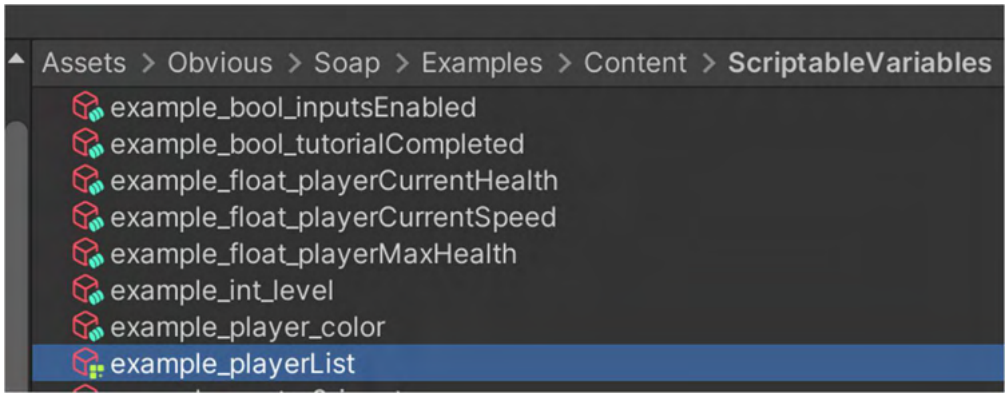
Soap:
3_ScriptableList_Example_Scene

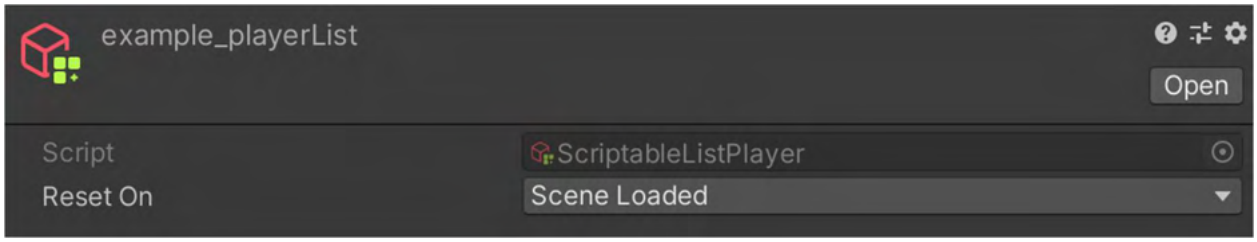
Table of Contents

Adding/removing elements

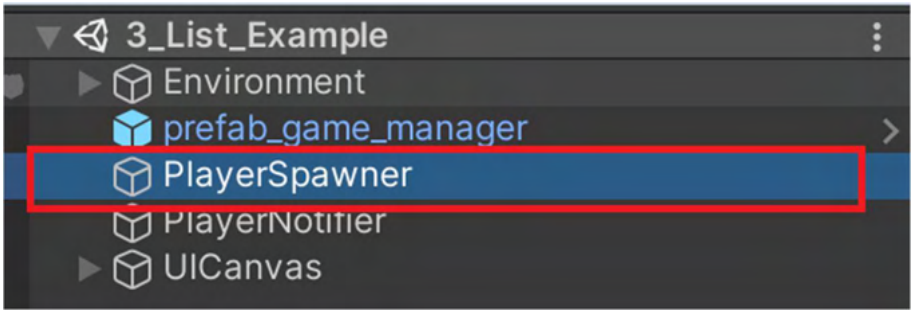
In this example we made a scriptable list of type **Player**. Find the **example_playerList** asset in the project.

