayed with Zero Touch approach in VB.net 8](#_Toc444012898)

[Figure 7 Next step showing two nodes and menu 9](#_Toc444012899)

[Figure 8 The basic functionality of pCOLADdynamo is working 10](#_Toc444012900)

[Figure 9 Result of hitting F5 (Run) and the Say “Hello Dynamo”button. 16](#_Toc444012901)

[Figure 10 Errors and warnings when building Dynamo solution 20](#_Toc444012902)

[Figure 11 Setting of Entrian plugin for attaching to process of Dynamo 21](#_Toc444012903)

[Figure 12 Choose right branch and Fork the repository 21](#_Toc444012904)

[Figure 13 pCOLAD in Rhino/Grasshopper 22](#_Toc444012905)

[Figure 14 The MVVM pattern and interactions 23](#_Toc444012906)

[Figure 15 Error in pCOLLECT 51](#_Toc444012907)

[Figure 16 So far so good… 55](#_Toc444012908)

[Figure 17 Through debugging when moving mouse over a variable you get all the properties, methods etc 57](#_Toc444012909)

[Figure 18 Using intelli sense to find the value of output of File Path node 59](#_Toc444012910)

[Figure 19 csv file with softreturns in row 7 loaded in DataTable 96](#_Toc444012911)

[Figure 20 Test for mouse hover on images 101](#_Toc444012912)

# Introduction

In 2013 Hans Hubers initiated a research called pCOLAD together with Michela Turrin, Irem Erbas and Ioannis Chatzikonstantinou. It was aimed at developing a method, prototype and case study for parametric COLaborative Architectural Design. pCOLAD was well received and presented at the eCAADe2014 conference in NewCastle. It used Grasshopper and VB.net as software. In 2013 this software was the best choice for a parametric solution. However in the course of 2014 a generative parametric solution became useful for Revit, called Dynamo. Since Revit is the most used BIM software, it seems adequate to investigate if the results of the pCOLAD project can be converted to Revit/Dynamo and if a way can be found to use the VB.net prototype in that environment or if adaption is needed or even a new prototype. A report was written about the first approach.

This second report about starting with Dynamo investigates the use of a VB.net prototype for pCOLAD in the Dynamo environment. Dynamo is being developed as open source on [GitHub](https://github.com/), a web based repository with versioning and collaboration functionality for software projects. To get acquainted with GitHub go [here](https://guides.github.com/). [There is a way](https://github.com/DynamoDS/Dynamo/wiki/How-To-Create-Your-Own-Nodes#recursion) to simply insert nodes through dll’s using the Zero Touch interface. However it is not using VB.net but C#. Does a VB.net dll work anyway, or do we have to translate pCOLAD nodes to C#? The Zero Touch approach seems not suitable for creating new objects. Does this imply that we cannot display a form with the content of a csv file? If we want to create new objects we can follow the example of the ColorRange component. In GitHub you can find the open source of Dynamo and many examples. If you don’t understand something type it in the search window and hit Enter (Figure 1).

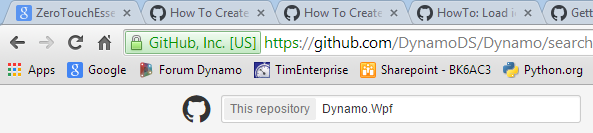


Figure 1 Search for things you don’t understand

However be aware that GitHub shows a default branch which is not necessarily the latest. E.g. if you work with Dynamo 0.7.5 then find that branch in GitHub because otherwise you would maybe find outdate examples of code (like I did, it cost me several days to find this out, Figure 2).

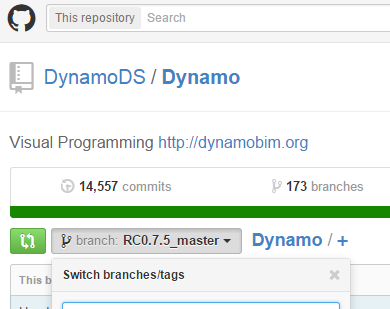


Figure 2 Find the right branch in the GitHub repository of Dynamo

The example code Dynamo\src\Libraries\CoreNodesUI\ColorRange.cs and Dynamo\src\Libraries\CoreNodesUI\UI\ColorRangeNodeViewCustomization.cs

can be found [here](https://github.com/DynamoDS/Dynamo/tree/master/src/Libraries/CoreNodesUI) (in the meanwhile moved to [here](https://github.com/DynamoDS/DynamoSamples/tree/master/src)). It also says: “When creating nodes that require custom UI, exposing static constructors and methods won’t work (HH, what is the case with Zero Touch approach – so useless if you want to display a form with lists). We need to implement a NodeModel class to represent the core logic of the node and provide a class implementing INodeViewCustomization<ColorRange>, which handles building the WPF-specific user interface logic.” Instead of <ColorRange> we probably will have to create a class INodeViewCustomization<Form>.

In order to delete custom nodes go here: C:\Users\jhubers\AppData\Roaming\Dynamo\0.7\definitions and delete the ones you don’t need anymore.

# For and While loops do work in Dynamo

Searching for ways to create plug-ins for Dynamo, I found that now For and While loops do work in Dynamo. However you must use an Imperative function for it to work ([Figure 3](http://dynamobim.com/forums/topic/while-loop-question/)).

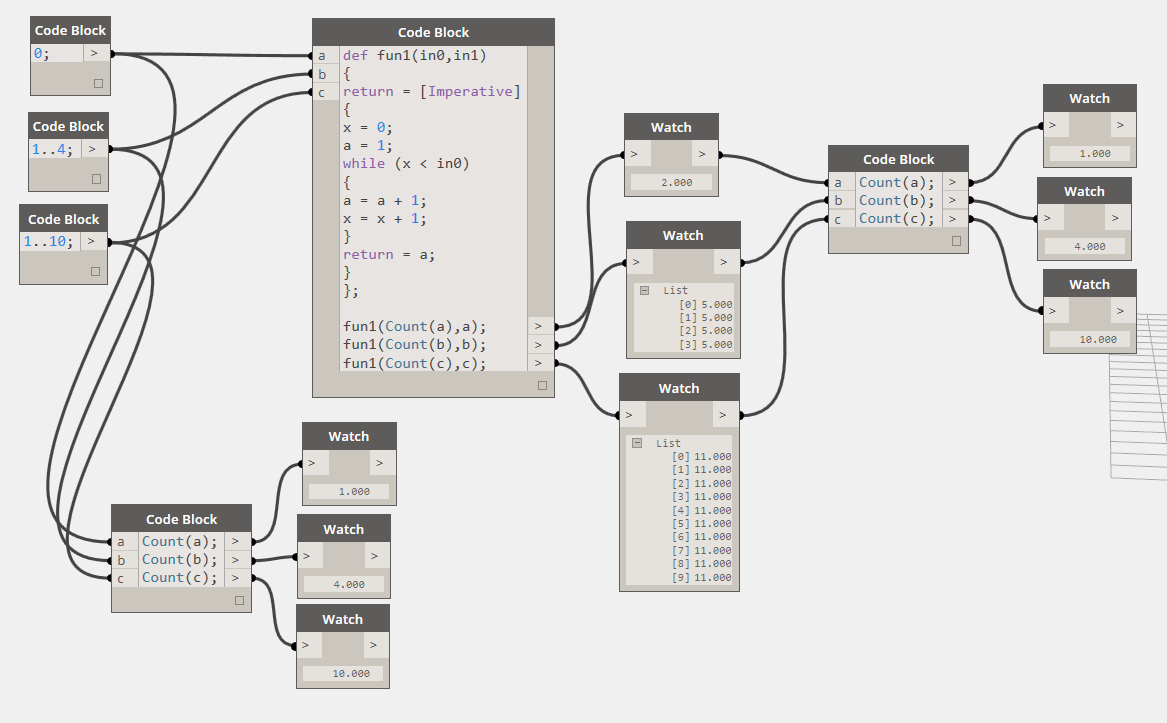


Figure 3 For and While loops [do work now](http://dynamobim.com/forums/topic/while-loop-question/) in Dynamo

But that is beside the point. [Here](https://github.com/DynamoDS/Dynamo/wiki/Getting-Started-with-Dynamo-Development) we can find the (outdated) instruction for creating your own Dynamo nodes. In C#, but probably easy to [convert](http://www.carlosag.net/Tools/CodeTranslator) to VB.net… [Here](https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development) you can find the procedure using Zero Touch for C# nodes.

But first try the Zero Touch approach. Copy the code to a new project and change the names and functions. E.g. to subtract and divide two doubles. This works ok. You can simply load the dll from the file menu of Dynamo (Figure 4).

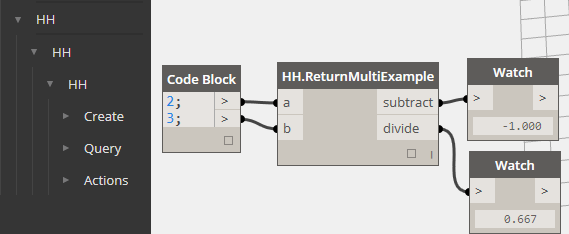


Figure 4 My first C# custom node for Dynamo

# C# Zero Touch example code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Autodesk.DesignScript.Runtime;

using Autodesk.DesignScript.Interfaces;

using Autodesk.DesignScript.Geometry;

///////////////////////////////////////////////////////////////////

/// NOTE: This project requires references to the ProtoInterface

/// and ProtoGeometry DLLs. These are found in the Dynamo install

/// directory.

///////////////////////////////////////////////////////////////////

namespace HH

{

public class HH

{

// Two private variables for example purposes

private double \_a;

private double \_b;

// We make the constructor for this object internal because the

// Dynamo user should construct an object through a static method

internal HH(double a, double b)

{

\_a = a;

\_b = b;

}

/// <summary>

/// An example of how to construct an object via a static method.

/// This is needed as Dynamo lacks a "new" keyword to construct a

/// new object

/// </summary>

/// <param name="a">1st number. This will be stored in the Class.</param>

/// <param name="b">2nd number. This will be stored in the Class</param>

/// <returns>A newly-constructed ZeroTouchEssentials object</returns>

public static HH MyFirst(double a, double b)

{

return new HH(a, b);

}

/// <summary>

/// Example property returning the value \_a inside the object

/// </summary>

public double A

{

get { return \_a; }

}

/// <summary>

/// Example property returning the value \_b inside the object

/// </summary>

public double B

{

get { return \_b; }

}

/// <summary>

/// An example showing how to return multiple values from a Zero-Touch imported node

/// The names in the attribute should match the keys in the returned dictionary.

/// </summary>

/// <param name="a">First number.</param>

/// <param name="b">Second number.</param>

/// <returns name="add">Number created by adding two inputs together.</returns>

/// <returns name="mult">Number created by multiplying two inputs together.</returns>

/// <search>example,multi</search>

[MultiReturn(new[] { "subtract", "divide" })]

public static Dictionary<string, object> ReturnMultiExample(double a, double b)

{

return new Dictionary<string, object>

{

{ "subtract", (a - b) },

{ "divide", (a / b) }

};

}

}

}

Now try with VB.net

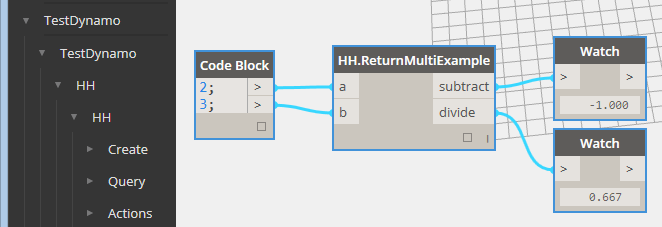


Figure 5 Zero Touch approach in VB.net works OK.

# Translated Zero Touch example code to VB.net:

Imports System

Imports System.Collections.Generic

Imports System.Linq

Imports System.Text

Imports System.Threading.Tasks

Imports Autodesk.DesignScript.Runtime

Imports Autodesk.DesignScript.Interfaces

Imports Autodesk.DesignScript.Geometry

'////////////////////////////////////////////////////////////////

' NOTE: This project requires references to the ProtoInterface

'nd ProtoGeometry DLLs. These are found in the Dynamo install

'irectory.

'////////////////////////////////////////////////////////////////

Namespace HH

Public Class HH

' Two private variables for example purposes

Private \_a As Double

Private \_b As Double

' We make the constructor for this object internal because the

' Dynamo user should construct an object through a static method

Friend Sub New(ByVal a As Double, ByVal b As Double)

MyBase.New()

\_a = a

\_b = b

End Sub

''' <summary>

''' An example of how to construct an object via a static method.

''' This is needed as Dynamo lacks a "new" keyword to construct a

''' new object

''' </summary>

''' <param name="a">1st number. This will be stored in the Class.</param>

''' <param name="b">2nd number. This will be stored in the Class</param>

''' <returns>A newly-constructed ZeroTouchEssentials object</returns>

Public Shared Function MyFirst(ByVal a As Double, ByVal b As Double) As HH

Return New HH(a, b)

End Function

''' <summary>

''' Example property returning the value \_a inside the object

''' </summary>

Public ReadOnly Property A As Double

Get

Return \_a

End Get

End Property

''' <summary>

''' Example property returning the value \_b inside the object

''' </summary>

Public ReadOnly Property B As Double

Get

Return \_b

End Get

End Property

''' <summary>

''' an example showing how to return multiple values from a zero-touch imported node

''' the names in the attribute should match the keys in the returned dictionary.

''' </summary>

''' <param name="a">first number.</param>

''' <param name="b">second number.</param>

''' <returns name="add">number created by adding two inputs together.</returns>

''' <returns name="mult">number created by multiplying two inputs together.</returns>

''' <search>example,multi</search>

<MultiReturn("subtract", "divide")> \_

Public Shared Function ReturnMultiExample(ByVal a As Double, ByVal b As Double) As Dictionary(Of String, Object)

Dim nD As New Dictionary(Of String, Object)()

nD.Add("subtract", a - b)

nD.Add("divide", a / b)

Return nD

End Function

End Class

End Namespace

We could also consider using Python. But first try to get as far as possible with the existing pCOLAD VB.net code.

Now try to display a form.

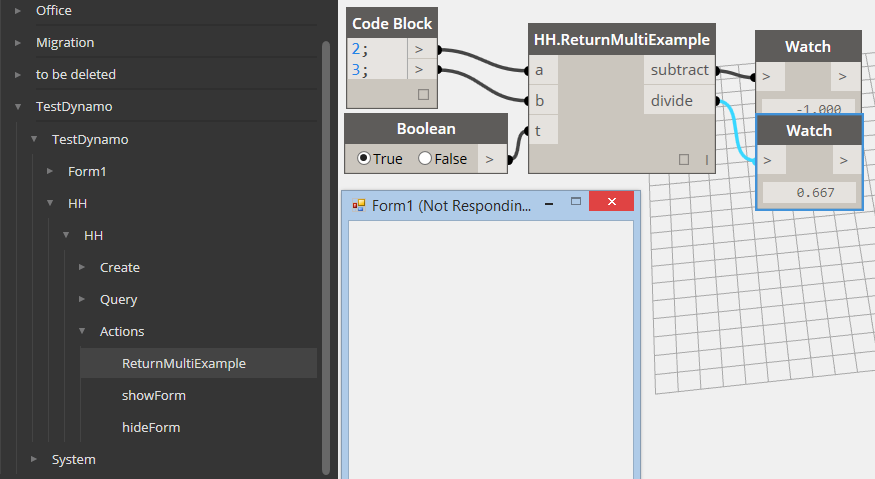


Figure 6 A form can be displayed with Zero Touch approach in VB.net

It turns out that a form can be displayed (Figure 6), but a lot of unnecessary nodes come with it… This changed while changing Public into Private or Friend for some of the statements in next code. [Here](http://msdn.microsoft.com/en-us/library/76453kax.aspx) you can find what this kind of access level modifiers do. If you set a class or module (sub or function) to Public, it shows up in the Dynamo Menu. So for anything else then the classes pCOLLECT, pSHARE, pPARAM and the corresponding action functions (e.g. Public Shared Function pSHARE(ByVal b As Boolean, ByVal p As List(Of String)) As Dictionary(Of String, Object)), set it to Private. So if you need to access variables throughout your class you will have to set them to Shared too. Read [here](http://msdn.microsoft.com/en-us/library/zc2b427x.aspx) what Shared is about.

If you try to close the Form, Dynamo crashes. The Boolean Toggle however works fine with showing and hiding the form. We probably have to add a form.close routine somewhere. Code until now:

Imports System

Imports System.Collections.Generic

Imports System.Linq

Imports System.Text

Imports System.Threading.Tasks

Imports Autodesk.DesignScript.Runtime

Imports Autodesk.DesignScript.Interfaces

Imports Autodesk.DesignScript.Geometry

'////////////////////////////////////////////////////////////////

' NOTE: This project requires references to the ProtoInterface

'nd ProtoGeometry DLLs. These are found in the Dynamo install

'irectory.

'////////////////////////////////////////////////////////////////

Namespace HH

Public Class HH

' Two private variables for example purposes

Private \_a As Double

Private \_b As Double

' Try with a boolean to display a form

Private \_t As Boolean

' We make the constructor for this object internal because the

' Dynamo user should construct an object through a static method

Friend Sub New(ByVal a As Double, ByVal b As Double, ByVal t As Boolean)

MyBase.New()

\_a = a

\_b = b

\_t = t

End Sub

''' <summary>

''' An example of how to construct an object via a static method.

''' This is needed as Dynamo lacks a "new" keyword to construct a

''' new object

''' </summary>

''' <param name="a">1st number. This will be stored in the Class.</param>

''' <param name="b">2nd number. This will be stored in the Class</param>

''' <returns>A newly-constructed ZeroTouchEssentials object</returns>

Public Shared Function MyFirst(ByVal a As Double, ByVal b As Double, ByVal t As Boolean) As HH

Return New HH(a, b, t)

End Function

''' <summary>

''' Example property returning the value \_a inside the object

''' </summary>

Public ReadOnly Property A As Double

Get

Return \_a

End Get

End Property

''' <summary>

''' Example property returning the value \_b inside the object

''' </summary>

Public ReadOnly Property B As Double

Get

Return \_b

End Get

End Property

Public ReadOnly Property T As Boolean

Get

Return \_t

End Get

End Property

''' <summary>

''' an example showing how to return multiple values from a zero-touch imported node

''' the names in the attribute should match the keys in the returned dictionary.

''' </summary>

''' <param name="a">first number.</param>

''' <param name="b">second number.</param>

''' <returns name="add">number created by adding two inputs together.</returns>

''' <returns name="mult">number created by multiplying two inputs together.</returns>

''' <search>example,multi</search>

<MultiReturn("subtract", "divide")> \_

Public Shared Function ReturnMultiExample(ByVal a As Double, ByVal b As Double, ByVal t \_

As Boolean) As Dictionary(Of String, Object)

Dim nD As New Dictionary(Of String, Object)()

nD.Add("subtract", a - b)

nD.Add("divide", a / b)

If t Then showForm() Else hideForm()

Return nD

End Function

Public Shared m\_form As Form1

Public Shared Sub showForm()

If m\_form Is Nothing Then m\_form = New Form1

If m\_form.Visible = False Then m\_form.Show()

End Sub

Public Shared Sub hideForm()

If Not m\_form Is Nothing Then m\_form.Visible = False

End Sub

End Class

End Namespace

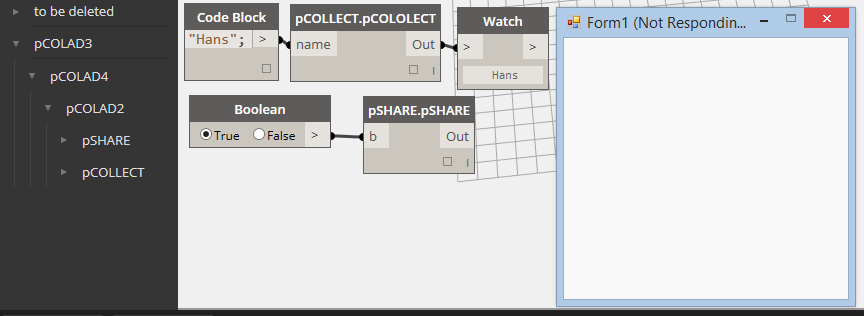


Figure 7 Next step showing two nodes and menu

Figure 7 shows that the main menu item (pCOLAD3) is determined by the name of the assembly (in Application properties of the project in VS). The name of the second level in the menu (pCOLAD4) is determined by the Root namespace (also set in Application properties of the project in VS). The name of the third level (pCOLAD2) comes from the namespace name. If you leave out the name of the Root namespace, you get rid of 1 level. Of course we would like to have pCOLAD as first level and pCOLLECT, pSHARE and pPARAM as second levels. So that should be name spaces. Or we should do without name spaces… only classes?

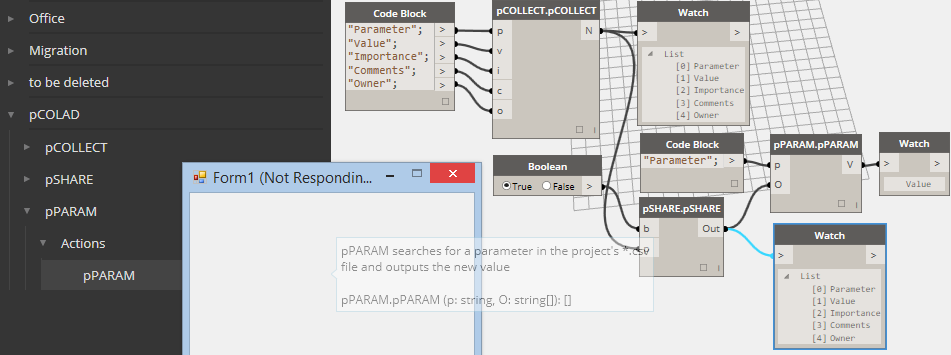


Figure 8 The basic functionality of pCOLADdynamo is working

# The basic functionality of pCOLADdynamo is working

Next VB.net basic code was used in Figure 8:

Imports System

Imports System.Collections.Generic

Imports System.Linq

Imports System.Text

Imports System.Threading.Tasks

Imports Autodesk.DesignScript.Runtime

Imports Autodesk.DesignScript.Interfaces

Imports Autodesk.DesignScript.Geometry

'////////////////////////////////////////////////////////////////

' NOTE: This project requires references to the ProtoInterface

' and ProtoGeometry DLLs. These are found in the Dynamo install

' directory.

'////////////////////////////////////////////////////////////////

Public Class pCOLLECT

Private \_p As String

Private \_v As String

Private \_i As String

Private \_c As String

Private \_o As String

Friend Sub New(ByVal p As String, ByVal v As String, ByVal i As String, ByVal c As String, ByVal o As String)

MyBase.New()

\_p = p

\_v = v

\_i = i

\_c = c

\_o = o

End Sub

''' <summary>

''' pCOLLECT get's the attributes of the connected parameter

''' the output should be connected a pSHARE node or through a list.add node

''' </summary>

''' <param name="p">the parameter name.</param>

''' <param name="v">the value of the parameter.</param>

''' <param name="i">the importance of the parameter.</param>

''' <param name="c">the comment of the parameter.</param>

''' <param name="o">the owner of the parameter.</param>

''' <returns name="N">the attributes of the parameter as a list of strings.</returns>

''' <search>collect,pCOLAD</search>

<MultiReturn("N")> \_

Public Shared Function pCOLLECT(ByVal p As String, ByVal v As String, ByVal i As String, ByVal c As String, \_

ByVal o As String) As Dictionary(Of String, Object)

Dim outputList As New List(Of String)

Dim nD2 As New Dictionary(Of String, Object)

'outputList = Nothing

outputList.Add(p)

outputList.Add(v)

outputList.Add(i)

outputList.Add(c)

outputList.Add(o)

nD2.Add("N", outputList)

Return nD2

End Function

End Class

Public Class pSHARE

Private \_b As Boolean

Private \_p As List(Of String)

Friend Sub New(ByVal b As Boolean, ByVal p As List(Of String))

MyBase.New()

\_b = b

\_p = p

End Sub

''' <summary>

''' pSHARE displays the content of the project's \*.csv file with the

''' shared parameters, their values etc.

''' </summary>

''' <param name="b">on or off togle.</param>

''' <param name="p">the connected parameters through pCOLLECT.</param>

''' <returns name="Out">the list of shared parameters.</returns>

''' <search>share,pCOLAD</search>

<MultiReturn("Out")> \_

Public Shared Function pSHARE(ByVal b As Boolean, ByVal p As List(Of String)) As Dictionary(Of String, Object)

Dim nD As New Dictionary(Of String, Object)()

nD.Add("Out", p)

If b Then showForm() Else hideForm()

Return nD

End Function

Private Shared m\_form As Form1

Private Shared Sub showForm()

If m\_form Is Nothing Then m\_form = New Form1

If m\_form.Visible = False Then m\_form.Show()

End Sub

Private Shared Sub hideForm()

If Not m\_form Is Nothing Then m\_form.Visible = False

End Sub

End Class

Public Class pPARAM

Private \_p As String

Private \_o As List(Of String)

Friend Sub New(ByVal p As String, ByVal o As List(Of String))

MyBase.New()

\_p = p

\_o = o

End Sub

''' <summary>

''' pPARAM searches for a parameter in the project's \*.csv file

''' and outputs the new value

''' </summary>

''' <param name="p">the parameter name to search for in the output of pSHARE.</param>

''' <param name="o">the output of pSHARE.</param>

''' <returns name="V">the parameter value as string.</returns>

''' <search>share,pCOLAD</search>

<MultiReturn("V")> \_

Public Shared Function pPARAM(ByVal p As String, ByVal O As List(Of String)) As Dictionary(Of String, Object)

Dim myValue As String = ""

Dim i As Integer = 0

Dim nD2 As New Dictionary(Of String, Object)()

For Each line As String In O

If line = p Then

myValue = O(i + 1)

Exit For

End If

i = i + 1

Next

nD2.Add("V", myValue)

Return nD2

End Function

End Class

We could probably create 1 menu (class) called pCOLAD and 3 actions pCOLLECT, pSHARE and pPARAM (Public Shared Functions). But then we get a difference with the structur of the pCOLAD Grasshopper project where every action is a separate class. So better stick to that structure for future maintenance.

Now gradually combine parts of the pCOLAD10.sln solution into the pCOLADdynamo.sln solution. First the most complicated: pSHARE. In order to have the complete Form that displays the \*.csv file just copy and paste that form. There is a whole bunch of compex addhandler and removehandler stuff going on when the LoadForm() and UnloadForm() routines are called. Since you call it from a Public Shared Function, these routines must be Shared too.

Sometimes you try things with intellisence (type a . and get a list of items). The icons next to the items have a meaning of course. You can find that [here](http://msdn.microsoft.com/en-us/library/y47ychfe.aspx).

For creating a node where you can add inputs you probably have to go [here](https://github.com/DynamoDS/Dynamo/blob/master/src/Libraries/CoreNodesUI/CreateList.cs). Be aware that importing namespaces like Dynamo.Models is only possible if the dll where this namespace is located is referenced. Good trick is to reference all the dll’s and exe files in the Dynamo folder and then right click in the code somewhere/Organize Usings/Remove unused references to get rid of the ones you don’t use. Translated as VB.net:

Imports System.Collections.Generic  
Imports System.Linq  
Imports Dynamo.Models  
Imports Dynamo.Nodes  
Imports Dynamo.Wpf  
Imports ProtoCore.AST.AssociativeAST  
  
Namespace DSCoreNodesUI  
      
    Public Class CreateListNodeViewCustomization  
        Inherits VariableInputNodeViewCustomization  
        Implements INodeViewCustomization(Of CreateList)  
          
        Public Sub CustomizeView(ByVal model As CreateList, ByVal nodeView As Dynamo.Controls.NodeView)  
            MyBase.CustomizeView(model, nodeView)  
        End Sub  
    End Class  
      
    <NodeName("List.Create"),  \_  
     NodeDescription("Makes a new list out of the given inputs"),  \_  
     NodeCategory(BuiltinNodeCategories.CORE\_LISTS\_CREATE),  \_  
     IsDesignScriptCompatible()>  \_  
    Public Class CreateList  
        Inherits VariableInputNode  
          
        Public Sub New()  
            MyBase.New  
            InPortData.Add(New PortData("index0", "Item Index #0"))  
            OutPortData.Add(New PortData("list", "A list"))  
            RegisterAllPorts  
            ArgumentLacing = LacingStrategy.Disabled  
        End Sub  
          
        Protected Overrides Function GetInputName(ByVal index As Integer) As String  
            Return ("index" + index)  
        End Function  
          
        Protected Overrides Function GetInputTooltip(ByVal index As Integer) As String  
            Return ("Item Index #" + index)  
        End Function  
          
        Protected Overrides Sub RemoveInput()  
            If (InPortData.Count > 1) Then  
                MyBase.RemoveInput  
            End If  
        End Sub  
          
        Public Overrides Function BuildOutputAst(ByVal inputAstNodes As List(Of AssociativeNode)) As IEnumerable(Of AssociativeNode)  
            If IsPartiallyApplied Then  
                Dim connectedInput As var = Enumerable.Range(0, InPortData.Count).Where(HasConnectedInput).Select(() => {  }, New IntNode(x), as, AssociativeNode).ToList  
                Dim paramNumNode As var = New IntNode(InPortData.Count)  
                Dim positionNode As var = AstFactory.BuildExprList(connectedInput)  
                Dim arguments As var = AstFactory.BuildExprList(inputAstNodes)  
                Dim functionNode As var = New IdentifierListNode() {LeftNode=newIdentifierNode(DSCore.ListUnknown, RightNode=newIdentifierNode(\_\_CreateUnknown}  
                Dim inputParams As var = New List(Of AssociativeNode)() {functionNode, paramNumNode, positionNode, arguments, AstFactory.BuildBooleanNode(false)}  
                Return  
                AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildFunctionCall("\_SingleFunctionObject", inputParams))  
                  
            End If  
            Return  
            AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes))  
              
        End Function  
    End Class  
End Namespace

The code between < > is an attribute list. You can read [here](http://msdn.microsoft.com/en-us/library/z0w1kczw(v=vs.140).aspx) when it is used.

However to start with the programming of custom nodes we find a simple example [here](https://github.com/DynamoDS/Dynamo/blob/master/src/Libraries/Samples/SampleLibraryUI/HelloDynamo.cs). But it turns out to be in an outdate branche of the repository. You will have to go [here](https://github.com/DynamoDS/Dynamo/tree/RC0.7.5_master/src/Libraries/Samples/SampleLibraryUI) instead. Or maybe a newer branch (see Figure 2). If you fork and download the project, you can start the Visual Project file by double clicking SampleLibraryUI.csproj in D:\Data\Research\Dynamo\Dynamo\src\Libraries\Samples\SampleLibraryUI\. Of course there are some libraries missing and you need the .xaml and xaml.cs file (you can not just copy the C# code). The libraries you need are: DynamoCore.dll, ProtoCore.dll, ProtoInterface.dll and Microsoft.Practices.Prism.dll. All to be found in C:\Program Files\Dynamo 0.7\. And the Windows.Base, System.Xaml, PresentationFramework and PresentationCore which is in the Assemblies/Framework part of the Add Reference dialogue. Make sure that in the properties of all the references Copy Local is set to false (otherwise you get a lot of unnecessary dlls in your Debug folder). Then you can build the solution and copy the dll that you made (in my case test.dll because that was the name of the solution) to the Dynamo/Nodes directory in C:\Program Files.

# The code of HelloDynamo.cs:

using System.Collections.Generic;

using System.Windows;

//Be aware that you need references DynamoCore, ProtoCore, ProtoInterface and Microsoft.Practices.Prism.dll

using Autodesk.DesignScript.Runtime;//you need to reference ProtoInterface.dll in C:\Program Files\Dynamo 0.7\

using Dynamo.Controls; //you need to reference DynamoCore.dll in C:\Program Files\Dynamo 0.7\

using Dynamo.Models;//you need to reference DynamoCore.dll in C:\Program Files\Dynamo 0.7\

using Dynamo.UI;//you need to reference DynamoCore.dll in C:\Program Files\Dynamo 0.7\

using Dynamo.UI.Commands;//you need to reference DynamoCore.dll in C:\Program Files\Dynamo 0.7\

using ProtoCore.AST.AssociativeAST;//you need to reference ProtoCore.dll in C:\Program Files\Dynamo 0.7\

namespace SamplesLibraryUI

{

/// <summary>

/// This exmple shows how to create a UI node for Dynamo

/// which loads custom data-bound UI into the node's view

/// at run time.

///

/// Nodes with custom UI follow a different loading path

/// than zero touch nodes. The assembly which contains

/// this node needs to be located in the 'nodes' folder in

/// Dynamo in order to be loaded at startup.

///

/// Dynamo uses the MVVM\* model of programming,

/// in which the UI is data-bound to the view model, which

/// exposes data from the underlying model. Custom UI nodes

/// are a hybrid because NodeModel objects already have an

/// associated NodeViewModel which you should never need to

/// edit. So here we will create a data binding between

/// properties on our class and our custom UI.

///

/// </summary>

///

// The NodeName attribute is what will display on

// top of the node in Dynamo

[NodeName("Hello Dynamo")]

// The NodeCategory attribute determines how your

// node will be organized in the library. You can

// specify your own category or use one of the

// built-ins provided in BuiltInNodeCategories.

[NodeCategory("Sample Nodes")]

// The description will display in the tooltip

// and in the help window for the node.

[NodeDescription("A sample UI node which displays custom UI.")]

[IsDesignScriptCompatible]

public class HelloDynamo : NodeModel, IWpfNode

{

#region private members

private string message;

private double awesome;

#endregion

#region properties

/// <summary>

/// A value that will be bound to our

/// custom UI's awesome slider.

/// </summary>

public double Awesome

{

get { return awesome; }

set

{

awesome = value;

RaisePropertyChanged("Awesome");//be aware that you need to reference Microsoft.Practices.Prism.dll in C:\Program Files\Dynamo 0.7\

RequiresRecalc = true;

}

}

/// <summary>

/// A message that will appear on the button

/// on our node.

/// </summary>

public string Message

{

get { return message; }

set

{

message = value;

// Raise a property changed notification

// to alert the UI that an element needs

// an update.

RaisePropertyChanged("NodeMessage");//be aware that you need to reference Microsoft.Practices.Prism.dll in C:\Program Files\Dynamo 0.7\

}

}

/// <summary>

/// DelegateCommand objects allow you to bind

/// UI interaction to methods on your data context.

/// </summary>

[IsVisibleInDynamoLibrary(false)]

public DelegateCommand MessageCommand { get; set; }

#endregion

#region constructor

/// <summary>

/// The constructor for a NodeModel is used to create

/// the input and output ports and specify the argument

/// lacing.

/// </summary>

/// <param name="workspace"></param>

public HelloDynamo(WorkspaceModel workspace)

: base(workspace)

{

// When you create a UI node, you need to do the

// work of setting up the ports yourself. To do this,

// you can populate the InPortData and the OutPortData

// collections with PortData objects describing your ports.

InPortData.Add(new PortData("something", "Input a string."));

// Nodes can have an arbitrary number of inputs and outputs.

// If you want more ports, just create more PortData objects.

OutPortData.Add(new PortData("something", "A result."));

OutPortData.Add(new PortData("some awesome", "A result."));

// This call is required to ensure that your ports are

// properly created.

RegisterAllPorts();

// The arugment lacing is the way in which Dynamo handles

// inputs of lists. If you don't want your node to

// support argument lacing, you can set this to LacingStrategy.Disabled.

ArgumentLacing = LacingStrategy.CrossProduct;

// We create a DelegateCommand object which will be

// bound to our button in our custom UI. Clicking the button

// will call the ShowMessage method.

MessageCommand = new DelegateCommand(ShowMessage, CanShowMessage);

// Setting our property here will trigger a

// property change notification and the UI

// will be updated to reflect the new value.

Message = "Say 'Hello Dynamo!'";

Awesome = 1;

}

#endregion

#region public methods

/// <summary>

/// If this method is not overriden, Dynamo will, by default

/// pass data through this node. But we wouldn't be here if

/// we just wanted to pass data through the node, so let's

/// try using the data.

/// </summary>

/// <param name="inputAstNodes"></param>

/// <returns></returns>

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

// When you create your own UI node you are responsible

// for generating the abstract syntax tree (AST) nodes which

// specify what methods are called, or how your data is passed

// when execution occurs.

// WARNING!!!

// Do not throw an exception during AST creation. If you

// need to convey a failure of this node, then use

// AstFactory.BuildNullNode to pass out null.

// Using the AstFactory class, we can build AstNode objects

// that assign doubles, assign function calls, build expression lists, etc.

return new[]

{

// In these assignments, GetAstIdentifierForOutputIndex finds

// the unique identifier which represents an output on this node

// and 'assigns' that variable the expression that you create.

// For the first node, we'll just pass through the

// input provided to this node.

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes)),

// For the second node, we'll build a double node that

// passes along our value for awesome.

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(1),

AstFactory.BuildDoubleNode(awesome))

};

}

/// <summary>

/// SetupCustomUIElements is part of the IWpfNode interface.

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="view">The view representing the node in the graph.</param>

[IsVisibleInDynamoLibrary(false)]

public void SetupCustomUIElements(dynNodeView view)

{

// The view variable is a reference to the node's view.

// In the middle of the node is a grid called the InputGrid.

// We reccommend putting your custom UI in this grid, as it has

// been designed for this purpose.

// Create an instance of our custom UI class (defined in xaml),

// and put it into the input grid.

var helloDynamoControl = new HelloDynamoControl();//be aware that you need the xaml and xaml.cs file for this

view.inputGrid.Children.Add(helloDynamoControl);

// Set the data context for our control to be this class.

// Properties in this class which are data bound will raise

// property change notifications which will update the UI.

helloDynamoControl.DataContext = this;

}

#endregion

#region command methods

private bool CanShowMessage(object obj)

{

// I can't think of any reason you wouldn't want to say Hello Dynamo!

// so I'll just return true.

return true;

}

private void ShowMessage(object obj)

{

MessageBox.Show("Hello Dynamo!");//be aware that you need to reference PresentationFramework

}

#endregion

}

}

\* If you are wondering what this MVVM model is about, have a look [here](http://www.codeproject.com/Articles/165368/WPF-MVVM-Quick-Start-Tutorial), [here](http://www.codeproject.com/Articles/278901/MVVM-Pattern-Made-Simple) or [here](http://www.paulspatterson.com/mvvm-and-wpf-for-vb-net-part-1/). Or more officially [here](https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx). MVVM stands for Model-View-ViewModel. It is a software design pattern (aka architecture). To be short it says: *”The goal of using design patterns is to create software that is easy to maintain, reusable, and more efficient and effective. MVVM is a design pattern that abstracts data from what is used to present that data as information. The Model represents the data that will be used. The View represents the presentation that the data is shown as information, and the ViewModel is the binding between the View and Model. Essentially a Model knows nothing about the View, and vice versa. It’s the ViewModel’s responsibility to be the liaison between the Model and the View.”* Next summaries are also from this source.

