ProjectDesigner+ Developer Guide

Contents

1 How to Add New Node Types? 2
1.1 How to Add Metadata to Nodes? 3

2 How to Add New Connection Types? 3
2.1 How to Add Metadata to Connections? 4

3 How to Add New Member Types? 4

4 How to Make Editor Actions Undo-Redoable? 5

5 How to Create Nodes From Assets? 6

6 How to Add New Options to Default Context Menu? 7

7 How to Save IDrawables? 8

1 How to Add New Node Types?

The main component of the Project Designer+ is an abstract class called NodeBase. New node types can be overridden by creating a class anywhere in your project and inheriting from NodeBase class.

NodeBase types are collected with reflection in the project after each domain reload. A new menu option will be available for your type when you right-click on an empty space in the editor.

This is pretty much all to it, you will be able to drag, select and interact with the new node after creating one with the context menu.

Here's an example from one of the built-in node types, "Notepad":

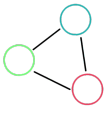
Raleway-TLF

```
 //Make sure to mark your classes as Serializable.
[Serializable]
public class Notepad : NodeBase
{
    // override node size
    public override Vector2 MinSize => new Vector2(440, 440);
    public override Vector2 MaxSize =>  new Vector2(440, 720);
    //override icon asset name
    public override string IconKey => "note";
    public override bool CanBeCopied => true;

    protected override int FooterHeight => 10;

    public Notepad(string header) : base()
    {
        HeaderText = header;
    }

    // virtual functions can be overriden to do something when the node is created.
    protected override void OnAddedInternal(IEditorContext context, DrawableCreationType drawableCreationType)
    {
        if (drawableCreationType == DrawableCreationType.Default)
        {
            AddMember<CommentMember>();
        }
    }
}
```

1.1 How to Add Metadata to Nodes?

To add basic metadata to node types, one can use NodeBaseMetaData attribute. NodeBaseMetaData attribute allows users to set display name, max count per project and other basic data for their NodeBase classes.

Raleway-TLF

```
[Serializable, NodeBaseMetaData("Note")]
public class Notepad : NodeBase
```

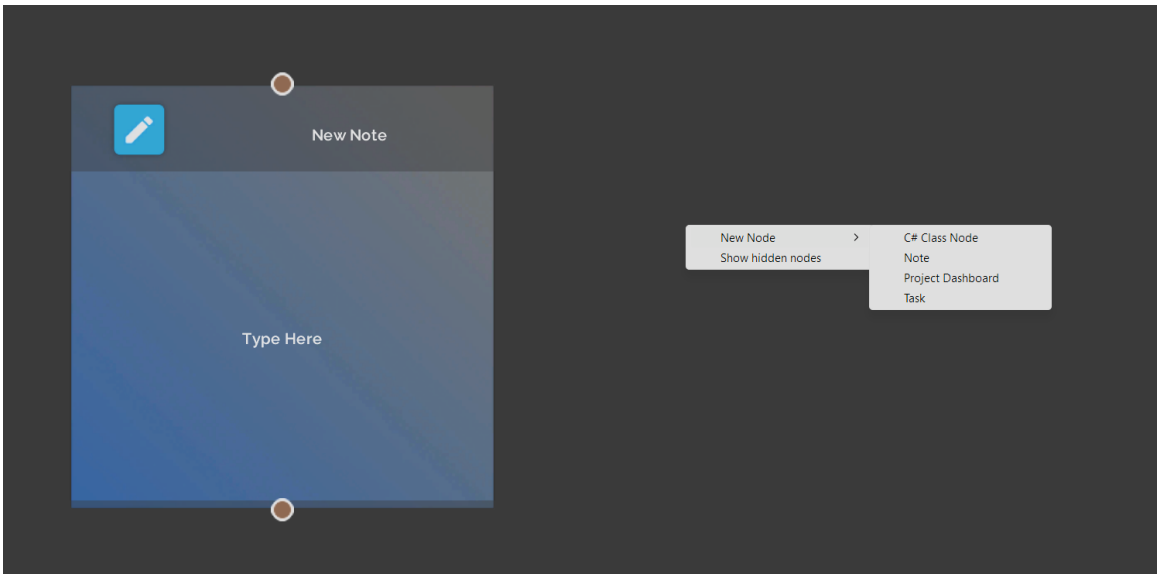


Figure 1: Notepad display name shows as "Note"