Obvious/Soap/Examples/Content/ScriptableVariables /

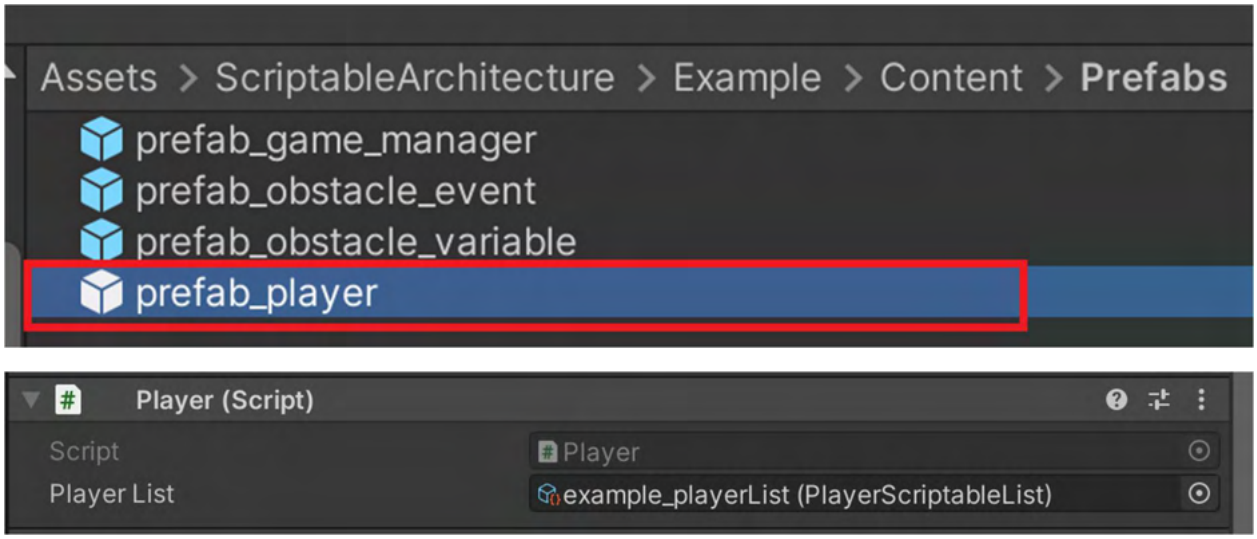




In the scene, we are going to spawn players and then destroy them. This behavior happens in the **PlayerSpawner.cs.** Using the 2 buttons on the screen (Spawn and Destroy) we are calling methods on this class.



If you check the **prefab_player**, you will notice that the Player.cs has a reference to this list.



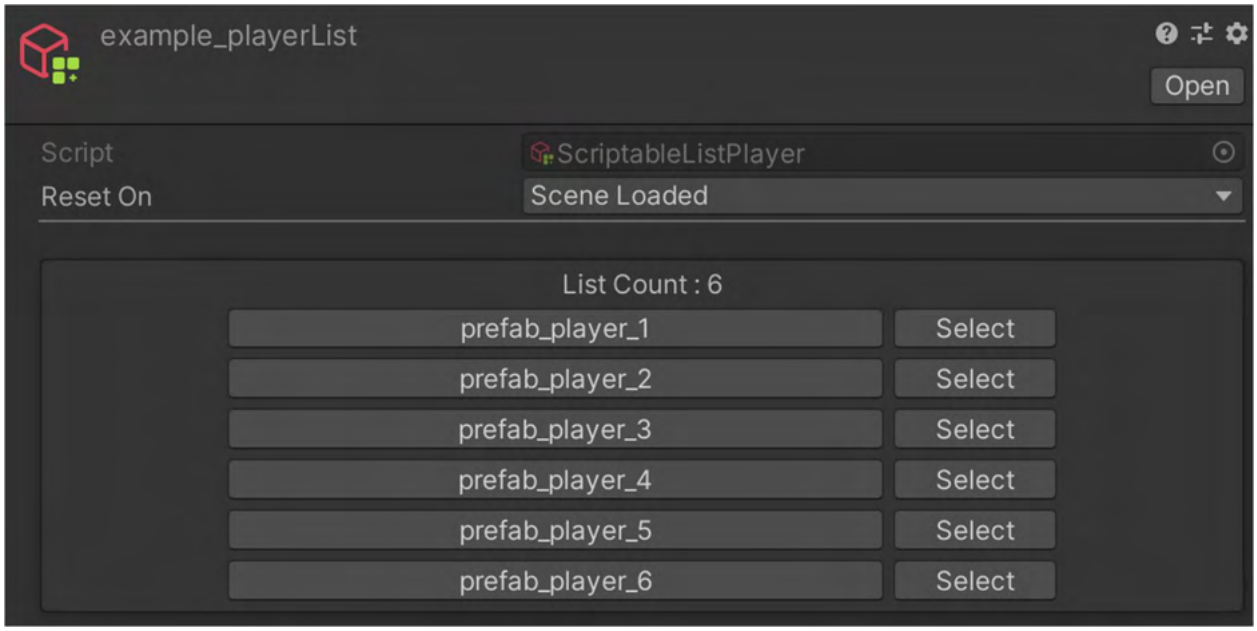
He adds himself on Awake() and remove itself OnDestroy().

```
private void Awake()
{
    scriptableListPlayer.Add(item: this);
}

private void OnDestroy()
{
    scriptableListPlayer.Remove(item: this);
}
```