*To summarize, the view has the following key characteristics:*

* *The view is a visual element, such as a window, page, user control, or data template. The view defines the controls contained in the view and their visual layout and styling.*
* *The view references the view model through its* ***DataContext*** *property. The controls in the view are data bound to the properties and commands exposed by the view model.*
* *The view may customize the data binding behavior between the view and the view model. For example, the view may use value converters to format the data to be displayed in the UI, or it may use validation rules to provide additional input data validation to the user.*
* *The view defines and handles UI visual behavior, such as animations or transitions that may be triggered from a state change in the view model or via the user's interaction with the UI.*
* *The view's code-behind may define UI logic to implement visual behavior that is difficult to express in XAML or that requires direct references to the specific UI controls defined in the view.*

*To summarize, the view model has the following key characteristics:*

* *The view model is a non-visual class and does not derive from any WPF base class. It encapsulates the presentation logic required to support a use case or user task in the application. The view model is testable independently of the view and the model.*
* *The view model typically does not directly reference the view. It implements properties and commands to which the view can data bind. It notifies the view of any state changes via change notification events via the INotifyPropertyChanged and INotifyCollectionChanged interfaces.*
* *The view model coordinates the view's interaction with the model. It may convert or manipulate data so that it can be easily consumed by the view and may implement additional properties that may not be present on the model. It may also implement data validation via the IDataErrorInfo or INotifyDataErrorInfo interfaces.*
* *The view model may define logical states that the view can represent visually to the user.*

*The model has the following key characteristics:*

* *Model classes are non-visual classes that encapsulate the application's data and business logic. They are responsible for managing the application's data and for ensuring its consistency and validity by encapsulating the required business rules and data validation logic.*
* *The model classes do not directly reference the view or view model classes and have no dependency on how they are implemented.*
* *The model classes typically provide property and collection change notification events through the INotifyPropertyChanged and INotifyCollectionChanged interfaces. This allows them to be easily data bound in the view. Model classes that represent collections of objects typically derive from the ObservableCollection<T> class.*
* *The model classes typically provide data validation and error reporting through either the IDataErrorInfo or INotifyDataErrorInfo interfaces.*
* *The model classes are typically used in conjunction with a service or repository that encapsulates data access and caching.*

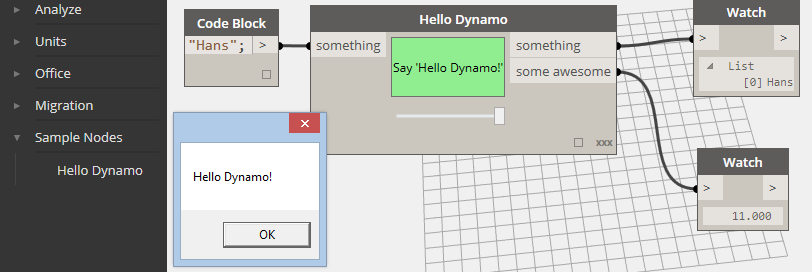


Figure 9 Result of hitting F5 (Run) and the Say “Hello Dynamo”button.

The main elements to learn for a MVVM are:

* How to creating a VB.Net WPF application,
* Understanding [XAML](http://www.wpf-tutorial.com/),
* Learn the concepts and practical approach to using the MVVM design pattern,
* Creating a Model,
* Creating the View,
* Wiring up the View and Model with a ViewModel
* Using Triggers and Delegating properties to take care of happenings that the user does; such as saving data.
* How to use Expression Blend to style something,
* Using the Entity Framework to Model the data,

After some reading, it becomes clear that pCOLAD is based on the oldfashioned Windows Form approach where the data and actions are mainly linked and initiated in the objects on the form. While in a WPF/MVVM (Windows Presentation Foundation/Model View ViewModel) approach the data and actions are completely seperated from the User Interface. [This](https://rachel53461.wordpress.com/2012/10/12/switching-from-winforms-to-wpfmvvm/) explains it well. And [this](https://rachel53461.wordpress.com/2011/05/08/simplemvvmexample/).

## Need to go to C# and XAML

Since the whole of Dynamo is in C# and based on MVVM, it might be wise to convert pCOLAD to this paradigm. Then the examples and the communication about them can be much easier implemented. This means starting over again, learning C# and XAML… ☹ sigh. You can find beginners courses [here](http://www.microsoftvirtualacademy.com/). A good start is Bob Tailor’s video for beginners [here](http://www.microsoftvirtualacademy.com/training-courses/c-fundamentals-for-absolute-beginners). To work with text it helps knowing [these](http://blogs.msdn.com/b/csharpfaq/archive/2004/03/12/what-character-escape-sequences-are-available.aspx?PageIndex=2) combinations for special characters. For the String.Format arguments have a look [here](https://msdn.microsoft.com/en-us/library/System.String.Format.aspx). E.g. in Bob Tailor’s video you learn in lessen 16 about classes. Then you understand better the part in the example code for the HelloDynamo node where there are regions called constructor. A region is just a collapsable part of the code, a functionality of Visual Studio. A constructor is infact a standard default property of the instance of the class. So every instance of the HelloDynamo node gets these properties: InPortData, OutPortData, ArgumentLacing, Message and Awesome. And a constructor is made by calling the name of the class (public HelloDynamo(WorkspaceModel workspace)) within the class (public class HelloDynamo : NodeModel, IWpfNode) and setting these properties to a default value. The : means that it inherit the properties of the class NodeModel and IWpfNode. The constructor also uses the default workspace property of the parent (or base) class by using the : base keyword and the property called *workspace*. This you can learn [here](https://msdn.microsoft.com/en-us/library/vstudio/ms173115(v=vs.110).aspx). Before continuïng better learn the basics of Object Oriented Programming [here](https://msdn.microsoft.com/en-us/library/dd460654.aspx).

Likewise the region called *public methodes* becomes understandable as it gives an action to the class that you can call. E.g. to setup the custom node (public void SetupCustomUIElements(dynNodeView view)). Another method is the public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes).

Override means that the methode with the same name of the base class, or parent class, or super class (all the same) was called *public virtual* which means that it allows to be overriden for (a part) of the methode. Or it might have been called *public abstract,* which means that it must be overriden. If a class is called *static* it means that you can not make an instance and that you don’t use the *new* keyword, but directly use it’s properties, methods etc. Same goes for the keyword *sealed.* E.g. a *String* is a sealed class. You set a variable to be a string and no need to make an instance.

Remember to use the try {} catch {} catch {} etc. finally {} error tracking. When using a List of a certain type. The syntax in *C# is List<class\_name> variable\_name = new List<class\_name>()*; and you go through the list with *foreach (class\_name other\_variable\_name in variable\_name) { do something}*. An interesting feature is the possibility to generate objects while populating the list. E.g. the list of cars:

List<Car> myCars = new List<Car>() {

new Car() { Make = "BMW", Model= "550i", Color=CarColor.Blue, StickerPrice=55000, Year=2009},

new Car() { Make="Toyota", Model="4Runner", Color=CarColor.White, StickerPrice=35000, Year=2010},

new Car() { Make="BMW", Model = "745li", Color=CarColor.Black, StickerPrice=75000, Year=2008},

new Car() {Make="Ford", Model="Escape", Color=CarColor.White, StickerPrice=25000, Year=2008},

new Car() {Make="BMW", Model="55i", Color=CarColor.Black, StickerPrice=57000, Year=2010}

};

Interesting for us is lessen 24 Understanding Event Driven Programming. Here the basics are explained about making a modern program based on Windows Presentation Foundation (WPF). It is a modern alternative for a Windows Form Application. pCOLAD was a Windows Form Application. But WPF doesn’t use Forms. It uses XAML. Exactly what we needed to develop our own nodes for Dynamo! Of course not all is explained and you need to search often the internet for answers. If you really really cannot find an answer, you can always post a question at <http://msdn.microsoft.com/forums> (hit Select a forum to find the suitable one).

It would be interesting to see if we could debug our pCOLADdynamo application within Visual Studio, because you then get feedback about the errors and you save a lot of time for opening Dynamo. Because our application is a class, which you can not run on itself, you need an application that references this class and simulates the actions you would take in Dynamo. For this you have to build a fake Dynamo application that gets around the errors like missing dll’s missing objects or properties like the input and output parameters. Best would be a WPF application, because Dynamo is that.

We could also try to fork and download a local copy of the GitHub DynamoDS project. However when you load the Dynamo.All.2013.sln file and hit the build button yout get 64 Errors and 23 warnings (Figure 10). After analysing it shows that very often the error is about a Source file (AssemblySharedInfo.cs) could not be found. Warning 76 gives a clue why this is: *A custom tool 'TextTemplatingFileGenerator' is associated with file 'AssemblySharedInfo.tt', but the output of the custom tool was not found in the project. You may try re-running the custom tool by right-clicking on the file in the Solution Explorer and choosing Run Custom Tool.*

It generates a file with next content.

using System;

using System.Reflection;

using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following

// set of attributes. Change these attribute values to modify the information

// associated with an assembly.

[assembly: AssemblyCompany("Autodesk, Inc")]

[assembly: AssemblyProduct("Dynamo")]

[assembly: AssemblyCopyright("Copyright © Autodesk, Inc 2014")]

[assembly: AssemblyTrademark("")]

// Make it easy to distinguish Debug and Release (i.e. Retail) builds;

// for example, through the file properties window.

#if DEBUG

[assembly: AssemblyConfiguration("Debug")]

[assembly: AssemblyDescription("Flavor=Debug")] // a.k.a. "Comments"

#else

[assembly: AssemblyConfiguration("Release")]

[assembly: AssemblyDescription("Flavor=Release")] // a.k.a. "Comments"

#endif

[assembly: CLSCompliant(true)]

// Setting ComVisible to false makes the types in this assembly not visible

// to COM components. If you need to access a type in this assembly from

// COM, set the ComVisible attribute to true on that type.

[assembly: ComVisible(false)]

// Note that the assembly version does not get incremented for every build

// to avoid problems with assembly binding (or requiring a policy or

// <bindingRedirect> in the config file).

//

// The AssemblyFileVersionAttribute is incremented with every build in order

// to distinguish one build from another. AssemblyFileVersion is specified

// in AssemblyVersionInfo.cs so that it can be easily incremented by the

// automated build process.

[assembly: AssemblyVersion("0.7.5.4003")]

// By default, the "Product version" shown in the file properties window is

// the same as the value specified for AssemblyFileVersionAttribute.

// Set AssemblyInformationalVersionAttribute to be the same as

// AssemblyVersionAttribute so that the "Product version" in the file

// properties window matches the version displayed in the GAC shell extension.

//[assembly: AssemblyInformationalVersion("1.0.0.0")] // a.k.a. "Product version"

// Version information for an assembly consists of the following four values:

//

// Major Version

// Minor Version

// Build Number

// Revision

//

// You can specify all the values or you can default the Build and Revision Numbers

// by using the '\*' as shown below:

// [assembly: AssemblyVersion("1.0.\*")]

[assembly: AssemblyFileVersion("0.7.5.4003")]

When you run this tool and try building again the build is more or less ok: ========== Build: 49 succeeded, 0 failed, 1 up-to-date, 0 skipped ========== Still a lot of warnings thoug. Much about not being CLS-compliant. CLS stands for Common Language Specification.

In the Libraries/Samples folder we find a SampleLibraryTests project and in there we find a HelloDynamoSystemTests.cs file which says : “/// IMPORTANT!

/// System tests have dependencies on Dynamo core dlls. In

/// order for these tests to work, your test dll needs to be

/// located in the Dynamo core directory. Set your build

/// output path to your Dynamo core directory, or add a copy step that

/// moves the output from this project to the Dynamo core directory.”

So searching for the SampleLibraryTests.dll and -.pdb (in earlier tries we found a lot of errors about missing pdb files, which stands for ProgramDataBase, they are also called symbol files, and they contain the debugging information, more info [here](https://msdn.microsoft.com/en-us/library/aa363368(v=vs.85).aspx)) it turns out they are created in the D:\Data\Research\Dynamo\Dynamo\bin\AnyCPU\Debug directory. Copy these files to the Dynamo Core directory (D:\Data\Research\Dynamo\Dynamo\src\DynamoCore). When trying to run the test you get the error: ---------------------------

*A project with an Output Type of Class Library cannot be started directly.*

*In order to debug this project, add an executable project to this solution which references the library project. Set the executable project as the startup project.*

---------------------------

It turns out that this test file is for internal testing during compilation. So we figure that what is meant with the Dynamo core directory is the directory where all the dll’s and exe’s go: in my case D:\Data\Research\Dynamo\Dynamo\bin\AnyCPU\Debug directory. Or even more likely: C:\Program Files\Dynamo 0.7\nodes.

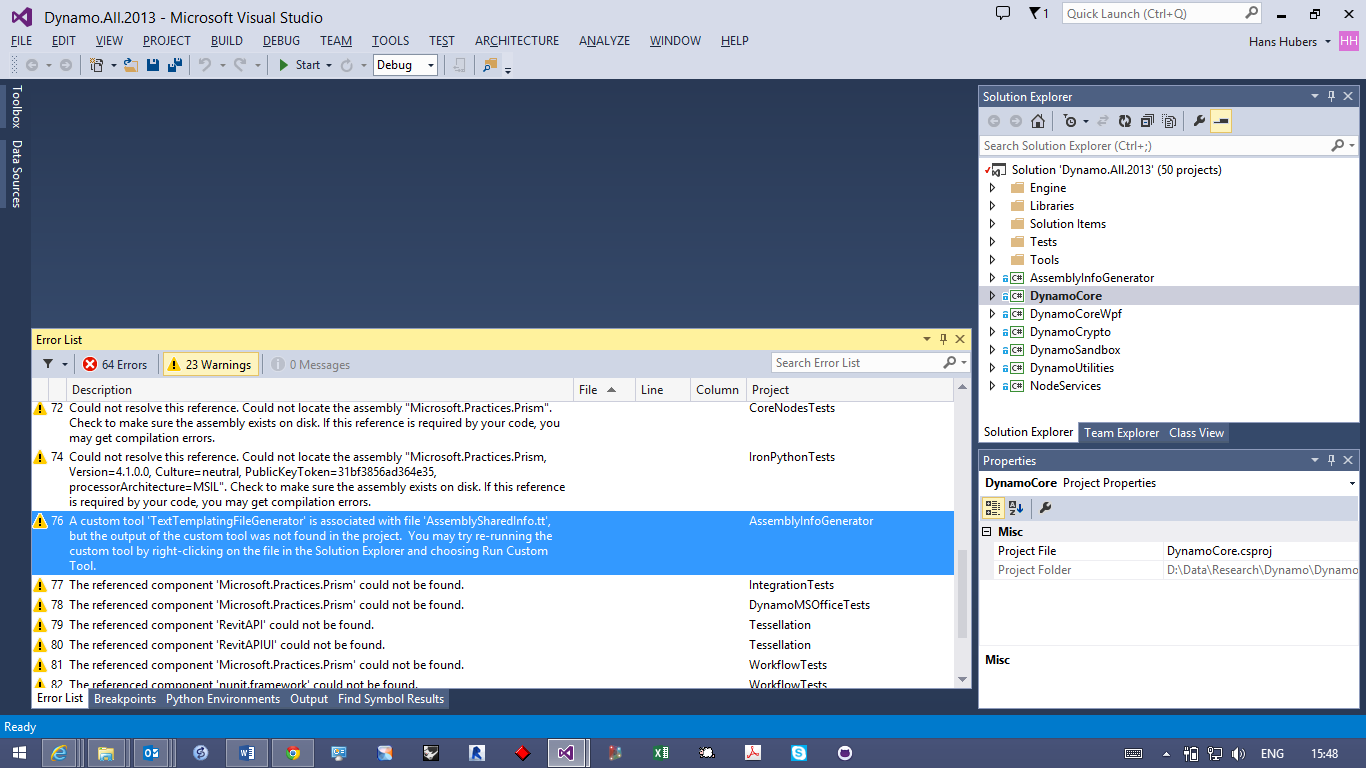


Figure 10 Errors and warnings when building Dynamo solution

Well can always try… So add a WPF project to it that uses the HelloDynamo class.

Some more reading shows that it should be possible to directly test your new node in Visual Studio Dynamo project. Also some more directions on [writing your first new node](https://github.com/DynamoDS/Dynamo/wiki/Getting-Started-with-Dynamo-Development) (scroll to 1/5th). But no, the page is outdated…

For meaning of the icons in the Visual Studio Explorer go [here](https://msdn.microsoft.com/en-us/library/ms181372(v=vs.80).aspx).

# What finally works is the following.

If you want to be able to contribute to the development of Dynamo

1. For absolute beginners: follow the course on C# [here](http://www.microsoftvirtualacademy.com/training-courses/c-fundamentals-for-absolute-beginners). It takes about 2 full days.
2. If you want to be able to contribute to the development of Dynamo, learn how to use GitHub [here](https://github.com/). Become a member.
3. Go to Dynamo on GitHub [here](https://github.com/DynamoDS/Dynamo). Find and select the right branch. If you want to contribute to Dynamo hit the Fork button (Figure 12) and make a local copy by hitting the Clone in Desktop button. Otherwise just download the zip file. Unzip.
4. Right click the start icon of Visual Studio (VS) twice in the task bar and choose Run as administrator (this is necessary for xcopy to program files on C drive at point 8).
5. Open project… … \Dynamo [the version you selected]\src\Libraries\Samples\SampleLibraryUI\SampleLibraryUI.csproj.
6. In the Solution Explorer right click the project SampleLibraryUI and choose Properties…
7. Go to the Debug tab at the left and hit the Start external program button. Hit the … button next to it and navigate to where you installed DynamoSandbox.exe.
8. In the same window go to the Build Events tab and put a post build command in the Build properties of the project that copies the dll and pdb file you will make to the nodes directory of Dynamo. That is where the nodes should be that show up in the menu of Dynamo. E.g.:

REM the echo f is needed if the file doesn't exist yet it tells the compiler that it is a file to be copied, and not a directory

echo f |xcopy /r /y “$(TargetPath)” “C:\Program Files\Dynamo 0.7\nodes\$(TargetName).dll”

1. Important: Set the “Run the post build event:” below it to Always.
2. In the Solution Explorer click the arrow next to References and notice that DynamoCore and ProtoCare have a yellow warning symbol. Delete them. Right click the References entry and choose Add Reference… Go to the Browse tab and hit the Browse… button at the bottom. Navigate to the Dynamo installation folder (normally C:\Program Files\Dynamo …\) and select the DynamoCore.dll and add it to the references. Do the same with ProtoCare.dll in that folder. Set Copy Local… to false in the properties window for both dlls.
3. Notice that in the Solution Explorer the AssemblySharedInfo.cs also has a problem. Delete it[[1]](#footnote-1). In fact it is a link to a file that is generated through a local tool in the complete Dynamo solution. But you can do without it.
4. Double click the HelloDynamo.cs file in the Solution Explorer and set a breakpoint almost at the bottom of the code in line 234 next to MessageBox.Show(“Hello Dynamo!”);
5. Now hit the green start arrow in the top bar of VS, but make sure that the window next to it reads Debug and not Release.
6. Now be patient while DynamoSandbox starts. In the output window you will see a lot of remarks. Also things like threads being exited etc. But wait some minutes until you see the Button of Dynamo show up or change in the task bar. Then go to Dynamo and hit the New button. You will find a new menu called “Sample Nodes” with the Hello Dynamo node in it.
7. Hit the node so it appears on the canvas. Hit the Say ‘Hello Dynamo!’ button in the middle and you will jump to the breakpoint in your code. Now with F10 you can step through your code and see what is going on. It is all very slow, but at least you can trace errors etc.

A faster way is to attach to the DynamoSandbox process. You can automate this with the Entrian Attach Addin. However if you want to change something to your code in both cases you will have to restart Dynamo. Settings for Entrian as in Figure 11. This makes the debugging start when you start DynamoSandbox.

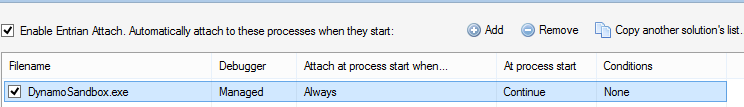


Figure 11 Setting of Entrian plugin for attaching to process of Dynamo

Hit the Rebuild option under BUILD menu (Ctrl+Shift+Alt+B). This forces the dll to be rebuild even if nothing changed. If you get errors, check if Dynamo Sandbox is still running (also in Task Manager = Ctrl+Alt+Del).

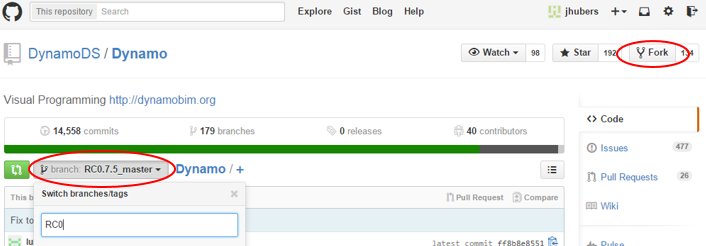


Figure 12 Choose right branch and Fork the repository

We can now start to modify the file towards the pCOLAD content. It should resemble Figure 13. However after some time it turns out that it was a trial version and that you have to pay to continue. A free alternative is found in … It remembers wich process you want to attach to, but the process must be running first. So this is the order of start debugging:

1. Set a Breakpoint where you want to start debugging.
2. Save your work (Ctrl+Shift+S)
3. Rebuild the solution (Ctrl+Shift+Alt+B)
4. Start Dynamo, but don't load a file yet
5. Go back to Visual Studio and hit Ctrl+Shift+Alt+F5
6. When the output shows that Dynamo is ready Go to Dynamo and open a New or existing file
7. Use your node and you will jump to Visual Studio when a method or property is used that you marked with a break point. With F11 you can now step through your code.

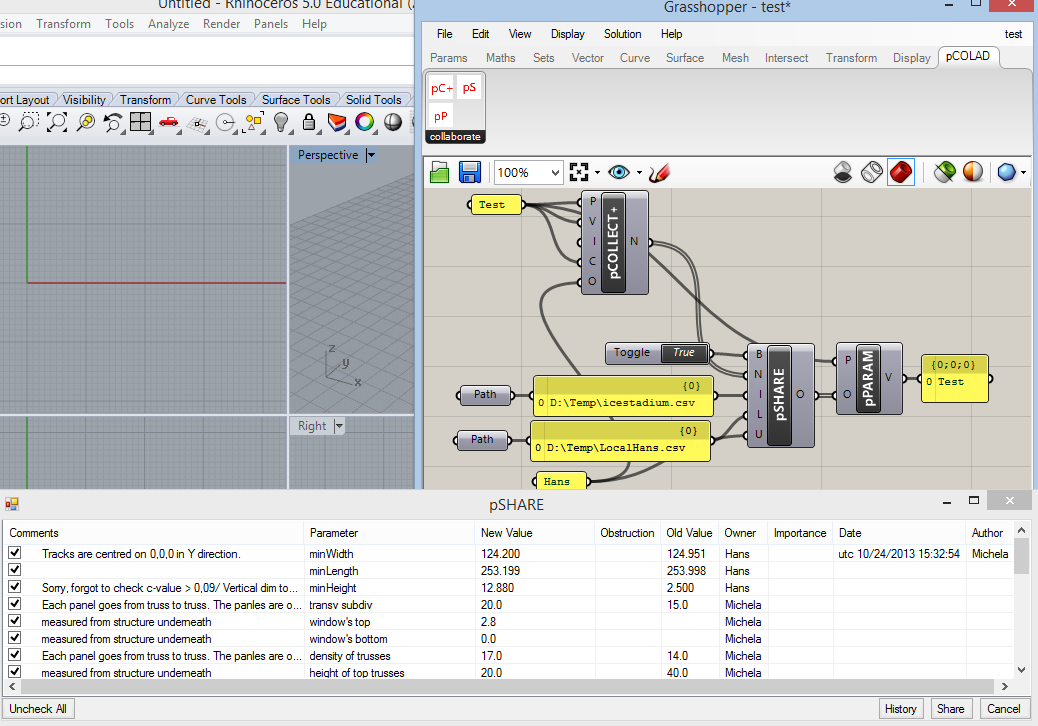


Figure 13 pCOLAD in Rhino/Grasshopper

Let’s start with pSHARE. Instead of showing a form, it should show a XAML display, but with same buttons and extensible grid or list view. Because we don’t want to have the form visible all the time, it is not a good idea to implement it as a node. So in fact with HelloDynamo we have a wrong example! For pSHARE we need a normal node. For pCOLLECT+ we need an adaptable node, but that is for later. It would be nice though to have an On/Off button instead of a Boolean toggle input. Maybe a green button with Off on it which turns red when you hit it with On on it. And the open circle image .

## A closer look to Model-View-ViewModel pattern

But I ran into a question how to make the image change on the button and assume that the delegatecommand has something to do with this. A closer look into the MVVM pattern is needed. In this case the View is pSHARE.xaml and the code behind it is pSHARE.xaml.cs. And as explained [here](https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx) only consists of the InitializeComponent();code.The Viewmodel is where the binding to the view takes place, so pSHARE.cs. The model is in fact outside pSHARE. It is referenced through different dlls. We might make a model for a parameter class and/or the loading of the csv file.

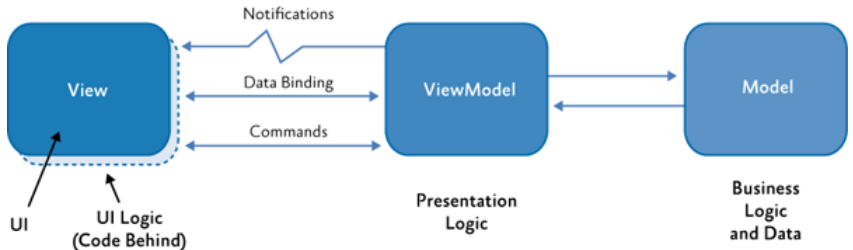


Figure 14 The MVVM pattern and interactions

First read this if you are not familiar with [WPF](http://www.wpftutorial.net) and then this about [MVVM](https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx). Later [this source](http://compositewpf.codeplex.com/) was very helpful. If you download the PRISM documentation, you find two well explained chapters about MVVM (the same as in the other link, but in a broader context and with examples). I think I finally got it:

The Model is the definition of the object classes. The ViewModel is the definition of the object instances. The View is the definition of the User Interface.

In order to display the \*csv file you might need what is written [here](http://www.wpftutorial.net/GridLayout.html) (it says : “The shared size feature of the grid layout allows it to synchronize the width of columns over multiple grids. The feature is very useful if you want to realize a multi-column listview by using a grid as layout panel within the data template. Because each item contains its own grid, the columns will not have the same width.”) But it works with fixed rows and colums. [Here](http://www.codeproject.com/Articles/35806/Model-View-ViewModel-in-WPF-Part-I) is a better approach. And [here](http://www.codeproject.com/Articles/35805/MVVM-in-WPF-Part-II) is part II. I got it [working](file:///D:\Data\Research\Csh%20lessons\articlezip\Article.sln). But it appears to rely on installing a database ole driver and I don't like users to first have to install that. So we'll have to use parsing of a csv file. But in combination with a MVVM pattern of course.

A simple example of MVVM can be found [here](http://www.codeproject.com/Articles/36545/WPF-MVVM-Model-View-View-Model-Simplified). Button example [here](http://www.c-sharpcorner.com/resources/629/wpf-button-with-image.aspx). And [here](https://msdn.microsoft.com/en-us/library/bb613598(v=vs.110).aspx) and [here](http://stackoverflow.com/questions/17657048/how-to-make-a-image-button-that-change-the-image-when-mouse-over-button-wpf).and [here](http://geekswithblogs.net/tburger/archive/2013/12/07/template-a-toggle-control-for-wpf-and-add-touch-support.aspx). Later I found [this](http://www.codeproject.com/Articles/165368/WPF-MVVM-Quick-Start-Tutorial) for beginners, which explains some things differently and e.g. doesn’t use delegates and dependency properties. It dates from 2012 and the example we will follow below from 2014, so maybe that’s why... The use of an ObservableCollection<> is something to remember though. From this information we derive the following. Through binding, the xaml form and the code work together. In our case this is done for the button: <Button Command="{Binding OnOff}"> OnOff is the property that is bound. In the pSHARE.cs file it is a delegate command: public DelegateCommand OnOff { get; set; }. A delegate command is an internal mechanism in WPF to update changes between the view and the model. A delegate is made in the class that you are referring to. It is a declaration of a method that is also defined there. E.g.

public delegate int PerformCalculation(int x, int y);

So when the OnOff button is hit this change in the Click property of the button is noticed and the DelegateCommand OnOff object is informed firing the ShowParams and CanShowParams procedures: OnOff = new DelegateCommand(ShowParams, CanShowParams); It takes two attributes. ShowParams is the action part and CanShowParams is the ability for action part. So in the ShowParams procedure we should put the action to change the image of the button and trigger all the actions we need to read and update the \*.csv file. But how can I address the button? The button is part of the usercontrol we defined in xaml, called pSHAREControl (find it in pSHARE.xaml.cs). Well you should not address the button here, let the View take care of itself??? [Here](http://www.codeproject.com/Articles/789957/WPF-Control-Patterns-WPF-and-XAML-Patterns-of-Code) we get some more insight in this problem. It is stated that the delegate object is bad practice… It explains that in order to use the delegate object you first have to define dependency properties (in fact that is explained somewhere else, but here we find an example) in the code behind a new control(WPF) that is made within a WPF project. So it is an example of making your own control, while also using existing standard WPF controls (in the HelloDynamo example we only use existing WPF controls). Let’s dive into this simple example and try to understand everything (my comments in green). First learn by hart the differences between value types and reference types [here](https://msdn.microsoft.com/en-us/library/ms173104(v=vs.120).aspx) (very important !).

## A simple WPF example

[Here](http://www.codeproject.com/Articles/789957/WPF-Control-Patterns-WPF-and-XAML-Patterns-of-Code) we find the example as mentioned. The example creates a C# WPF application named UserControlWithCodeBehind. You will have to set the name while you create the project, otherwise on several places you get wrong references to the class, causing e.g. the error “InitialiseComponent() is not part of this context” or something like that.

When you start a new WPF project you get an App.xaml, App.xaml.cs and a MainWindow.xaml and MainWindow.xaml.cs. Right Click the projectname in the explorer and choose Add…/New Item… choosing a Control(WPF) while you name it “EditableTextAndLabelControl”. In the EditableTextAndLabelControl.xaml we replace the Grid with a StackPanel containing a TextBlock named "TheLabelTextBlock", a TextBox named "TheEditableTextTextBox" and a Button named “SaveButton”. The text of the TextBlock and TextBox we will bind to dependency properties (object properties that you can set and get during run time) resp. TheLabel and TheEditableText. More about that below.

<UserControl x:Class="UserControlWithCodeBehind.EditableTextAndLabelControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:this="clr-namespace:UserControlWithCodeBehind"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<StackPanel Orientation="Horizontal">

<TextBlock x:Name="TheLabelTextBlock"

Text="{Binding Path=TheLabel, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom"/>

<TextBox x:Name="TheEditableTextTextBox"

Grid.Column="1"

Width="100"

Text="{Binding Path=TheEditableText, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom"

Margin="10,0,10,0"/>

<Button x:Name="SaveButton"

Content="Save"

Width="70"

Grid.Column="2"

VerticalAlignment="Bottom"/>

</StackPanel>

</UserControl>

You might wonder what is the difference between a TextBlock and a TextBox. "TextBoxes are used for displaying text more focused for content input or when content is needed to be made selectable by the user. The TextBox can only be set to one colour, one font size, one font type etc. TextBoxes also have fixed Line Spacing. The TextBox can also be set to a fixed height and width but also have scrollbars switched on to allow content to expand.(quote from Stack Overflow website)".

Now you will have to define the (dependency) properties and methods of the custom control EditableTextAndLabelControl. This is done in the EditableTextAndLabelControl.xaml.cs by defining a partial class that inherites from UserControl:

public partial class EditableTextAndLabelControl : UserControl

You need a mechanism that at run time binds the input of the user to xaml controls. Microsoft developed the Delegate type for that. Read [here](https://msdn.microsoft.com/en-us/library/ms173172.aspx) about the use of Delegates. Summary:

// Declare a delegate.

public delegate void Del(string message);

// Create a method for a delegate.

public static void DelegateMethod(string message)

{

System.Console.WriteLine(message);

}

// Instantiate the delegate.

Del handler = DelegateMethod;

// Call the delegate.

handler("Hello World");

A Delegate is a special type of which the instance can use any compatible method (compatible means same type and number of arguments and return). Delegates are therefor used to add or change methods of objects during run time. This makes it possible to keep View and Model separated pieces of code. [Here](https://msdn.microsoft.com/en-us/library/ms173171.aspx) it says: “Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs.” That is exactly what happens in the code below where it reads: SaveButton.Click += SaveButton\_Click; Clicking the SaveButton is attached through the += operator to the event handler SaveButton\_Click. This event handler is then coded as a method that returns a delegate: SaveEvent. The delegate is declared through a special object called Action of type Event. It is a short way for declaring a delegate. You could also do it like this:

public delegate void SaveEvent(string string1, string string2);

And then use this type to instantiate an object etc. etc.

Here you declare an event variable as an action (which by definition creates a delegate, in this case SaveEvent):

public event Action<string, string> SaveEvent = null; So SaveEvent is defined as delegate through this line of code in the (partial) class EditableTextAndLabelControl and is a type that passes the methods TheLabel and TheEditableText. Those in turn get and set the dependency properties (use propdp code snippet). To read more about dependency properties start [here](https://msdn.microsoft.com/en-us/library/ms752914(v=vs.110).aspx). Remember this “The purpose of dependency properties is to provide a way to compute the value of a property based on the value of other inputs.” E.g. through binding input through a xaml control. So it is like a normal property, only with extra possibilities. Why is it called dependency? Because its value *depends* on what goes on in other code. **So a Delegate is a type that passes methods through arguments and these arguments could be dependency properties that are depending on e.g. user input through a xaml control.**

Here is the code of EditableTextAndLabelControl.xaml.cs:

using System;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace UserControlWithCodeBehind

{

/// <summary>

/// Interaction logic for EditableTextAndLabelControl.xaml

/// </summary>

public partial class EditableTextAndLabelControl : UserControl

{

// defines the SaveEvent delegate

public event Action<string, string> SaveEvent = null;

// constructs the default properties and methods of the control

public EditableTextAndLabelControl()

{

InitializeComponent();

// attaches event handler to the button click

SaveButton.Click += SaveButton\_Click;

}

// fires SaveEvent of the control

void SaveButton\_Click(object sender, RoutedEventArgs e)

{

if (SaveEvent != null)

{

// the delegate SaveEvent passes the dependency properties as strings

SaveEvent(TheLabel, TheEditableText);

}

}

#region TheLabel Dependency Property

// defines the string property TheLabel using the dependency property

public string TheLabel

{

get { return (string)GetValue(TheLabelProperty); }

set { SetValue(TheLabelProperty, value); }

}

// registers the dependency property

public static readonly DependencyProperty TheLabelProperty =

DependencyProperty.Register

(

"TheLabel",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheLabel Dependency Property

#region TheEditableText Dependency Property

public string TheEditableText

{

get { return (string)GetValue(TheEditableTextProperty); }

set { SetValue(TheEditableTextProperty, value); }

}

public static readonly DependencyProperty TheEditableTextProperty =

DependencyProperty.Register

(

"TheEditableText",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheEditableText Dependency Property

}

}

In the MainWindow.xaml you define the layout of the whole interface. You put in your own control by using <this:nameOfYourControlTemplate x:name “nameOfYourControl” and on new lines the visual properties of your control. For this to work you will have to add xmlns:this="clr-namespace:theNameOfYourClass" . Notice also the special custom made dependency property called TheLabel, defined in the code of the class (see above).

The MainWindow.xaml:

<Window x:Class="UserControlWithCodeBehind.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:this="clr-namespace:UserControlWithCodeBehind"

Title="MainWindow"

Height="350"

Width="525">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<!--2 custom controls, of course you get blue wobly lines under TheLabel, because

you will define this depency property during run time-->

<this:EditableTextAndLabelControl x:Name="MyUserControl1"

HorizontalAlignment="Center"

VerticalAlignment="Center"

TheLabel="MyText"

Margin="0,10"/>

<this:EditableTextAndLabelControl x:Name="MyUserControl2"

Grid.Row="1"

HorizontalAlignment="Center"

VerticalAlignment="Center"

TheLabel="MyOtherText"

Margin="0,10"/>

<TextBlock x:Name="TheSaveEventLog"

Grid.Row="2"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"/>

</Grid>

</Window>

You could put the interaction logic in the MainWindow.xaml.cs using the names of the elements in MainWindow.xaml. That is convenient because you can simply use their names, but again, this is considered bad practice, because you couple the model and the view too tight. If you want to change the View, you will have to do a lot of work in the ViewModel too. Better would be if your controls are following a template that is defined in the new control’s cs-file. But to understand this better, let’s first continue with the tightly coupled example. The MainWindow.xaml.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace UserControlWithCodeBehind

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

//attach event handler to the SaveEvent delegate of the user controls

MyUserControl1.SaveEvent += MyUserControl\_SaveEvent;

MyUserControl2.SaveEvent += MyUserControl\_SaveEvent;

}

// add a method to display the input of the user in the user control in normal text block

void MyUserControl\_SaveEvent(string arg1, string arg2)

{

TheSaveEventLog.Text += "\nSaved string \"" + arg2 + "\" for label \"" + arg1 + "\"";

}

}

}

Now why this is bad practice? What is better? As mentioned already the interface is too tightly coupled to the control. E.g. if you want to change the layout of the second control so the button is below the text input box, you would have to create a two custom controls with almost the same code behind it. Not practical to maintain! A solution is to make a lookless control.

