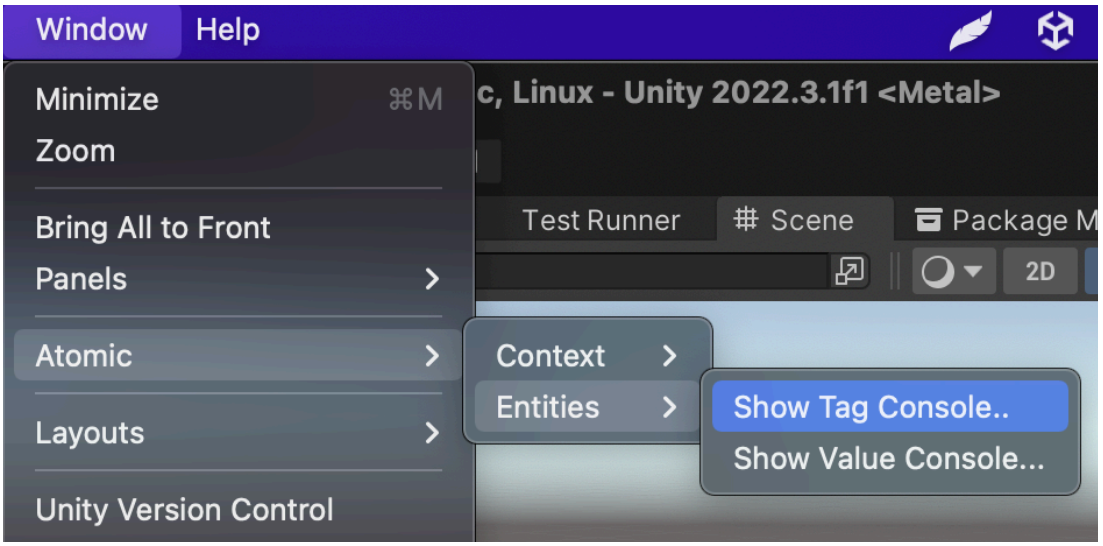
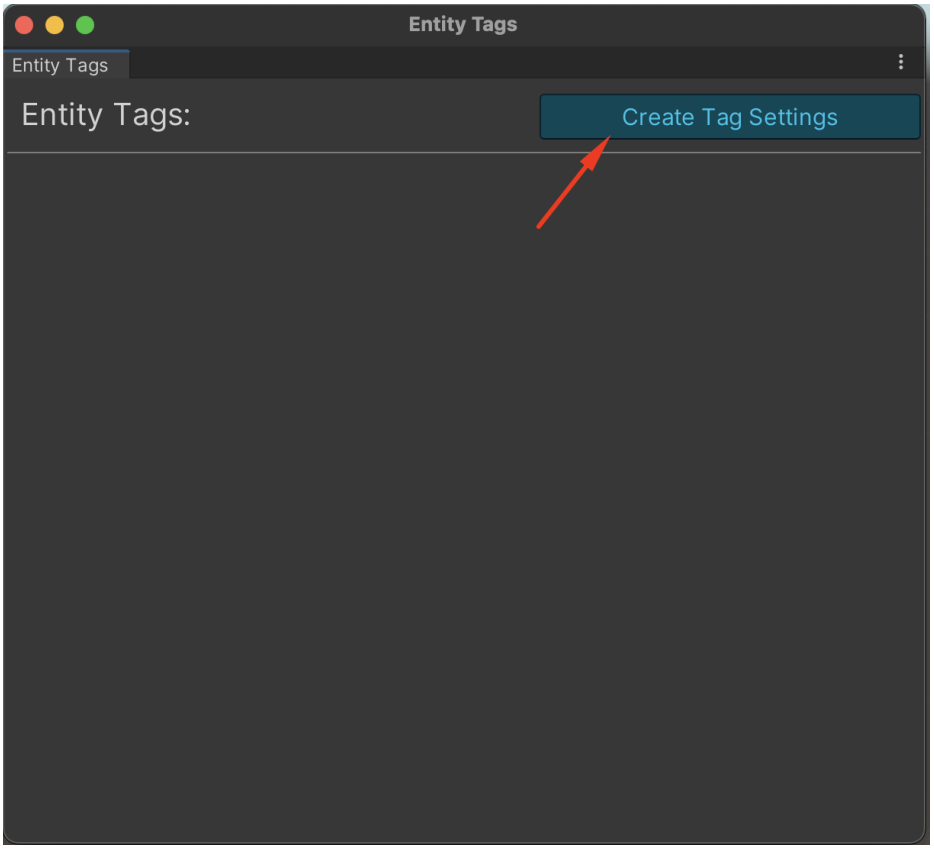
By analogy with value identifiers, you can also set up tag identifiers for your game objects.