You can reference this list in other classes and access it without needing to know how it is populated, and without needing a “manager” to hold it.

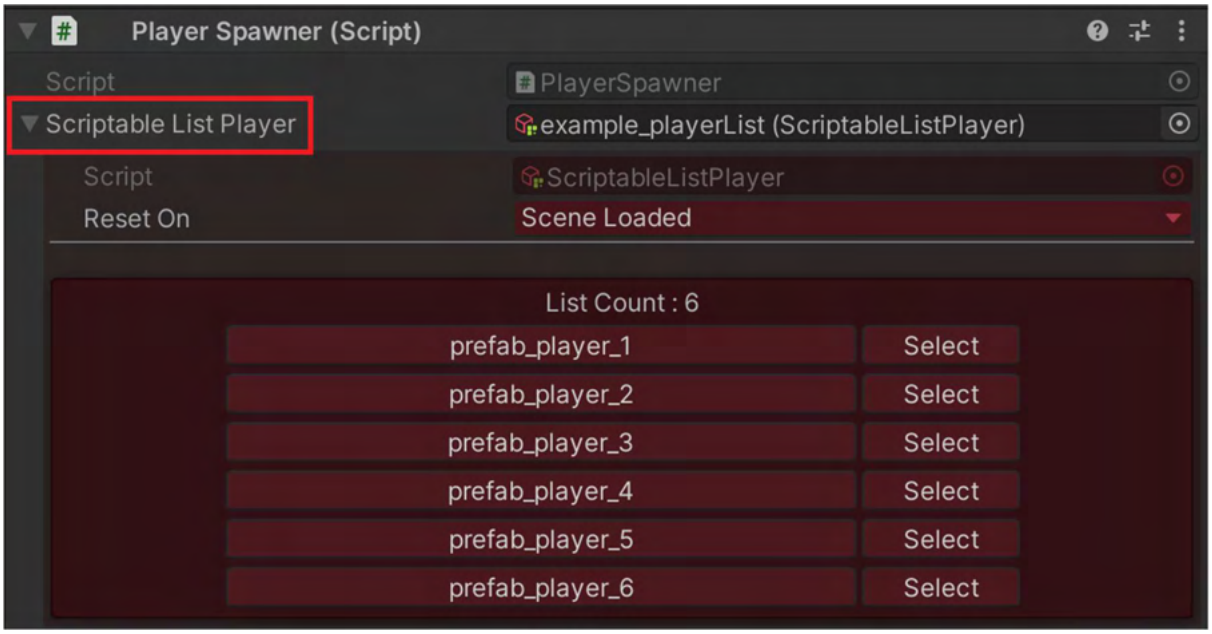
If you select the ScriptableList in the inspector, you will see the list of players in real time.



By clicking on the first button (with the name of the object), it pings the object in the hierarchy. By clicking on the "Select" button, it will select the object in the hierarchy.

You can also call common methods that you would call on a normal list, like Add, Remove, AddRange, RemoveRange and Clear.

Moreover, by clicking on the small arrow you can expand the inspector showing the Scriptable list inspector.