2 How to Add New Connection Types?

Very similarly to the NodeBase, all connections are created from ConnectionBase abstract class. Defining the class is a simple process:

Raleway-TLF

```
public class InheritanceConnection : ConnectionBase
{
    protected override void DrawConnection(IEditorContext context, Vector2 fromOutputScreenPos, Vector2 toInputScreenPos, Vector2 fromCenterScreenPos, Vector2 toCenterScreenPos, Color color)
    {
        Vector2 start = fromOutputScreenPos + (toInputScreenPos - fromOutputScreenPos).normalized * 10;
        Vector2 end = toInputScreenPos - (toInputScreenPos - fromOutputScreenPos).normalized * 20;
        GUIUtilities.DrawLine(start, end, color);
        GUIUtilities.DrawTriangle(end, 15f, color, toInputScreenPos - fromOutputScreenPos, false);

        Vector2 midPoint = (fromOutputScreenPos + toInputScreenPos) / 2;
        GUI.Label(new Rect(midPoint, new Vector2(60, 18)), "inherits");
    }
}
```

Overriding the DrawConnection function is more than enough to create the new Connection type. The only thing to do now is to enable the connection type for the NodeBase you want.

Raleway-TLF

```
//This must return true for the NodeBase to have any connections at all.
protected override bool CanHaveConnections()
{
    return true;
}

//If you want a NodeBase to have only InheritanceConnection.
protected override bool CanHaveConnectionOfType(Type connectionType)
{
    return connectionType is InheritanceConnection;
}
```

2.1 How to Add Metadata to Connections?

To change the display name of the connection type, one can use ConnectionBaseMetaData attribute.

Raleway-TLF

```
 // The menu option will show "inherit".
 [Serializable, ConnectionBaseMetaData("Inherit")]
 public class InheritanceConnection : ClassConnection
```

3 How to Add New Member Types?

A member is the smallest part of the Node that accomplishes a simple task or visualizes basic data about the node or the project. The class defining the members is... yes MemberBase class. MemberBases are created the same way as NodeBase and ConnectionBase.

Raleway-TLF

```
[Serializable]
 public class CommentMember : MemberBase
 {
     // Save the text.
     [SerializeField]
     private string _text;
     [SerializeField]
     private string _header;

     public override void Draw(IEditorContext context, NodeBase parent, float
   width)
     {
         // Enable text area when the node is expanded.
         if (parent.IsExpanded)
         {
             _text = CustomGUILayout.TextArea(_text, Id, LabelStyle);
         }
         // And just show the label when it's not.
         else
         {
             CustomGUILayout.Label(_text, LabelWithRichTextStyle);
         }
     }

     public override MemberBase Copy(NodeBase parent)
     {
         return new CommentMember(this);
     }
```


MemberBases must be serializable and copyable. Also it's advisable do editable operations when the Node is **expanded**. Expanded state means the node is inflated to its max size and is in edit mode.

MemberBases should be added manually to NodeBases. One can either take advantage of OnContextClick virtual function in NodeBases or just add the member on creation.

Raleway-TLF

```
    // Add the member on creation...
    protected override void OnAddedInternal(IEditorContext context, DrawableCreationType drawableCreationType)
    {
        // Default Creation type means the node is not created by copying or from an asset. More information can be found on code comments.
        if (drawableCreationType == DrawableCreationType.Default)
        {
            AddMember<CommentMember>();
        }
    }

    // Or add with a context action.
    protected override void OnContextClick(IEditorContext context, Vector2 position, GenericMenu menu)
    {
        // ProcessAction will be explained later.
        menu.AddItem(new GUIContent("New Comment"), false, context.ProcessAction, new AddMemberAction(this, new CommentMember()));
        menu.AddItem(new GUIContent("New Image"), false, context.ProcessAction, new AddMemberAction(this, new ImageMember()));
    }
```

4 How to Make Editor Actions Undo-Redoable?

Most actions done on the editor window is undo-redoable. This is done with a class called EditorContextAction. Each EditorContextAction is saved to the EditorContextHistory after being "done" so it can be undone later.

EditorContextAction classes must be feed into IEditorContext.ProcessAction function to behave properly.