For this you start a new WPF project named “LooklessControlSample” and add a new item. This time not a control(WPF) but a simple C# class, named “EditableTextAndLabelControl”. You only put the action and dependency properties in it and add a method (Save()) that returns the dependcy properties through the delegate. So you inherit from WPF Control instead of the UserControl, leave out the constructor for the default control and the event handler:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace LooklessControlSample

{public class EditableTextAndLabelControl : Control

{

public event Action<string, string> SaveEvent = null;

public void Save()

{

if (SaveEvent != null)

SaveEvent(TheLabel, TheEditableText);

}

#region TheLabel Dependency Property

public string TheLabel

{

get { return (string)GetValue(TheLabelProperty); }

set { SetValue(TheLabelProperty, value); }

}

public static readonly DependencyProperty TheLabelProperty =

DependencyProperty.Register

(

"TheLabel",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheLabel Dependency Property

#region TheEditableText Dependency Property

public string TheEditableText

{

get { return (string)GetValue(TheEditableTextProperty); }

set { SetValue(TheEditableTextProperty, value); }

}

public static readonly DependencyProperty TheEditableTextProperty =

DependencyProperty.Register

(

"TheEditableText",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheEditableText Dependency Property

}

}

The code of MainWindow.xaml and MainWindow.xaml.cs is the same as above, except that we add a WPF resource dictionary named “SampleControls.xaml” in a folder named “Themes”. Because we are going to use a control template and style to define the visual representation of the control, we add a section Window.Resources with the ResourceDictionary:

<Window x:Class="LooklessControlSample.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:this="clr-namespace:LooklessControlSample"

Title="MainWindow"

Height="350"

Width="525">

<Window.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Themes/SampleControls.xaml" />

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Window.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<this:EditableTextAndLabelControl x:Name="MyControl1"

HorizontalAlignment="Center"

VerticalAlignment="Center"

TheLabel="MyText"

Margin="0,10" />

<this:EditableTextAndLabelControl x:Name="MyControl2"

Grid.Row="1"

HorizontalAlignment="Center"

VerticalAlignment="Center"

Style="{StaticResource TheVerticalStyleForEditableTextAndLabelControl}"

TheLabel="MyOtherText"

Margin="0,10" />

<TextBlock x:Name="TheSaveEventLog"

Grid.Row="2"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" />

</Grid>

</Window>

Then within the ResourceDictionary we can define control templates:

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"

xmlns:this="clr-namespace:LooklessControlSample">

<ControlTemplate x:Key="TheHorizontalTemplateForEditableTextAndLabelControl"

TargetType="this:EditableTextAndLabelControl">

<StackPanel Orientation="Horizontal">

<TextBlock x:Name="TheLabelTextBlock"

Text="{Binding Path=TheLabel, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom" />

<TextBox x:Name="TheEditableTextTextBox"

Grid.Column="1"

Width="100"

Text="{Binding Path=TheEditableText, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom"

Margin="10,0,10,0" />

<Button x:Name="SaveButton"

Content="Save"

Width="70"

Grid.Column="2"

VerticalAlignment="Bottom">

<i:Interaction.Triggers>

<i:EventTrigger EventName="Click"

SourceObject="{Binding ElementName=SaveButton}">

<ei:CallMethodAction MethodName="Save"

TargetObject="{Binding RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}" />

</i:EventTrigger>

</i:Interaction.Triggers>

</Button>

</StackPanel>

</ControlTemplate>

<ControlTemplate x:Key="TheVerticalTemplateForEditableTextAndLabelControl"

TargetType="this:EditableTextAndLabelControl">

<StackPanel Orientation="Vertical">

<TextBlock x:Name="TheLabelTextBlock"

Text="{Binding Path=TheLabel, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom" />

<TextBox x:Name="TheEditableTextTextBox"

Grid.Column="1"

Width="100"

Text="{Binding Path=TheEditableText, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom"

Margin="10,0,10,0" />

<Button x:Name="SaveButton"

Content="Save"

Width="70"

Grid.Column="2"

VerticalAlignment="Bottom">

<i:Interaction.Triggers>

<i:EventTrigger EventName="Click"

SourceObject="{Binding ElementName=SaveButton}">

<ei:CallMethodAction MethodName="Save"

TargetObject="{Binding RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}" />

</i:EventTrigger>

</i:Interaction.Triggers>

</Button>

</StackPanel>

</ControlTemplate>

<!-- Horizontal Style for EditableTextAndLabelControl (this is a default style for the

control - a style without a resource key) -->

<Style TargetType="this:EditableTextAndLabelControl">

<Setter Property="Template"

Value="{StaticResource TheHorizontalTemplateForEditableTextAndLabelControl}" />

</Style>

<!-- Vertical Style for EditableTextAndLabelControl -->

<Style TargetType="this:EditableTextAndLabelControl"

x:Key="TheVerticalStyleForEditableTextAndLabelControl">

<Setter Property="Template"

Value="{StaticResource TheVerticalTemplateForEditableTextAndLabelControl}" />

</Style>

</ResourceDictionary>

Notice that we added a vertical control template and that we had to add references to Expression Blend SDK dlls: xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity" xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"

For this you need to [download](http://www.microsoft.com/en-us/download/details.aspx?id=3062) and reference the Blend SDK files: Microsoft.Expression.Interactions.dll

System.Windows.Interactivity.dll

Of course you will have to add those dlls when you give your application to somebody else, which is not so handy. This is one of the reasons why it is interesting to see that there is a way to access parts of the template from C# code of a lookless control.

The ControlTemplate defined in SampleControls.xaml file becomes only simpler. It does not need references to the Expression Blend SDK and its button does not have any triggers:

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:this="clr-namespace:FindingTemplatePartInCodeSample">

<ControlTemplate x:Key="TheTemplateForEditableTextAndLabelControl"

TargetType="this:EditableTextAndLabelControl">

<StackPanel Orientation="{Binding Path=TheOrientation, RelativeSource={RelativeSource TemplatedParent}}">

<TextBlock x:Name="TheLabelTextBlock"

Text="{Binding Path=TheLabel, RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom" />

<TextBox x:Name="TheEditableTextTextBox"

Grid.Column="1"

Width="100"

Text="{Binding Path=TheEditableText,

RelativeSource={RelativeSource AncestorType=this:EditableTextAndLabelControl}}"

VerticalAlignment="Bottom"

Margin="10,0,10,0" />

<Button x:Name="PART\_SaveButton"

Content="Save"

Width="70"

Grid.Column="2"

VerticalAlignment="Bottom" />

</StackPanel>

</ControlTemplate>

<!-- Style for EditableTextAndLabelControl (this is a default style for the

control - a style without a resource key) -->

<Style TargetType="this:EditableTextAndLabelControl">

<Setter Property="Template"

Value="{StaticResource TheTemplateForEditableTextAndLabelControl}" />

</Style>

</ResourceDictionary>

Note that the name of the button was changed to have prefix "PART\_". This is a general convention for parts of the control that C# code expects to find.

The EditableTextAndLabelControl.cs file becomes:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace FindingTemplatePartInCodeSample

{

public class EditableTextAndLabelControl : Control

{

public event Action<string, string> SaveEvent = null;

Button \_saveButton = null;

public override void OnApplyTemplate()

{

base.OnApplyTemplate();

if (this.Template == null)

return;

\_saveButton = this.Template.FindName("PART\_SaveButton", this) as Button;

if (\_saveButton != null)

\_saveButton.Click += \_saveButton\_Click;

}

void \_saveButton\_Click(object sender, RoutedEventArgs e)

{

if (SaveEvent != null)

SaveEvent(TheLabel, TheEditableText);

}

#region TheLabel Dependency Property

public string TheLabel

{

get { return (string)GetValue(TheLabelProperty); }

set { SetValue(TheLabelProperty, value); }

}

public static readonly DependencyProperty TheLabelProperty =

DependencyProperty.Register

(

"TheLabel",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheLabel Dependency Property

#region TheEditableText Dependency Property

public string TheEditableText

{

get { return (string)GetValue(TheEditableTextProperty); }

set { SetValue(TheEditableTextProperty, value); }

}

public static readonly DependencyProperty TheEditableTextProperty =

DependencyProperty.Register

(

"TheEditableText",

typeof(string),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(null)

);

#endregion TheEditableText Dependency Property

#region TheOrientation Dependency Property

public Orientation TheOrientation

{

get { return (Orientation)GetValue(TheOrientationProperty); }

set { SetValue(TheOrientationProperty, value); }

}

public static readonly DependencyProperty TheOrientationProperty =

DependencyProperty.Register

(

"TheOrientation",

typeof(Orientation),

typeof(EditableTextAndLabelControl),

new PropertyMetadata(Orientation.Horizontal)

);

#endregion TheOrientation Dependency Property

}

}

OnApplyTemplate() method pulls the button control out of the template by its name "PART\_SaveButton". It also adds \_saveButton\_Click handler to the Button's Click event. All very interesting and very complicated. But it is explained that in fact this is against the paradigm of Skeleton not knowing about the Meat (Model seperated from the View). So what we were looking for, how to change the image of the button, should be done in xaml. That much became clear. [The part 2](http://www.codeproject.com/Articles/801363/View-View-Model-based-WPF-and-XAML-Implementationa) seems more interesting for loading the csv file. E.g. "Every FrameworkElement object has DataContext dependency property. Unless overridden, this property is inherited from the object's parents within the logical tree. DataContext object is the default source object for the WPF Binding in a sense that if neither RelativeSource, nor Source, nor ElementName is specified for a binding, the binding assumes that the its Source object is provided by the DataContext of the binding's target. This was done on purpose by the Microsoft people who implemented WPF, in order to make it easier for a visual object to bind its properties to a non-visual one provided by its DataContext property." This explains why in some examples you only find {Binding} without a path etc.

Searching for a WPF example with changing image button [here](http://geekswithblogs.net/tburger/archive/2013/12/07/template-a-toggle-control-for-wpf-and-add-touch-support.aspx) is one using Expression Blend. Addapting the principle to an On/Off button we get next MainWindow.xaml code:

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions" x:Class="TestButton.MainWindow"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<ControlTemplate x:Key="ToggleButtonControlTemplate1" TargetType="{x:Type ToggleButton}">

<Grid>

<VisualStateManager.CustomVisualStateManager>

<ei:ExtendedVisualStateManager/>

</VisualStateManager.CustomVisualStateManager>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CheckStates">

<VisualState x:Name="Checked">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OffText">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OnImage">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OffImage">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unchecked">

<Storyboard>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OffImage">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OffText">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Visible}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OnText">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="OnImage">

<DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Indeterminate"/>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Image x:Name="OffImage" Margin="0,0,474,276" Source="OnOff\_Images/Layer 5.png" Stretch="Fill"/>

<TextBlock x:Name="OffText" HorizontalAlignment="Left" Margin="10,48,0,0" TextWrapping="Wrap" VerticalAlignment="Top"><Run Language="nl-nl" Text="Off"/></TextBlock>

<TextBlock x:Name="OnText" HorizontalAlignment="Left" Margin="10.723,48,0,0" TextWrapping="Wrap" VerticalAlignment="Top" RenderTransformOrigin="-4.546,-3.759"><Run Language="nl-nl" Text="On"/></TextBlock>

<Image x:Name="OnImage" Margin="0,0,474,276" Source="OnOff\_Images/Layer 4.png" Stretch="Fill"/>

</Grid>

</ControlTemplate>

</Window.Resources>

<Grid>

<ToggleButton Content="ToggleButton" HorizontalAlignment="Left" VerticalAlignment="Top" Template="{DynamicResource ToggleButtonControlTemplate1}"/>

</Grid>

</Window>

## Simple solution for image changing button

That is rather complex, but maybe we can simplify by leaving out the animations. [Here](http://stackoverflow.com/questions/1532573/customizing-the-toggle-state-of-a-toggle-button-in-wpf) we find a simpler solution:

<ToggleButton

VerticalAlignment="Center" HorizontalAlignment="Center"

IsChecked="{Binding Status}">

<ToggleButton.Resources>

<Image x:Key="OnImage" Source="OnOff\_Images/Layer 4.png" />

<Image x:Key="OffImage" Source="OnOff\_Images/Layer 5.png" />

</ToggleButton.Resources>

<ToggleButton.Style>

<Style TargetType="ToggleButton">

<Style.Triggers>

<Trigger Property="IsChecked" Value="True">

<Setter Property="Content" Value="{StaticResource OnImage}">

</Setter>

</Trigger>

<Trigger Property="IsChecked" Value="False">

<Setter Property="Content" Value="{StaticResource OffImage}">

</Setter>

</Trigger>

</Style.Triggers>

</Style>

</ToggleButton.Style>

</ToggleButton>

The MainWindow.xaml.cs file is very simple:

sing System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TestButton

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

}

So the code of pCOLAD pSHARE.xaml should be:

<UserControl x:Class= "pCOLADnamespace.pSHAREControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<Grid x:Name="CustomUIGrid">

<StackPanel>

<ToggleButton x:Name="myButton"

IsChecked="{Binding Status}"

Command="{Binding OnOff}">

<ToggleButton.Resources>

<Image x:Key="OnImage" Source="OnOff\_Images/On.png" Stretch="Fill"/>

<Image x:Key="OffImage" Source="OnOff\_Images/Off.png" Stretch="Fill"/>

</ToggleButton.Resources>

<ToggleButton.Effect>

<DropShadowEffect BlurRadius="3" ShadowDepth="1"/>

</ToggleButton.Effect>

<ToggleButton.Style>

<Style TargetType="ToggleButton">

<Style.Triggers>

<Trigger Property="IsChecked" Value="True">

<Setter Property="Content" Value="{StaticResource OnImage}">

</Setter>

</Trigger>

<Trigger Property="IsChecked" Value="False">

<Setter Property="Content" Value="{StaticResource OffImage}">

</Setter>

</Trigger>

</Style.Triggers>

</Style>

</ToggleButton.Style>

</ToggleButton>

<Label HorizontalAlignment="Center" VerticalAlignment="Top">

<Label.Style>

<Style TargetType="{x:Type Label}">

<Style.Triggers>

<DataTrigger Binding="{Binding IsChecked,ElementName = myButton}" Value="True">

<Setter Property="Content" Value="On"/>

</DataTrigger>

<DataTrigger Binding="{Binding IsChecked,ElementName = myButton}" Value="False">

<Setter Property="Content" Value="Off"/>

</DataTrigger>

</Style.Triggers>

</Style>

</Label.Style>

</Label>

</StackPanel>

</Grid>

</UserControl>

However the control should not trigger the loading of the \*.csv file but the state of the button should allow or not to load that file when Dynamo runs (F5). So we should react on the Run button. In the Dynamo project we find that a RunEnabled propertychange is used and a DynamicRunEnabled one (must be AutoRun). Tried several things: in DynamoViewModel.cs a propertyChanged event is raised called “RunEnabled” which is bound to the Run button in DynamoView.xaml. But I don’t know how to catch the event in an instance of DynamoSandbox. Next code doesn’t work…

private void listner(object sender, System.ComponentModel.PropertyChangedEventArgs e)

{

if (e.PropertyName == "RunEnabled")

{

MessageBox.Show("Got ya");

}

However when you hit Run the first time it reacts and also if something changed, so maybe you can use that to set the solution to dirty, so it reacts the next run.

Or maybe find the solution [here](http://www.codeproject.com/Articles/27559/Tracing-Events-Raised-by-Any-C-Object)???Nope. Keep it for later. Posted a question on the [GitHub issues section of DynamoDS](https://github.com/DynamoDS/Dynamo/issues/).

Now first try to load a csv file in a xaml datagrid the MVVM way. [Here](http://stackoverflow.com/questions/8148772/databinding-a-wpf-grid-control-to-a-csv-file) it says: DataGrid is based on the [ItemsControl](http://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol.aspx) control, and items are represented by [ItemsControl.ItemsSource](http://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol.itemssource.aspx) which type is IEnumerable. So if you simply want to display CSV in a DataGrid you should read CSV in any data structure which implements IEnumerable and set it to the DataGrid.ItemsSource property through the code behind on more preferrable way -binding from the XAML. So your grid would have a single column which represent an entire CSV file line. If you want to show each value in a separate column - each item of IEnumerable data structure (I would suggest IList<>) should represent IEnumerable as well, basically IList<IEnumerable<object>>.

I already thought using a List so that seams a good idea. But above we saw that an observablecollection, notifies the XAML when something changes. If I understand it correctly IList is an implementation of observablecollection type so let's try. It further says: To read CSV in the local data structure I would suggest using some free CSV parsing library rather than doing this yourself since parsing of the CSV sometimes tricky thing. Take a look at the [KBCsv library](http://kbcsv.codeplex.com/) as a case.

But the KBCsv project moved to GitHub, [here](https://github.com/kentcb/KBCsv/blob/master/Doc/quick-start-guide.md). Would be nice to use because it gives easy to read code. But again would imply to ship the dlls. So no thanks. Just read the file and try to put it in a IList<IEnumerable<Object>>. But it turned out to be very complicated. Looking for a way to embed a dll in my own dll I found [this](https://www.youtube.com/watch?v=x-KK7bmo1AM). So it is possible!

This is the code:

AppDomain.CurrentDomain.AssemblyResolve += new ResolveEventHandler (CurrentDomain\_AssemblyResolve);

And the handler outside the area where the error occurs

Assembly CurrentDomain\_AssemblyResolve(object sender, ResolveEventArgs args)

{

using (var stream = Assembly.GetExecutingAssembly().GetManifestResourceStream("EmbedAssembly.Kent.Boogaart.KBCsv.dll"))

{

byte[] assemblyData = new byte[stream.Length];

stream.Read(assemblyData, 0, assemblyData.Length);

return Assembly.Load(assemblyData);

}

}

However it only works then for 1 resource. Looking for a solution with several resources it appears to make a difference if code is managed or not. [Here](https://social.msdn.microsoft.com/Forums/vstudio/en-US/a3e28547-4791-4394-b450-29c82cd70f70/managed-code-vs-unmanaged-code?forum=csharpgeneral) you can find what that means: managed code is platform independent e.g. C#,VB.Net, Java and targets a managed run time environement (like Common Language Runtime (CLR) in the .NET framework). Unmanaged code targets the operating system in a specific architecture e.g. C++, C and VB 6.0. Probably we only have to do with managed code. In that case we could try  [ILMerge](http://research.microsoft.com/~mbarnett/ILMerge.aspx), which is often recommended. However the management of the code, the reusability by others get's more complex… I prefer the simple method I used in pCOLAD for Grasshopper:

Dim csvSubItems As String() = lines(0).Split(";"c)

For Each line As String In lines

If x = 0 Then 'the first line should contain the header titles

Dim hd As String() = (line.Split(";"c))

For Each hdn In hd

m\_form.ListView1.Columns.Add(hdn, 100, HorizontalAlignment.Left).Name = hdn

If hdn = "Obstruction" Then R = True

Next

If R = False Then

MessageBox.Show("There is no column named 'Obstruction'. Please make sure that this word is in the first line of the \*.csv file.")

'm\_formClose() if you close the form in this way you get stuck with object not set to an instance etc.

m\_form.Hide()

'Exit Sub 'is this preventing to start over?

'Return

''ExpireSolution(True)

formPopulated = False

Return

End If

Else

Dim item As New ListViewItem(line.Split(";"c))

Dim userInObstruction As Boolean = False

item.UseItemStyleForSubItems = False

'if more new items then old items oldLines(x) will throw an out of bounds error

'oldLine will be split into items and ASSUMED to have equal number of subItems -

'WHICH is not the case if input file format changed...!!!To be checked.

If x > oldLines.Count Then x = oldLines.Count

If oldLines.Count = 0 Then oldLine = "" Else oldLine = oldLines(x)

Dim oldItem As New ListViewItem(oldLine.Split(";"c))

m\_form.ListView1.Items.Add(item)

'MsgBox("Just added an item...")

For counter = 0 To csvSubItems.Count - 1

'but if you add an attribute and in the csv file this field is empty you get an invalid argument error

'if this item is new then set the color to red

'if you start with an emptly local file, oldfile is empty, oldLine is empty so oldItem has no SubItems

If item.SubItems.Count - 1 >= counter Then

item.SubItems(counter).Name = csvSubItems(counter)

'MsgBox("Invalidargument="&counter&"?itemSubItems.Count="&item.SubItems.Count&".csvSubItems.Count="&csvSubItems.Count)

If item.SubItems(counter).Text <> oldItem.SubItems(counter).Text Then subItemNotEqual = True

If oldLines.Count = 0 Then subItemNotEqual = False

If item.SubItems(counter).Name = "Date" Then subItemNotEqual = False

If item.SubItems(counter).Name = "Author" Then subItemNotEqual = False

If item.SubItems(counter).Text = "" Then subItemNotEqual = False

'MsgBox(String.Format("item.SubItems({0}).Text = " & item.SubItems(counter).Text & \_

' ". While oldItem.SubItems({0}).Text = " & oldItem.SubItems(counter).Text, counter))

If subItemNotEqual Then

'MsgBox(String.Format("item.SubItems({0}).Name = ", counter) & item.SubItems(counter).Name)

item.SubItems(counter).ForeColor = Drawing.Color.Red

subItemNotEqual = False

newData = newData + 1 'only one message for several red subitems

If newData = 1 Then MsgBox("The input file has changed since you last opened it. Changes are in red.")

End If

'MsgBox("Added subitem: " & item.SubItems(counter).Name)

If item.SubItems(counter).Name = "Obstruction" And item.SubItems(counter).Text.Contains(userName) Then userInObstruction = True

End If

Next

If userInObstruction = False Then

'MsgBox("item no. " & item.Index & " will be checked...It is named: " & item.Name)

item.Checked = True

'MsgBox("Checking of item no. " & item.Index & " passed.")

End If

End If

'MsgBox("x = " & x)

x = x + 1

Next

End If

Catch Ex As Exception

formPopulated = False

'when there is a new line - so difference between ppp and old\_qqq -> out of bounds exception

MessageBox.Show("2.We have a problem. Original error: " & Ex.Message & " Please start again...")

m\_formClose()

Finally

' Check this again, since we need to make sure we didn't throw an exception on open.

If (myStream IsNot Nothing) Then

myStream.Close()

End If

If (myOldStream IsNot Nothing) Then

myOldStream.Close()

End If

End Try

MyColumnResize(m\_form.ListView1)

One thing to consider is that in Grasshopper you can connect several outputs to one input. Therefore you can build a common multiple from all pSHARE outputs and the \*.csv file. In Dynamo we have to solve this with an intermediate List. Trying this shows that a semicolon (;) within a tekst in a code block automaticaly inserts a Return character after the semicolon. This means that a we can not use ; as seperator in the \*.csv file. But this is problematic, because if we want to exchange a point as x,y,z we get wrong results! Colin from Autodesk answered in forum that they will address this in next version. Using a String node and Concat node builds a correct line for the csv file though. So no worries…

Let's first also see if we can format and fill a DataTable, which we then could databind to a xaml gridview. Databinding is a way to connect a xaml view to a C# viewmodel. E.g. has a DataTable a method that adds a column? [Here](https://msdn.microsoft.com/en-us/library/system.data.datatable(v=vs.110).aspx) it says: If you are creating a DataTable programmatically, you must first define its schema by adding [DataColumn](https://msdn.microsoft.com/en-us/library/system.data.datacolumn(v=vs.110).aspx) objects to the [DataColumnCollection](https://msdn.microsoft.com/en-us/library/system.data.datacolumncollection(v=vs.110).aspx) (accessed through the [Columns](https://msdn.microsoft.com/en-us/library/system.data.datatable.columns(v=vs.110).aspx) property). For more information about adding [DataColumn](https://msdn.microsoft.com/en-us/library/system.data.datacolumn(v=vs.110).aspx) objects, see [Adding Columns to a DataTable](https://msdn.microsoft.com/en-us/library/hfx3s9wd(v=vs.110).aspx). But that is not necessary at all, you can just set a variable as a table and add columns and rows. No idea why people make things more complicated than necessary…

To add rows to a DataTable, you must first use the [NewRow](https://msdn.microsoft.com/en-us/library/system.data.datatable.newrow(v=vs.110).aspx) method to return a new [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) object. The [NewRow](https://msdn.microsoft.com/en-us/library/system.data.datatable.newrow(v=vs.110).aspx) method returns a row with the schema of the DataTable, as it is defined by the table's [DataColumnCollection](https://msdn.microsoft.com/en-us/library/system.data.datacolumncollection(v=vs.110).aspx). The maximum number of rows that a DataTable can store is 16,777,216. For more information, see [Adding Data to a DataTable](https://msdn.microsoft.com/en-us/library/z16c79x4(v=vs.110).aspx).

The DataTable also contains a collection of [Constraint](https://msdn.microsoft.com/en-us/library/system.data.constraint(v=vs.110).aspx) objects that can be used to ensure the integrity of the data. For more information, see [DataTable Constraints](https://msdn.microsoft.com/en-us/library/st1t2c35(v=vs.110).aspx). And there is also an example in C#.

Ater having the datatable build with the csv file, we have to bind it to a xaml datagridview. This took me days… The xaml even didn't show. I forgot that you need a mechanism to bind to it, like a delegate and dependency properties. Or a DelegateCommand (Dynamo uses that) as explained [here](https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx).

Looking for examples of showing a dll in a Wpf application, [here](http://www.codeproject.com/Articles/32825/How-to-Creating-a-WPF-User-Control-using-it-in-a-W) is one that simply adds a reference to the dll and then adds xlmns: [a name]= "clr-namespace:[the namespace of the dll];assembly=[the assembly name of the dll]". And then within the window of the host application <[a name]:[name of user control constructor]>. But in our case we can not access the Dynamo main window. Or can we? The custom node examples uses view.inputGrid.Children.Add(pSHAREControl); could we similarly add a control to the canvas – or what is it called in Dynamo? Yes, in the WorkspaceView.xaml we find that "Canvas" is the name of the control. But after some trial and error and google searches it turned out that simply changing UserControl in the MyControl.xaml into Window and putting Window test = new Window(); test.Show(); does the trick. Now still remains the binding to the datatable…

[Interesting search result](https://www.google.nl/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=How+to+create+a+separate+control+in+dll+for+WPF+host+application+with+C%23) for if you want to mix Windows Form and WPF application.

After a weekend break I realized that what I need is a *fundamental* understanding of how to set up a MVVM patern and how to bind a datagrid to a csv file. I vaguely remember that for this binding you will have to implement INotifyCollectionChanged interface or ObservableCollection<T>. Does the HelloDynamo sample and thus pCOLAD already implement the INotifyCollectionChanged interface? By searching for the definitions of the implementations of the HelloDynamo class: NodeModel and IWpfNode we find that the NodeModel implements: : ModelBase, IBlockingModel. The ModelBase implements: NotificationObject which implements: INotifyPropertyChanged which is a MS Interface. But it is not ment for collections. You need the INotifyCollectionChanged interface or ObservableCollection<T> which derives from System.Collections.Specialized while INotifyPropertyChanged derives from System.ComponentModel. [Here](https://msdn.microsoft.com/en-us/library/vstudio/system.collections.specialized.inotifycollectionchanged(v=vs.100).aspx) it says: "You can enumerate over any collection that implements the IEnumerable interface." And further: WPF provides the [ObservableCollection<T>](https://msdn.microsoft.com/en-us/library/vstudio/ms668604(v=vs.100).aspx) class, which is a built-in implementation of a data collection that exposes the INotifyCollectionChanged interface. For an example, see [How to: Create and Bind to an ObservableCollection](https://msdn.microsoft.com/en-us/library/vstudio/ms748365(v=vs.100).aspx). So instead of loading the csv file in a datatable we should load it in an ObservableCollection. Let's try first with a simple Wpf application.

You need to add using System.Collections.ObjectModel; and you have to put a type between <>: class CSVloader : ObservableCollection<string>. But [here](https://social.msdn.microsoft.com/Forums/vstudio/en-US/3cfcd9ae-384d-4a58-a190-63762d5dbc6d/load-a-csv-file-data-into-collection-in-wpf?forum=wpf) it shows that it is better to first build your list and then convert it to an ObservableCollection (OC). Also it turns out that you need properties of objects to fill the OC. So turn the rows of the datatable into objects. [Here](http://stackoverflow.com/questions/27489000/convert-datatable-to-observablecollection-without-specific-class) we find a generic solution for converting a datatable into an ObservableCollection. Also I had to go again to The World's Simplest C# WPF MVVM Example [here](http://www.markwithall.com/programming/2013/03/01/worlds-simplest-csharp-wpf-mvvm-example.html). Remember that an ObservableCollection<T> is a collection of objects of type T; it is not a table with columns etc. So the objects could be of a custom type where the properties will become the names of the columns and the value the content of the fields. Then you can bind the XAML DataGrid control to the collection with AutoGenerateColumns="True". But how do you use a csv to generate properties of a class?

The answer is somewhere between [this](http://stackoverflow.com/questions/16759603/datatable-to-observable-collection) and [this](http://stackoverflow.com/questions/27489000/convert-datatable-to-observablecollection-without-specific-class). Also [interesting](https://blog.udemy.com/wpf-datagrid/).

But more interesting is the example [here](https://social.msdn.microsoft.com/Forums/vstudio/en-US/f4cd1a62-4178-466f-ae49-b94f9a6c4697/not-able-to-display-data-loaded-from-csv-file-datagrid-in-wpf?forum=wpf). Though it builds an OLE based methode called GetDataTableFromCSV(string strFileName), which uses settings that might not correspond to a users settings it simply states: dataGrid.ItemsSource = table.DefaultView to display the table. So there should be no need to convert to an ObservableCollection… Indeed there isn't. MAJOR BREAKTHROUGH achieved with next setup.

## The most simple MVVM CSV LOADER

Next setup works. Only once debug ran whenever you switch to the MainWindow.xaml an extra myXaml window is created…

1. Start a new C# WPF application. Call it MyCSVLoader2
2. Add a C# class. Call it CSVloader.cs
3. Add a WPF window. Call it myXaml
4. In the MainWindow.xaml :

<Window x:Class="MyCSVLoader2.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:ViewModel ="clr-namespace:MyCSVLoader2"

Title="MainWindow" Height="350" Width="525">

<Window.DataContext>

<ViewModel:CSVloader/>

</Window.DataContext>

</Window>

1. In the myXaml.xaml:

<Window x:Class="MyCSVLoader2.myXaml"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="myXaml">

<Grid>

<DataGrid Name="myXamlTable"/>

</Grid>

</Window>

1. In CSVloader.cs:

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace MyCSVLoader2

{

class CSVloader

{

string inputFile = "D:\\Temp\\test.csv";

DataTable myDataTable = new DataTable();

public DataTable myPropDataTable { get; set; }

//constructor

public CSVloader()

{

this.myPropDataTable = openCSV();

myXaml \_MyControl = new myXaml();

\_MyControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;

\_MyControl.Show();

}

public DataTable openCSV()

{

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

//Console.WriteLine(line);

if (i == 0) //this contains the headers.

{

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, System.Type.GetType("System.String"));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 0;

foreach (var word in words)

{

row[x] = word;

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

return myDataTable;

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

return myDataTable;

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

return myDataTable;

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set to a etc.

return myDataTable;

}

}

}

}

## Basic script for pSHARE for Dynamo

Translating The most simple MVVM CSV LOADER into pSHARE for Dynamo gives next basic script that works alright.

using System.Collections.Generic;

using System.Windows;

using Autodesk.DesignScript.Runtime;

using Dynamo.Controls;

using Dynamo.Models;

using Dynamo.UI;

using Dynamo.UI.Commands;

using ProtoCore.AST.AssociativeAST;

using System;

//using System.Diagnostics;

//using System.ComponentModel;

//using System.Diagnostics.CodeAnalysis;

//using System.Reflection;

//using System.Collections;

using System.IO;

using System.Data;

//using System.Security.Permissions;

namespace pCOLADnamespace

{

#region some node settings

/// pSHARE takes care of the communication of parameterchanges through a shared \*.csv file.

[NodeName("pSHARE")]

[NodeCategory("pCOLAD")]

[NodeDescription("Load and share changes to parameters.")]

[IsDesignScriptCompatible]

#endregion

public class pSHARE : NodeModel, IWpfNode

{

bool On = false;

// later replace with an input

string inputFile = "D:\\Temp\\test.csv";

DataTable myDataTable;

private string \_OnOffButton;

#region properties

public string OnOffButton

{

get { return \_OnOffButton; }

set

{

\_OnOffButton = value;

//Raise a property changed notification to alert the UI that an element needs an update

RaisePropertyChanged("NodeMessage");

}

}

/// DelegateCommand objects allow you to bind UI interaction to methods on your data context.

[IsVisibleInDynamoLibrary(false)]

public DataTable myPropDataTable { get; set; }

public DelegateCommand OnOff { get; set; }

[IsVisibleInDynamoLibrary(false)]

#endregion

#region constructor

/// The constructor for a NodeModel is used to create the input and output ports and specify the argument lacing.

public pSHARE(WorkspaceModel workspace)

: base(workspace)

{

InPortData.Add(new PortData("something", "Input a string."));

OutPortData.Add(new PortData("something", "A result."));

RegisterAllPorts();

ArgumentLacing = LacingStrategy.CrossProduct;

OnOff = new DelegateCommand(ShowParams, CanShowParams);

// update UI

OnOffButton = "Share";

}

#endregion

#region public methods

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

return new[] { AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes)), };

}

public void openCSV()

{

//make sure that the table doesn't exist. Otherwise just show it.

if (myDataTable == null)

{

myDataTable = new DataTable();

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

if (i == 0) //this contains the headers. Also use to create properties for Parameter class.

{

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, System.Type.GetType("System.String"));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 0;

foreach (var word in words)

{

row[x] = word;

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

//return myDataTable;

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

//return myDataTable;

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

//return myDataTable;

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set to a etc.

//return myDataTable;

}

//make sure that control doesn't exist.

}

try

{

//check if the control exist already

bool isMyControlOpen = false;

foreach (Window w in Application.Current.Windows)

{

if (w is MyControl)

{

isMyControlOpen = true;

w.Activate();

}

}

if (!isMyControlOpen)

{

this.myPropDataTable = myDataTable;

MyControl \_MyControl = new MyControl();

\_MyControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;

\_MyControl.Show();

}

}

catch (System.Exception e)

{

MessageBox.Show("Exception source: {0}", e.Source);

}

}

[IsVisibleInDynamoLibrary(false)]

//probably Dynamo has a method that makes it go here asa pSHARE is loaded

public void SetupCustomUIElements(dynNodeView view)

{

var pSHAREControl = new pSHAREControl();

view.inputGrid.Children.Add(pSHAREControl);

pSHAREControl.DataContext = this;

}

public void recalc(DynamoModel actual)

{

//if pSHARE is ON the solution should be recalculated. Doesn't acutally work.

actual.ResetEngine(true);

}

#endregion

#region command methods

private bool CanShowParams(object obj)

{

return true;

}

private void ShowParams(object obj)

{

//switch the On boolean to show or not the \*.csv file

if (On == false)

{

On = true;

//and show the \*.csv file

openCSV();

}

else

{

On = false;

//close \*.csv display

closeMyControl();

}

}

#endregion

public void closeMyControl()

{

foreach (Window w in Application.Current.Windows)

{

if (w is MyControl)

{

w.Close();

}

}

}

}

}

Next we should try to set the columnwidth of MyControl. That is simply done in the xaml:

<Window x:Class="pCOLADnamespace.MyControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<StackPanel >

<DataGrid Name="myXamlTable" ColumnWidth="100"/>

</StackPanel>

</Window>

Now lets try to add a checkbox column. Also simple: myDataTable.Columns.Add("Accepted", typeof(bool)); Because it is a bool type, the conversion automatically makes it a checkbox column in datagrid.

Alternatively you could add next statements to your xaml:

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<CheckBox IsChecked="{Binding Path=IsSelected, UpdateSourceTrigger=PropertyChanged}" />

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

# Add a checkbox for obstruction of parameter value

Now try to get the name in the Obstruction field when you unselect a checkbox.

[Here](http://blog.galasoft.ch/posts/2010/05/handling-datagrid-selecteditems-in-an-mvvm-friendly-manner/) we find an example. However be aware that you have to add a xlm namespace and a reference resp:

xmlns:i=<http://schemas.microsoft.com/expression/2010/interactivity>

and

System.Windows.Interactivity

Also you have to replace the cmd: EventCommand by i:InvokeCommandAction. More necessary info can be found [here](http://blog.pluralsight.com/how-to-use-prism-5%E2%80%99s-invokecommandaction-behavior). Be aware that you have to install Prism through Tools/NuGet Package Manager/ Mange NuGet Packages for Solution… and xmlns:prism=<http://www.codeplex.com/prism> to your xaml control.

But using two windows, like we have to in Dynamo, doesn't work. Turns out that you have to show the custom window after constructing the behaviours. Be aware that The [SelectionChanged](https://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.selector.selectionchanged.aspx) event is not raised if the new or old selection contains no rows but only cells. So it says [here](https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid.selectedcells.aspx). But I want to select 1 cell. What finally worked:

<DataGrid Name="myXamlTable" AutoGeneratingColumn="myXamlTable\_AutoGeneratingColumn"

CanUserSortColumns="False" SelectionUnit="Cell">

<i:Interaction.Triggers>

<i:EventTrigger EventName="SelectedCellsChanged">

<prism:InvokeCommandAction Command="{Binding SelectionChangedCommand}"

TriggerParameterPath="AddedCells" />

</i:EventTrigger>

</i:Interaction.Triggers>

</DataGrid>

And in Viewmodel:

using Microsoft.Practices.Prism.Commands;

using Microsoft.Practices.Prism.Mvvm;

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace MyCSVLoader2

{

class CSVloader

{

string inputFile = "D:\\Temp\\test.csv";

DataTable myDataTable = new DataTable();

public DataTable myPropDataTable { get; set; }

//constructor

public CSVloader()

{

this.myPropDataTable = openCSV();

myXaml \_MyControl = new myXaml();

\_MyControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;

\_MyControl.DataContext = this;

//IList<DataGridCellInfo> selectedCells;

SelectionChangedCommand = new DelegateCommand<IList<DataGridCellInfo>>(OnSelection);

\_MyControl.Show();

//\_MyControl.Topmost = true;

}

public DelegateCommand<IList<DataGridCellInfo>> SelectionChangedCommand { get; private set; }

private void OnSelection(IList<DataGridCellInfo> addedCells)

{

string msg = "";

if (addedCells != null)

{

DataRowView dgrv = (DataRowView)addedCells[0].Item;

DataRow dgr = dgrv.Row;

DataGridColumn dgc = (DataGridColumn)addedCells[0].Column;

//int rI = Convert.ToInt16(dgr.ToString());

int rI = myPropDataTable

int cI = addedCells[0].Column.DisplayIndex;

//check in which column is clicked

msg = addedCells[0].Column.Header.ToString();

if (msg=="Accepted")

{

DataRow r = myDataTable.Rows[rI];

r["Obstruction"]= "Hans";

}

////this works to get the content of the cells

//foreach (DataGridCellInfo di in addedCells)

//{

// //Cast the DataGridCellInfo.Item to the source object type

// //In this case the ItemsSource is a DataTable and individual items are DataRows

// DataRowView dvr = (DataRowView)di.Item;

// msg = dvr.Row.ItemArray[0].ToString();

//}

}

MessageBox.Show(msg);

}

public DataTable openCSV()

{

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

//Console.WriteLine(line);

if (i == 0) //this contains the headers. Also use to create properties for Parameter class.

{

// add a checbox column for easy setting obstruction field

myDataTable.Columns.Add("Accepted", typeof(bool));

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, typeof(string));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 1;

foreach (var word in words)

{

row[x] = word;

if (myDataTable.Columns.IndexOf("Obstruction") == x)

{

if (word == "")

{

row["Accepted"] = true;

}

else

{

row["Accepted"] = false;

}

}

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

return myDataTable;

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

return myDataTable;

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

return myDataTable;

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set to a etc.

return myDataTable;

}

}

}

}

Well I couldn't get the row index of the selected cells. And in fact this method violates the MVVM rule that the ViewModel should not know about the View objects. [Here](http://stackoverflow.com/questions/20080130/how-to-bind-currentcell-in-wpf-datagrid-using-mvvm-pattern) I found the solution.