Subscribe to list Events

The Scriptable List has several events that you can subscribe to by code.

1) OnItemAdded / OnItemRemoved

Useful if you want to do different things depending on if an item is added or removed. To see an example of this, click on the **VfxSpawner.cs.**

```
public void Awake()
{
    scriptableListPlayer.OnItemRemoved += OnPlayerDestroyed;
    scriptableListPlayer.OnItemAdded += OnPlayerSpawned;
}

public void OnDestroy()
{
    scriptableListPlayer.OnItemRemoved -= OnPlayerDestroyed;
    scriptableListPlayer.OnItemAdded -= OnPlayerSpawned;
}

private void OnPlayerSpawned(Player player)
{
    Instantiate(_spawnVFXPrefab, player.transform.position, Quaternion.identity, transform);
}

private void OnPlayerDestroyed(Player player)
{
    Instantiate(_destroyVFXPrefab, player.transform.position, Quaternion.identity, transform);
}
```

Here we want to display a different VFX depending on if a player has been spawned or destroyed. The player argument is useful, as we can get its position and spawn the VFX there.

2) OnItemCountChanged

This event is useful if you only care about when the number of elements in the list is changed.

The example for this one is the **ListCount.cs**

Scene Hierarchy: *UICanvas/Bottom/ListCount*

```
void Awake()
{
    scriptableListPlayer.OnItemCountChanged += UpdateText;
}

private void OnDestroy()
{
    scriptableListPlayer.OnItemCountChanged -= UpdateText;
}

private void UpdateText(Player player)
{
    _text.text = $"Count : {scriptableListPlayer.Count}";
}
```

We simply update the text with the number of elements in the list.

3) OnItemsAdded / OnItemsRemoved

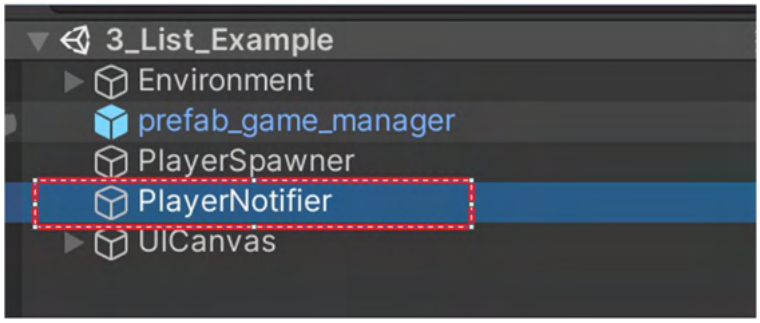
This event is useful if you don't want the OnItemAdded/OnItemRemoved event triggered every time you add/remove an item. This will be called once after a range of items have been added/removed.

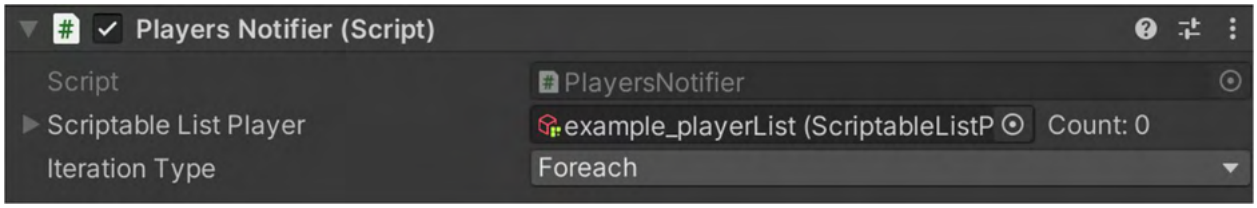
Auto clearing

Additionally to the reset mode, in the Editor, ScriptableLists automatically clear themselves when exiting play mode.

Iteration

Find the **PlayerNotifier.cs** in the scene.





In this case, we iterate through the list (using a foreach or an index iteration) and call a method on each player.

This can be useful in different cases. If you need to iterate over a list of enemies but you want independent code for things like making abilities, power ups etc.).