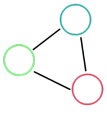
Raleway-TLF

```
//EditorContextActions must be serializable.
[Serializable]
public class HideNodeAction : EditorContextAction
{
    [SerializeReference]
    private NodeBase _nodeBase;

    public HideNodeAction(NodeBase node) : base("Hide Nodes", $"{node} is hidden.")
    {
        Debug.Assert(node != null);
        _nodeBase = node;
    }

    //Implement Do function,
    public override void Do(IEditorContext context)
    {
        Debug.Assert(_nodeBase != null);
        _nodeBase.Hide();
    }

```

```
        //And implement Undo function.
        public override void Undo(IEditorContext context)
        {
            Debug.Assert(_nodeBase != null);
            _nodeBase.Show();
        }
    }
```

After creating the class, use IEditorContext to pass actions. IEditorContext is accessible with most of the defined virtual functions.

Raleway-TLF

```
        // When the user hits Ctrl+Z, the Undo from HideNodeAction will be called.
        protected override void OnContextClick(IEditorContext context, Vector2 position, GenericMenu menu)
        {
            menu.AddItem(new GUIContent("Hide"), false, context.ProcessAction, new HideNodeAction(this));
        }
```

5 How to Create Nodes From Assets?

One neat feature of the Project Designer+ is users can create nodes directly from assets. NodeBaseAssetMap attribute is called for a certain asset type (UnityEngine.Object), when said asset is dragged into the editor.

NodeBaseAssetMap can be used on static methods anywhere to register for asset creation. Below code example is used for creating a "Nodepad" from a texture in project browser.

Raleway-TLF

```
[NodeBaseAssetMap(typeof(Texture2D))]
private static NodeBase CreateFromTexture2DAsset(UnityEngine.Object textureAsset)
{
    Texture2D texture = (Texture2D)textureAsset;
    Notepad notepad = new Notepad();
    notepad.AddMember(new ImageMember(texture));
    return notepad;
}
```

NodeBaseAssetMap constructor also accepts an int as priority to override default implementations.

Figure 2: A "Class" node created from NodeBaseAssetMapAttribute

6 How to Add New Options to Default Context Menu?

It is possible to add completely new type of drawable items to the Project Designer+ using IDrawable interface.

Raleway-TLF

```
public interface IDrawable
{
    //Draw and rect are the important functions for element to be drawable on the editor.
    void Draw(IEditorContext context, Vector2 screenPosition, Vector2 mousePosition);
    Rect Rect { get; }
    string Id { get; }
    bool IsHidden { get; }
    void Hide();
    void Show();
    void OnAdded(IEditorContext context, DrawableCreationType drawableCreationType);
    void OnRemoved(IEditorContext context);
    bool Match(string searchQuery);
}
```


Once you have a new IDrawable class to work with, we have to add it to the editor somehow, as it won't be collected with reflection like NodeBases. At this point, another attribute comes to our help. DefaultContextHandler attribute.

DefaultContextHandler is an attribute used on static methods to be called when the user right clicks on an empty space on editor. Below example shows one of the default options added to the context menu.

Raleway-TLF

```
[DefaultContextHandler]
static void ShowHiddenDrawables(IEditorContext context, Vector2 position,
  GenericMenu menu)
{
    // Performs an undo-redoable action.
    menu.AddItem(new GUIContent("Show hidden nodes"), false, context.
  ProcessAction, new ShowNodesAction(context.GetDrawables<NodeBase>()));
}
```

In conclusion, using DefaultContextHandler, IDrawable, and EditorContextActions, one can add completely unrelated things to the editor without breaking anything.

7 How to Save IDrawables?

The Project Designer+ enables saveability of IDrawables by using **SerializeField** and **SerializeReference** attributes Unity provides. But, sometimes we may not want to save IDrawables directly, and instead want to save them some else. In fact, this is the case for ConnectionBase. even though it's an IDrawable, it is not saved directly and instead it is saved in NodeBases where it's stored.

For this exact purpose, IDrawables can also implement ISerialized interface. It is a simple interface to implement. Below shows the default implementation for NodeBase.

Raleway-TLF

```
public void Serialize(List<ISerialized> serializeds)
{
    //mark to be saved.
    serializeds.Add(this);
}

//Add connections to drawables as they aren't saved in the project.
public void Deserialize(List<IDrawable> drawables)
{
    drawables.Add(this);
    foreach (var connection in GetConnections())
    {
        drawables.Add(connection);
    }
}
```

For any problem, feature request, advice or documentation request reach us:

birch_games@hotmail.com
furkancaglayan15@hotmail.com

Visit our site at:
birchgames.com