Almost perfect solution in MyCSVLoader2 project (selection works only after selecting other cell):

XAML:

<Window x:Class="MyCSVLoader2.myXaml"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

xmlns:prism="http://www.codeplex.com/prism"

xmlns:local ="clr-namespace:MyCSVLoader2"

Title="myXaml">

<Grid>

<DataGrid Name="myXamlTable" AutoGeneratingColumn="myXamlTable\_AutoGeneratingColumn"

CanUserSortColumns="False" CanUserReorderColumns="False" CanUserAddRows="False"

CanUserDeleteRows="False"

ItemsSource="{Binding myPropDataTable}"

CurrentCell="{Binding CellInfo, Mode=TwoWay}"

SelectedIndex="{Binding RowIndex}" SelectionMode="Single"

>

</DataGrid>

</Grid>

</Window>

Code behind XAML (myXaml.xaml.cs):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Controls.Primitives;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace MyCSVLoader2

{

/// <summary>

/// Interaction logic for myXaml.xaml

/// </summary>

public partial class myXaml : Window

{

public myXaml()

{

InitializeComponent();

}

private void myXamlTable\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

//Set properties on the columns during auto-generation

if (e.Column is DataGridCheckBoxColumn && !e.Column.IsReadOnly)

//this makes every checkbox column in datagrid 1 click reponsive

{

var checkboxFactory = new FrameworkElementFactory(typeof(CheckBox));

checkboxFactory.SetValue(FrameworkElement.HorizontalAlignmentProperty, HorizontalAlignment.Center);

checkboxFactory.SetValue(FrameworkElement.VerticalAlignmentProperty, VerticalAlignment.Center);

checkboxFactory.SetBinding(ToggleButton.IsCheckedProperty, new Binding(e.PropertyName) { UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged });

//checkboxFactory.SetBinding(myXamlTable.SelectedIndex, new Binding(e.PropertyName) { UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged });

//how to hookup to the state of this checkbox????????????

e.Column = new DataGridTemplateColumn

{

Header = e.Column.Header,

CellTemplate = new DataTemplate { VisualTree = checkboxFactory },

SortMemberPath = e.Column.SortMemberPath

};

}

switch (e.Column.Header.ToString())

{

case "Accepted":

//e.Column.CanUserSort = false;

e.Column.Width = 20;

//e.Column.Visibility = Visibility.Visible;

break;

case "Parameter":

//e.Column.CanUserSort = false;

//e.Column.Width = 20;

//e.Column.Visibility = Visibility.Visible;

break;

default:

//e.Column.CanUserSort = false;

e.Column.Width = 100;

//e.Column.Visibility = Visibility.Visible;

break;

}

}

}

}

ViewModel:

using Microsoft.Practices.Prism.Commands;

using Microsoft.Practices.Prism.Mvvm;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace MyCSVLoader2

{

public class CSVloader : INotifyPropertyChanged

{

string inputFile = "D:\\Temp\\test.csv";

string userName = "Hans";

DataTable myDataTable = new DataTable();

public DataTable myPropDataTable { get; set; }

//constructor

public CSVloader()

{

this.myPropDataTable = openCSV();

myXaml \_MyControl = new myXaml();

//\_MyControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;//now set in xaml

\_MyControl.DataContext = this;

\_MyControl.Show();

//\_MyControl.Topmost = true;

}

private int \_rowIndex;

public int RowIndex

{

get { return \_rowIndex; }

set

{

\_rowIndex = value;

OnPropertyChanged("RowIndex");

// MessageBox.Show(string.Format("Row: {0}", \_rowIndex.ToString()));

// if checkbox is clicked

if (\_cellInfo != null && \_cellInfo.Column.DisplayIndex == 0)

{

DataRow dr = myDataTable.Rows[\_rowIndex];

string cellContent = dr["Obstruction"].ToString();

if (cellContent == "")

{

dr["Obstruction"] = userName;

}

else if (cellContent.Contains(userName))

{

// remove username from the cell

cellContent = cellContent.Replace(userName,"");

if (cellContent.EndsWith(","))

{

cellContent = cellContent.Substring(0, cellContent.Length - 1);

}

if (cellContent.Contains(",,"))

{

cellContent.Replace(",,",",");

}

dr["Obstruction"] = cellContent;

}

else

{

dr["Obstruction"] += "," + userName;

}

}

}

}

private DataGridCellInfo \_cellInfo;

public DataGridCellInfo CellInfo

{

get { return \_cellInfo; }

set

{

\_cellInfo = value;

OnPropertyChanged("CellInfo");

//MessageBox.Show(string.Format("Column: {0}",

//\_cellInfo.Column != null ? \_cellInfo.Column.DisplayIndex.ToString() : "Index out of range!"));

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null) // if there is any subscribers

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

public DataTable openCSV()

{

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

//Console.WriteLine(line);

if (i == 0) //this contains the headers.

{

// add a checbox column for easy setting obstruction field

myDataTable.Columns.Add("Accepted", typeof(bool));

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, typeof(string));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 1;

foreach (var word in words)

{

row[x] = word;

if (myDataTable.Columns.IndexOf("Obstruction") == x)

{

if (word.Contains(userName))

{

row["Accepted"] = false;

}

else

{

row["Accepted"] = true;

}

}

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

return myDataTable;

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

return myDataTable;

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

return myDataTable;

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set etc.

return myDataTable;

}

}

}

}

In order to solve the problem of not being able to react directly on the checkbox. Try the solution with adding a checkbox column in XAML. Question is if you then can find out which row is clicked. Indeed this gives an error that you can see in the output window: System.Windows.Data Error: 40 : BindingExpression path error: 'isChecked' property not found on 'object' ''DataRowView' (HashCode=14918965)'. BindingExpression:Path=isChecked; DataItem='DataRowView' (HashCode=14918965); target element is 'CheckBox' (Name='Accepted'); target property is 'IsChecked' (type 'Nullable`1').

The binding is not correct. The reason is explained [here](http://www.codeproject.com/Articles/27432/Artificial-Inheritance-Contexts-in-WPF): "…when you try to bind a property on an object that is not in the element tree." Only the solutions seem rather complex. Maybe easier to create an extra object and bind to that in the right way? Or can I set the datacontext of the extra column to the same property as the datagrid? YES! That is the solution. Set the Binding of the checkbox to the DataContext.isChecked property of the type DataGrid as follows:

<CheckBox Name="Accepted" IsChecked="{Binding Path = DataContext.isChecked, RelativeSource={RelativeSource FindAncestor,AncestorType=DataGrid} ,Mode= TwoWay, UpdateSourceTrigger = PropertyChanged }"/>

However the next problem then is how to set the state of the extra checkbox according to the username in the "Obstruction" column. It took me days to not find a good solution.

Maybe find a way to bind to the Autogenerated checkboxcolumn? Other solution would be to make an ObservableCollection with Parameter objects, but then it becomes difficult to add properties on the fly. Maybe just make objects with a checked property for the binding of the checkbox? But how do you bind to these objects? Binding needs a source and a property. If e.g. we make an ObservableCollection with objects Check and bool property CheckProp we get an error about conversion from Collection…'System.Nullable`1[System.Boolean]' to …'System.Nullable`1[System.Boolean]' or something like that. It would then need a converter and becomes very complex again. There must be a way to use the values in an autogenerated column. Maybe during generation. But I have the impression that the added checkbox is generated later. So maybe use an event when the checkbox is initiated or loaded? YES! Next code works:

In the xaml control: <CheckBox Loaded="y\_Loaded"…

In the code behind:

private void y\_Loaded(object sender, RoutedEventArgs e)

{

CheckBox cb = (CheckBox)sender;

if (cb.DataContext.GetType() == typeof(DataRowView))

{

DataRowView drv = (DataRowView)cb.DataContext;

DataRow dr = drv.Row;

bool? b = (bool?)dr["Accepted"];

cb.IsChecked = b;

}

i += 1;

}

And the column "Accepted" can be hidden during AutogeneratingColmns.

So the complete code for the CSVloader becomes as follows.

myXaml.xaml:

<Window x:Class="MyCSVLoader2.myXaml"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

xmlns:prism="http://www.codeplex.com/prism"

xmlns:local ="clr-namespace:MyCSVLoader2"

Title="myXaml">

<Grid >

<DataGrid Name="myXamlTable" AutoGeneratingColumn="myXamlTable\_AutoGeneratingColumn"

CanUserSortColumns="False" CanUserReorderColumns="False" CanUserAddRows="True"

CanUserDeleteRows="False"

CurrentCell="{Binding CellInfo, Mode=TwoWay}"

SelectedIndex="{Binding RowIndex}" SelectionMode="Extended">

<DataGrid.Columns >

<DataGridTemplateColumn Header="x" >

<DataGridTemplateColumn.CellTemplate>

<DataTemplate >

<CheckBox Name="y" Loaded="y\_Loaded"

IsChecked="{Binding Path = DataContext.isChecked,

RelativeSource={RelativeSource FindAncestor,AncestorType=DataGrid},

Mode= TwoWay}">

</CheckBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

Code behind (myXaml.xaml.cs):

using System.Data;

using System.Windows;

using System.Windows.Controls;

namespace MyCSVLoader2

{

/// <summary>

/// Interaction logic for myXaml.xaml

/// </summary>

public partial class myXaml : Window

{

public myXaml()

{

InitializeComponent();

}

private void myXamlTable\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

//Set properties on the columns during auto-generation

switch (e.Column.Header.ToString())

{

case "Accepted":

//e.Column.CanUserSort = false;

//e.Column.Width = 20;

e.Column.Visibility = Visibility.Collapsed;

break;

case "Parameter":

//e.Column.CanUserSort = false;

//e.Column.Width = 20;

//e.Column.Visibility = Visibility.Visible;

break;

default:

//e.Column.CanUserSort = false;

e.Column.Width = 100;

//e.Column.Visibility = Visibility.Visible;

break;

}

}

private void y\_Loaded(object sender, RoutedEventArgs e)

{

//set the checkbox to the value in the hidden column "Accepted"

CheckBox cb = (CheckBox)sender;

if (cb.DataContext.GetType() == typeof(DataRowView))

{

DataRowView drv = (DataRowView)cb.DataContext;

DataRow dr = drv.Row;

bool? b = (bool?)dr["Accepted"];

cb.IsChecked = b;

}

}

}

}

Viewmodel (CSVloader.cs):

using System;

using System.ComponentModel;

using System.Data;

using System.IO;

using System.Windows;

using System.Windows.Controls;

using System.Text.RegularExpressions;

namespace MyCSVLoader2

{

public class CSVloader : INotifyPropertyChanged

{

string inputFile = "D:\\Temp\\test.csv";

string userName = "Hans";

DataTable myDataTable = new DataTable();

//the property CSVloader.myPropDataTable is used as itemsSource for the datagrid

public DataTable myPropDataTable { get; set; }

//constructor

public CSVloader()

{

this.myPropDataTable = openCSV();

myXaml \_MyControl = new myXaml();

\_MyControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;

\_MyControl.DataContext = this;

\_MyControl.Show();

}

private int \_rowIndex;

public int RowIndex

{

get { return \_rowIndex; }

set

{

\_rowIndex = value;

OnPropertyChanged("RowIndex");

// MessageBox.Show(string.Format("Row: {0}", \_rowIndex.ToString()));

}

}

private DataGridCellInfo \_cellInfo;

public DataGridCellInfo CellInfo

{

get { return \_cellInfo; }

set

{

\_cellInfo = value;

OnPropertyChanged("CellInfo");

}

}

private bool \_isChecked;

public bool isChecked

{

get { return \_isChecked; }

set

{

\_isChecked = value;

//OnPropertyChanged("isChecked"); //this sets all checkboxes to checked...

// if checkbox is clicked

DataRow dr = myDataTable.Rows[\_rowIndex];

string cellContent = dr["Obstruction"].ToString();

if (\_cellInfo != null && !\_isChecked) //add the userName

{

if (cellContent == "")

{

dr["Obstruction"] = userName;

}

else

{

dr["Obstruction"] += "," + userName;

}

}

else

{

// remove username from the cell

cellContent = cellContent.Replace(userName, "");

//remove double and end commas

cellContent = Regex.Replace(cellContent, ",{2,}", ",").Trim(',');

dr["Obstruction"] = cellContent.Trim();

}

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null) // if there is any subscribers

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

public DataTable openCSV()

{

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

//Console.WriteLine(line);

if (i == 0) //this contains the headers. Also use to create properties for Parameter class.

{

//// add a checbox column for easy setting obstruction field

myDataTable.Columns.Add("Accepted", typeof(bool));

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, typeof(string));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 1;

foreach (var word in words)

{

row[x] = word;

if (myDataTable.Columns.IndexOf("Obstruction") == x)

{

if (word.Contains(userName))

{

row["Accepted"] = false;

}

else

{

row["Accepted"] = true;

}

}

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

return myDataTable;

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

return myDataTable;

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

return myDataTable;

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set to a etc.

return myDataTable;

}

}

}

}

Let's now try to make it work in pCOLAD. It turns out that there is a problem afterall in CSVloader. When you set the IsChecked property of a checkbox in the code behind you also invoke a change in the isChecked property of the CSVloader instance. You need that property to bind the action of a single click on the checkbox. That is no problem the first time, but when you close the control and create a new one (like we do in pSHARE) then yout get a misfit with the autogenerated (and hidden) column "Accepted". The reason is that when you create a new control, first the binding is used. So the control gets the \_rowIndex property and sets the SelectedIndex accordingly. And then because the relativeSource of the checkBox is the datagrid, the control gets the IsChecked property of all the rows, which at start is false, and sets the checkBox accordingly of all rows to false. Then when the CheckBox is loaded it sets the IsChecked property to the value of the cells in column "Accepted". So you must keep the column "Accepted" up to date.

However there is a warning about hiding the PropertyChanged eventhandler. This is probably because pSHARE inheridits it through NodeModel or IWpfNode. Maybe we should use the DelegateCommand instead. That might be a good idea for the buttons (avoiding the RelayCommand class), but we can use the RaisePropertyChanged instead of the OnPropertyChanged and then leave out the whole PropertyChanged eventhandler.

# Get the output of pSHARE in right format

Also we should try to get the output right. It should be a list of strings where the name of the parameters is altered with their new value. Then pPARAM can find the name and use the next string in the list as value. Of course we must add the actual inputs of pSHARE to the list. And they should be added to the DataGrid too, so in fact add them to the DataTable. Then take the name column, check if there is no obstruction and if there is a new value and then add those to the OutputList. But where in the code should this happen? Well, after adding the input to the DataTable. The added parameters list should have the format: Parameter;Value;Importance;Comments;Owner followed on next line by the values:ParameterName1;Value1;Importance1;Comments1;Owner1 followed on next line by Parameter;Value;Importance;Comments;Owner followed on next line by the values: ParameterName2;Value2;Importance2;Comments2;Owner1 (of course same owner) etc.. In fact maybe better to now first develop pCOLLECT.

Here we run into the problem of how to get the inputs and add the semicolon. On GitHub Yu Ke (@ke-yu) [answered a solution](https://github.com/DynamoDS/Dynamo/issues/4776). It turns out that in a custom node the inputs are not directly accessible as strings or other types. They are of type AssociativeNode and stored in a List< AssociativeNode > inputAstNodes. To add a semicolon you have to convert them into Abstract Syntax Tree (AST) nodes. For this you use AstFactory:

var semiColon = AstFactory.BuildStringNode(";"); And then build a new List< AssociativeNode > as follows:

List<AssociativeNode> pCOLLECToutputList = new List<AssociativeNode>();

foreach (AssociativeNode InputItem in inputAstNodes)

{

pCOLLECToutputList.Add(InputItem);

pCOLLECToutputList.Add(semiColon);

}

//remove the last semiColon

if (pCOLLECToutputList.Count > 0)

{

pCOLLECToutputList.RemoveAt(pCOLLECToutputList.Count - 1);

}

However then we would get an output with the semicolon on every second line. Therefore we need a function that adds the AssociativeNodes together:

var funcNode = AstFactory.BuildFunctionCall("%add", pCOLLECToutputList);

And then we can prepare the output as follows:

return new[]

{AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), funcNode)};

However it gives the error in Figure 15. Since Dynamo has a new version, I installed 0.8. But then all kinds of things go wrong. First of course the post build events must put the dll in the nodes folder of Dynamo 0.8. Also the References DynamoCore, ProtoCore and ProtoInterface must be changed. Then compiling gave a list of errors:

Error 5 'Dynamo.Models.NodeModel' does not contain a constructor that takes 1 arguments D:\Data\Research\Dynamo\pCOLAD for Dynamo\pCOLLECT.cs 106 15 pCOLAD02

Error 11 'Dynamo.Models.NodeModel' does not contain a constructor that takes 1 arguments D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 127 15 pCOLAD02

Error 1 The command "REM the echo f is needed if the file doesn't exist yet it tells the compiler that it is a file to be copied, and not a directory

echo f |xcopy /r /y “D:\Data\Research\Dynamo\pCOLAD for Dynamo\bin\Debug\pCOLAD.dll” “C:\Program Files\Dynamo 0.8\nodes\pCOLAD.dll”

echo f |xcopy /r /y “D:\Data\Research\Dynamo\pCOLAD for Dynamo\bin\Debug\pCOLAD.pdb” “C:\Program Files\Dynamo 0.7\nodes\pCOLAD.pdb”" exited with code 4. pCOLAD02

Error 3 The type or namespace name 'Commands' does not exist in the namespace 'Dynamo.UI' (are you missing an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pCOLLECT.cs 9 17 pCOLAD02

Error 8 The type or namespace name 'Commands' does not exist in the namespace 'Dynamo.UI' (are you missing an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 7 17 pCOLAD02

Error 2 The type or namespace name 'Controls' does not exist in the namespace 'Dynamo' (are you missing an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pCOLLECT.cs 6 14 pCOLAD02

Error 7 The type or namespace name 'Controls' does not exist in the namespace 'Dynamo' (are you missing an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 4 14 pCOLAD02

Error 10 The type or namespace name 'DelegateCommand' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 116 16 pCOLAD02

Error 12 The type or namespace name 'DelegateCommand' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 136 25 pCOLAD02

Error 6 The type or namespace name 'dynNodeView' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pCOLLECT.cs 230 43 pCOLAD02

Error 13 The type or namespace name 'dynNodeView' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 278 43 pCOLAD02

Error 4 The type or namespace name 'IWpfNode' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pCOLLECT.cs 43 40 pCOLAD02

Error 9 The type or namespace name 'IWpfNode' could not be found (are you missing a using directive or an assembly reference?) D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs 25 38 pCOLAD02

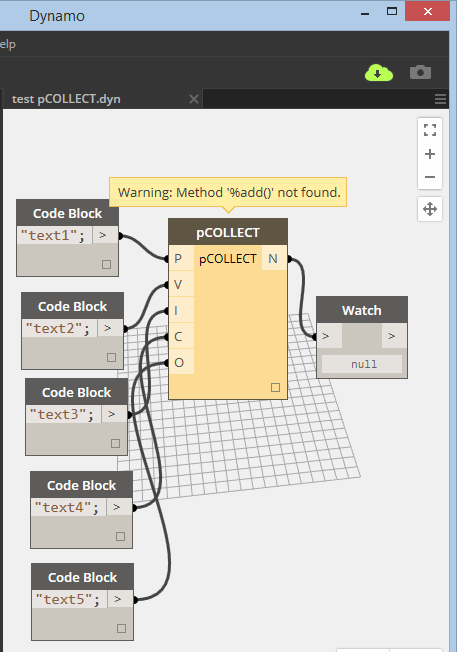


Figure 15 Error in pCOLLECT

So we had to download the Dynamo-RC0.8.0\_master.zip and check everything again. BTW the error in Figure 15 was due to the fact that I tried to add up more than two items. What finally worked:

using System.Collections.Generic;

using System.Windows;

using Autodesk.DesignScript.Runtime;

using Dynamo.Controls;

using Dynamo.Models;

using Dynamo.UI;

using Dynamo.UI.Commands;

using ProtoCore.AST.AssociativeAST;

using Dynamo.Wpf;

namespace pCOLADnamespace

{

/// <summary>

/// Dynamo uses the MVVM model of programming,

/// in which the UI is data-bound to the view model, which

/// exposes data from the underlying model. Custom UI nodes

/// are a hybrid because NodeModel objects already have an

/// associated NodeViewModel which you should never need to

/// edit. So here we will create a data binding between

/// properties on our class and our custom UI.

///

/// </summary>

///

// The NodeName attribute is what will display on

// top of the node in Dynamo

[NodeName("pCOLLECT")]

// The NodeCategory attribute determines how your

// node will be organized in the library. You can

// specify your own category or use one of the

// built-ins provided in BuiltInNodeCategories.

[NodeCategory("pCOLAD")]

// The description will display in the tooltip

// and in the help window for the node.

[NodeDescription("Collects parameters and their attributes for pSHARE.")]

[IsDesignScriptCompatible]

public class pCOLLECT : NodeModel

{

#region constructor

/// <summary>

/// The constructor for a NodeModel is used to create

/// the input and output ports and specify the argument

/// lacing.

/// </summary>

/// <param name="workspace"></param>

public pCOLLECT()

{

// When you create a UI node, you need to do the

// work of setting up the ports yourself. To do this,

// you can populate the InPortData and the OutPortData

// collections with PortData objects describing your ports.

InPortData.Add(new PortData("P", "Parameter name as string."));

InPortData.Add(new PortData("V", "Value as string."));

InPortData.Add(new PortData("I", "Importance as string."));

InPortData.Add(new PortData("C", "Comment as string."));

InPortData.Add(new PortData("O", "Owner as string."));

// Nodes can have an arbitrary number of inputs and outputs.

// If you want more ports, just create more PortData objects.

OutPortData.Add(new PortData("N", "List of strings."));

//OutPortData.Add(new PortData("some awesome", "A result."));

// This call is required to ensure that your ports are

// properly created.

RegisterAllPorts();

// The arugment lacing is the way in which Dynamo handles

// inputs of lists. If you don't want your node to

// support argument lacing, you can set this to LacingStrategy.Disabled.

//ArgumentLacing = LacingStrategy.CrossProduct;

ArgumentLacing = LacingStrategy.Disabled;

}

#endregion

#region public methods

/// <summary>

/// If this method is not overriden, Dynamo will, by default

/// pass data through this node. But we wouldn't be here if

/// we just wanted to pass data through the node, so let's

/// try using the data.

/// </summary>

/// <param name="inputAstNodes"></param>

/// <returns></returns>

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

// When you create your own UI node you are responsible

// for generating the abstract syntax tree (AST) nodes which

// specify what methods are called, or how your data is passed

// when execution occurs.

// WARNING!!!

// Do not throw an exception during AST creation. If you

// need to convey a failure of this node, then use

// AstFactory.BuildNullNode to pass out null.

// Using the AstFactory class, we can build AstNode objects

// that assign doubles, assign function calls, build expression lists, etc.

//build a new output List<AssociativeNode>consisting of fieldnames seperated by ';' and

// on next inputAstNodes with ';' added

List<AssociativeNode> pCOLLECTtempList = new List<AssociativeNode>();

var headings = AstFactory.BuildStringNode("Parameter;Value;Importance;Comments;Owner");

var semiColon = AstFactory.BuildStringNode(";");

foreach (AssociativeNode InputItem in inputAstNodes)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(InputItem);

arguments.Add(semiColon);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

//don't add ';' to the last one

if (inputAstNodes.IndexOf(InputItem) == inputAstNodes.Count - 1)

{

pCOLLECTtempList.Add(InputItem);

}

else

{

pCOLLECTtempList.Add(funcNode);

}

}

// now pCOLLECTtempList has the inputs followed by ';'

// but it should become one string so add the items together

List<AssociativeNode> pCOLLECToutputList = new List<AssociativeNode>();

AssociativeNode A = pCOLLECTtempList[0];

for (int i = 0; i < pCOLLECTtempList.Count - 1; i++)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(A);

arguments.Add(pCOLLECTtempList[i + 1]);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

A = funcNode;

}

pCOLLECToutputList.Add(A);

pCOLLECToutputList.Insert(0, headings);

return new[]

{

// In these assignments, GetAstIdentifierForOutputIndex finds

// the unique identifier which represents an output on this node

// and 'assigns' that variable the expression that you create.

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(pCOLLECToutputList))

};

}

/// <summary>

/// View customizer for HelloDynamo Node Model.

/// </summary>

public class pCOLLECTNodeViewCustomization : INodeViewCustomization<pCOLLECT>

{

/// <summary>

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="model">The NodeModel representing the node's core logic.</param>

/// <param name="nodeView">The NodeView representing the node in the graph.</param>

public void CustomizeView(pCOLLECT model, NodeView nodeView)

{

// The view variable is a reference to the node's view.

// In the middle of the node is a grid called the InputGrid.

// We reccommend putting your custom UI in this grid, as it has

// been designed for this purpose.

// Create an instance of our custom UI class (defined in xaml),

// and put it into the input grid.

var \_pCOLLECTcontrol = new pCOLLECTcontrol();

nodeView.inputGrid.Children.Add(\_pCOLLECTcontrol);

// Set the data context for our control to be this class.

// Properties in this class which are data bound will raise

// property change notifications which will update the UI.

\_pCOLLECTcontrol.DataContext = model;

}

/// <summary>

/// Here you can do any cleanup you require if you've assigned callbacks for particular

/// UI events on your node.

/// </summary>

public void Dispose() { }

}

#endregion

}

}

This works alright for generating the output of pCOLLECT, but e.g. in pSHARE we need to add the input to a DataTable. Then we need strings. Probably the TryGetValue or TryGetInput method should be used. This works on a Dictionary. You can find the info [here](https://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.100).aspx). So we need to know what Dictionary to use. After some trial and error it turns out that the Inputs object is a Dictionary< int, System.Tuple<int,NodeModel>> To find out what should be in the NodeModel we iterate trough the dictionary as follows:

foreach (KeyValuePair<int, System.Tuple<int, NodeModel>> InputItem in Inputs)

{

var ttt = InputItem.Value.Item2;

}

And find that ttt = {Dynamo.Nodes.CodeBlockNodeModel}.

So now we can use

foreach (KeyValuePair<int, System.Tuple<int, NodeModel>> InputItem in Inputs)

{

Dynamo.Nodes.CodeBlockNodeModel ttt = (Dynamo.Nodes.CodeBlockNodeModel)InputItem.Value.Item2;

string sss = ttt.Code;

}

However it turns out that sss doesn't show the input strings in the right order. That is because of the nature of a Dictionary. So we should iterate through the keys:

for (int i = 0; i < Inputs.Count ; i++)

{

var item = Inputs[i];

Dynamo.Nodes.CodeBlockNodeModel itemValue = (Dynamo.Nodes.CodeBlockNodeModel)item.Item2;

string s = itemValue.Code;

}

Next step is see if we can get the filewatcher working. So when somebody changes the csv file, you get a warning and must close MyControl. Let's first change that name into CSVControl. StackOverflow however advises to use fileInfo instead, because of OS callbacks etc. In fact if I only check at the moment of trying to save to the csv file if it was changed by somebody else, that would be enough. During the pCOLAD Grasshopper testcase we experienced some issues with filewatcher. Sot try the fileInfo solution. However Dynamo already has a FileWatcher in the FilePath node (see [here](https://github.com/DynamoDS/Dynamo/pull/2572)).

# Save to csv file and History file

Hmm, first the save option ofcourse. Must add some buttons for that. To keep the buttons at the bottom of the window, you need a Grid with Grid.RowDefinitions and the first row you give height "\*", meaning that it will fill the vertical space that is left over in the parent (the Window)and the second row you give height "Auto". Inside the first row we put the DataGrid inside the second row we put a Grid with GridColumnDefinitions resp. "Auto", "\*", "Auto" and there we put the buttons with some margins to keep them apart. In XAML this gives:

<Window x:Class="pCOLADnamespace.CSVControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="530">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<DataGrid Grid.Row="0" Name="myXamlTable" AutoGeneratingColumn="myXamlTable\_AutoGeneratingColumn"

CanUserSortColumns="False" CanUserReorderColumns="False" CanUserAddRows="False"

CanUserDeleteRows="False"

CurrentCell="{Binding CellInfo, Mode=TwoWay}"

SelectedIndex="{Binding RowIndex}" SelectionMode="Extended" VerticalAlignment="Top">

<DataGrid.Columns >

<DataGridTemplateColumn Header="x" >

<DataGridTemplateColumn.CellTemplate>

<DataTemplate >

<CheckBox Name="y" Loaded="y\_Loaded"

IsChecked="{Binding Path = DataContext.isChecked,

RelativeSource={RelativeSource FindAncestor,AncestorType=DataGrid},

Mode=TwoWay}">

</CheckBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

<Grid Grid.Row="1" VerticalAlignment="Bottom" Height="25">

<Grid.ColumnDefinitions >

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Button x:Name="Cancel" Grid.Column ="2" Content="Cancel" VerticalAlignment="Bottom"

HorizontalAlignment="Right"

/>

<Button x:Name="Share" Grid.Column ="2" Content="Share" VerticalAlignment="Bottom" Margin="299,0,44,0"

/>

<Button x:Name="History" Grid.Column ="2" Content="History" VerticalAlignment="Bottom"

Margin="252,0,91,0"

/>

<Button x:Name="UnCheckAll" Grid.Column ="0" Content="UnCheckAll" VerticalAlignment="Bottom"

Margin="55,0,5,0"

/>

<Button x:Name="CheckAll" Grid.Column ="0" Content="CheckAll" VerticalAlignment="Bottom"

HorizontalAlignment="Left"

/>

</Grid>

</Grid>

</Window>

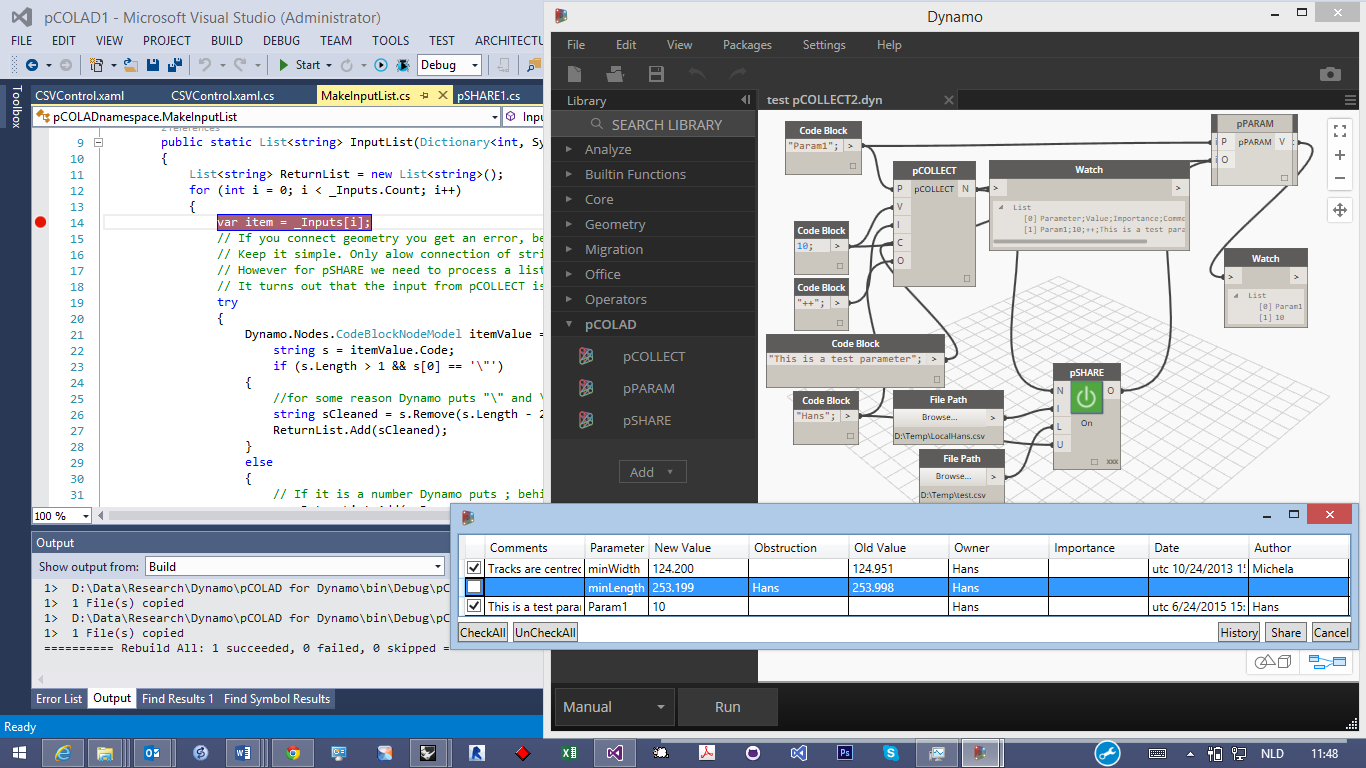


Figure 16 So far so good…

Figure 16 shows the state on 26th of June 2015 when I continued the [discussion on GitHub](https://github.com/DynamoDS/Dynamo/issues/4776) with Michael Kirschner (@holyjewsus) about how to get the Inputs from custom nodes. For some reasons he and Matt Jezyk (@tatlin) find that I shouldn't interact directly with the nodes in the graph. So I explained the project as follows:

First I have to state that I am not a programmer. I picked up some knowledge by doing. I am working on a research project called pCOLAD, as presented and published at the eCAADe 2014 conference. I want to extend it from Grasshopper to Dynamo. That meant from vb.net and Windows Forms to C#, xaml and MVVM (quite a step!).

pCOLAD is about multidisciplinary parametric design. The idea is that different disciplines use some crucial shared parameters and are immediately informed if a value of a shared parameter is changed. Since a CSV file is easy for all disciplines to handle, I need a pCOLLECT node to gather the crucial parameters and values in the Dynamo solution. And link that to a pSHARE node, that displays a copy of the CSV file (a xaml control with a DataGrid bound to a DataTable) and adds the output of pCOLLECT. A pPARAM node uses the output of pSHARE to find the parameters and values that are needed from other team members (see attached image). Of course it is more complex than this. In order to understand what a parameter is about, you have to add several attributes. This must be flexible. So pCOLLECT should have the possibility to add parameter inputs. And some smart mapping of different attributes is needed.

As written above I found a way to turn the input of pCOLLECT into CSV format output. Now I linked that to an input of pSHARE, but again have difficulties to use it (in the image it is faked). I use the Inputs Dictionary to find the data. But need to know the type of course. It turns out to be of type pCOLLECT. I was expecting a Dynamo type like Dynamo.Nodes.CodeBlockNodeModel.

But you say this is an unsafe way. Do you mean that it is not reliable, or that in future versions of Dynamo it will not work (as I noticed going from 0.7 to 0.8). The last is not a big problem. It is a research project. Once Dynamo is more stable I plan to find funding for a test case project.

And I attached Figure 16.

In the meanwhile I figured out how to make a function that takes care of reading the Inputs. It turns out that they are of the type of the connected nodes. So you have to check all of the possibilities or use the var type. However if the input comes from a CodeBlock then the Type.Name is "CodeBlockNodeModel" and you can get the content with the .Code property. When I connect pCOLLECT to pSHARE then I can of course check for that type and use a property of pCOLLECT which contains the output list: outputListProp. If we connect a List.Create node then the Type.Name is "CreateList". And the object you get is Dynamo.Models.NodeModel {DSCoreNodesUI.CreateList}. To get the content you get its Inputs and check what type they are and get their content accordingly. If you use the StringFromObject node you must know that testType = {Name = "FromObject" FullName = "DSCoreNodesUI.StringNodes.FromObject"}. And to get the value use the CachedValue.Data property. If you connect String.Substring then you get testType = {Name = "DSFunction" FullName = "Dynamo.Nodes.DSFunction"} and you get the value also with the CachedValue.Data property. However it seems that this only works when you run it a second time… So there must be a better way. [Here](https://github.com/DynamoDS/Dynamo/issues/3770)  Ian Keough (@ikeough) says: If the node is only modifying the items in a list, then passing the modified list out, I would make a method that modifies the list as you would like and returns a list, then use a FunctionCallNode in your generated AST to call that method. Using the GetMirror() method should be avoided unless you need to know something about the incoming list to populate your UI. It's not great to use that because your node model then requires a reference to the EngineController, which is possible, but is something that we don't like to recommend.

So similar to what Yu-Ke answered make a funcNode…….

var semiColon = AstFactor.BuildStringNode(";");

var arguments = new List<AssociativeNode>() {

// suppose custom node model only has one input

inputAstNodes[0],

semiColon};

// To append ";" to input string, use +. For example, "hello" + ";"

// + operation is actually a function call %add(x, y), so we create a function call node

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

return new[]

{

AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), funcNode);

};

+ test2 {DSCoreNodesUI.CreateList} Dynamo.Models.NodeModel {DSCoreNodesUI.CreateList}

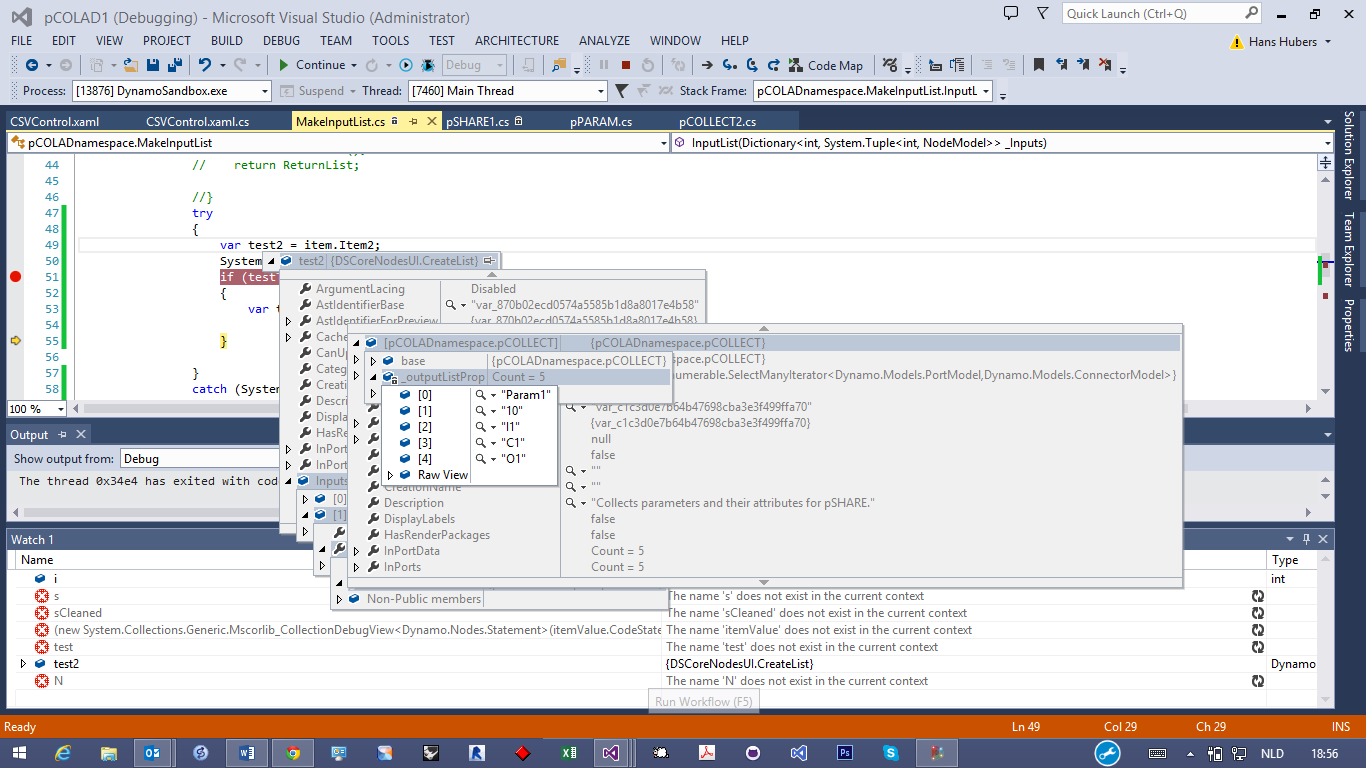


Figure 17 Through debugging when moving mouse over a variable you get all the properties, methods etc

Figure 17 shows that when detecting that the N input of pSHARE is the output of a List.CreateList you can find the item2 of item2 etc. to get the pCOLLECT outputs.

BTW when using the ObjectBrowser it is handy to know [what the symbols mean](https://msdn.microsoft.com/en-us/library/vstudio/y47ychfe(v=vs.100).aspx). Below some trial and error to get the content of the inputs before using a switch statement; just to avoid trying these again.

using System.Collections.Generic;

using Dynamo.Models;

using System.Windows;

using ProtoCore.AST.AssociativeAST;

using System.Collections.ObjectModel;

namespace pCOLADnamespace

{

public static class MakeInputList

{

public static List<string> InputList(Dictionary<int, System.Tuple<int, NodeModel>> \_Inputs)

{

List<string> ReturnList = new List<string>();

for (int i = 0; i < \_Inputs.Count; i++)

{

//the Inputs object is a Dictionary< int, System.Tuple<int,NodeModel>>

var item = \_Inputs[i];

// If you connect geometry you get an error, because type is then Dynamo.Nodes.DSFunction

// However for pSHARE we need to process a list<string> or the output of pCOLLECT directly.

