Magic Auto Linker

(Anyone can comment)

Please check online documentation for most updated version:
https://docs.google.com/document/d/1hCXerp9d1OQlYknq5roHTgNa9YUYg3jIoLuurTUhELA

Index

- Index
- What is MagicAutoLinker?
- Story
- How to use
  - Installation
  - Implementation
- Known Issues
- Extra
- Reference
  - AutoLink attribute
    - Syntax
    - Properties
  - AutoLinker component
    - Properties
    - Buttons
- Troubleshooting
  - PostSharp Serialization error on iOS runtime

What is MagicAutoLinker?

MagicAutoLinker links your scene objects to your code references automatically in runtime. You don't have to drag&drop hundreds of scene objects to inspector fields of your script component. It also supports subclasses and collections(array or list) in your scripts. **This is very useful especially on GUI objects and scripts because they tend to have hundreds of references between them.**

All the magical linking work is done in runtime. It doesn't mess with your game objects or scene structure in edit mode. However MagicAutoLinker introduces a preview window in edit mode for you to be able to see the possible runtime links in your scene even before starting the play mode.

MagicAutoLinker is for everyone who can write simple scripts.

Story

(My paraphrasing from Stephen Hawking's article. For original : Link)

In the beginning, there was only darkness, water, and the great god Bumba. One day Bumba, in pain from a stomach ache, vomited up the Visual Basic. Still in pain, Bumba vomited up the Delphi and other visual programming environments. In those early days, programmers were creating some visual objects in their project and the visual environment was creating the links between the visual objects and the code behind it automatically. Those were the great awesome days. When the Unity came to be, the good days were over. For developing games on Unity, the programmers had to do a lot of ugly and repetitive work to be able to reference their scene objects in the code.

Those repetitive work included many steps. First, they were creating the objects in the scene. Second, they were writing their scripts with reference fields. Then they were dragging&dropping hundreds (maybe thousands) of scene objects onto the reference fields in the inspector window. These were very painful and hard to maintain way of working.

MagicAutoLinker is created for you to end this great pain of dragging&dropping scene objects. You just need to mark the reference fields in your scripts with [AutoLink] attribute and AutoLinker does the rest magically :) :) :)

How to use

Installation

1. Just add the CreatedMessageSender prefab into your scene. You can read WeNeedCreatedMessage's readme file for more information.
2. Decorate your reference fields with [AutoLink] attribute. See the implementation section for more info.
3. Add AutoLinker component to game objects which contains your AutoLinker dependant scripts.
4. (iOS only) Set your project's API compatibility level to full .NET. Don't use the subset level.

That's it! When you start the play mode, your scene objects will be linked to the reference fields in code automatically. The reference fields don't even have to be "public" to get processed by AutoLinker. They can be private. If you plan to serialize them, you can still make them public or decorate them with [SerializeField] attribute.

Implementation

AutoLinker component must be added to the same game object along with this script. Then all you need to do is decorate your fields with [AutoLink] attribute.

```
[RequireComponent(typeof(AutoLinker))]
public class YourScript : MonoBehaviour
{
	[AutoLink]
	TextMesh UserName;
	// A game object named as "UserName" must be the child of the object
	// which this script is attached to in scene hierarchy.

	[AutoLink(GameObjectName = "AvatarPortrait")]
	MeshRenderer Avatar;
	// You can override the gameobject name with GameObjectName property
	// of AutoLink attribute so the AutoLinker looks for the new name
	// instead of the default field name.

	void Start()
	{
		UserName.text = "Guest User";
		// You don't get a null exception here because
		// AutoLinker links the "UserName" gameobject to
		// the UserName field in this script even before
		// the start event sent by Unity engine in runtime.
```

```
        }
}
```

Known Issues

- You can see “GUI error” when you switch between scenes in the edit mode if the preview window was open. This is totally harmless and will be fixed in an upcoming version.

Extra

MagicAutoLinker includes WeNeedCreatedMessage package which can be found here. It registers itself to CreatedMessageScriptExecutionOrder object to guarantee that AutoLinker runs before all other scripts so you don't get any null exceptions in your scripts.

Reference

AutoLink attribute

This attribute indicates that the class member needs to be processed by AutoLinker. It is the only thing you need to declare in your code. No other coding is required for AutoLinker to work.

Syntax

```
namespace Atesh.AutoLinker
{
	[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property)]
	sealed class AutoLinkAttribute : Attribute
	{
		// members...
	}
}
```

Properties

All properties are optional.

GameObjectName : You can use this property to override the default member name to seach in the scene. If the member is a collection (Array or List) the value of this property is considered as container name in the scene.

CollectionElementName : If the member is a collection, this property defines the each collection item's name prefix.

IndexOffsetForCollectionElement : Defines the first index number of the element names of a collection. Default is 1.

NumberOfDigitsForCollectionElement : Defines the number of digits at the end of the element names of a collection. Default is 1.

Description : A simple description for the element that will be auto linked. It also appears in the debug messages if the AutoLinker can't find any object for that member. You can use this to show a warning message to your scene/level editors.

AutoLinker component

This is the only component you need to add to the game objects which contains scripts you want to be processed by AutoLinker.

Properties

Self Destroy: If this property is checked (true), AutoLinker removes itself from the gameobject as soon as it completes the auto linking process. This is useful in some cases like template objects because you only want the template objects to be auto linked. The instance objects which are created from the template shouldn't be auto linked. This example is only true if every member of every script on the template object is serializable. Unity will handle the preserving of links on the created instance objects. If the script members aren't serializable, you can't use self destroy option and each instance of the template object must be processed by the AutoLinker separately.

Buttons

Preview and Preview all buttons open up the AutoLinker Preview window so you can see the possible runtime links even before starting the play mode.

Troubleshooting

1. PostSharp Serialization error on iOS runtime

You may encounter this runtime error on iOS platform if your project's API compatibility level is .NET subset. Please use full .NET level as your compatibility level. Also note that this rule only applies to iOS platform.