// It turns out that the input N of pSHARE from pCOLLECT is type pCOLLECT so use its property \_outputListProp

//try if every input can be converted in an AssiociativeNode, we then can always use the AstFactory

//nope, doesn't work

//try to get the PortData of the NodeModel. Is null.

//try

//{

// NodeModel test2 = item.Item2;

// ObservableCollection<PortData> tt2 = test2.InPortData;

// //PortData tt3 = tt2[0];

// System.Tuple<int,NodeModel> tt5;

// HashSet<System.Tuple<int,Dynamo.Models.NodeModel>> tt6;

// test2.TryGetOutput(0, out tt6);

// test2.TryGetInput(0,out tt5);

//}

//catch (System.Exception)

//{

// MessageBox.Show("Somthing wrong");

// ReturnList.Clear();

// return ReturnList;

//}

// Make a switch construct !!!!!!!!!!!!!!! v now is the outputlist van pCOLLECT

// If List.CreateList is input of pSHARE then recursively run the method to get v

// You have to add v to ReturnList and return the ReturnList finally.

switch (switch\_on)

{

default:

}

try

{

var test2 = item.Item2;

System.Type testType = test2.GetType();

if (testType.Name == "CreateList")

{

Dictionary<int,System.Tuple<int,NodeModel>> t = test2.Inputs;

List<string> t1 = MakeInputList.InputList(t);;

}

}

catch (System.Exception)

{

MessageBox.Show("Please connect only output of pCOLLECT or List.CreateList to N input of pSHARE.");

ReturnList.Clear();

return ReturnList;

}

try

{

//Dynamo.Nodes.CustomNodeController<Dynamo.CustomNodeDefinition> tt = (Dynamo.Nodes.CustomNodeController<Dynamo.CustomNodeDefinition>)item.Item2;

var test2 = item.Item2;

System.Type testType = test2.GetType();

if (testType.Name == "pCOLLECT")

{

pCOLLECT test = (pCOLLECT)item.Item2;

var v = test.outputListProp;

}

}

catch (System.Exception)

{

MessageBox.Show("Please connect only output of pCOLLECT to N input of pSHARE.");

ReturnList.Clear();

return ReturnList;

}

try

{

Dynamo.Nodes.CodeBlockNodeModel itemValue = (Dynamo.Nodes.CodeBlockNodeModel)item.Item2;

string s = itemValue.Code;

if (s.Length > 1 && s[0] == '\"')

{

//for some reason Dynamo puts "\" and \";" around the string

string sCleaned = s.Remove(s.Length - 2).Remove(0, 1);

ReturnList.Add(sCleaned);

}

else

{

// If it is a number Dynamo puts ; behind it.

ReturnList.Add(s.Remove(s.Length-1));

}

}

catch (System.Exception)

{

MessageBox.Show("Please connect only strings or numbers.");

ReturnList.Clear();

return ReturnList;

}

}

return ReturnList;

}

}

}

Testing the type of the File Path node output shows it is = {DSCore.File.Filename}. But if I try to use this as a type I get the error: "The type name 'Filename' does not exist in the type 'DSCore.File'". If I try to use DSCore.IO.FilePath I get the error: "Cannot declare a variable of static type 'DSCore.IO.FilePath'". How do I get the file path in my custom UI node? Figure 18 shows that it should be possible to get it.

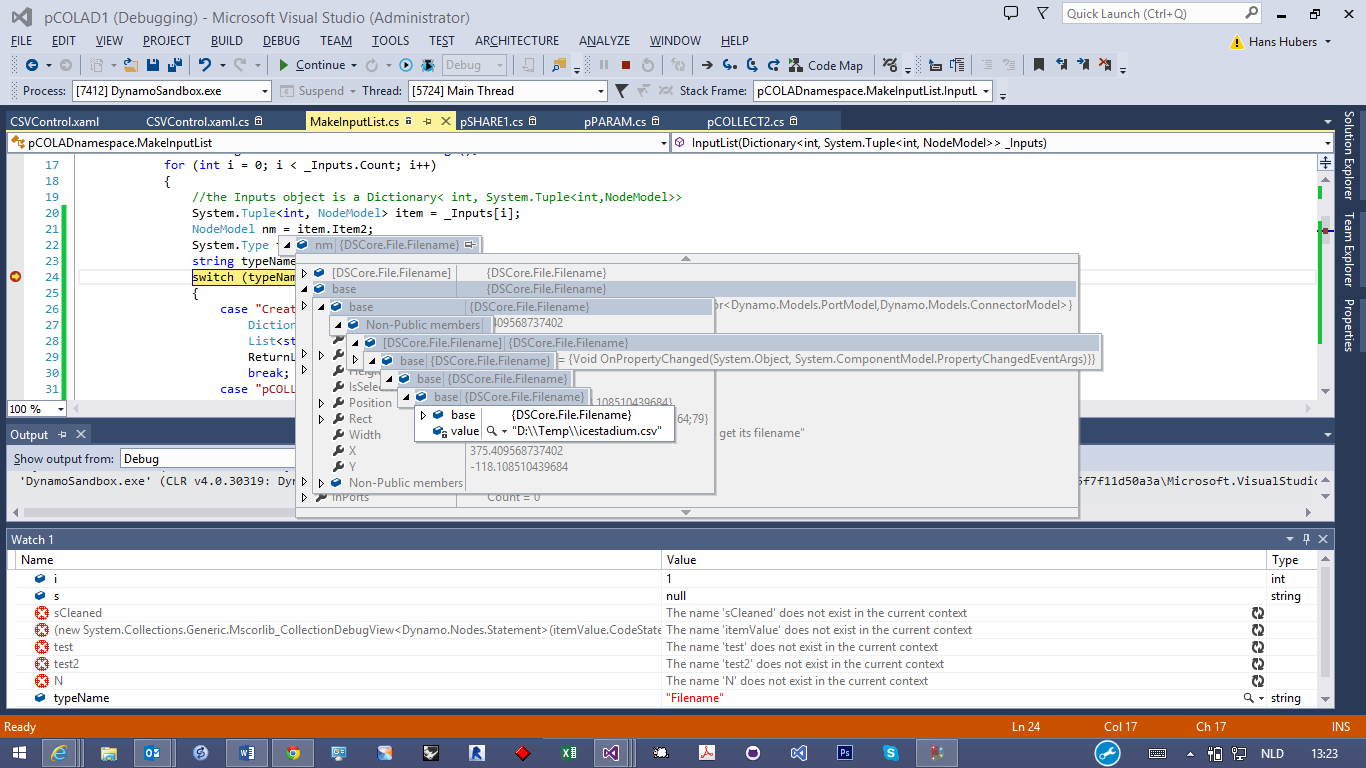


Figure 18 Using intelli sense to find the value of output of File Path node

DSCore.IO.FilePath.FileName(string, [bool])

However Yu-Ke answered:

hi [**@jhubers**](https://github.com/jhubers) , DSCore.File.Filename is the type of File Path node, but File Path node should only output string.

I'm afraid you have to write a little bit AST nodes, similar toSampleLibraryUI.Examples.CustomNodeModel.BuildOutputAst(), where

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

....

var funcNode = AstFactory.BuildFunctionCall(

new Func<double,double>(SampleUtilities.MultiplyInputByNumber),

new List<AssociativeNode>(){doubleNode});

....

}

You may implement your logic in a static class, for example:

public static class MyDataCollector

{

[IsVisibleInDynamoLibrary(false)]

public static string pCollect(string filePath, int foo, double bar);

{

...

}

}

And in you custom node model's BuildOutputAst(), change that line to:

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

....

var funcNode = AstFactory.BuildFunctionCall(

// Create my function MyDataCollector.pCollect with some inputs

// Func<string, int, double, string> is my function's signature

new Func<string, int, dobule, string>(MyDataCollector.pCollect),

inputAstNodes);

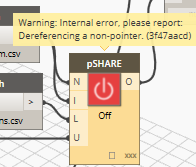
....

}

And that's it! If you connect File Path to, say the first input port of custom node, you should get a string.

We don't directly deal with specific nodes in a custom node's implementation (for example, won't care about if the input is CodeBlockNodeModel or whatever else). The inputs are either primitive types like integer, string, a List of string or some public types that defined in custom node library and will be imported by Dynamo.

BTW it is good to know that these greyed code snippets in the issues at GitHub can be created by putting `…` (back ticks) around parts of the text or ``` then some lines ``` for a whole code block. Look [here](https://help.github.com/articles/basic-writing-and-formatting-syntax/) for more mark ups format.



using Autodesk.DesignScript.Runtime;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace pCOLADnamespace

{

public static class MyDataCollector

{

[IsVisibleInDynamoLibrary(false)]

public static string filePath(string \_filePath)

{

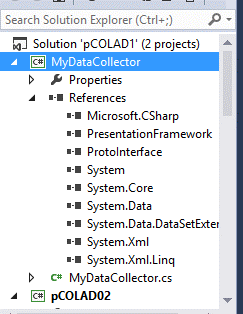
//System.Windows.MessageBox.Show("I can make something happen here...");

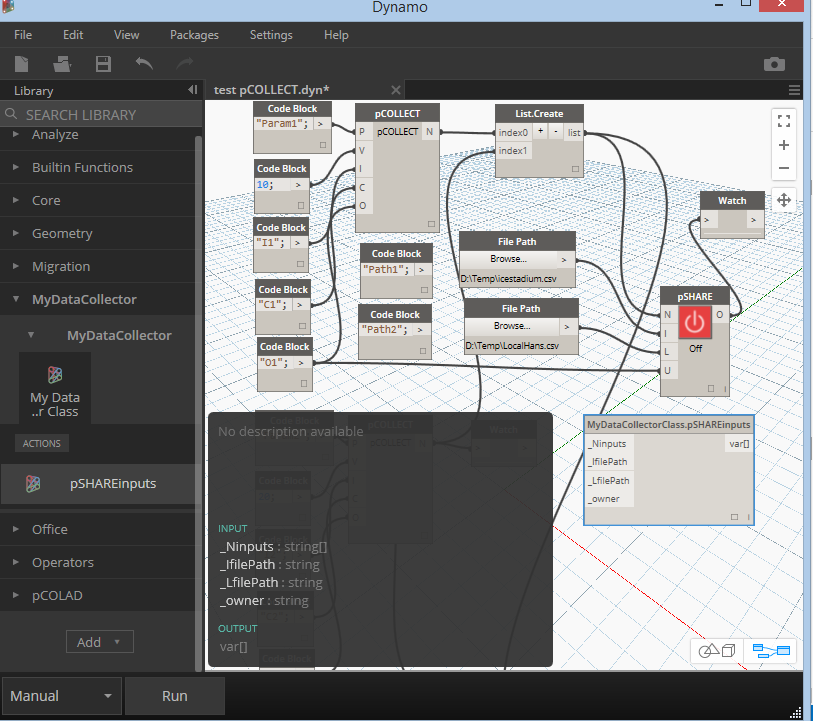
return \_filePath;

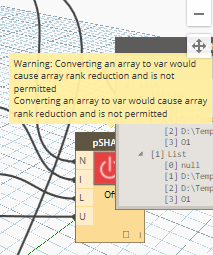
}

}

}







# How to put your solution on GitHub

Be aware that in order to load an existing VS solution to GitHub there are [some specific steps to be taken](http://stackoverflow.com/questions/19982053/how-do-i-add-an-existing-solution-to-github-from-visual-studio-2013):

1. Open the solution in Visual Studio 2013
2. Select File | Add to Source Control
3. Select the Microsoft Git Provider  
   That creates a local GIT repository
4. Surf to GitHub
5. Create a new repository DO NOT SELECT Initialize this repository with a README  
   That creates an empty repository with no Master branch
6. Once created open the repository and copy the URL (it's on the right of the screen in the current version)
7. Go back to Visual Studio
   * Make sure you have the Microsoft Git Provider selected under Tools/Options/Source Control/Plug-in Selection
8. Open Team Explorer
9. Select Home | Unsynced Commits
10. Enter the GitHub URL into the yellow box
11. Click Publish
12. Select Home | Changes
13. Add a Commit comment
14. Select Commit and Push from the drop down

Your solution is now in GitHub

pSHARE.cs before loading solution to GitHub:

using System.Collections.Generic;

using System.Windows;

using Autodesk.DesignScript.Runtime;

using Dynamo.Controls;

using Dynamo.Models;

using Dynamo.UI.Commands;

using ProtoCore.AST.AssociativeAST;

using System;

using System.IO;

using System.Data;

using System.Text.RegularExpressions;

using System.Windows.Controls;

using System.Windows.Input;

using Dynamo.Wpf;

//using MyDataCollector;

namespace pCOLADnamespace

{

#region some node settings

/// pSHARE takes care of the communication of parameter changes through a shared \*.csv file.

[NodeName("pSHARE")]

[NodeCategory("pCOLAD")]

[NodeDescription("Load and share changes to parameters.")]

[IsDesignScriptCompatible]

#endregion

public class pSHARE : NodeModel

{

#region properties

bool On = false;

private string \_OnOffButton;

// later replace with an input

string inputFile = "D:\\Temp\\test2.csv";

string userName = "Hans";

DataTable myDataTable;

/// <summary>

/// the property pSHARE.myPropDataTable is used as itemsSource for the datagrid

/// </summary>

public DataTable myPropDataTable { get; set; }

private int \_rowIndex;

public int RowIndex

{

get { return \_rowIndex; }

set

{

\_rowIndex = value;

RaisePropertyChanged("RowIndex");

// MessageBox.Show(string.Format("Row: {0}", \_rowIndex.ToString()));

}

}

private DataGridCellInfo \_cellInfo;

/// <summary>

/// property of pSHARE about which cell is selected

/// </summary>

public DataGridCellInfo CellInfo

{

get { return \_cellInfo; }

set

{

\_cellInfo = value;

RaisePropertyChanged("CellInfo");

//MessageBox.Show(string.Format("Column: {0}",

}

}

private bool \_isChecked;

/// <summary>

/// property of pSHARE telling if a row is checked and so a value obstructed

/// </summary>

public bool isChecked

{

get { return \_isChecked; }

set

{

\_isChecked = value;

//OnPropertyChanged("isChecked"); //this sets all checkboxes to checked...

// if checkbox is clicked

DataRow dr = myDataTable.Rows[\_rowIndex];

//also change the value in the hidden column "Accepted"

dr["Accepted"] = value;

string cellContent = dr["Obstruction"].ToString();

if (\_cellInfo != null && !\_isChecked) //add the userName

{

if (cellContent == "")

{

dr["Obstruction"] = userName;

}

else

{

dr["Obstruction"] += "," + userName;

}

}

else

{

// remove username from the cell

cellContent = cellContent.Replace(userName, "");

//remove double and end commas

cellContent = Regex.Replace(cellContent, ",{2,}", ",").Trim(',');

dr["Obstruction"] = cellContent.Trim();

}

}

}

/// <summary>

/// the property pSHARE.OnOffButton is used to open or close the CSV display

/// </summary>

public string OnOffButton

{

get { return \_OnOffButton; }

set

{

\_OnOffButton = value;

//Raise a property changed notification to alert the UI that an element needs an update

//RaisePropertyChanged("NodeMessage");

}

}

/// DelegateCommand objects allow you to bind UI interaction to methods on your data context.

[IsVisibleInDynamoLibrary(false)]

public DelegateCommand OnOff { get; set; }

/// <summary>

/// Don't know why this is here. Maybe it was needed to hide a DelegateCommand

/// </summary>

/// <param name="workspace"></param>

[IsVisibleInDynamoLibrary(false)]

#endregion

#region constructor

/// The constructor for a NodeModel is used to create the input and output ports and specify the argument lacing.

public pSHARE()

{

InPortData.Add(new PortData("N", "Input (a List.CreateList) of pCOLLECT output(s)"));

InPortData.Add(new PortData("I", "Input a FilePath for the shared csv files."));

InPortData.Add(new PortData("L", "Input a FilePath for the local copy of the csv file."));

InPortData.Add(new PortData("U", "Input a the user namen (Code Block)."));

OutPortData.Add(new PortData("O", "Output of parameter name and value on next line; two by two."));

RegisterAllPorts();

ArgumentLacing = LacingStrategy.CrossProduct;

OnOff = new DelegateCommand(ShowParams, CanShowParams);

// update UI

OnOffButton = "Share";

}

#endregion

#region public methods

/// <summary>

/// If this method is not overriden, Dynamo will, by default

/// pass data through this node. So we should later use it to pass the OutputList.

/// </summary>

/// <param name="inputAstNodes"></param>

/// <returns></return

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

//When you run the graph and inputs have changed you get here, so add the input to the DataTable

//For this we first need to convert the inputs, however the Inputs can be of many types and it is

//hard to find out how to handle them. Mabe it is easier to use the inputAstNodes which is a List<AssociativeNode>

//In order to add this List to the CSV file we need to make that a List<AssociativeNode> too.

//For the output of pSHARE we only need the parameter names and values (if they are not obstructed).

//So we first have to work on the N input which consist of lines of headers and lines of corresponding

//values; two by two. It is inputAstNodes[0]. But no way to get the content

//Try use InPortData, which is an ObservableCollection<PortData>. Also no way to get the content

//Try use InPorts, which is an ObservableCollection<PortModel>. Also no content.

//Tuple<int,NodeModel> t1;

//this.TryGetInput(0, out t1);

//AssociativeNode N = inputAstNodes[0];

//PortData N = InPortData[0];

//PortModel N = InPorts[0];

//Try first make an AstNode from the inputs and then check the content. Also no way to get content.

//List<AssociativeNode> t1 = inputAstNodes;

//var t2 = AstFactory.BuildExprList(inputAstNodes);

//var t4 = new[] { AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes)), };

//IEnumerable<AssociativeNode> t3 = (IEnumerable<AssociativeNode>)AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes));

//get the file paths !!!!!!!!!!!

//object o = Inputs[1].Item2;

//System.Type testType = o.GetType();

//string typeName = testType.Name;

//string test;

// this works

//List<string> ls = new List<string> { "een", "twee", "drie" };

//List<object> test = new List<object> { ls, "a string", "another string", "last string" };

var t = new Func<List<string>, string, string, string, List<object>>(MyDataCollector.MyDataCollectorClass.pSHAREinputs);

//var t = new Func<ls, s1, s2, s3, List<object>>(MyDataCollector.MyDataCollectorClass.pSHAREinputs);

var funcNode = AstFactory.BuildFunctionCall(t, inputAstNodes);

//var funcNode = AstFactory.BuildFunctionCall(t, new List<AssociativeNode>() {StringNode});

//string fp = o.Value;

List<string> InputList = MakeInputList.InputList(InputNodes);

//now take out only the parameter names and on the next line the value

return new[] { AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), funcNode) };

//return new[] { AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(OutputList)), };

}

///// <summary>

///// the PropertyChange event is used to update the binding to CSV display

///// but it turns out that it was inherited by the NodeModel class

///// and you can simply use the RaisPropertyChanged method

///// </summary>

//public event PropertyChangedEventHandler PropertyChanged;

///// <summary>

///// the OnPropertyChanged interface is used to update the binding to CSV display

///// </summary>

///// <param name="propertyName"></param>

//protected void OnPropertyChanged(string propertyName)

//{

// if (PropertyChanged != null) // if there is any subscribers

// PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

//}

/// <summary>

/// opens the csv file, turns it into a dataTable, and shows the CSV display

/// </summary>

public void openCSV()

{

//make sure that the table doesn't exist. Otherwise just show it.

//however!!!!! if through the control it was changed...

if (myDataTable == null)

{

myDataTable = new DataTable();

try

{

StreamReader myStream = new StreamReader(inputFile);

string line = "";

int i = 0;

while (line != null)

{

line = myStream.ReadLine();

if (line == null)

{

break;

}

string[] words = line.Split(new Char[] { ';' });

if (i == 0) //this contains the headers. Also might be used to create properties for Parameter class.

{

// add a checbox column for easy setting obstruction field

myDataTable.Columns.Add("Accepted", typeof(bool));

foreach (var word in words)

{

//add a column for every header with (name, text)

myDataTable.Columns.Add(word, typeof(string));

}

}

else

{

//add a row to the datatable

DataRow row = myDataTable.NewRow();

int x = 1;

foreach (var word in words)

{

row[x] = word;

if (myDataTable.Columns.IndexOf("Obstruction") == x)

{

if (word == "")

{

row["Accepted"] = true;

}

else

{

row["Accepted"] = false;

}

}

x += 1;

}

myDataTable.Rows.Add(row);

}

i += 1;

}

}

catch (FileNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure it exists?");

}

catch (DirectoryNotFoundException)

{

MessageBox.Show("We couldn't find the file. Are you sure the directory exists?");

}

catch (Exception e)

{

MessageBox.Show(string.Format("We found a problem: {0}", e));//instance not set to a etc.

}

//make sure that control doesn't exist.

}

try

{

//check if the control exist already

bool isCSVControlOpen = false;

foreach (Window w in Application.Current.Windows)

{

if (w is CSVControl)

{

isCSVControlOpen = true;

w.Activate();

}

}

if (!isCSVControlOpen)

{

this.myPropDataTable = myDataTable;

CSVControl \_CSVControl = new CSVControl();

//bind the datatable to the xaml datagrid

\_CSVControl.myXamlTable.ItemsSource = this.myPropDataTable.DefaultView;

\_CSVControl.DataContext = this;

\_CSVControl.Show();

}

}

catch (System.Exception e)

{

MessageBox.Show("Exception source: {0}", e.Source);

}

}

public class pSHARENodeViewCustomization : INodeViewCustomization<pSHARE>

{

/// <summary>

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="model">The NodeModel representing the node's core logic.</param>

/// <param name="nodeView">The NodeView representing the node in the graph.</param>

//probably Dynamo has a method that makes it go here asa pSHARE is loaded

public void CustomizeView(pSHARE model, NodeView nodeView)

{

var pSHAREControl = new pSHAREcontrol();

nodeView.inputGrid.Children.Add(pSHAREControl);

pSHAREControl.DataContext = model;

}

/// <summary>

/// Here you can do any cleanup you require if you've assigned callbacks for particular

/// UI events on your node.

/// </summary>

public void Dispose() { }

}

/// <summary>

/// try to get Dynamo recalculate the solution when you hit On button

/// </summary>

/// <param name="actual"></param>

public void recalc(DynamoModel actual)

{

//if pSHARE is ON the solution should be recalculated. Doesn't acutally work.

actual.ResetEngine(true);

}

/// <summary>

/// close the CSV display

/// </summary>

public void closeCSVControl()

{

foreach (Window w in Application.Current.Windows)

{

if (w is CSVControl)

{

w.Close();

}

}

}

#endregion

#region command methods

private bool CanShowParams(object obj)

{

return true;

}

private void ShowParams(object obj)

{

//switch the On boolean to show or not the \*.csv file

if (On == false)

{

On = true;

//and show the \*.csv file

openCSV();

}

else

{

On = false;

//close \*.csv display

//myDataTable = null;

closeCSVControl();

//in order to not check the last selected row

//!!!!!!!!!!!!!still not right

if (\_rowIndex == 0)

{

\_rowIndex = 1;

}

else

{

\_rowIndex = 0;

}

}

}

private ICommand \_ShareClicked;

public ICommand ShareClicked

{

get

{

if (\_ShareClicked == null)

{

\_ShareClicked = new RelayCommand(

param => this.Share(),

param => this.CanShare()

);

}

return \_ShareClicked;

}

}

private bool CanShare()

{

return true;

}

private void Share()

{

MessageBox.Show("Share button is Clicked");

}

#endregion

}

}

Last pCOLLECT.cs before starting with GitHub repository pCOLAD for Dynamo

using System.Collections.Generic;

using System.Windows;

using Autodesk.DesignScript.Runtime;

using Dynamo.Controls;

using Dynamo.Models;

using Dynamo.UI;

using Dynamo.UI.Commands;

using ProtoCore.AST.AssociativeAST;

using Dynamo.Wpf;

using System;

using Dynamo.Nodes;

namespace pCOLADnamespace

{

/// <summary>

/// Dynamo uses the MVVM model of programming,

/// in which the UI is data-bound to the view model, which

/// exposes data from the underlying model. Custom UI nodes

/// are a hybrid because NodeModel objects already have an

/// associated NodeViewModel which you should never need to

/// edit. So here we will create a data binding between

/// properties on our class and our custom UI.

///

/// </summary>

///

// The NodeName attribute is what will display on

// top of the node in Dynamo

[NodeName("pCOLLECT")]

// The NodeCategory attribute determines how your

// node will be organized in the library. You can

// specify your own category or use one of the

// built-ins provided in BuiltInNodeCategories.

[NodeCategory("pCOLAD")]

// The description will display in the tooltip

// and in the help window for the node.

[NodeDescription("Collects parameters and their attributes for pSHARE.")]

[IsDesignScriptCompatible]

public class pCOLLECT : NodeModel

{

private List<string> \_outputListProp;

public List<string> outputListProp

{

get { return \_outputListProp; }

set { \_outputListProp = value; }

}

#region constructor

/// <summary>

/// The constructor for a NodeModel is used to create

/// the input and output ports and specify the argument

/// lacing.

/// </summary>

/// <param name="workspace"></param>

public pCOLLECT()

{

// When you create a UI node, you need to do the

// work of setting up the ports yourself. To do this,

// you can populate the InPortData and the OutPortData

// collections with PortData objects describing your ports.

InPortData.Add(new PortData("P", "Parameter name as int."));

InPortData.Add(new PortData("V", "Value as string."));

InPortData.Add(new PortData("I", "Importance as string."));

InPortData.Add(new PortData("C", "Comment as string."));

InPortData.Add(new PortData("O", "Owner as string."));

// Nodes can have an arbitrary number of inputs and outputs.

// If you want more ports, just create more PortData objects.

OutPortData.Add(new PortData("N", "List of strings."));

//OutPortData.Add(new PortData("some awesome", "A result."));

// This call is required to ensure that your ports are

// properly created.

RegisterAllPorts();

// The arugment lacing is the way in which Dynamo handles

// inputs of lists. If you don't want your node to

// support argument lacing, you can set this to LacingStrategy.Disabled.

//ArgumentLacing = LacingStrategy.CrossProduct;

ArgumentLacing = LacingStrategy.Disabled;

}

#endregion

#region public methods

/// <summary>

/// If this method is not overriden, Dynamo will, by default

/// pass data through this node. But we wouldn't be here if

/// we just wanted to pass data through the node, so let's

/// try using the data.

/// </summary>

/// <param name="inputAstNodes"></param>

/// <returns></returns>

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

// When you create your own UI node you are responsible

// for generating the abstract syntax tree (AST) nodes which

// specify what methods are called, or how your data is passed

// when execution occurs.

// WARNING!!!

// Do not throw an exception during AST creation. If you

// need to convey a failure of this node, then use

// AstFactory.BuildNullNode to pass out null.

// Make a list from the inputs, using the MakeInputList class.

// In fact not used in pCOLLECT but needed in pSHARE

List<string> InputList = MakeInputList.InputList(InputNodes);

\_outputListProp = InputList;

//Dynamo.CustomNodeDefinition pCOLLECTdefinition = (Dynamo.CustomNodeDefinition)InputList;

//List<string> InputList = new List<string>();

//for (int i = 0; i < Inputs.Count ; i++)

//{

// var item = Inputs[i];

// Dynamo.Nodes.CodeBlockNodeModel itemValue = (Dynamo.Nodes.CodeBlockNodeModel)item.Item2;

// string s = itemValue.Code;

// //for some reason Dynamo puts "\" and \";" around the string

// string sCleaned = s.Remove(s.Length - 2).Remove(0, 1);

// InputList.Add(sCleaned);

//}

// Using the AstFactory class, we can build AstNode objects

// that assign doubles, assign function calls, build expression lists, etc.

//build a new output List<AssociativeNode>consisting of fieldnames seperated by ';' and

// on next inputAstNodes with ';' added

List<AssociativeNode> pCOLLECTtempList = new List<AssociativeNode>();

// the headings should become flexible in future!!!!!!!!!!

var headings = AstFactory.BuildStringNode("Parameter;Value;Importance;Comments;Owner");

var semiColon = AstFactory.BuildStringNode(";");

foreach (AssociativeNode InputItem in inputAstNodes)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(InputItem);

arguments.Add(semiColon);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

//don't add ';' to the last one

if (inputAstNodes.IndexOf(InputItem) == inputAstNodes.Count - 1)

{

pCOLLECTtempList.Add(InputItem);

}

else

{

pCOLLECTtempList.Add(funcNode);

}

}

// now pCOLLECTtempList has the inputs followed by ';'

// but it should become one string so add the items together

List<AssociativeNode> pCOLLECToutputList = new List<AssociativeNode>();

AssociativeNode A = pCOLLECTtempList[0];

for (int i = 0; i < pCOLLECTtempList.Count - 1; i++)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(A);

arguments.Add(pCOLLECTtempList[i + 1]);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

A = funcNode;

}

pCOLLECToutputList.Add(A);

pCOLLECToutputList.Insert(0, headings);

//var test4 = TryGetInput(0, out System.Tuple < 0, NodeModel);

//var test3 = GetValue(0);

//string test = pCOLLECTtempList[0];

//also gives a temp and large number.........

// var test2 = AstFactory.BuildStringNode(pCOLLECTtempList[0].ToString()).value;

//var funcNode = AstFactory.BuildFunctionCall("%add", pCOLLECTtempList);

return new[]

{

// In these assignments, GetAstIdentifierForOutputIndex finds

// the unique identifier which represents an output on this node

// and 'assigns' that variable the expression that you create.

//// For the first node, we'll just pass through the

//// input provided to this node.

//AstFactory.BuildAssignment(

// GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes)),

// we output the headers and on next line the values seperated by ';'

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(pCOLLECToutputList)),

//AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), funcNode)

//// For the second node, we'll build a double node that

//// passes along our value for awesome.

//AstFactory.BuildAssignment(

// GetAstIdentifierForOutputIndex(1),

// AstFactory.BuildDoubleNode(awesome))

};

}

/// <summary>

/// View customizer for HelloDynamo Node Model.

/// </summary>

public class pCOLLECTNodeViewCustomization : INodeViewCustomization<pCOLLECT>

{

/// <summary>

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="model">The NodeModel representing the node's core logic.</param>

/// <param name="nodeView">The NodeView representing the node in the graph.</param>

public void CustomizeView(pCOLLECT model, NodeView nodeView)

{

// The view variable is a reference to the node's view.

// In the middle of the node is a grid called the InputGrid.

// We reccommend putting your custom UI in this grid, as it has

// been designed for this purpose.

// Create an instance of our custom UI class (defined in xaml),

// and put it into the input grid.

var \_pCOLLECTcontrol = new pCOLLECTcontrol();

nodeView.inputGrid.Children.Add(\_pCOLLECTcontrol);

// Set the data context for our control to be this class.

// Properties in this class which are data bound will raise

// property change notifications which will update the UI.

\_pCOLLECTcontrol.DataContext = model;

}

/// <summary>

/// Here you can do any cleanup you require if you've assigned callbacks for particular

/// UI events on your node.

/// </summary>

public void Dispose() { }

}

#endregion

#region command methods

//private bool CanShowMessage(object obj)

//{

// // I can't think of any reason you wouldn't want to say Hello Dynamo!

// // so I'll just return true.

// return true;

//}

//private void ShowMessage(object obj)

//{

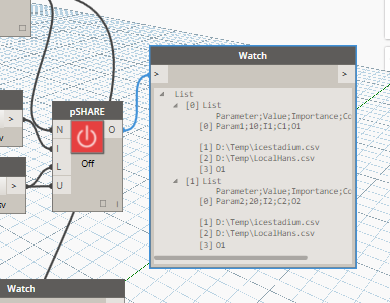
// MessageBox.Show("Hello Dynamo!");

//}

#endregion

}

}



In order to try [the package solution](https://github.com/DynamoDS/Dynamo/wiki/Building-a-Package-for-Dynamo-in-Visual-Studio) I tried to change the folder structure in the pCOLAD-for-Dynamo solution in VS. But that doesn't change it in the GitHub repository. And also not in the Windows File Explorer. So the folder structure should be changed there. But will VS then still find the right files? To be sure, make a copy first of the whole directory and then change it an see what VS can make of it. It took me one day and didn't succeed to get it working. The idea is to put some copy commands in the \*.proj files of VS using the path "DestinationFolder="$(SolutionDir)..\packages\MyDynamoPackage\bin\"

The SolutionDir is the folder where the \*.sln file is. The ..\ is a DOS command causing to go one directory up. So if the packages folder is in the same directory as the solution, leave out the ".." . However there is already such a folder with a folder inside that is called Microsoft.Bcl.Build.1.0.14. So change ..\packges etc. to \DynamoPackages etc.

To test the package on Dynamo, simply copy the MyDynamoPackage folder into your <user>\AppData\Roaming\Dynamo\0.x\packagesfolder and run Dynamo.

Doesn't work. Probably need a pkg.json file. Tried to make one on the basis of the Zip List package:

{"file\_hash":null,"name":"pCLOAD","version":"0.0.1","description":"Tools for on-line sharing of parameters.","group":"pCOLAD","keywords":["pCOLAD","share"],"dependencies":[],"license":"HH","contents":"pCOLAD - Tools for on-line sharing of parameters","engine\_version":"0.8.1.1823","engine\_metadata":"","engine":"dynamo"}

But now Dynamo even doesn't want to start. This was due to a lot of files in the package bin folder that should not be there. There was stll an error because of loading the assembly from MyDataCollector.dll, which of course was removed from the Dynamo nodes folder. Don't forget to remove the dll's from the Dynamo nodes folder and REM the after build commands. Now with the packages it works alright. However to automise the updating of the package it would be handy to copy also the package by code. Next lines at the end of the pCOLAD.csproj file does the thrick:

<Target Name="AfterBuild">

<ItemGroup>

<Dlls Include="$(OutDir)\*.dll" />

<Pdbs Include="$(OutDir)\*.pdb" />

<Xmls Include="$(OutDir)\*.xml" />

<Configs Include="$(OutDir)\*.config" />

<Pack Include="$(SolutionDir)\DynamoPackages\pCOLADpackage\\*\*\\*.\*" />

</ItemGroup>

<Copy SourceFiles="@(Dlls)" DestinationFolder="$(SolutionDir)\DynamoPackages\pCOLADpackage\bin\" />

<Copy SourceFiles="@(Pdbs)" DestinationFolder="$(SolutionDir)\DynamoPackages\pCOLADpackage\bin\" />

<Copy SourceFiles="@(Xmls)" DestinationFolder="$(SolutionDir)\DynamoPackages\pCOLADpackage\bin\" />

<Copy SourceFiles="@(Configs)" DestinationFolder="$(SolutionDir)\DynamoPackages\pCOLADpackage\bin\" />

<Copy SourceFiles="@(Pack)" DestinationFolder="C:\Users\jhubers\AppData\Roaming\Dynamo\0.8\packages\pCOLADpackage\%(RecursiveDir)" />

<MakeDir Directories="$(SolutionDir)\DynamoPackages\pCOLADpackage\dyf" />

<MakeDir Directories="$(SolutionDir)\DynamoPackages\pCOLADpackage\extra" />

</Target>

</Project>

Now we have to find a way to use the result of the Func method.

The resulting DataTable is stored in a property of MyDataCollector class. However it is only there accessable during the creation of the pSHARE output node, which is the last thing you can do with pSHARE. However, since we have to use the on/off button to show the csv+parameters display, we can then acces it there.

//is there a problem if a cell is empty????

//loop through all cells of a datatable and check for null

foreach (DataRow ro in newParamTable.Rows)

{

foreach (DataColumn col in newParamTable.Columns)

{

if (ro[col.ColumnName] == DBNull.Value)

{

ro[col.ColumnName] = "xyz";

}

}

}

//In order to merge tables they should have same starting columns? No, problema was about different primaryKeyColomns

//CSV: Accepted;Comments;Parameter;New Value;Obstruction;Old Value;Owner;Importance;Date;Author

//pCOLLECT: Accepted;Parameter;Value;Importance;Comments;Owner

newParamTable.Columns.Add("Obstruction", typeof(string));

newParamTable.Columns.Add("Old Value", typeof(string));

newParamTable.Columns.Add("Date", typeof(string));

newParamTable.Columns.Add("Author", typeof(string));

newParamTable.Columns["Value"].ColumnName = "New Value";

//0 2 3 7 1 6 4 5 8 9

//now it is: Accepted;Parameter;New Value;Importance;Comments;Owner;Obstruction;Old Value;Date;Author

//0 1 2 3 4 5 6 7 8 9

//shoud become: Accepted;Comments;Parameter;New Value;Obstruction;Old Value;Owner;Importance;Date;Author

newParamTable.Columns["Comments"].SetOrdinal(1);

newParamTable.Columns["Owner"].SetOrdinal(7);

newParamTable.Columns["Importance"].SetOrdinal(7);

Good trick: CTRL+K and then S to surround the selection with a code snippet

ItemsSource="{Binding myPropDataTable, Mode=TwoWay}"

Seems a problem is that using a static class you cannot implement the INotifyPropertyChanged interface, because that needs an instance. The problem is that we then can not RaisePropertyChanged("MyPropDataTable") to notify the CSVcontrol that it should update changes form myDataTable. E.g. after changing inputs of pSHARE or changes in the csv file. The way to go would be to use a singleton? But I don't know if that would work with the

Func<List<List<string>>, string, string, string, List<string>>(MyDataCollectorClass.pSHAREinputs).

Solved it by adding a UpdateCSVControl event.

24 aug. 15

After 4 weeks of holiday trying to pick-up where I left. It shows that displaying and editing the csv-file looks ok. But when I turn on automatic running in Dynamo, I get an error in the MakeInputList script (but I was using an older version of pCOLAD!). This was anyway not the right way to make it, so start with replacing this script. For the record, it was:

//This method is used by all pCOLAD nodes.

//pCOLLECT uses it to build an outputListProp which is used by pSHARE.

//pSHARE uses it to build an input list from a (list of) PCOLLECT's output(s). NOT

//pPARAM uses it to build an input list of the output of pSHARE

using System.Collections.Generic;

using Dynamo.Models;

using System.Windows;

using ProtoCore.AST.AssociativeAST;

using System.Collections.ObjectModel;

namespace pCOLADnamespace

{

public static class MakeInputList

{

public static List<string> InputList(IDictionary<int, System.Tuple<int, NodeModel>> \_Inputs)

{

List<string> ReturnList = new List<string>();

for (int i = 0; i < \_Inputs.Count; i++)

{

//the Inputs object is a Dictionary< int, System.Tuple<int,NodeModel>>

System.Tuple<int, NodeModel> item = \_Inputs[i];

NodeModel nm = item.Item2;

System.Type testType = nm.GetType();

string typeName = testType.Name;

switch (typeName)

{

case "CreateList":

IDictionary<int, System.Tuple<int, NodeModel>> item1 = nm.InputNodes;

List<string> l1 = MakeInputList.InputList(item1); ;

ReturnList.AddRange(l1);

break;

case "pCOLLECT":

pCOLLECT pC = (pCOLLECT)item.Item2;

List<string> l2 = pC.outputListProp;

ReturnList.AddRange(l2);

break;

case "Filename":

//we do nothing with the filename; catch it with the inputAstNodes!!!have to work on this

// no have to open the file in code so need the path

break;

case "CodeBlockNodeModel":

// If the Inputs are from pCOLLECT then this is OK

// If the Inputs are from pSHARE you should make it recognizable in the ReturnList

// so you can make a difference with the inputs from pCOLLECTs!!!

// or maybe make a case for the paths?

Dynamo.Nodes.CodeBlockNodeModel cbnm = (Dynamo.Nodes.CodeBlockNodeModel)nm;

string s = cbnm.Code;

if (s.Length > 1 && s[0] == '\"')

{

//for some reason Dynamo puts "\" and \";" around the string

string sCleaned = s.Remove(s.Length - 2).Remove(0, 1);

ReturnList.Add(sCleaned);

}

else

{

// If it is a number Dynamo puts ; behind it.

ReturnList.Add(s.Remove(s.Length - 1));

}

break;

default:

MessageBox.Show("Please connect only output of pCOLLECT or List.CreateList to N input of pSHARE.");

ReturnList.Clear();

break;

}

//try

//{

// Dynamo.Nodes.CodeBlockNodeModel itemValue = (Dynamo.Nodes.CodeBlockNodeModel)item.Item2;

// string s = itemValue.Code;

// if (s.Length > 1 && s[0] == '\"')

// {

// //for some reason Dynamo puts "\" and \";" around the string

// string sCleaned = s.Remove(s.Length - 2).Remove(0, 1);

// ReturnList.Add(sCleaned);

// }

// else

// {

// // If it is a number Dynamo puts ; behind it.

// ReturnList.Add(s.Remove(s.Length-1));

// }

//}

//catch (System.Exception)

//{

// MessageBox.Show("Please connect only strings or numbers.");

// ReturnList.Clear();

// return ReturnList;

//}

}

return ReturnList;

}

}

}

# Make a variable input node

But it should become a dynamic node, meaning that you can add or delete inputs, like the List.Create node. The script of which in version 8.1 is:

using System.Collections.Generic;

using System.Linq;

using Dynamo.Models;

using Dynamo.Nodes;

using DSCoreNodesUI.Properties;

using ProtoCore.AST.AssociativeAST;

namespace DSCoreNodesUI

{

[NodeName("List.Create")]

[NodeDescription("ListCreateDescription", typeof(DSCoreNodesUI.Properties.Resources))]

[NodeSearchTags("ListCreateSearchTags", typeof(DSCoreNodesUI.Properties.Resources))]

[NodeCategory(BuiltinNodeCategories.CORE\_LISTS\_CREATE)]

[IsDesignScriptCompatible]

public class CreateList : VariableInputNode

{

public CreateList()

{

InPortData.Add(new PortData("index0", Resources.CreateListPortDataIndex0ToolTip));

OutPortData.Add(new PortData("list", Resources.CreateListPortDataResultToolTip));

RegisterAllPorts();

ArgumentLacing = LacingStrategy.Disabled;

}

protected override string GetInputName(int index)

{

return "index" + index;

}

protected override string GetInputTooltip(int index)

{

return string.Format(Resources.ListCreateInPortToolTip, index);

}

protected override void RemoveInput()

{

if (InPortData.Count > 1)

base.RemoveInput();

}

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

if (IsPartiallyApplied)

{

var connectedInput = Enumerable.Range(0, InPortData.Count)

.Where(HasConnectedInput)

.Select(x => new IntNode(x) as AssociativeNode)

.ToList();

var paramNumNode = new IntNode(InPortData.Count);

var positionNode = AstFactory.BuildExprList(connectedInput);

var arguments = AstFactory.BuildExprList(inputAstNodes);

var functionNode = new IdentifierListNode

{

LeftNode = new IdentifierNode("DSCore.List"),

RightNode = new IdentifierNode("\_\_Create")

};

var inputParams = new List<AssociativeNode>

{

functionNode,

paramNumNode,

positionNode,

arguments,

AstFactory.BuildBooleanNode(false)

};

return new[]

{

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0),

AstFactory.BuildFunctionCall("\_SingleFunctionObject", inputParams))

};

}

return new[]

{

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0),

AstFactory.BuildExprList(inputAstNodes))

};

}

}

}

The pCOLLECT2.cs was:

using System.Collections.Generic;

using System.Windows;

using Autodesk.DesignScript.Runtime;

using Dynamo.Controls;

using Dynamo.Models;

using Dynamo.UI;

using Dynamo.UI.Commands;

using ProtoCore.AST.AssociativeAST;

using Dynamo.Wpf;

using System;

using Dynamo.Nodes;

using pCOLADnamespace;

namespace pCOLADnamespace

{

/// <summary>

/// Dynamo uses the MVVM model of programming,

/// in which the UI is data-bound to the view model, which

/// exposes data from the underlying model. Custom UI nodes

/// are a hybrid because NodeModel objects already have an

/// associated NodeViewModel which you should never need to

/// edit. So here we will create a data binding between

/// properties on our class and our custom UI.

///

/// </summary>

///

// The NodeName attribute is what will display on

// top of the node in Dynamo

[NodeName("pCOLLECT")]

// The NodeCategory attribute determines how your

// node will be organized in the library. You can

// specify your own category or use one of the

// built-ins provided in BuiltInNodeCategories.

[NodeCategory("pCOLAD")]

// The description will display in the tooltip

// and in the help window for the node.

[NodeDescription("Collects parameters and their attributes for pSHARE.")]

[IsDesignScriptCompatible]

public class pCOLLECT : VariableInputNode//NodeModel

{

private List<string> \_outputListProp;

public List<string> outputListProp

{

get { return \_outputListProp; }

set { \_outputListProp = value; }

}

#region constructor

/// <summary>

/// The constructor for a NodeModel is used to create

/// the input and output ports and specify the argument

/// lacing.

/// </summary>

/// <param name="workspace"></param>

public pCOLLECT()

{

// When you create a UI node, you need to do the

// work of setting up the ports yourself. To do this,

// you can populate the InPortData and the OutPortData

// collections with PortData objects describing your ports.

InPortData.Add(new PortData("P", "Parameter name as int."));

InPortData.Add(new PortData("V", "Value as string."));

InPortData.Add(new PortData("I", "Importance as string."));

InPortData.Add(new PortData("C", "Comment as string."));

InPortData.Add(new PortData("O", "Owner as string."));

// Nodes can have an arbitrary number of inputs and outputs.

// If you want more ports, just create more PortData objects.

OutPortData.Add(new PortData("N", "List of strings."));

//OutPortData.Add(new PortData("some awesome", "A result."));

// This call is required to ensure that your ports are

// properly created.

RegisterAllPorts();

// The arugment lacing is the way in which Dynamo handles

// inputs of lists. If you don't want your node to

// support argument lacing, you can set this to LacingStrategy.Disabled.

//ArgumentLacing = LacingStrategy.CrossProduct;

ArgumentLacing = LacingStrategy.Disabled;

}

#endregion

#region public methods

/// <summary>

/// If this method is not overriden, Dynamo will, by default

/// pass data through this node. But we wouldn't be here if

/// we just wanted to pass data through the node, so let's

/// try using the data.

/// </summary>

/// <param name="inputAstNodes"></param>

/// <returns></returns>

[IsVisibleInDynamoLibrary(false)]

public override IEnumerable<AssociativeNode> BuildOutputAst(List<AssociativeNode> inputAstNodes)

{

// When you create your own UI node you are responsible

// for generating the abstract syntax tree (AST) nodes which

// specify what methods are called, or how your data is passed

// when execution occurs.

// WARNING!!!

// Do not throw an exception during AST creation. If you

// need to convey a failure of this node, then use

// AstFactory.BuildNullNode to pass out null.

// Make a list from the inputs, using the MakeInputList class.

// In fact not used in pCOLLECT but needed in pSHARE

List<string> InputList = MakeInputList.InputList(InputNodes);

\_outputListProp = InputList;

//Dynamo.CustomNodeDefinition pCOLLECTdefinition = (Dynamo.CustomNodeDefinition)InputList;

//List<string> InputList = new List<string>();

//for (int i = 0; i < Inputs.Count ; i++)

//{

// var item = Inputs[i];

// Dynamo.Nodes.CodeBlockNodeModel itemValue = (Dynamo.Nodes.CodeBlockNodeModel)item.Item2;

// string s = itemValue.Code;

// //for some reason Dynamo puts "\" and \";" around the string

// string sCleaned = s.Remove(s.Length - 2).Remove(0, 1);

// InputList.Add(sCleaned);

//}

// Using the AstFactory class, we can build AstNode objects

// that assign doubles, assign function calls, build expression lists, etc.

//build a new output List<AssociativeNode>consisting of fieldnames seperated by ';' and

// on next inputAstNodes with ';' added

List<AssociativeNode> pCOLLECTtempList = new List<AssociativeNode>();

// the headings should become flexible in future!!!

// also use the creation of output similar to pSHARE and pPARAM!!!

var headings = AstFactory.BuildStringNode("Parameter;Value;Importance;Comments;Owner");

var semiColon = AstFactory.BuildStringNode(";");

foreach (AssociativeNode InputItem in inputAstNodes)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(InputItem);

arguments.Add(semiColon);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

//don't add ';' to the last one

if (inputAstNodes.IndexOf(InputItem) == inputAstNodes.Count - 1)

{

pCOLLECTtempList.Add(InputItem);

}

else

{

pCOLLECTtempList.Add(funcNode);

}

}

// now pCOLLECTtempList has the inputs followed by ';'

// but it should become one string so add the items together

List<AssociativeNode> pCOLLECToutputList = new List<AssociativeNode>();

AssociativeNode A = pCOLLECTtempList[0];

for (int i = 0; i < pCOLLECTtempList.Count - 1; i++)

{

List<AssociativeNode> arguments = new List<AssociativeNode>();

arguments.Add(A);

arguments.Add(pCOLLECTtempList[i + 1]);

var funcNode = AstFactory.BuildFunctionCall("%add", arguments);

A = funcNode;

}

pCOLLECToutputList.Add(A);

pCOLLECToutputList.Insert(0, headings);

//var test4 = TryGetInput(0, out System.Tuple < 0, NodeModel);

//var test3 = GetValue(0);

//string test = pCOLLECTtempList[0];

//also gives a temp and large number.........

// var test2 = AstFactory.BuildStringNode(pCOLLECTtempList[0].ToString()).value;

//var funcNode = AstFactory.BuildFunctionCall("%add", pCOLLECTtempList);

return new[]

{

// In these assignments, GetAstIdentifierForOutputIndex finds

// the unique identifier which represents an output on this node

// and 'assigns' that variable the expression that you create.

//// For the first node, we'll just pass through the

//// input provided to this node.

//AstFactory.BuildAssignment(

// GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(inputAstNodes)),

// we output the headers and on next line the values seperated by ';'

//Have to find a way to make it a dynamic node, meaning add or delete inputs!!!

//ProtoCore.AST.AssociativeAST.DynamicNode dn= new DynamicNode();

AstFactory.BuildAssignment(

GetAstIdentifierForOutputIndex(0), AstFactory.BuildExprList(pCOLLECToutputList)),

//AstFactory.BuildAssignment(GetAstIdentifierForOutputIndex(0), funcNode)

//// For the second node, we'll build a double node that

//// passes along our value for awesome.

//AstFactory.BuildAssignment(

// GetAstIdentifierForOutputIndex(1),

// AstFactory.BuildDoubleNode(awesome))

};

}

/// <summary>

/// View customizer for HelloDynamo Node Model.

/// </summary>

public class pCOLLECTNodeViewCustomization : INodeViewCustomization<pCOLLECT>

{

/// <summary>

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="model">The NodeModel representing the node's core logic.</param>

/// <param name="nodeView">The NodeView representing the node in the graph.</param>

public void CustomizeView(pCOLLECT model, NodeView nodeView)

{

// The view variable is a reference to the node's view.

// In the middle of the node is a grid called the InputGrid.

// We reccommend putting your custom UI in this grid, as it has

// been designed for this purpose.

// Create an instance of our custom UI class (defined in xaml),

// and put it into the input grid.

var \_pCOLLECTcontrol = new pCOLLECTcontrol();

nodeView.inputGrid.Children.Add(\_pCOLLECTcontrol);

// Set the data context for our control to be this class.

// Properties in this class which are data bound will raise

// property change notifications which will update the UI.

\_pCOLLECTcontrol.DataContext = model;

}

/// <summary>

/// Here you can do any cleanup you require if you've assigned callbacks for particular

/// UI events on your node.

/// </summary>

public void Dispose() { }

}

#endregion

#region command methods

//private bool CanShowMessage(object obj)

//{

// // I can't think of any reason you wouldn't want to say Hello Dynamo!

// // so I'll just return true.

// return true;

//}

//private void ShowMessage(object obj)

//{

// MessageBox.Show("Hello Dynamo!");

//}

#endregion

}

}

Now first check if there is a possibility to work with an unknown number of parameters when constructing the output.

The basic idea can be found [here](https://msdn.microsoft.com/en-us/library/ms228391%28v=vs.90%29.aspx?f=255&MSPPError=-2147217396). It explains how to make a method with unknown parameters. Adapted a little for using a List:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication1

{

class Program

{

private static void Average(string title, params int[] values)

{

int sum = 0;

System.Console.Write("Average of {0} (", title);

for (int i = 0; i < values.Length; i++)

{

sum += values[i];

System.Console.Write(values[i] + ", ");

}

System.Console.WriteLine("): {0}", (float)sum / values.Length);

}

static void Main()

{

List<int> li1 = new List<int> { 5, 10, 15 };

int[] l1 = li1.ToArray();

//int[] l1 = new int[] { 5, 10, 15 };

int[] l2 = new int[] { 5, 10, 15, 20, 25, 30 };

Average("List One", l1);

Average("List Two", l2);

}

}

}

Now if you can make a method with unknown number of parameters. You can make a delegate with unknown number of parameters, but infortunately you can not make a Func delegate. Also the BuildFunctionCall, used to make an Associative node only accepts 27 overloads with Actions or Funcs using upto 11 parameters. I could then of course make a construct with cases depending on the number of parameters… First have a look again at the Func delegate. [Here](http://www.codeproject.com/Tips/771746/Uses-of-Func-in-Csharp) it says:

Func in short is parameterized delegate. In C#, a delegate instance points towards a method. When a caller invokes the delegate, it calls its target method. This way, the caller is not invoking the target method rather invoking the delegate which can call the target method. We do it because it creates an abstraction on invoking the target method. We of course always can invoke a method directly but decoupling of the client and target method is sometimes a need or gives us more flexibility to make things clean and simple.

We can use Func delegate to represent a method that can be passed as a parameter without explicitly declaring a custom delegate.

Why I said it's a parameterized delegate:

Hide   Copy Code

delegate TResult Func <out TResult> ();

delegate TResult Func <in T, out TResult> (T arg);

delegate TResult Func <in T1, in T2, out TResult> (T1 arg1, T2 arg2);

... and so on, up to T16

[Here](http://simpleprogrammer.com/2010/09/24/explaining-what-action-and-func-are/) it is explained even better.

Also check [this](http://stackoverflow.com/questions/290061/what-does-the-syntax-in-c-sharp-mean) out. It explains how Lambda expressions are equivalent to anonymous expressions.

According to Yu Ke we can use a function with indetermined parameters instead of the Func<…>. He writes:

[**@jhubers**](https://github.com/jhubers) , to do that, you need a c# function which accepts any input, e.g.,

class MyClass

{

public static int Sum(params int[] vs)

{

int sum = 0;

foreach (int v in vs)

sum += v;

return sum;

}

}

And create a variable input UI node, which could derive from VariableInputNode

And:

[**@jhubers**](https://github.com/jhubers) , you don't need to use Func<...> stuff, it just helps to automatically figure out its class name and function name (AstFactory.BuildFunctionCall() has a bunch of overloaded version, the basic two are AssociativeNode BuildFunctionCall(string className, string functionName, List<AssociativeNode> arguments, Core core = null) and AssociativeNode BuildFunctionCall(string functionName, List<AssociativeNode> arguments, Core core = null). You could explicitly do that:

var funcNode = AstFactory.BuildFunctionCall("MyClass", "pCollect", inputAstNodes);

So I tried: public static List<string> pCOLLECToutputs(params string[] ss).

But it gives errors, stating that the method or constructor is not found.

Now in pSHARE when you use var t = new Func<List<List<string>>, string, string, string, List<string>>(MyDataCollectorClass.pSHAREinputs); in fact you create a delegate t (an object type) that during execution returns a List<string> (that is the yellow highlited parameter). As you can see above the part between () are the arguments of the Func delegate = the parameters, that should have the same number and type as the Func<types>. In this case MyDataCollectorClass.pSHAREinputs, which is the static function public static List<string> pSHAREinputs(List<List<string>> \_Ninputs, string \_IfilePath, string \_LfilePath, string \_owner).

The thing is that \_Ninputs contains the first input of the pSHARE node (which is a List of several pCOLLECT outputs, so a List<List<string>>. But if we use a function with undetermined number of parameters, how do we get the input ToolTips (that is where the names are stored).

# Add the Histroy button that hides the others if clicked and shows the history of csv changes.

I wanted to change the background color of the History button. In fact it should be a Toggle Button. When you press it it should stay checked and get a nice green color. Also the other buttons should dissapear. To do this with XAML is incredable difficult. It turns out that you will have to override the default ControlTemplate. Best is to start with the default ControlTemplate and change some settings. You can find the default templates of controles [here](https://msdn.microsoft.com/fr-fr/library/cc278075(v=vs.95).aspx).

You can put the ControlTemplate in several places, but most used is in a Style. For this you first have to make a resource in the part of the control that you want to use the ControlTemplate. E.g. if you have your buttons at the bottom of a window and you use a grid you have to put in the following.  
<Grid VerticalAlignment="Bottom" Height="25">

<Grid.Resources>

<Style TargetType="Button" x:Key="newTemplate">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<!--Define the ControlTemplate here.-->

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Grid.Resources>

</Grid>

The default ControlTemplate of a button is:

<Style TargetType="Button">

<Setter Property="Background" Value="#FF1F3B53"/>

<Setter Property="Foreground" Value="#FF000000"/>

<Setter Property="Padding" Value="3"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="BorderBrush">

<Setter.Value>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="#FFA3AEB9" Offset="0"/>

<GradientStop Color="#FF8399A9" Offset="0.375"/>

<GradientStop Color="#FF718597" Offset="0.375"/>

<GradientStop Color="#FF617584" Offset="1"/>

</LinearGradientBrush>

</Setter.Value>

</Setter>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Grid>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="Normal"/>

<VisualState x:Name="MouseOver">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="BackgroundAnimation" Storyboard.TargetProperty="Opacity" To="1"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#F2FFFFFF"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#CCFFFFFF"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#7FFFFFFF"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Pressed">

<Storyboard>

<ColorAnimation Duration="0" Storyboard.TargetName="Background" Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Color)" To="#FF6DBDD1"/>

<DoubleAnimation Duration="0" Storyboard.TargetName="BackgroundAnimation" Storyboard.TargetProperty="Opacity" To="1"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#D8FFFFFF"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#C6FFFFFF"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#8CFFFFFF"/>

<ColorAnimation Duration="0" Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="(Rectangle.Fill).(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#3FFFFFFF"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Disabled">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="DisabledVisualElement" Storyboard.TargetProperty="Opacity" To=".55"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="FocusStates">

<VisualState x:Name="Focused">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="FocusVisualElement" Storyboard.TargetProperty="Opacity" To="1"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Unfocused" />

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Border x:Name="Background" CornerRadius="3" Background="White" BorderThickness="{TemplateBinding BorderThickness}" BorderBrush="{TemplateBinding BorderBrush}">

<Grid Background="{TemplateBinding Background}" Margin="1">

<Border Opacity="0" x:Name="BackgroundAnimation" Background="#FF448DCA" />

<Rectangle x:Name="BackgroundGradient" >

<Rectangle.Fill>

<LinearGradientBrush StartPoint=".7,0" EndPoint=".7,1">

<GradientStop Color="#FFFFFFFF" Offset="0" />

<GradientStop Color="#F9FFFFFF" Offset="0.375" />

<GradientStop Color="#E5FFFFFF" Offset="0.625" />

<GradientStop Color="#C6FFFFFF" Offset="1" />

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

</Grid>

</Border>

<ContentPresenter

x:Name="contentPresenter"

Content="{TemplateBinding Content}"

ContentTemplate="{TemplateBinding ContentTemplate}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

Margin="{TemplateBinding Padding}"/>

<Rectangle x:Name="DisabledVisualElement" RadiusX="3" RadiusY="3" Fill="#FFFFFFFF" Opacity="0" IsHitTestVisible="false" />

<Rectangle x:Name="FocusVisualElement" RadiusX="2" RadiusY="2" Margin="1" Stroke="#FF6DBDD1" StrokeThickness="1" Opacity="0" IsHitTestVisible="false" />

</Grid>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

But if you only want to change e.g. the border you can use:

<ControlTemplate TargetType="Button">

<Border x:Name="RootElement">

<Border.Background>

<SolidColorBrush x:Name="BorderBrush" Color="Black"/>

</Border.Background>

<!--Create a border that has a different color by adding smaller grid.

The background of this grid is specificied by the button's Background

property.-->

<Grid Margin="4" Background="{TemplateBinding Background}">

<!--Use a ContentPresenter to display the Content of

the Button.-->

<ContentPresenter

HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

Margin="4,5,4,4" />

</Grid>

</Border>

</ControlTemplate>

And then apply this to buttons as follows:

<Button Style="{StaticResource newTemplate}"

Background="Navy" Foreground="White" FontSize="14"

Content="Button1"/>

<Button Style="{StaticResource newTemplate}"

Background="Purple" Foreground="White" FontSize="14"

Content="Button2" Click="Button\_Click"/>

If you want to change the colour of a togglebutton (replace the <Button…\> in the code with <ToggleButton …\>)best is to use Blend: Right click on the xaml of the control in project browser and choose Open in Blend (after saving your project). In Blend right click the toggle button and choose Edit Template. If you want to use the new style you are going to make for several toggle buttons then select Edit a Copy… and define it in a Window resource. If you only want to use the style for this toggle button then select Edit current… Or if you want to preserve the default style but still want to only define the style for this button, then choose Edit a Copy… and define it in ToggleButton: [name].

I wanted to use the default style of other buttons (no need to have different normal style for a ToggleButton than for a normal Button – only the checked state should be different). To do this I first made a normal Button, then choose Edit Template/Edit a Copy… and then right click on the button and choose View Source. Then in the code of this Button change everywhere "Button" into "ToggleButton".

Now in the States Tab choose Checked and you will see red dots on several places. This means that Blend is recording the changes you are going to make. You want to change the BackGround and the BackGroundAnimation and the BackGroundGradient. In the right pane set for all of these properties the color to the one you like. Save your work. Go back to Visual Studio and accept the changes.

The code of my ToggleButton now is:

<ToggleButton x:Name="History" Grid.Column="2" Content="History" Command="{Binding HistoryCommand}"

VerticalAlignment="Bottom" Margin="252,0,91,0" Style="{DynamicResource MyToggleButtonStyle}">

<ToggleButton.Resources>

<Style x:Key="MyToggleButtonStyle" TargetType="{x:Type ToggleButton}">

<Setter Property="Background" Value="#FF1F3B53"/>

<Setter Property="Foreground" Value="#FF000000"/>

<Setter Property="Padding" Value="3"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="BorderBrush">

<Setter.Value>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="#FFA3AEB9" Offset="0"/>

<GradientStop Color="#FF8399A9" Offset="0.375"/>

<GradientStop Color="#FF718597" Offset="0.375"/>

<GradientStop Color="#FF617584" Offset="1"/>

</LinearGradientBrush>

</Setter.Value>

</Setter>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type ToggleButton}">

<Grid>

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="Normal"/>

<VisualState x:Name="MouseOver">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="BackgroundAnimation" Storyboard.TargetProperty="Opacity" To="1"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Pressed">

<Storyboard>

<ColorAnimation Duration="0" Storyboard.TargetName="Background" Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Color)" To="#FF6DBDD1"/>

<DoubleAnimation Duration="0" Storyboard.TargetName="BackgroundAnimation" Storyboard.TargetProperty="Opacity" To="1"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Disabled">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="DisabledVisualElement" Storyboard.TargetProperty="Opacity" To=".55"/>

</Storyboard>

</VisualState>

</VisualStateGroup>

<VisualStateGroup x:Name="FocusStates">

<VisualState x:Name="Focused">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="FocusVisualElement" Storyboard.TargetProperty="Opacity" To="1"/>

</Storyboard>

</VisualState>

<VisualState x:Name="Unfocused" />

</VisualStateGroup>

<VisualStateGroup x:Name="CheckStates">

<VisualState x:Name="Checked">

<Storyboard>

<ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)" Storyboard.TargetName="Background">

<EasingColorKeyFrame KeyTime="0" Value="#FF14F082"/>

</ColorAnimationUsingKeyFrames>

<ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(SolidColorBrush.Color)" Storyboard.TargetName="BackgroundAnimation">

<EasingColorKeyFrame KeyTime="0" Value="#FF14F082"/>

</ColorAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill)" Storyboard.TargetName="BackgroundGradient">

<DiscreteObjectKeyFrame KeyTime="0">

<DiscreteObjectKeyFrame.Value>

<SolidColorBrush Color="#FF14F082"/>

</DiscreteObjectKeyFrame.Value>

</DiscreteObjectKeyFrame>

</ObjectAnimationUsingKeyFrames>

</Storyboard>

</VisualState>

<VisualState x:Name="Unchecked"/>

<VisualState x:Name="Indeterminate"/>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Border x:Name="Background" CornerRadius="3" Background="White" BorderThickness="{TemplateBinding BorderThickness}" BorderBrush="{TemplateBinding BorderBrush}">

<Grid Background="{TemplateBinding Background}" Margin="1">

<Border Opacity="0" x:Name="BackgroundAnimation" Background="#FF448DCA" />

<Rectangle x:Name="BackgroundGradient" >

<Rectangle.Fill>

<LinearGradientBrush StartPoint=".7,0" EndPoint=".7,1">

<GradientStop Color="#FFFFFFFF" Offset="0" />

<GradientStop Color="#F9FFFFFF" Offset="0.375" />

<GradientStop Color="#E5FFFFFF" Offset="0.625" />

<GradientStop Color="#C6FFFFFF" Offset="1" />

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

</Grid>

</Border>

<ContentPresenter

x:Name="contentPresenter"

Content="{TemplateBinding Content}"

ContentTemplate="{TemplateBinding ContentTemplate}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

Margin="{TemplateBinding Padding}"/>

<Rectangle x:Name="DisabledVisualElement" RadiusX="3" RadiusY="3" Fill="#FFFFFFFF" Opacity="0" IsHitTestVisible="false" />

<Rectangle x:Name="FocusVisualElement" RadiusX="2" RadiusY="2" Margin="1" Stroke="#FF6DBDD1" StrokeThickness="1" Opacity="0" IsHitTestVisible="false" />

</Grid>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</ToggleButton.Resources>

</ToggleButton>

Now it still has to set the visibility of the other buttons… For this you can use the BooleanToVisibility item that you can set in a Window.Resource. But standard it works the wrong way around. Only the path=IsChecked can be used. So you have to build your own BooleanToVisibility code. I have put it in pCOLADnamespace. In order to use it you have to make a xmlns: entry

xmlns:myBoolConverter="clr-namespace:pCOLADnamespace"

And then in the Windows.Resources make an instance of the class:

<Window.Resources>

<myBoolConverter:BoolToVisibilityConverter x:Key="BoolToVisibleConverter"

FalseValue= "Visible"

TrueValue="Hidden" />

The code:

[ValueConversion(typeof(bool), typeof(Visibility))]

public class BoolToVisibilityConverter : IValueConverter

{

public Visibility TrueValue { get; set; }

public Visibility FalseValue { get; set; }

public BoolToVisibilityConverter()

{

// set defaults

FalseValue = Visibility.Hidden;

TrueValue = Visibility.Visible;

}

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

return (bool)value ? TrueValue : FalseValue;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

# Add CheckAll and UncheckAll buttons

Next issue was how to use the CheckAll and UncheckAll buttons. Finaly solved it wit DelegateCommands setting the CheckAllButton and UnCheckAllButton properties and using that in the isChecked property which is bound to the checkbox. You call this procedure by simply setting this.isChecked = false or true, while setting this.\_isChecked property to the right value.

# Changes in red

More complex is how to set the changed values in the csv to red colour. Interesting reference is found [here](https://msdn.microsoft.com/en-us/library/cc903950%28v=vs.95%29.aspx?f=255&MSPPError=-2147217396). We want to show in red changes in the comments, the new value, the importance. So we will have to replace the autogenerated columns by column templates with cell templates that are bound to a property that shows which cells have changed compared to the copy of the csv file.

In order to find errors in binding your control to your viewmodel read [this](http://blogs.msdn.com/b/mikehillberg/archive/2006/09/14/wpftracesources.aspx) (but doesn't work with attached processes debugging). Anyway the problem became that you can bind to a static class, but then all the textblocks text will be the same value, because a static class can not implement INotifyPropertyChanged and so you can not use RaisePropertyChanged.

xmlns:myForeGroundConverter ="clr-namespace:MyDataCollector;assembly=MyDataCollector"

<DataTemplate x:Key="changedComments">

<TextBlock

Text="{Binding Source={x:Static myForeGroundConverter:MyDataCollectorClass.Comments}}"

Foreground="{Binding Source={x:Static myForeGroundConverter:MyDataCollectorClass.newComments}, Converter={StaticResource ObjectToForegroundConverter}}"

/>

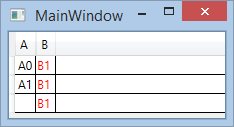
</DataTemplate>

Will have to move the whole comparison of the csv files to another class, or to pSHARE it selve. But couldn't get the binding working. Finally found the answer [here](http://www.thomaslevesque.com/2011/03/21/wpf-how-to-bind-to-data-when-the-datacontext-is-not-inherited/). Because you create a DataGridColumn outside the DataContext it is not in the visual tree. Through a Freezable object you can still bind it. Hurray for WPF… However I didn't get it working and finally decided to develop a small application to test why and post it on [Stackoverflow](http://stackoverflow.com/questions/33200227/show-on-a-datagrid-in-red-changed-text-in-a-datatable-wpf):

Title: <http://stackoverflow.com/questions/33200227/show-on-a-datagrid-in-red-changed-text-in-a-datatable-wpf>

How can I show in red text the value of a cell in a DataTable, when it is different from the same cell in another DataTable? In the final application the tables will be generated from changing csv files. So I have to replace autogenerated columns. I understood that you need a DataGridTemplateColumn for that, setting the CellTemplate to a resource. However this replaced column then is not part of the visual tree and therefore binding doesn't work.

http://www.thomaslevesque.com/2011/03/21/wpf-how-to-bind-to-data-when-the-datacontext-is-not-inherited/ shows that a converter implementing the Freezable object should solve that. Trying to figure out this solution, I made next simplified example. But it displays the last value of the cell in all cells of the column where the change occurs, and all in red. What am I doing wrong?



The ViewModel:

using System;

using System.ComponentModel;

using System.Data;

namespace WpfApplication1

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

//private Model \_Model; //for clarity left out

private DataTable \_propDataTable;

public DataTable propDataTable

{

get { return \_propDataTable; }

set

{

\_propDataTable = value;

NotifyPropertyChanged("propDataTable");

}

}

private DataTable propCopyDataTable;

private string \_sB;

public string sB

{

get { return \_sB; }

set

{

\_sB = value;

NotifyPropertyChanged("sB");

}

}

private bool \_bB = false;

public bool bB

{

get { return \_bB; }

set

{

\_bB = value;

NotifyPropertyChanged("bB");

}

}

public ViewModel()

{

propDataTable = new DataTable();

propDataTable.Columns.Add("A", typeof(string));

propDataTable.Columns.Add("B", typeof(string));

DataRow row0 = propDataTable.NewRow();

DataRow row1 = propDataTable.NewRow();

row0[0] = "A0";

row0[1] = "B0";

row1[0] = "A1";

row1[1] = "B1";

propDataTable.Rows.Add(row0);

propDataTable.Rows.Add(row1);

propCopyDataTable = propDataTable.Copy();

//now set a different value in propCopyDataTable

propCopyDataTable.Rows[1][1] = "Changed";

//find out which cells in column B are different

//try to show in red text which cell changed

for (int i = 0; i < propDataTable.Rows.Count; i++)

{

DataRow dr = propDataTable.Rows[i];

DataRow drc = propCopyDataTable.Rows[i];

sB = (string) dr["B"];

if (dr["B"].ToString().Equals(drc["B"].ToString()))

{

bB = true;

}

else

{

bB = false;

}

}

}

}

}

The ObjectToForegroundConverter:

using System;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

using System.Windows.Media;

namespace WpfApplication1

{

[ValueConversion(typeof(object), typeof(SolidColorBrush))]

public class ObjectToForegroundConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

SolidColorBrush b = new SolidColorBrush(Colors.Black);

try

{

bool changedValue = (bool)value;

if (changedValue)

{

b = Brushes.Red;

}

}

catch (Exception e)

{

MessageBox.Show(string.Format("Error: {0}", e));//instance not set to a etc.

}

return b;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

The BindingProxy converter:

using System.Windows;

namespace WpfApplication1

{

public class BindingProxy : Freezable

{

protected override Freezable CreateInstanceCore()

{

return new BindingProxy();

}

public object Data

{

get { return (object)GetValue(DataProperty); }

set { SetValue(DataProperty, value); }

}

public static readonly DependencyProperty DataProperty =

DependencyProperty.Register("Data", typeof(object), typeof(BindingProxy), new UIPropertyMetadata(null));

}

}

The XAML control:

<Window x:Class="WpfApplication1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication1"

Title="MainWindow" Height="350" Width="525">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:ObjectToForegroundConverter x:Key="MyObjectToForegroundConverter"/>

<myViewModel:BindingProxy x:Key="proxy" Data="{Binding}" />

<DataTemplate x:Key="changedBColumn" >

<TextBlock

Text="{Binding Data.sB,Source={StaticResource proxy},Mode=OneWay}"

Foreground="{Binding Data.bB,Converter={StaticResource MyObjectToForegroundConverter},Source={StaticResource proxy},Mode=OneWay}"

/>

</DataTemplate>

</Window.Resources>

<Grid>

<DataGrid x:Name="myXAMLtable" AutoGeneratingColumn="DataGrid\_AutoGeneratingColumn"

ItemsSource="{Binding propDataTable}">

</DataGrid>

</Grid>

</Window>

The code behind:

using System.Windows;

using System.Windows.Controls;

namespace WpfApplication1

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void DataGrid\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

switch (e.Column.Header.ToString())

{

case "B":

{

DataGridTemplateColumn BTemplateColumn = new DataGridTemplateColumn();

BTemplateColumn.Header = "B";

BTemplateColumn.CellTemplate = (DataTemplate)Resources["changedBColumn"];

e.Column = BTemplateColumn;

break;

}

}

}

}

}

|  |  |
| --- | --- |
| up vote0down vote[accept](javascript:void(0);) | The problem here is each cell need a bool to indicate the state of changing. But you have just 1 property for a column in your view-model. That means all the cells in the same column will have the same state depending on that property. That explains why all the cells in the column B are red because the property bB is set to true and bound to all cells in the column B.  You need an item class (instead of just simple string value) to hold the state, something like this:  public class Item {  public string Value {get;set;}  public Item(string value){  Value = value;  }  public bool IsChanged { get; private set;}  public void SetChanged(){  IsChanged = true;  }  public override string ToString(){  return Value;  }  public override bool Equals(object other){  var item = other as Item;  if(item == null) return false;  return item.Value == Value;  }  public override int GetHashCode(){  if(Value == null) return base.GetHashCode();  return Value.GetHashCode();  }  }  Now your DataTable should be created like this:  propDataTable = new DataTable();  propDataTable.Columns.Add("A", typeof(Item));  propDataTable.Columns.Add("B", typeof(Item));  DataRow row0 = propDataTable.NewRow();  DataRow row1 = propDataTable.NewRow();  row0[0] = new Item("A0");  row0[1] = new Item("B0");  row1[0] = new Item("A1");  row1[1] = new Item("B1");  propDataTable.Rows.Add(row0);  propDataTable.Rows.Add(row1);  propCopyDataTable = propDataTable.Copy();  //now set a different value in propCopyDataTable  propCopyDataTable.Rows[1][1] = new Item("Changed");  //find out which cells in column B are different  //try to show in red text which cell changed  for (int i = 0; i < propDataTable.Rows.Count; i++) {  DataRow dr = propDataTable.Rows[i];  DataRow drc = propCopyDataTable.Rows[i];  if (!object.Equals(dr["B"], drc["B"])) {  (dr["B"] as Item).SetChanged();  }  }  Now you also need to modify your converter like this:  public object Convert(object value, Type targetType, object parameter, CultureInfo culture)  {  SolidColorBrush b = new SolidColorBrush(Colors.Black);  var item = (Item)value;  if (item.IsChanged) {  b = Brushes.Red;  }  return b;  }  The XAML should also be modified like this:  <DataTemplate x:Key="changedBColumn" >  <TextBlock Text="{Binding [B], Mode=OneWay}"  Foreground="{Binding [B],Converter={StaticResource MyObjectToForegroundConverter},  Mode=OneWay}"  />  </DataTemplate>  NOTE: You don't need to use a proxy **unless** you want to bind the controls in DataTemplate to the main view-model. But actually that's wrong. What actually you need to bind your controls to here is each data row. In each cell template, the implicit DataContext is actually a DataRowView. So you don't need to set the Source explicitly. Just bind it normally. In the above XAML, I use the Path [B] meaning the indexer [] with a string key B is passed in directly - it is equivalent to the call someDataRowView["B"]. Also the Converter has the passed-in value as an Item (not a bool). Every time changing a value for a cell, you need to set it to a new Item, don't simply change the Item.Value property because it does not support INotifyPropertyChanged. Although in this case looks like you simply want to test it.  If you want to test the proxy technique, try creating another example. In fact that technique **does**work in your original code, when it successfully binds all cells to the bB property of your main view-model which is passed in via the proxy. |

Implementation of this in pCOLAD was not simple. Everywhere I had to replace strings by (new) Item(string). But got it working. It turned out that setting the Background to Pink colour was better.

Next issue. When you change the name of a connected parameter and hit RUN, you create an extra line in myDataTable. If you change it back this line will not disappear. A solution might be to rebuild myDataTable everytime you hit the OnOff button. However this should be without loading the csv files, in order to avoid to many times opening of the shared file. Well you could copy the copyTable (changed it to copyDataTable) to myDataTable… No! of course not, then you don't have any difference any more. So make a loadedDataTable to go back to.

Couldn't get rid of Pink cells after hitting the Share button. Probably has to do with INotifyPropertyChanged not being implemented. Maybe it is simpler to make a DataTable where you put in the Brushes.Pink or Brushes.White, depending on the difference with between myDataTable and copyDataTable? And bind the Background of the cells to that table? That doesn't work. You can set the ItemsSource of a DataGrid to a DataTable property, but for the cells' background you'll need an object property. So stick with the Item class.

An interesting approach is found [here](http://stackoverflow.com/questions/2830521/wpf-toolkit-datagridcell-style-datatrigger). It allows to set the colour of cells in all columns (also new ones). Have to translate it to the autogeneratingColumns event in code behind and change ".IsDirty" to ".IsChanged"

foreach (var column in columns)

{

var dataColumn =

new DataGridTextColumn

{

Header = column.Caption,

Binding = new Binding(column.FieldName),

CellStyle =

new Style

{

TargetType = typeof (DataGridCell),

Triggers =

{

new DataTrigger

{

Binding = new Binding(column.FieldName + ".IsDirty"),

Setters =

{

new Setter

{

Property = Control.BackgroundProperty,

Value = Brushes.Yellow,

}

}

}

}

}

};

\_dataGrid.Columns.Add(dataColumn);

}

However it is not clear to me how you can get the FieldName property of a column. Has something to do with generating DataGrid from sql database. Not our case. Try something else. [Here](https://social.msdn.microsoft.com/Forums/vstudio/en-US/8b2e94b7-3c44-4642-8acc-851de5285062/bind-datatable-to-wpf-datagrid-using-datagridtemplatecolumn-programatically?forum=wpf) we find a solution for binding to DataGridCells:

Sorry for the late reply. The problem here is that each cell's data context is a DataRowView instead of the an MyData instance. So the question now is 'how to extract the correct MyData instance from the DataRowView'.

In order to do that, we need to 1) know which column is being generated; 2) Change the default binding in the DataGridTemplateColumn so its content is bound to a specific data column in the DataRowView instead of the DataRowView itself.

This leads to the following code:

public class MyDataGridTemplateColumn : DataGridTemplateColumn

{

public string ColumnName

{

get;

set;

}

protected override System.Windows.FrameworkElement GenerateElement(DataGridCell cell, object dataItem)

{

// The DataGridTemplateColumn uses ContentPresenter with your DataTemplate.

ContentPresenter cp = (ContentPresenter)base.GenerateElement(cell, dataItem);

// Reset the Binding to the specific column. The default binding is to the DataRowView.

BindingOperations.SetBinding(cp, ContentPresenter.ContentProperty, new Binding(this.ColumnName));

return cp;

}

}

Since here your DataGrid is read-only, I just override the GenerateElement method; if there is editing template, then override GenerateEditingElement as well.

Now in the AutoGeneratingColumn event handler, use MyDataGridTemplateColumn instead:

private void dataGrid1\_AutoGeneratingColumn(object sender, Microsoft.Windows.Controls.DataGridAutoGeneratingColumnEventArgs e)

{

if (e.PropertyType == typeof(MyData))

{

MyDataGridTemplateColumn col = new MyDataGridTemplateColumn();

col.ColumnName = e.PropertyName; // so it knows from which column to get MyData

col.CellTemplate = (DataTemplate)FindResource("MyDataTemplate");

e.Column = col;

e.Column.Header = e.PropertyName;

}

}

Please let me know how it works for you.

Regards,  
Jie

Implementing this first in the simple test applications works:

ViewModel:

using System;

using System.ComponentModel;

using System.Data;

using System.Windows.Media;

namespace WpfApplication2

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

//private Model \_Model; //for clarity left out

private DataTable \_propDataTable;

public DataTable propDataTable

{

get { return \_propDataTable; }

set

{

\_propDataTable = value;

NotifyPropertyChanged("propDataTable");

}

}

private DataTable propCopyDataTable;

public ViewModel()

{

propDataTable = new DataTable();

propDataTable.Columns.Add("A", typeof(Item));

propDataTable.Columns.Add("B", typeof(Item));

DataRow row0 = propDataTable.NewRow();

DataRow row1 = propDataTable.NewRow();

row0[0] = new Item("A0");

row0[1] = new Item("B0");

row1[0] = new Item("A1");

row1[1] = new Item("B1");

propDataTable.Rows.Add(row0);

propDataTable.Rows.Add(row1);

propCopyDataTable = propDataTable.Copy();

//now set a different value in propCopyDataTable

propCopyDataTable.Rows[1][1] = new Item("Changed");

propCopyDataTable.Rows[0][0] = new Item("Also Changed");

//find out which cells in column B are different

//try to show in red text which cell changed

for (int i = 0; i < propDataTable.Rows.Count; i++)

{

DataRow dr = propDataTable.Rows[i];

DataRow drc = propCopyDataTable.Rows[i];

for (int j = 0; j < propDataTable.Columns.Count; j++)

{

if (!Object.Equals(dr[j], drc[j]))

{

(dr[j] as Item).SetChanged();

}

}

}

}

}

}

The Item class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WpfApplication2

{

public class Item

{

public string Value { get; set; }

public Item(string value)

{

Value = value;

}

public bool IsChanged { get; set; }

public void SetChanged()

{

IsChanged = true;

}

public override string ToString()

{

return Value;

}

public override bool Equals(object other)

{

var item = other as Item;

if (item == null) return false;

return item.Value == Value;

}

public override int GetHashCode()

{

if (Value == null) return base.GetHashCode();

return Value.GetHashCode();

}

}

}

The ObjectToForeGroundConvert:

using System;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

using System.Windows.Media;

namespace WpfApplication2

{

[ValueConversion(typeof(object), typeof(SolidColorBrush))]

public class ObjectToForegroundConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

SolidColorBrush b = new SolidColorBrush(Colors.Black);

if (value.Equals(DBNull.Value))

{

return b;

}

var item = (Item)value;

if (item.IsChanged)

{

b = Brushes.Red;

}

return b;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

The XAML control:

<Window x:Class="WpfApplication2.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication2"

Title="MainWindow" Height="350" Width="525">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:ObjectToForegroundConverter x:Key="MyObjectToForegroundConverter"/>

<DataTemplate x:Key="changedColumns" >

<TextBlock

Text="{Binding}"

Foreground="{Binding Converter={StaticResource MyObjectToForegroundConverter}}"

/>

</DataTemplate>

</Window.Resources>

<Grid>

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" AutoGeneratingColumn="myXAMLtable\_AutoGeneratingColumn"

ItemsSource="{Binding propDataTable}" CanUserAddRows="False">

</DataGrid>

</Grid>

</Window>

The DataGridColumnTemplate with overiding binding:

using System.Windows.Controls;

using System.Windows.Data;

namespace WpfApplication2

{

public class MyDataGridTemplateColumn : DataGridTemplateColumn

{

public string ColumnName

{

get;

set;

}

protected override System.Windows.FrameworkElement GenerateElement(DataGridCell cell, object dataItem)

{

// The DataGridTemplateColumn uses ContentPresenter with your DataTemplate.

ContentPresenter cp = (ContentPresenter)base.GenerateElement(cell, dataItem);

// Reset the Binding to the specific column. The default binding is to the DataRowView.

BindingOperations.SetBinding(cp, ContentPresenter.ContentProperty, new Binding(this.ColumnName));

return cp;

}

}

}

The Code behind the xaml:

using System.Windows;

using System.Windows.Controls;

namespace WpfApplication2

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void myXAMLtable\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

MyDataGridTemplateColumn col = new MyDataGridTemplateColumn();

col.ColumnName = e.PropertyName; // so it knows from which column to get the Item

col.CellTemplate = (DataTemplate)FindResource("changedColumns");

e.Column = col;

e.Column.Header = e.PropertyName;

}

}

}

pCOLLECT should know the added inputs when you restart. But then every instance of pCOLLECT should store the input names somewhere. But how do you know wich pCOLLECT is using which file? Yu Ke answered it [here](https://github.com/DynamoDS/Dynamo/issues/4776). Also have a look [here](https://github.com/DynamoDS/Dynamo/issues/5365). And basic information [here](https://msdn.microsoft.com/en-us/library/ms973893.aspx?f=255&MSPPError=-2147217396). So a \*.dyn file is just an xml file. Opening it with VS shows:

<Workspace Version="0.8.1.1823" X="-15.2362787219823" Y="-74.945324724499" zoom="0.671637979689955" Name="Home" RunType="Manual" RunPeriod="1000" HasRunWithoutCrash="False">

<NamespaceResolutionMap>

<ClassMap partialName="List.Create" resolvedName="DSCore.List" assemblyName="DSCoreNodes.dll" />

<ClassMap partialName="List" resolvedName="DSCore.List" assemblyName="DSCoreNodes.dll" />

<ClassMap partialName="List.Flatten" resolvedName="DSCore.List" assemblyName="DSCoreNodes.dll" />

</NamespaceResolutionMap>

<Elements>

<pCOLADnamespace.pCOLLECT guid="9d81e054-18af-490a-bc99-12b702d3fb87" type="pCOLADnamespace.pCOLLECT" nickname="pCOLLECT" x="218.818666578652" y="121.026467984284" isVisible="true" isUpstreamVisible="true" lacing="Disabled" inputcount="6" />

<Dynamo.Nodes.CodeBlockNodeModel guid="9370d06c-a14e-4cd6-b3e6-e44c35ebd976" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="120.086039006327" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;ParameterName1&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="5da6cf16-c3b2-4770-87ea-f41ae5ca482d" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="203.028114215722" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="1;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="6b6884f5-07ea-4c65-b802-c88b936812e1" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="288.028114215722" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;belangrijk1&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="b38fc4b5-7f11-4d85-bb9f-a60ce25bcb47" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="535.557271883109" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;extra1&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="91c8084b-b4c3-4f43-8606-07b0fe6ac0f2" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="370.028114215722" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;commentaar1&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.Watch guid="50c9dfe6-b4c4-458f-acbc-9e58b11447b1" type="Dynamo.Nodes.Watch" nickname="Watch" x="383.176221052378" y="501.032875634814" isVisible="true" isUpstreamVisible="true" lacing="Disabled" />

<Dynamo.Nodes.CodeBlockNodeModel guid="038a93ce-4798-4654-b4af-cbd3e4841098" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="452.54647274704" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;eigenaar&quot;;" ShouldFocus="false" />

<pCOLADnamespace.pSHARE guid="45f3ce24-162d-4110-b61d-481b3c2d9ab0" type="pCOLADnamespace.pSHARE" nickname="pSHARE" x="595.173765737405" y="253.411088867937" isVisible="true" isUpstreamVisible="true" lacing="CrossProduct" />

<pCOLADnamespace.pPARAM guid="45215231-10d8-4781-b0d7-acb30c97f6b1" type="pCOLADnamespace.pPARAM" nickname="pPARAM" x="708.13971845979" y="420.731973763868" isVisible="true" isUpstreamVisible="true" lacing="Disabled" />

<DSCore.File.Filename guid="e28501d4-f414-4e2e-8073-bf19a971cdba" type="DSCore.File.Filename" nickname="File Path" x="360.510771669148" y="160.268237628988" isVisible="true" isUpstreamVisible="true" lacing="Disabled">

<System.String>D:\Temp\test2.csv</System.String>

</DSCore.File.Filename>

<DSCore.File.Filename guid="628f9c29-45e7-4919-87b7-d2ccd02beaf1" type="DSCore.File.Filename" nickname="File Path" x="360.427774848009" y="242.983966322687" isVisible="true" isUpstreamVisible="true" lacing="Disabled">

<System.String>D:\Temp\test2 - Copy.csv</System.String>

</DSCore.File.Filename>

<DSCoreNodesUI.CreateList guid="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" type="DSCoreNodesUI.CreateList" nickname="List.Create" x="228.757609111999" y="411.016471035273" isVisible="true" isUpstreamVisible="true" lacing="Disabled" inputcount="2" />

<pCOLADnamespace.pCOLLECT guid="d0258267-1d06-4a9b-868a-3da093e90e0e" type="pCOLADnamespace.pCOLLECT" nickname="pCOLLECT" x="221.862045947228" y="639.138319973683" isVisible="true" isUpstreamVisible="true" lacing="Disabled" inputcount="6" />

<Dynamo.Nodes.CodeBlockNodeModel guid="45ee71fa-6173-4079-826f-ae11334bb189" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="713.598950468303" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="2;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="99e7a3cd-30ba-4184-b808-86e73c456634" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="798.598950468303" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;belangrijk2&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="3d64feba-b818-4b37-a478-f5dddd74e6cf" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="880.598950468303" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;commentaar2&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="1d8efdb3-056b-4c86-80f4-eb1b59ada4f9" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="27" y="636.598950468303" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;ParameterName2&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="85a0fd57-bd55-4222-89e9-05b7dbfbeb56" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="28.7904952410116" y="968.042176222131" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;extra2&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.Watch guid="c58ca5d1-5a43-4020-9e7f-f137c6bf2de7" type="Dynamo.Nodes.Watch" nickname="Watch" x="1006.30278179629" y="319.98348518954" isVisible="true" isUpstreamVisible="true" lacing="Disabled" />

<Dynamo.Nodes.CodeBlockNodeModel guid="4abf83c4-2ed0-4b18-83c2-eb5d3c50ab1d" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="434.91327774832" y="419.345772074617" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;ParameterName1&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="cb13cbef-f584-41dd-a987-3592cc9c5468" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="705.059392952776" y="520.826784980631" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="MyDataCollector.pCOLLECTinputs(p,v,i,c,o,e);" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="1bd3be5f-42ab-4439-9bf1-b8c8b6dc9349" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="683.423637026105" y="702.52609321334" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="MyDataCollector.pSHAREinputs(n,i,l,u);" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="68029f46-f246-48ad-aaae-7dd7e7580d22" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="422.17968889146" y="801.711837604365" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="MyDataCollector.pPARAMinputs(P,O);" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="25352530-eb30-48f9-b735-728eb7f39836" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="506.676351229782" y="122.059906718086" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="MyDataCollector.pCOLLECToutputs();" ShouldFocus="false" />

<Dynamo.Nodes.Watch guid="5c2b41e0-1701-4111-a4ba-a7ebeaef1263" type="Dynamo.Nodes.Watch" nickname="Watch" x="973.069336858374" y="151.149960252688" isVisible="true" isUpstreamVisible="true" lacing="Disabled" />

<pCOLADnamespace.pCOLLECT guid="b440a677-a607-4d82-b1d6-56e8542835ee" type="pCOLADnamespace.pCOLLECT" nickname="pCOLLECT" x="224.347735447126" y="857.955541910642" isVisible="true" isUpstreamVisible="true" lacing="Disabled" inputcount="6" />

<Dynamo.Nodes.CodeBlockNodeModel guid="4379ae3f-4ea4-4d17-870f-09df57b852b8" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="32.4102794044034" y="1219.21287581613" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;belangrijk5&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="51c39c79-1127-4e76-be9f-1f4d9849249c" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="32.4102794044034" y="1301.21287581613" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;commentaar5&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="a346c96c-ce74-4e23-82d8-7b763da63dc6" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="32.4102794044034" y="1057.21287581613" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;minWidth&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="74aa9f1d-2ede-41e4-a5e0-1f1501b32fee" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="34.200774645415" y="1388.65610156995" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="&quot;extra5&quot;;" ShouldFocus="false" />

<Dynamo.Nodes.CodeBlockNodeModel guid="5e592be8-cdc6-4946-8964-0c569eb41269" type="Dynamo.Nodes.CodeBlockNodeModel" nickname="Code Block" x="32.4102794044034" y="1134.21287581613" isVisible="true" isUpstreamVisible="true" lacing="Disabled" CodeText="5;" ShouldFocus="false" />

<Dynamo.Nodes.Watch guid="5c0b81cb-21e8-4362-bc8d-84594a14e577" type="Dynamo.Nodes.Watch" nickname="Watch" x="770.969109226907" y="235.138090378943" isVisible="true" isUpstreamVisible="true" lacing="Disabled" />

</Elements>

<Connectors>

<Dynamo.Models.ConnectorModel start="9d81e054-18af-490a-bc99-12b702d3fb87" start\_index="0" end="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="9370d06c-a14e-4cd6-b3e6-e44c35ebd976" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="9370d06c-a14e-4cd6-b3e6-e44c35ebd976" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="5da6cf16-c3b2-4770-87ea-f41ae5ca482d" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="5da6cf16-c3b2-4770-87ea-f41ae5ca482d" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="6b6884f5-07ea-4c65-b802-c88b936812e1" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="6b6884f5-07ea-4c65-b802-c88b936812e1" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="b38fc4b5-7f11-4d85-bb9f-a60ce25bcb47" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="5" portType="0" />

<Dynamo.Models.ConnectorModel start="b38fc4b5-7f11-4d85-bb9f-a60ce25bcb47" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="6" portType="0" />

<Dynamo.Models.ConnectorModel start="91c8084b-b4c3-4f43-8606-07b0fe6ac0f2" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="91c8084b-b4c3-4f43-8606-07b0fe6ac0f2" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="4" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="9d81e054-18af-490a-bc99-12b702d3fb87" end\_index="4" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="45f3ce24-162d-4110-b61d-481b3c2d9ab0" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="4" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="cb13cbef-f584-41dd-a987-3592cc9c5468" end\_index="5" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="1bd3be5f-42ab-4439-9bf1-b8c8b6dc9349" end\_index="4" portType="0" />

<Dynamo.Models.ConnectorModel start="038a93ce-4798-4654-b4af-cbd3e4841098" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="4" portType="0" />

<Dynamo.Models.ConnectorModel start="45f3ce24-162d-4110-b61d-481b3c2d9ab0" start\_index="0" end="45215231-10d8-4781-b0d7-acb30c97f6b1" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="45f3ce24-162d-4110-b61d-481b3c2d9ab0" start\_index="0" end="68029f46-f246-48ad-aaae-7dd7e7580d22" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="45f3ce24-162d-4110-b61d-481b3c2d9ab0" start\_index="0" end="5c0b81cb-21e8-4362-bc8d-84594a14e577" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="45215231-10d8-4781-b0d7-acb30c97f6b1" start\_index="0" end="c58ca5d1-5a43-4020-9e7f-f137c6bf2de7" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="e28501d4-f414-4e2e-8073-bf19a971cdba" start\_index="0" end="45f3ce24-162d-4110-b61d-481b3c2d9ab0" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="e28501d4-f414-4e2e-8073-bf19a971cdba" start\_index="0" end="1bd3be5f-42ab-4439-9bf1-b8c8b6dc9349" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="628f9c29-45e7-4919-87b7-d2ccd02beaf1" start\_index="0" end="45f3ce24-162d-4110-b61d-481b3c2d9ab0" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="628f9c29-45e7-4919-87b7-d2ccd02beaf1" start\_index="0" end="1bd3be5f-42ab-4439-9bf1-b8c8b6dc9349" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" start\_index="0" end="45f3ce24-162d-4110-b61d-481b3c2d9ab0" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" start\_index="0" end="50c9dfe6-b4c4-458f-acbc-9e58b11447b1" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" start\_index="0" end="1bd3be5f-42ab-4439-9bf1-b8c8b6dc9349" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="d0258267-1d06-4a9b-868a-3da093e90e0e" start\_index="0" end="0b99501e-49e4-4ff0-b32e-a24a3c988d3a" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="45ee71fa-6173-4079-826f-ae11334bb189" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="99e7a3cd-30ba-4184-b808-86e73c456634" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="3d64feba-b818-4b37-a478-f5dddd74e6cf" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="1d8efdb3-056b-4c86-80f4-eb1b59ada4f9" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="85a0fd57-bd55-4222-89e9-05b7dbfbeb56" start\_index="0" end="d0258267-1d06-4a9b-868a-3da093e90e0e" end\_index="5" portType="0" />

<Dynamo.Models.ConnectorModel start="4abf83c4-2ed0-4b18-83c2-eb5d3c50ab1d" start\_index="0" end="45215231-10d8-4781-b0d7-acb30c97f6b1" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="4abf83c4-2ed0-4b18-83c2-eb5d3c50ab1d" start\_index="0" end="68029f46-f246-48ad-aaae-7dd7e7580d22" end\_index="1" portType="0" />

<Dynamo.Models.ConnectorModel start="25352530-eb30-48f9-b735-728eb7f39836" start\_index="0" end="5c2b41e0-1701-4111-a4ba-a7ebeaef1263" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="4379ae3f-4ea4-4d17-870f-09df57b852b8" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="2" portType="0" />

<Dynamo.Models.ConnectorModel start="51c39c79-1127-4e76-be9f-1f4d9849249c" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="3" portType="0" />

<Dynamo.Models.ConnectorModel start="a346c96c-ce74-4e23-82d8-7b763da63dc6" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="0" portType="0" />

<Dynamo.Models.ConnectorModel start="74aa9f1d-2ede-41e4-a5e0-1f1501b32fee" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="5" portType="0" />

<Dynamo.Models.ConnectorModel start="5e592be8-cdc6-4946-8964-0c569eb41269" start\_index="0" end="b440a677-a607-4d82-b1d6-56e8542835ee" end\_index="1" portType="0" />

</Connectors>

<Notes />

<Annotations />

</Workspace>

E.g. we find inside <Elements>:

<pCOLADnamespace.pCOLLECT guid="9d81e054-18af-490a-bc99-12b702d3fb87" type="pCOLADnamespace.pCOLLECT" nickname="pCOLLECT" x="218.818666578652" y="121.026467984284" isVisible="true" isUpstreamVisible="true" lacing="Disabled" inputcount="6" />

We see that attribute 'inputcount' is saved with value "6". We should try to save the inputNicknames. Finally succeeded with:

protected override void SerializeCore(XmlElement element, SaveContext context)

{

base.SerializeCore(element, context);

var xmlDocument = element.OwnerDocument;

var subNode = xmlDocument.CreateElement("ExtraInputs");

foreach (var item in InPortData)

{

if (item.NickName=="P"|item.NickName=="V"|item.NickName=="I"|item.NickName=="C")

{

continue;

}

subNode.SetAttribute(item.NickName,item.ToolTipString);

}

element.AppendChild(subNode);

}

protected override void DeserializeCore(XmlElement element, SaveContext context)

{

base.DeserializeCore(element, context); //Base implementation must be called.

foreach (XmlNode subNode in element.ChildNodes)

{

if (!subNode.Name.Equals("ExtraInputs"))

continue;

if (subNode.Attributes == null || (subNode.Attributes.Count <= 0))

continue;

foreach (XmlAttribute attr in subNode.Attributes)

{

InPortData.Add(new PortData(attr.Name, attr.Value));

}

RegisterAllPorts();

break;

}

}

# Thumbnails and full screen images

Now start with trying to get thumbnails explaining the parameter. Have a look [here](http://mtaulty.com/CommunityServer/blogs/mike_taultys_blog/archive/2013/01/25/windows-8-making-a-simple-photo-viewer-in-c-and-xaml.aspx). Important to know is the needed hierarchy to get to the Image XAML tag in a proper way:

Grid

DataGrid

DataGrid.Columns

DataGridTemplateColumn

DataGridTemplateColumn.CellTemplate

DataTemplate

ItemsControl

ItemsControl.ItemTemplate

DataTemplate

Border

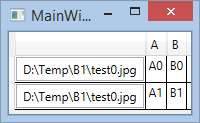
Image

But it was not easy to get it working. Finally posed next question on Stackoverflow.com:

Question about images in grid.

Title: How to bind datagridcells in a template to nested objects.

The real plug-in for a MVVM application is based on a csv file in a shared DropBox. One of the columns (in below simplified example "B") might have cell values that need explanation with images. Therefore we add a folder if needed, named with the cell value ("B1" in the example), containing images. Eventually these images should become thumbnails, and full screen images if a thumbnail is selected. But for now it would be great if the added column could display the file paths belonging to a cell value in column "B" in an adjacent cell. However after days of research and trial and error the result is:



While the cell in front of A0 should be empty and in front of A1 should read:

D:\Temp\B1\test0.jpg

D:\Temp\B1\test1.jpg

I have next questions:

Why do I have to set the DataContext again for the ListBox and for the TextBlock? It was already set for the Window. Are they not in the logical tree?

Why is only the first file displayed and why in all cells of the column?

What am I doing wrong?

The View:

<Window x:Class="WpfApplication3.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication3"

Title="MainWindow" Height="120" Width="200">

<Window.Resources>

<myViewModel:testConverter x:Key="myTestConverter"/>

</Window.Resources>

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Grid>

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" CanUserAddRows="False"

ItemsSource="{Binding PropDataTable}">

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<ListBox x:Name="folder" ItemsSource="{Binding MyImageFolderList,

Converter={StaticResource myTestConverter}}">

<ListBox.DataContext>

<myViewModel:ViewModel/>

</ListBox.DataContext>

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding MyImageFolderList/MyImageList/MyImagePath}">

<TextBlock.DataContext>

<myViewModel:ViewModel/>

</TextBlock.DataContext>

</TextBlock>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

The ViewModel:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

namespace WpfApplication3

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

//private Model \_Model; //for clarity left out

private DataTable propDataTable;

public DataTable PropDataTable

{

get { return propDataTable; }

set

{

propDataTable = value;

NotifyPropertyChanged("PropDataTable");

}

}

private List<MyImageFolder> myImageFolderList;

public List<MyImageFolder> MyImageFolderList

{

get { return myImageFolderList; }

set

{

myImageFolderList = value;

NotifyPropertyChanged("MyImageFolderList");

}

}

public ViewModel()

{

DataTable tempPropDataTable = new DataTable();

tempPropDataTable.Columns.Add("A", typeof(string));

tempPropDataTable.Columns.Add("B", typeof(string));

DataRow row0 = tempPropDataTable.NewRow();

DataRow row1 = tempPropDataTable.NewRow();

row0[0] = "A0";

row0[1] = "B0";

row1[0] = "A1";

row1[1] = "B1";

tempPropDataTable.Rows.Add(row0);

tempPropDataTable.Rows.Add(row1);

PropDataTable = tempPropDataTable;

MyImageFolderList = new List<MyImageFolder>();

//in D:\Temp\B1 there are two filesP test0.jpg and test1.jpg

string B0 = "D:\\Temp\\B1\\test0.jpg";

string B1 = "D:\\Temp\\B1\\test1.jpg";

MyImageFolder mif = new MyImageFolder("B1");

MyImage mi0 = new MyImage(B0);

MyImage mi1 = new MyImage(B1);

mif.MyImageList = new List<MyImage>();//did you forget this???

mif.MyImageList.Add(mi0);

mif.MyImageList.Add(mi1);

MyImageFolderList.Add(mif);

}

}

}

The MyImageFolder class:

using System;

using System.Collections.Generic;

using System.ComponentModel;

namespace WpfApplication3

{

public class MyImageFolder : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

private string myImageFolderPath = "";

public string MyImageFolderPath

{

get { return myImageFolderPath; }

set

{

myImageFolderPath = value;

NotifyPropertyChanged("MyImageFolderPath");

}

}

private List<MyImage> myImageList = new List<MyImage>();

public List<MyImage> MyImageList

{

get { return myImageList; }

set

{

myImageList = value;

NotifyPropertyChanged("MyImageList");

}

}

public MyImageFolder(string fp)

{

this.MyImageFolderPath = fp;

}

}

}

The MyImage class:

public class MyImage : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

private string myImagePath = "";

public string MyImagePath

{

get { return myImagePath; }

set

{

myImagePath = value;

NotifyPropertyChanged("MyImagePath");

}

}

//constructor

public MyImage(string ip)

{

MyImagePath = ip;

}

}

The answer was:

Everywhere you are writing something like

<ListBox.DataContext>

<myViewModel:ViewModel/>

</ListBox.DataContext>

You are creating a new instance of the ViewModel class and setting it to the DataContext behind the UI object. This is almost never what you want, because the end result is something like this (note the many instances of the ViewModel object) :

<DataGridRow DataContext=ViewModel1.PropDataTable[0]>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[0]>

<TextBlock DataContext=ViewModel3.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[1]>

<TextBlock DataContext=ViewModel4.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[2]>

<TextBlock DataContext=ViewModel5.MyImageFolderPath />

</ListBoxItem>

</DataGridItem>

<DataGridRow DataContext=ViewModel1.PropDataTable[1]>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[0]>

<TextBlock DataContext=ViewModel7.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[1]>

<TextBlock DataContext=ViewModel8.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[2]>

<TextBlock DataContext=ViewModel.MyImageFolderPath />

</ListBoxItem>

</DataGridRow>

The DataContext is inherited from the parent control, so you should write you bindings with that assumption.

What you want is ONE copy of your ViewModel object, which as a property called PropDataTable that is a collection of objects. Each object in PropDataTable should have a collection of objects called MyImageFolderList, and each item in MyImageFolderList should have a property called MyImageFolderPath.

So if you have the following (note the lack of assigning .DataContext to a new instance of ViewModel)

<DataGrid ItemsSource="{Binding PropDataTable"}>

...

<ListBox ItemsSource="{Binding MyImageFolderList"}>

...

<TextBlock Text="{Binding MyImageFolderPath}" />

...

</ListBox>

</DataGrid>

It actually renders as something along these lines :

<DataGridRow DataContext=ViewModel1.PropDataTable[0]>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[0]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[0].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[1]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[1].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[2]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[2].MyImageFolderPath />

</ListBoxItem>

</DataGridItem>

<DataGridRow DataContext=ViewModel1.PropDataTable[1]>

<ListBoxItem DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[0]>

<TextBlock DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[0].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[1]>

<TextBlock DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[1].MyImageFolderPath />

</ListBoxItem>

</DataGridRow>

I have an article on my blog that goes into the DataContext in a bit more detail if you're struggling to understand it as well. [What is this "DataContext" you speak of?](https://rachel53461.wordpress.com/2012/07/14/what-is-this-datacontext-you-speak-of/)

|  |  |
| --- | --- |
| [share](http://stackoverflow.com/a/33416651/2406292)[edit](http://stackoverflow.com/posts/33416651/edit) | answered 19 mins ago  [[https://www.gravatar.com/avatar/01b235a8024b2dd10f427c301c876a0b?s=32&d=identicon&r=PG](http://stackoverflow.com/users/302677/rachel)Rachel](http://stackoverflow.com/users/302677/rachel) **63.8k**28167280 |

Thanks a lot. Nice blog also! Do I understand it right if I say that by inserting a tag inside a tag you go down the Data layer tree when binding? The PropDataTable was just filled with strings. But when I change them into objects after a day of trial and error I still have empty cells. When I try to set the binding through the Properties window of VS it also doesn't show the properties I need. Strangely it shows a tree: ViewModel – PropDataTable – PropDataTable – DataView etc.

Is the following reasoning ok?

So TextBox should be bound to string property: string \*\*ImagePath\*\* of object\*\*Image\*\*. The parent of TextBox (ListBox) is bound to property: List<Image>\*\*ImageList\*\*. The parent of ListBox (DataGrid) is bound to the property: DataTable \*\*PropDataTable\*\* which contains the objects that has the ImageList property. I call these objects \*\*Item\*\*. So \_Item\_ has a property List<Image> \*\*ImageList\*\* and \_Image\_ has a property string \*\*ImagePath\*\*. If the reasoning is right, what can be wrong with my code? I have put it on GitHub [link]( https://github.com/jhubers/test)

Answer of Rachel:

@H.H. PropDataTable is a DataTable object, and DataTable does not have a property called ImageList. Change PropDataTable to a collection of objects which contain a public List<Image> property calledImageList – [Rachel](http://stackoverflow.com/users/302677/rachel) [2 hours ago](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54692182_33416651).

My comment:

I don't get this. If PropDataTable is not a DataTable, how can I use it to make a DataGrid? Or is every row of the DataGrid filled with the objects in the collection until the row is full and then the next row? Also in the real plug-in I don't know how many columns there are in the changing csv file...

I could of course automatically generate the columns on the basis of PropDataTable and replace them in the code behind the xaml. And then use PropDataTable to make a collection. It would be good if the imagepaths are also written to the History file. But a list in an Excel cell? You use a softreturn for that. What does it looks like in csv? Test shows that then the content of the cell is put between "" and normal returns between the lines. How can I handle this in C#? If I try it in pCOLAD I get an error about primary keys in the datatable (Figure 19). Of course because there are no unique values in column Parameter in row 7 and 8. Row 7,8 and 9 should be one row. Is that possible?

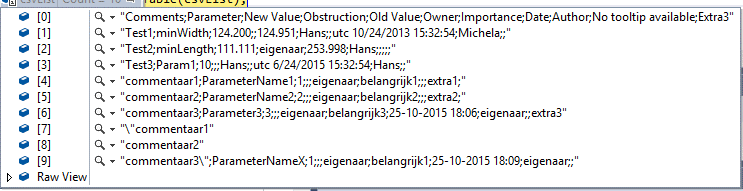


Figure 19 csv file with softreturns in row 7 loaded in DataTable

The problen lies in: line = myStream.ReadLine(); you'll have to test if there are returns between the "". On the [msdn site](https://msdn.microsoft.com/en-us/library/vstudio/system.io.streamreader.readline(v=vs.100).aspx) an alternative is presented:

function code:

string ReadNextMultiline(StreamReader mlReader)  
        {  
            bool MultilineDetected;  
            string res = "", mLine = "";              
            do  
            {  
                MultilineDetected = false;  
                mLine = mlReader.ReadLine();  
                res = String.Concat(  
                                        res,   
                                        (res.Length > 0?" ":""),    // add a space where there was a linebreak.  
                                        mLine);  
                string[] broken = res.Split(';');  
                // if the RES features unfinished multiliner, then the LAST element will contain exactly 1 " symbol:  
                if ((broken[broken.Length - 1].IndexOf('\"') >= 0) &&               // there's some " symbol  
                    (broken[broken.Length - 1].IndexOf('\"') == broken[broken.Length - 1].LastIndexOf('\"'))    // there's exactly 1 " on that row.  
                   )  
                {  
                    MultilineDetected = true;  
                }  
            } while (MultilineDetected);  
            return res;                          
        }

That works, but have to apply the recommended style for streamreader:

using (StreamReader sr = new StreamReader(filename))

{

string line = "";

while (sr.Peek() >= 0)

{

line = ReadNextMultiline(sr);

csvList.Add(line);

}

sr.Dispose();

return csvList;

}

And you have to replace the adde space by "\n".

Still the problems stay with displaying the result. Next comments on stackoverflow site lead to the answer.

Usually I would recommend making the property a List<T> of MyClass objects, where MyClass contains public properties of the fields you want, however since the columns aren't known at runtime and could change, a DataTable might be a bit better. Try manually adding an extra column to your DataTable with the fieldname of ImageList... if you can't set the datatype of the column to a List<Image> then you might need to leave it as an object and use an IValueConverter to convert it from object to List<Image> in the binding – [Rachel](http://stackoverflow.com/users/302677/rachel) [Oct 31 at 21:43](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54700309_33416651)

I can add a column to the DataTable of type List<Image>, but the datagrid then shows (Collection) in the cell. How do I get to the content of the collection? – [H.H.](http://stackoverflow.com/users/2406292/h-h) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54776814_33416651)

You'll have to use a DataGridTemplateColumn like you have in your original question, and I would probably use an IValueConverter to convert the DataRow (which will be the DataContext of your Template) to your List<Image>... it sounds like your grid will have variable columns, but the List<Image> will always be static. Btw if your grid does NOT have variable columns, I would highly recommend making a custom class for it instead of using the DataTable – [Rachel](http://stackoverflow.com/users/302677/rachel) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54788578_33416651)

var item = (DataRowView)value; return item[0]; this works. Thanks. – [H.H.](http://stackoverflow.com/users/2406292/h-h) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54793793_33416651)

The final version can be found here: [Link](https://github.com/jhubers/test) – [H.H.](http://stackoverflow.com/users/2406292/h-h) [21 hours ago](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54821399_33416651).

Later I added also the showing of a full screen image that is zoomable like in Revit and Dynamo:

The full screen control:

<Window x:Class="WpfApplication3.FullScreenImage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:WpfApplication3"

SizeToContent="WidthAndHeight"

Title="FullScreenImage" >

<Grid>

<local:ZoomBorder x:Name="border" ClipToBounds="True" Background="Gray">

<Image Name="fullImage" >

</Image>

</local:ZoomBorder>

</Grid>

</Window>

The code behind:

using System.Windows;

namespace WpfApplication3

{

public partial class FullScreenImage : Window

{

public FullScreenImage()

{

InitializeComponent();

}

}

}

The ZoomBorder:

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace WpfApplication3

{

public class ZoomBorder : Border

{

private UIElement child = null;

private Point origin;

private Point start;

private TranslateTransform GetTranslateTransform(UIElement element)

{

return (TranslateTransform)((TransformGroup)element.RenderTransform)

.Children.First(tr => tr is TranslateTransform);

}

private ScaleTransform GetScaleTransform(UIElement element)

{

return (ScaleTransform)((TransformGroup)element.RenderTransform)

.Children.First(tr => tr is ScaleTransform);

}

public override UIElement Child

{

get { return base.Child; }

set

{

if (value != null && value != this.Child)

this.Initialize(value);

base.Child = value;

}

}

public void Initialize(UIElement element)

{

this.child = element;

if (child != null)

{

TransformGroup group = new TransformGroup();

ScaleTransform st = new ScaleTransform();

group.Children.Add(st);

TranslateTransform tt = new TranslateTransform();

group.Children.Add(tt);

child.RenderTransform = group;

child.RenderTransformOrigin = new Point(0.0, 0.0);

this.MouseWheel += child\_MouseWheel;

this.MouseDown += child\_MouseDown;

this.MouseUp += child\_MouseUp;

this.MouseMove += child\_MouseMove;

this.PreviewMouseRightButtonDown += new MouseButtonEventHandler(

child\_PreviewMouseRightButtonDown);

}

}

public void Reset()

{

if (child != null)

{

// reset zoom

var st = GetScaleTransform(child);

st.ScaleX = 1.0;

st.ScaleY = 1.0;

// reset pan

var tt = GetTranslateTransform(child);

tt.X = 0.0;

tt.Y = 0.0;

}

}

#region Child Events

private void child\_MouseWheel(object sender, MouseWheelEventArgs e)

{

if (child != null)

{

var st = GetScaleTransform(child);

var tt = GetTranslateTransform(child);

double zoom = e.Delta > 0 ? .2 : -.2;

if (!(e.Delta > 0) && (st.ScaleX < .4 || st.ScaleY < .4))

return;

Point relative = e.GetPosition(child);

double abosuluteX;

double abosuluteY;

abosuluteX = relative.X \* st.ScaleX + tt.X;

abosuluteY = relative.Y \* st.ScaleY + tt.Y;

st.ScaleX += zoom;

st.ScaleY += zoom;

tt.X = abosuluteX - relative.X \* st.ScaleX;

tt.Y = abosuluteY - relative.Y \* st.ScaleY;

}

}

private void child\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (e.ChangedButton == MouseButton.Middle | e.ChangedButton == MouseButton.Left)

{

if (child != null)

{

var tt = GetTranslateTransform(child);

start = e.GetPosition(this);

origin = new Point(tt.X, tt.Y);

this.Cursor = Cursors.Hand;

child.CaptureMouse();

}

}

}

private void child\_MouseUp(object sender, MouseButtonEventArgs e)

{

if (e.ChangedButton == MouseButton.Middle | e.ChangedButton == MouseButton.Left)

{

if (child != null)

{

child.ReleaseMouseCapture();

this.Cursor = Cursors.Arrow;

}

}

}

void child\_PreviewMouseRightButtonDown(object sender, MouseButtonEventArgs e)

{

this.Reset();

}

private void child\_MouseMove(object sender, MouseEventArgs e)

{

if (child != null)

{

if (child.IsMouseCaptured)

{

var tt = GetTranslateTransform(child);

Vector v = start - e.GetPosition(this);

tt.X = origin.X - v.X;

tt.Y = origin.Y - v.Y;

}

}

}

#endregion

}

}

The MainWindow xaml:

<Window x:Class="WpfApplication3.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication3"

SizeToContent="WidthAndHeight"

Title="MyWindow">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:testConverter x:Key="myTestConverter"/>

<myViewModel:DataRowToListConverter x:Key="myDataRowToListConverter"/>

<DataTemplate x:Key="convertedImage">

<ListBox ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}">

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding ImagePath}"/>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</Window.Resources>

<Grid >

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" CanUserAddRows="False"

ItemsSource="{Binding PropDataTable}" AutoGeneratingColumn="myXAMLtable\_AutoGeneratingColumn">

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<ListBox ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}">

<ListBox.ItemTemplate>

<DataTemplate>

<Image Source="{Binding ImagePath}" Width="60"

HorizontalAlignment="Left" MouseUp="Image\_MouseUp">

</Image>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

The code behind MainWindow:

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace WpfApplication3

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void myXAMLtable\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

switch (e.Column.Header.ToString())

{

case "Images":

// Create a new template column.

DataGridTemplateColumn imageTemplateColumn = new DataGridTemplateColumn();

imageTemplateColumn.Header = "Images";

imageTemplateColumn.CellTemplate = (DataTemplate)Resources["convertedImage"];

// Replace the auto-generated column with the templateColumn.

e.Column = imageTemplateColumn;

e.Column.Width = 200;

break;

default:

e.Column.Width = 100;

break;

}

}

private void Image\_MouseUp(object sender, MouseButtonEventArgs e)

{

var im = (System.Windows.Controls.Image)sender;

FullScreenImage myFullScreenImage = new FullScreenImage();

myFullScreenImage.fullImage.Source = im.Source;

myFullScreenImage.Show();

}

}

}

The DataRowToListConverter:

using System;

using System.Collections.Generic;

using System.Data;

using System.Globalization;

using System.Windows.Data;

namespace WpfApplication3

{

[ValueConversion(typeof(DataRowView), typeof(List<Image>))]

class DataRowToListConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

List<Image> li = new List<Image>();

if (value==null)

{

return li;

}

else

{

var item = (DataRowView)value;

return item[0];

}

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

The ViewModel:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

namespace WpfApplication3

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

//private Model \_Model; //for clarity left out

private DataTable propDataTable;

public DataTable PropDataTable

{

get { return propDataTable; }

set

{

propDataTable = value;

NotifyPropertyChanged("PropDataTable");

}

}

//constructor

public ViewModel()

{

PropDataTable = new DataTable();

Image B1I0 = new Image("D:\\Temp\\B1\\test0.jpg");

Image B1I1 = new Image("D:\\Temp\\B1\\test1.jpg");

List<Image> B1L = new List<Image>();

B1L.Add(B1I0);

B1L.Add(B1I1);

Item A0 = new Item("A0");

Item B0 = new Item("B0");

Item A1 = new Item("A1");

Item B1 = new Item("B1");

PropDataTable.Columns.Add("Images", typeof(List<Image>));

PropDataTable.Columns.Add("A", typeof(Item));

PropDataTable.Columns.Add("B", typeof(Item));

DataRow row0 = PropDataTable.NewRow();

DataRow row1 = PropDataTable.NewRow();

row0[1] = A0;

row0[2] = B0;

row1[1] = A1;

row1[2] = B1;

row1[0] = B1L;

PropDataTable.Rows.Add(row0);

PropDataTable.Rows.Add(row1);

}

}

}

The Item:

namespace WpfApplication3

{

public class Item

{

public string Svalue { get; set; }

//overiding the ToString method takes care of displaying the string value in the DataGrid

public override string ToString()

{

return Svalue;

}

//constructor

public Item(string s)

{

Svalue = s;

}

}

}

The Image:

namespace WpfApplication3

{

public class Image

{

public string ImagePath { get; set; }

public Image(string ip)

{

ImagePath = ip;

}

}

}

Now integrate in pCOLAD. Careful. Check step by step. Test also effect if you start at 0. E.g. turned out that you get errors with primary column again, because table is empty. And when you switch from History to CSV you get warnings that parameters already exist. This was because the owner was added after the comparison.

The listing of images however takes too much space. Or they become too small. In fact they should be 14 pixels high with a border of 1 pixel to fit with the 16 pixels high text rows. A mouse hover should then show them in bigger format(Figure 20) and a mouse click in full format. Of course on mouse leave the images and borders should go to start dimensions and the column should go to fit the content again.

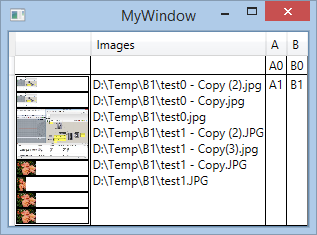


Figure 20 Test for mouse hover on images

This led to the next code:

Mainwindow xaml:

<Window x:Class="WpfApplication3.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication3"

SizeToContent="WidthAndHeight"

Title="MyWindow">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:testConverter x:Key="myTestConverter"/>

<myViewModel:DataRowToListConverter x:Key="myDataRowToListConverter"/>

<DataTemplate x:Key="convertedImagePath">

<ItemsControl ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding ImagePath}"/>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</DataTemplate>

</Window.Resources>

<Grid >

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" CanUserAddRows="False"

ItemsSource="{Binding PropDataTable}" AutoGeneratingColumn="myXAMLtable\_AutoGeneratingColumn" >

<DataGrid.Columns x:Uid="ttt">

<DataGridTemplateColumn Width="Auto" x:Name="dgtc">

<DataGridTemplateColumn.CellTemplate x:Uid="ppp">

<DataTemplate x:Name="dt">

<ItemsControl ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}"

x:Name="ic">

<ItemsControl.ItemTemplate x:Uid="sss">

<DataTemplate x:Name="dt2">

<Border x:Name="imageBorder" BorderThickness="1" BorderBrush="Black"

MouseLeave="imageBorder\_MouseLeave"

MouseEnter="imageBorder\_MouseEnter">

<Image x:Name="myImage" Source="{Binding ImagePath}" Height="14"

MouseUp="Image\_MouseUp" HorizontalAlignment="Left">

</Image>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

Code behind:

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace WpfApplication3

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void myXAMLtable\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

switch (e.Column.Header.ToString())

{

case "Images":

// Create a new template column.

DataGridTemplateColumn imageTemplateColumn = new DataGridTemplateColumn();

imageTemplateColumn.Header = "Images";

imageTemplateColumn.CellTemplate = (DataTemplate)Resources["convertedImagePath"];

// Replace the auto-generated column with the templateColumn.

e.Column = imageTemplateColumn;

//e.Column.Width = 200;

break;

default:

//e.Column.Width = 100;

break;

}

}

private void Image\_MouseUp(object sender, MouseButtonEventArgs e)

{

var im = (System.Windows.Controls.Image)sender;

//if (e.ClickCount == 2)

FullScreenImage myFullScreenImage = new FullScreenImage();

myFullScreenImage.fullImage.Source = im.Source;

myFullScreenImage.Show();

}

private void imageBorder\_MouseEnter(object sender, MouseEventArgs e)

{

var b = (Border)sender;

var im = (System.Windows.Controls.Image)b.Child;

im.Height = 50;

b.Height = 52;

}

private void imageBorder\_MouseLeave(object sender, MouseEventArgs e)

{

var b = (Border)sender;

var im = (System.Windows.Controls.Image)b.Child;

im.Height = 14;

b.Height = 16;

var dgtc = (DataGridTemplateColumn)this.FindName("dgtc");

//apparently you have to first reset the DataGridTemplateColumn

dgtc.Width = new DataGridLength();

var md = Mouse.DirectlyOver;

if (md == null)

{

dgtc.Width = new DataGridLength(1, DataGridLengthUnitType.SizeToCells);

return;

}

if (md.GetType() != typeof(Border))

{

dgtc.Width = new DataGridLength(1, DataGridLengthUnitType.SizeToCells);

}

else

{

Border mdb = (Border)md;

string mdn = mdb.Name;

if (mdn != "imageBorder")

{

dgtc.Width = new DataGridLength(1, DataGridLengthUnitType.SizeToCells);

}

}

}

static public void BringToFront(Panel pParent, ContentPresenter pToMove)

{

try

{

int currentIndex = Canvas.GetZIndex(pToMove);

int zIndex = 0;

int maxZ = 0;

ContentPresenter child;

for (int i = 0; i < pParent.Children.Count; i++)

{

if (pParent.Children[i] is ContentPresenter &&

pParent.Children[i] != pToMove)

{

child = pParent.Children[i] as ContentPresenter;

zIndex = Canvas.GetZIndex(child);

maxZ = Math.Max(maxZ, zIndex);

if (zIndex > currentIndex)

{

Canvas.SetZIndex(child, zIndex - 1);

}

}

}

Canvas.SetZIndex(pToMove, maxZ);

}

catch (Exception ex)

{

}

}

public static T FindVisualParent<T>(DependencyObject child) where T : DependencyObject

{

// get parent item

DependencyObject parentObject = VisualTreeHelper.GetParent(child);

// we’ve reached the end of the tree

if (parentObject == null) return null;

// check if the parent matches the type we’re looking for

T parent = parentObject as T;

if (parent != null)

{

return parent;

}

else

{

// use recursion to proceed with next level

return FindVisualParent<T>(parentObject);

}

}

}

}

When integrating in pCOLAD remember that for imagepaths you have a column with softreturns in the csv file. You already solved the problem of getting them right in the datatable. However in order to get the images right in the DataGrid you need a ListBox or an ItemsControl and so you need a List<MyImage> that is filled with objects that have a MyImagePath property. You have to take care of two things:

1. In the csv file you have content in the Images column. That should correspond to the content of the folder with the name of the Parameter column. If there is inconsistency you'll have to act on that. Or you always recreate the content of the Images column on the basis of the folder. Then you get the same situation as the second case:
2. You add, delete or change images. When you add an image, the path (or the file name) should appear in the Images column. When you delete an image, the corresponding path or file name should be deleted. When you change an image the path or file name should become red. This all implies that changes in the folders with Parameter names should be watched. So maybe simpler to add a folder called Images to the DropBox with inside Folders with Parameter names. Then you can more easily watch this Images folder and subfolders.

Or the other way around. We can make a List<String> Images column in the bound myPropDataTable on the basis of the Images folder. Then you don't need the DataRowToListConverter. But didn't I try that already? Yes! The ItemsControl and ListBox couldn't bind to List<strings> only to List<objects>. And then you need a converter. Isn't it strange!

So we have to add a List<MyImage> column to myPropDataTable myPropDataTable.Columns.Add("Images", typeof(List<MyImage>));

When we add rows, we fill the cells in this column with the paths of images in the folder belonging to the parameter of that row. We forget about marking changes in the images – only in the paths. It becomes too complex. We add a file watcher though (how can you avoid a message if you change the files yourself?).

We add a column without name where we show in overlap the images.

Implementing ran into problem with putting changes in red. Try to convert the solution to a property of Item. Then change the DataRowToListConverter into an ItemToListConverter. You bind the first column with the thumbnails like you do for the other columns only add this converter. First test this with the WpfApplication3. Indeed it works. You set the row[0] to a new Item and you give Item the Im property of type List<Image>. For the test create an Item Z0 and set the Im property to a list of images. The only thing you have to do is then check in the converter of the DataRowView item is of type Item (and not of string) and if so return it's Im property. This changes the converter into:

using System;

using System.Collections.Generic;

using System.Data;

using System.Globalization;

using System.Windows.Data;

namespace WpfApplication3

{

[ValueConversion(typeof(DataRowView), typeof(List<Image>))]

class DataRowToListConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

List<Image> li = new List<Image>();

if (value==null)

{

return li;

}

else

{

var item = (DataRowView)value;

//now the row consists only of items, but in the firs row we want the list of images

//which is now a property of Item, so in that case replace the row item by this property

if (item[0].GetType() == typeof(Item))

{

Item thisItem = (Item)item[0];

return thisItem.Im;

}

else

{

return item[0];

}

}

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

Works more or less. Have to check reading and writing image paths to csv file. Would be better to display only the filename… Solved by making an extra myImageFileName property to MyImage.

Writing to CSV with multilines added. But something goes wrong when you then open the CSVControl again by hitting the Red button. When you go to History file and back there is no problem. And after that everything is ok too. Also the new images in test.csv are shown in History. The old are not shown. The new should not be shown too. Well best would be if all changes to images were shown… Maybe with file version? Becomes too slow. So try to only display the names of the imagefiles when HistoryOn is true.

When a WPF application becomes slow you can use the following tools to find out where the problem is:

## Use WPF Profiling Tools to Profile a WPF Applicaton

Profiling a WPF application is an important step to understanding its behavior. There are lots of tools in the market for profiling WPF applications.

Here is a small compilation of such tools:

[Snoop](http://snoopwpf.codeplex.com/)  
[WPFPerf](http://www.microsoft.com/downloads/details.aspx?FamilyID=c17ba869-9671-4330-a63e-1fd44e0e2505&displaylang=en)  
[Perforator](http://msdn.microsoft.com/en-us/library/aa969767.aspx)  
[Visual Profiler](http://msdn.microsoft.com/en-us/library/aa969767.aspx)

The Perforator, Visual Profiler is part of the WPF Performance Suite. Have a look [here](https://msdn.microsoft.com/en-us/library/aa969767.aspx?f=255&MSPPError=-2147217396).

When implementing the FileSystemWatcher to get warnings when the csv file is changed by somebody else or the files in the Images folder, it turned out that in Automatic mode the Dynamo kept showing the warnings. As a 'quick' solution the runningtype of Dynamo is detected. With some help from Yu Ke and others it was found that in the UI customising procedure this setting can be detected as follows:

public static DynamoModel dm;

public void CSVUpdateHandler(object o, EventArgs e)

{

Compare();

myPropDataTable = MyDataCollectorClass.myDataTable;

RaisePropertyChanged("MyPropDataTable");

//update the solution

this.OnNodeModified(forceExecute: true);

runtype(dm);

}

public class pSHARENodeViewCustomization : INodeViewCustomization<pSHARE>

{

/// <summary>

/// At run-time, this method is called during the node

/// creation. Here you can create custom UI elements and

/// add them to the node view, but we recommend designing

/// your UI declaratively using xaml, and binding it to

/// properties on this node as the DataContext.

/// </summary>

/// <param name="model">The NodeModel representing the node's core logic.</param>

/// <param name="nodeView">The NodeView representing the node in the graph.</param>

//probably Dynamo has a method that makes it go here asa pSHARE is loaded

public void CustomizeView(pSHARE model, NodeView nodeView)

{

var pSHAREControl = new pSHAREcontrol();

nodeView.inputGrid.Children.Add(pSHAREControl);

pSHAREControl.DataContext = model;

Dynamo.ViewModels.NodeViewModel vm = nodeView.ViewModel;

Dynamo.Models.NodeModel nm = vm.NodeModel;

Dynamo.ViewModels.DynamoViewModel dvm = vm.DynamoViewModel;

pSHARE.dm = dvm.Model;

}

/// <summary>

/// Here you can do any cleanup you require if you've assigned callbacks for particular

/// UI events on your node.

/// </summary>

public void Dispose() { }

}

public void runtype(DynamoModel actual)

{

//!!!check if Automatic running is on

DynamoModel dm = actual;

foreach (var item in dm.Workspaces)

{

if (item.GetType() == typeof(HomeWorkspaceModel))

{

HomeWorkspaceModel hm = (HomeWorkspaceModel)item;

RunType rt = hm.RunSettings.RunType;

if (rt == RunType.Automatic)

{

MyDataCollectorClass.AutoPlay = true;

hm.RunSettings.RunType = RunType.Manual;//is needed to avoid hanging when filesystemwatcher fires

}

else

{

MyDataCollectorClass.AutoPlay = false;

}

}

}

}

## Updating Windows, Visual Studio and Dynamo

Had to reinstall Windows and went to Visual Studio 2015 and Dynamo 9.0. But problems with missing dlls of course (yellow warnings in the References folder). You can find a list of references and their paths in the \*.csproj files. Be sure that the paths in the different files are the same. In order to use the latest Prism 6 had to move to Net 4.5 too. Fortunately Dynomo 9.0 did too. However more difficult error:

Unknown build error, 'Cannot resolve dependency to assembly 'DynamoUtilities, Version=0.9.0.3067, Culture=neutral, PublicKeyToken=null' because it has not been preloaded. When using the ReflectionOnly APIs, dependent assemblies must be pre-loaded or loaded on demand through the ReflectionOnlyAssemblyResolve event. It turned out that Dynamo.Utilities was missing in the References (while it was there before).

There was also: warning MSB3277: Found conflicts between different versions of the same dependent assembly that could not be resolved. That propably means that I am referenceing different versions of reference in pCOLAD.csproj and myDataCollector.csproj. If you open these files with NotePad and search for ‘Reference’ you can copy the references and their paths and settings. It turned out that one had <Reference Include="System.Windows.Interactivity, Version=4.0.0.0 and the other <Reference Include="System.Windows.Interactivity, Version=4.5.0.0. When both were set to 4.5 the problem was gone.

But pCOLAD didn’t work in Dynamo anymore. The components didn’t show up. The packages were not working. The dll’s had to be copied to the nodes folder. But for this remember to run VS as Administrator. In order to debug you have to attach to the DynamoSandbox process. When you hit CTRL + ALT + P you get a list of processes you can attach to. It turned out the Func<> didn’t work anymore. That was due to the fact that MyDataCollector.dll didn’t load. When I loaded manually everything worked again. After that I changed the pkg.json file to the adapted content of the pkg.json file of DynamoSamples-0.9.0 files. Those are now separate and not part of the Dynamo repository anymore. Maybe also things didn’t work because some dll’s were in the packages folder that shouldn’t be there. Make sure that references are not copied local (setting in properties window).

## Test pCOLAD in Grasshoper and Dynamo together

Now check with pCOLAD10.sln for Grasshopper and the solution for the Ice Stadium. See if we can make the two systems work together. BTW also check the functionality of pCOLAD for Grasshopper in the report about that [research](file:///D:\Data\Research\pCOLAD\Research%20report%2003.docx). Well, for Grasshopper to work with pCOLAD, you have to put the pCOLAD10.gha file in C:\Users\jhubers\AppData\Roaming\Grasshopper\Libraries and start Rhino over again. The test file test.gh in D:\Temp shows the csv file, but not the images. Saving the csv file causes problems in the History.csv file: the columns are written without a return in front. Also obstruction is different: Grasshopper pCOLAD writes ‘/’ between names.

In order to redevelop and debug pCOLAD10.snl replace the references with the newer ones and set the Application in the properties of the project to .net 4. In the Debug section set Start External Program to the Rhino5.exe and the Command Line arguments to: /nosplash /runscript="-\_LoadScript D:\Temp\DebugGH.rvb. In the Tools/Options/Debugging uncheck Just My Code.

Would it be possible to reuse some of the code developed for Dynamo/pCOLAD? E.g. the display of several images in the csv display in Grasshopper/pCOLAD was cumbersome, because the ListView in the WindowsForm doesn’t allow multi-line nor lists in fields. A solution was found to combine all pictures belonging to a parameter into one, but the consequence was that all rows had to be of this height. A ListView has only 1 row height. Therefore it would be better to only display all the pictures if an image in the ListView is clicked. A little experiment (D:\Data\Research\Grasshopper\pCOLAD\_C\pCOLAD\_C\) showed that it is possible to combine a windows WPF MVVM application with Grasshopper/pCOLAD (see all the way down at pSHARE.vb). For this the WPF \*.exe file must be referenced and copied with a post build event to the directory of the \*.gha file (C:\Users\[your user nam]\AppData\Roaming\Grasshopper\Libraries. A static property of the MainWindow can be used to set the path of the image directory belonging to a parameter. Also some used references like PresentationFramework need to be added to the references.

Some other problems: History button doesn’t hide the others. Multi-line is not handled. Single obstruction click in Dynamo/pCOLAD doesn’t work correctly anymore: only on first line appears the user name. The latter was due to setting the binding of the RowIndex to OneWay. Changed to.

CurrentCell="{Binding CellInfo, Mode=TwoWay}" x:FieldModifier="public"

SelectedIndex="{Binding RowIndex, Mode=TwoWay}" VerticalAlignment="Top">

How to load and save the multi-line content of the Images column in the History.csv file in Grasshopper/pCOLAD? In the project csv file the image names are not written. They are generated on the basis of the content of the Images folders. So for reading the History.csv file one solution might be to detect the multi-line by checking the first and last char. If it is “ and “ then it is a multi-line. Then replace the return character by | e.g. And when you save you generate the multi-lines on the basis of the content of the Images folder. So there is no problem then.

By testing the use of both Grasshopper/ and Dynamo/pCOLAD an error occurred while displaying the change message in Grasshopper when Dynamo shares the csv file. Something about not being allowed to use Grasshopper.Instances.DocumentEditor calling it from another thread. After some googling the solution seemed to be to first check if Grasshopper.Instances.DocumentEditor.InvokeRequired and if so then use MessageBox.Show(Grasshopper.Instances.DocumentEditor.Invoke(method As Delegate, ParamArray args As Object()), "message”). But it gave all kinds of errors. The delegate was set to Delegate Sub OnChangedCallBack(source As Object, e As FileSystemEventArgs).And the ParamArray to Dim pa(1) As Object

pa(0) = New Object()

pa(1) = e

It worked for the first message, but the second gave errors like System.Reflection.AmbiguousMatchException or Overload resolution failed because no Public 'Show' is most specific for these arguments: 'Public Shared Function Show(text As String, caption As String) As System.Windows.Forms.DialogResult'. Strange enough it turned out in that last case that pa became the username as text. Also not matching parametercount was sometimes an error. It was more or less solved by putting an if statement after the callback: if pa Is username etc.

After some days of trial and error, it seems like that if you want to use a control (like the Grasshopper.Instances.DocumentEditor) when it is created on another thread then where you want to use it (as in this case the thread started by the filesystemwatcher) you have to invoke it. Meaning that you need a delegate. The execution of the code then kind of jumps back to the start and executes the code. So in next code when the line is executed that reads:  
If Grasshopper.Instances.DocumentEditor.InvokeRequired the first time it is true. Then after executing the line: Grasshopper.Instances.DocumentEditor.Invoke (d, pa) the execution jumps back to the if statement and now it is false. And execution continues after the Else statement.

Code:

Delegate Sub OnChangedCallBack(source As Object, e As FileSystemEventArgs)

Private Sub OnChanged(source As Object, e As FileSystemEventArgs)

watcher.EnableRaisingEvents = False 'en weer aanzetten na afhandeling van deze routine

Dim pa(1) As Object

pa(0) = New Object()

pa(1) = e

If authorName = "" Then 'you are not the one that hit Share button

If Grasshopper.Instances.DocumentEditor.InvokeRequired Then

Dim d As New OnChangedCallBack(AddressOf OnChanged)

Grasshopper.Instances.DocumentEditor.Invoke(d, pa)

Else

MessageBox.Show(Grasshopper.Instances.DocumentEditor, "Message for " & userName & ": Somebody changed the input file, please start over again...")

End If

Else

authorName = "" 'otherwise authorName will always be not ""

End If

m\_formClose()

watcher.EnableRaisingEvents = True

End Sub

In order to debug all process you have to check the project properties/Debug/Enable native code debugging. Otherwise you can’t set break points in some code. And in Debugging/Options/check Supress JIT optimization on module load (Managed only) in order to be able to put variables in the watch window and/or use intellisence.

Next problem: if in Grasshopper/pCOLAD you click on a checkbox and there are no images, you get an error about value can not be null pointing at the path. In fact if you hit the checkbox, WpfApplication3 should not be activated… But also clicking on an empty picture should not give an error of course. So:

Dim ip As String = myDir & "\" & parameterName

Dim myDirInfo As DirectoryInfo = New DirectoryInfo(ip)

If myDirInfo.Exists Then

If (myDirInfo.EnumerateFiles().Any()) Then

Next problem: the history file is not correctly displayed. In Dynamo it shows an extra column (Column1) so probably Grasshopper/pCOLAD did add it, but doesn’t display it? It also turns out that if no History.csv file exists we get a serious error. If you solve it by sending a message asking for hitting the Share button first. You get a History.csv file that starts with an empty line. If there is an image in a line, then the rest of the items are shifted to the right. Solved by If newHistoryFile Boolean. And in the multilines code removed the “;” at the end.

Next problem: in Dynamo changes are not red anymore. That is normal of course if you make the changes with Grasshopper (also the local file is then changed). Also obstruction names are comma separated, while in Grasshopper separation char is /. Also the sharing date is written differently.

Next problem: in Dynamo showing a big History.csv file is too slow. Find a way to not showing the images. This needed a [discussion on Stackoverflow](http://stackoverflow.com/questions/35204920/conditional-datatemplate-for-images?noredirect=1#comment58161839_35204920). The final solution was using a DataTemplateSelector as suggested. The code:

using System.Data;

using System.Windows;

using System.Windows.Controls;

namespace pCOLADnamespace

{

public class ImageTemplateSelector : DataTemplateSelector

{

public DataTemplate WithImage

{ get; set; }

public DataTemplate WithoutImage

{ get; set; }

public override DataTemplate SelectTemplate(object item, DependencyObject container)

{

//pSHARE pS = item as pSHARE;

DataRowView drv = item as DataRowView;

if (drv != null)

{

if (pSHARE.historyOn)

{

return WithoutImage;

}

else

{

return WithImage;

}

}

else

{

return base.SelectTemplate(item, container);

}

}

}

}

And in Window.Resources in the XAML:

<DataTemplate x:Key="imagesOn">

<ItemsControl ItemsSource="{Binding

Converter={StaticResource myDataRowToListConverter}}">

<ItemsControl.ItemTemplate>

<DataTemplate >

<Border x:Name="imageBorder1" BorderThickness="1" BorderBrush="Black"

MouseLeave="imageBorder\_MouseLeave"

MouseEnter="imageBorder\_MouseEnter" Height="16">

<!-- try RenderOptions.BitmapScalingMode="LowQuality" if it is too slow -->

<Image x:Name="myImage" Source="{Binding MyImagePath}"

MouseUp="Image\_MouseUp" HorizontalAlignment="Center">

</Image>

</Border>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</DataTemplate>

<DataTemplate x:Key="imagesOff">

<!--just empty cells-->

</DataTemplate>

And in <DataGrid.Columns> in the XAML:

<DataGridTemplateColumn x:Name="dgtc">

<DataGridTemplateColumn.CellTemplateSelector>

<local:ImageTemplateSelector

WithImage="{StaticResource imagesOn}"

WithoutImage="{StaticResource imagesOff}"/>

</DataGridTemplateColumn.CellTemplateSelector>

</DataGridTemplateColumn>

Next problem. The comments are written cumulatively in Grasshopper, not in Dynamo. But in fact that is not a problem. In Dynamo you could also add comments instead of changing them. You type them in the control, while in Grasshopper you have to add them through the parameters.

Also the save date is differently formatted.

In Grasshopper warning also if you save yourself. Looked again at the FileSystemWatcher. It is a known problem that it fires several times. Even if you only watch the LastWrite and disable the watcher as soon as the first event is detected you get several calls at the event handler. It appears to depend on the software that is writing to the file. Maybe not smart to write to the file in a loop? Changed it to writing the text all lines at once and that solved the issue. But this showed next problem. When there are less columns in the project csv file than in the history file you get errors when showing the history file. This appears if at some point in time the project csv file was written to the history file with less columns than the first time. Solved by taking the smallest count for the loop. But it also shows that the case was not treated where an extra property of the parameter is not used anymore. Check what happens then! Well not much. The property column and the value just stays as it was. In the comment one could mention that it is not used anymore and set the objection to true. But this showed the next shortcoming in pCOLAD-for-Dynamo. The comment is overwritten because it is also a property of the parameter. So you have to change it in the solution not only in the display (minor thing for later). The warnings in pD are still multiple (pG is OK now - so do the same in pD - OK now too).

Now what if you add and remove a property in pG? That works more or less (when you share the data, it is not updated in the history view in pG. In pD everything works immediately OK. The problem in pG is probably that the history listview now shows the smallest amount of column heads. Indeed, changed it into longest.

Now try if pD works alright in Revit. No! Get error:

Object reference not set to an instance of an object.

at pCOLADnamespace.pSHARE.closeCSVControl() in D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs:line 505

at pCOLADnamespace.pSHARE.ShowParams(Object obj) in D:\Data\Research\Dynamo\pCOLAD for Dynamo\pSHARE1.cs:line 564

at MS.Internal.Commands.CommandHelpers.CriticalExecuteCommandSource(ICommandSource commandSource, Boolean userInitiated)

at System.Windows.Controls.Primitives.ButtonBase.OnClick()

at System.Windows.Controls.Primitives.ButtonBase.OnMouseLeftButtonUp(MouseButtonEventArgs e)

at System.Windows.RoutedEventArgs.InvokeHandler(Delegate handler, Object target)

at System.Windows.RoutedEventHandlerInfo.InvokeHandler(Object target, RoutedEventArgs routedEventArgs)

at System.Windows.EventRoute.InvokeHandlersImpl(Object source, RoutedEventArgs args, Boolean reRaised)

at System.Windows.UIElement.ReRaiseEventAs(DependencyObject sender, RoutedEventArgs args, RoutedEvent newEvent)

at System.Windows.UIElement.OnMouseUpThunk(Object sender, MouseButtonEventArgs e)

at System.Windows.RoutedEventArgs.InvokeHandler(Delegate handler, Object target)

at System.Windows.RoutedEventHandlerInfo.InvokeHandler(Object target, RoutedEventArgs routedEventArgs)

at System.Windows.EventRoute.InvokeHandlersImpl(Object source, RoutedEventArgs args, Boolean reRaised)

at System.Windows.UIElement.RaiseEventImpl(DependencyObject sender, RoutedEventArgs args)

at System.Windows.UIElement.RaiseTrustedEvent(RoutedEventArgs args)

at System.Windows.Input.InputManager.ProcessStagingArea()

at System.Windows.Input.InputProviderSite.ReportInput(InputReport inputReport)

at System.Windows.Interop.HwndMouseInputProvider.ReportInput(IntPtr hwnd, InputMode mode, Int32 timestamp, RawMouseActions actions, Int32 x, Int32 y, Int32 wheel)

at System.Windows.Interop.HwndMouseInputProvider.FilterMessage(IntPtr hwnd, WindowMessage msg, IntPtr wParam, IntPtr lParam, Boolean& handled)

at System.Windows.Interop.HwndSource.InputFilterMessage(IntPtr hwnd, Int32 msg, IntPtr wParam, IntPtr lParam, Boolean& handled)

at MS.Win32.HwndWrapper.WndProc(IntPtr hwnd, Int32 msg, IntPtr wParam, IntPtr lParam, Boolean& handled)

at MS.Win32.HwndSubclass.DispatcherCallbackOperation(Object o)

at System.Windows.Threading.ExceptionWrapper.InternalRealCall(Delegate callback, Object args, Int32 numArgs)

at System.Windows.Threading.ExceptionWrapper.TryCatchWhen(Object source, Delegate callback, Object args, Int32 numArgs, Delegate catchHandler)

Maybe it is related to the earlier error, that didn’t affect DynamoSandbox (Dynamo not as addin in Revit):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Activated | Event | Time | Duration | Thread |
|  | Exception thrown: 'System.ArgumentException' in ProtoCore.dll ("Object of type 'System.Boolean' cannot be converted to type 'Dynamo.UI.Commands.DelegateCommand'.") | 20.89s |  | [14964] |

Maybe not. The problem was here: foreach (Window w in Application.Current.Windows) Application stands for WPF applications, and Current is null. Probably because Revit is not a WPF application? Checking the processes doesn’t help either (only Revit can be found as process).

But how can I then avoid several CSVControls to be open? Maybe automatically close whenever Revit Dynamo closes? For this we could use events. [Here](https://msdn.microsoft.com/en-us/library/aa645739(v=vs.71).aspx) is some background information. I also asked the question in the Dynamo GitHub and indeed the answer is using events from the DynamoView or DynamoModel. Also the Dispose method of the node could be used. But since both are already properties of the public class pSHARENodeViewCustomization : INodeViewCustomization<pSHARE> hooking up to the Dynamo events seems most simple. It is complex stuff though. First you need a delegate. It seems that public delegate void DynamoModelHandler(DynamoModel model); is the one. Note that it needs a DynamoModel as parameter. Then we need an event. That could well be: public event DynamoModelHandler ShutdownStarted; So ShutdownStarted is an event of the DynamoModel. We already used the DynamoModel when we needed to detect if Automatic mode was on. It is a property of pSHARE called dm. So in the pSHARE constructor we can put: dm.ShutdownStarted += new DynamoModelHandler(closeCSVcontrolFrom\_dm); And the method that uses dm:

public void closeCSVcontrolFrom\_dm(DynamoModel myDm)

{

if (\_CSVControl != null)

{

\_CSVControl.Close();

}

}

But this doesn’t work because dm, the DynamoModel, is null during instanciation of pSHARE. The DynamoModel, dm, is instanciated in the nested class pSHARENodeViewCustomization. It appears that you can access the methods of the parent class by passing that to the constructor. But be aware that you then also have to generate a default instance. So that brings us to next code within the pSHARENodeViewCustomization class:

public pSHARENodeViewCustomization()

{

}

public pSHARENodeViewCustomization(pSHARE parent)

{

this.parent = parent;

}

public void Dispose()

{

//maybe here you can close any open controls?

if (parent.\_CSVControl != null)

{

parent.\_CSVControl.Close();

}

}

But it doesn’t work, because before Dispose() the parent (pSHARE) is already null.

In the meanwhile Dynamo is update to the 0.9.1 version. Had to rename several usings and the InPortData property became obsolete: It says: 'NodeModel.InPortData' is obsolete: 'InPortData is deprecated, please use the InPortNamesAttribute, InPortDescriptionsAttribute, and InPortTypesAttribute instead.' It took some time to understand how to use these. You have to declare them before the declaration of the class. As follows:

[InPortNamesAttribute("N", "I", "L", "U")]

[InPortDescriptionsAttribute(

"Input (a List.CreateList) of pCOLLECT output(s)",

"Input a FilePath for the shared csv files.",

"Input a FilePath for the local copy of the csv file.",

"Input a the user namen (Code Block).")]

[InPortTypesAttribute ("string", "string", "string", "string")]

[OutPortNamesAttribute ("O")]

[OutPortDescriptionsAttribute ("Output of parameter name and value on next line; two by two.")]

[OutPortTypesAttribute ("string")]

#endregion

public class pSHARE : NodeModel

Etc.

But how can we then add inports dynamically? Maybe next code from [here](http://stackoverflow.com/questions/14663763/how-to-add-an-attribute-to-a-property-at-runtime) can help?

public class SomeAttribute : Attribute

{

public SomeAttribute(string value)

{

this.Value = value;

}

public string Value { get; set; }

}

public class SomeClass

{

public string Value = "Test";

}

[TestMethod]

public void CanAddAttribute()

{

var type = typeof(SomeClass);

var aName = new System.Reflection.AssemblyName("SomeNamespace");

var ab = AppDomain.CurrentDomain.DefineDynamicAssembly(aName, AssemblyBuilderAccess.Run);

var mb = ab.DefineDynamicModule(aName.Name);

var tb = mb.DefineType(type.Name + "Proxy",System.Reflection.TypeAttributes.Public, type);

var attrCtorParams = new Type[] { typeof(string) };

var attrCtorInfo = typeof(SomeAttribute).GetConstructor(attrCtorParams);

var attrBuilder = new CustomAttributeBuilder(attrCtorInfo, new object[] { "Some Value" });

tb.SetCustomAttribute(attrBuilder);

var newType = tb.CreateType();

var instance = (SomeClass)Activator.CreateInstance(newType);

Assert.AreEqual("Test", instance.Value);

var attr = (SomeAttribute)instance.GetType().GetCustomAttributes(typeof(SomeAttribute), false).SingleOrDefault();

Assert.IsNotNull(attr);

Assert.AreEqual(attr.Value, "Some Value");

}

Couldn’t get it to work. Also very persistent error occurs during debugging:

error CS1704: An assembly with the same simple name 'ProtoCore' has already been imported. Try removing one of the references (e.g. 'ProtoCore.dll') or sign them to enable side-by-side. But ProtoCore is referenced only once. Alos copy local has been disabled. ProtoCore has been found on C-drive in several places:

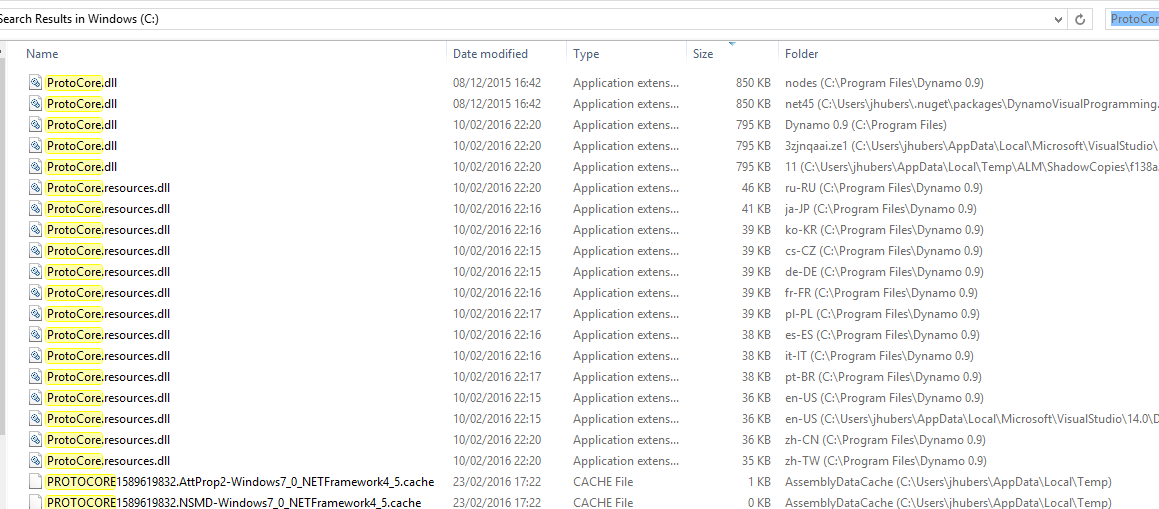


Figure 21 Too many ProtoCore references

The solution was to uninstall Dynamo and delete all of the files in Figure 21. And reinstall Dynamo.

Continuing tests in Revit/Dynamo showed that when you connect only 1 pCOLLECT, you get the message that you have to put a List.Create in between to connect to pSHARE. But if you do for only 1 pCOLLECT the parameter does not show up in the display. This was due to if statements checking if pCOLLECT was not directly connected to pSHARE by checking the number of DataTables to be merged. But now while saving you get errors with myDataTable and the merging of the tables when you save changes. One of the tables is null!! Changed it to a warning in the output pSHARE.

Next problem is that when sharing the parameters if (!Application.Current.Dispatcher.CheckAccess()) gives an error in Revit because Application.Current is null. Revit is not a WPF application. Solved it by putting inside if (Application.Current != null) but that of course is not a solution, because then you don’t run the code that gives a warning that the csv file has been changed. However, you immediately see in red the changes somebody else made… Maybe that is enough? Maybe not, because if you don’t have the display open or visible, you easily can miss this.

## Reminder for working with the code after some time.

When you start again developing Dynamo/pCOLAD:

1. Open the VS pCOLAD.sln file.
2. Set in Tools/Options/Debugging/Enable Just My Code to true. To speed up. Maybe safer not to change it, because Grasshopper/pCOLAD needs it to be false. Only the first time debugging and loading the symbols of the used dlls takes much time. Next time (because caching is turned on in options) is much faster. With Enable Just My Code to true you’ll miss some intellisense feedback during debugging and some impossible locations to break.
3. Start Dynamo.
4. Go back to VS and hit Shift + Ctrl + Alt + F5 to start debugging. Or hit Ctrl + Alt + P and attach the DynamoSandbox process by hand. Wait until everything is loaded.
5. Go to Dynamo and start the pCOLAD test file.
6. Hit RUN and hit the Red start button in pSHARE.

When you start with Grasshopper/pCOLAD:

1. Open the pCOLAD10.sln file.
2. Set in Tools/Options/Debugging/Enable Just My Code to false.
3. Hit F5.

1. Try first to open the whole Dynamo project solution (actual Dynamo.All.2013.sln) and run the custom tool (right click on AssemblyInfo.tt file in project AssemblyInfoGenerator in Solution Explorer). And see if indeed you can do without it. Yes, no difference. [↑](#footnote-ref-1)