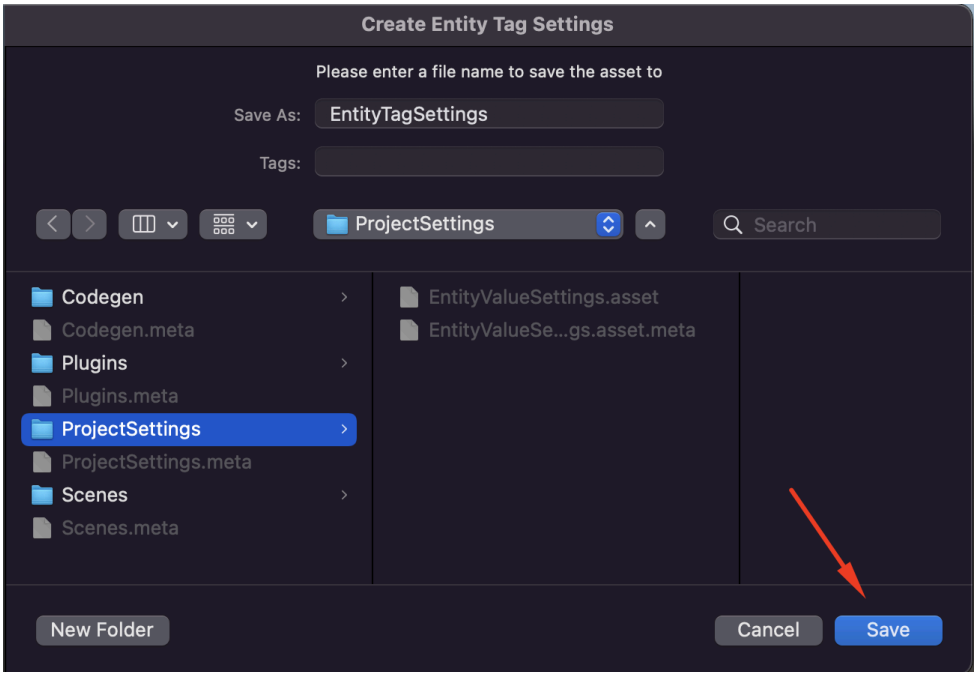
The first step is to click Window→Atomic→Entities→Show Tag Console…



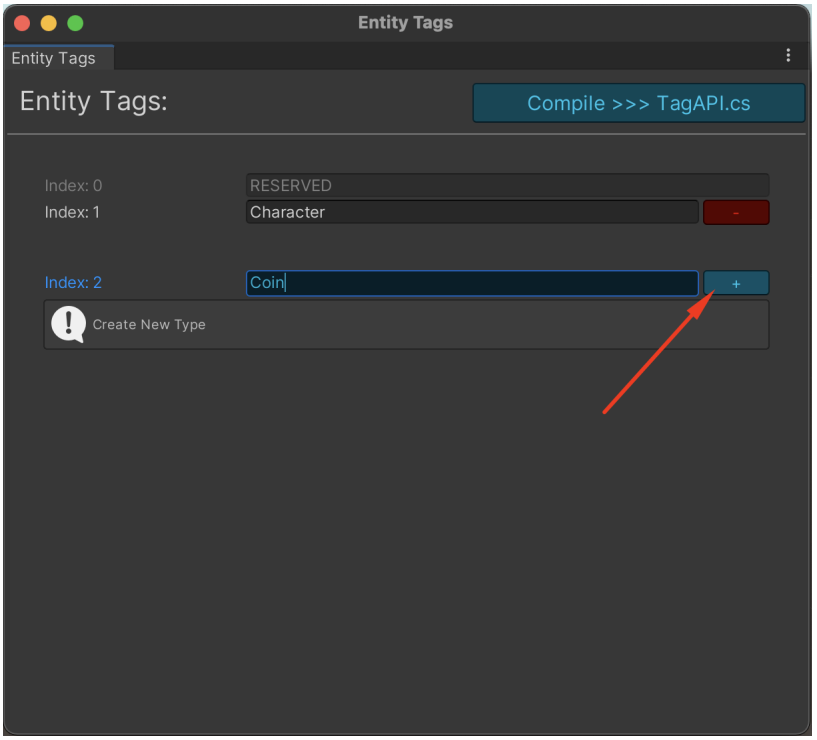
Next, click on “Create Tag Settings”, to create an asset that stores various identifiers for entity tags



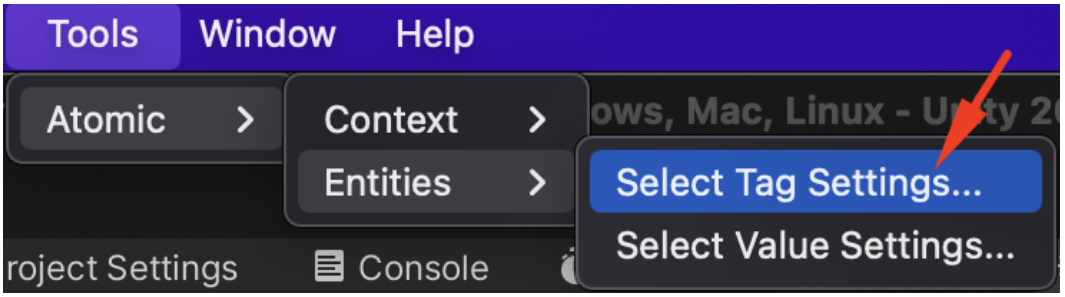
Next, save asset to the “ProjectSettings” folder



After creating an asset, the developer can create various tags for game entities

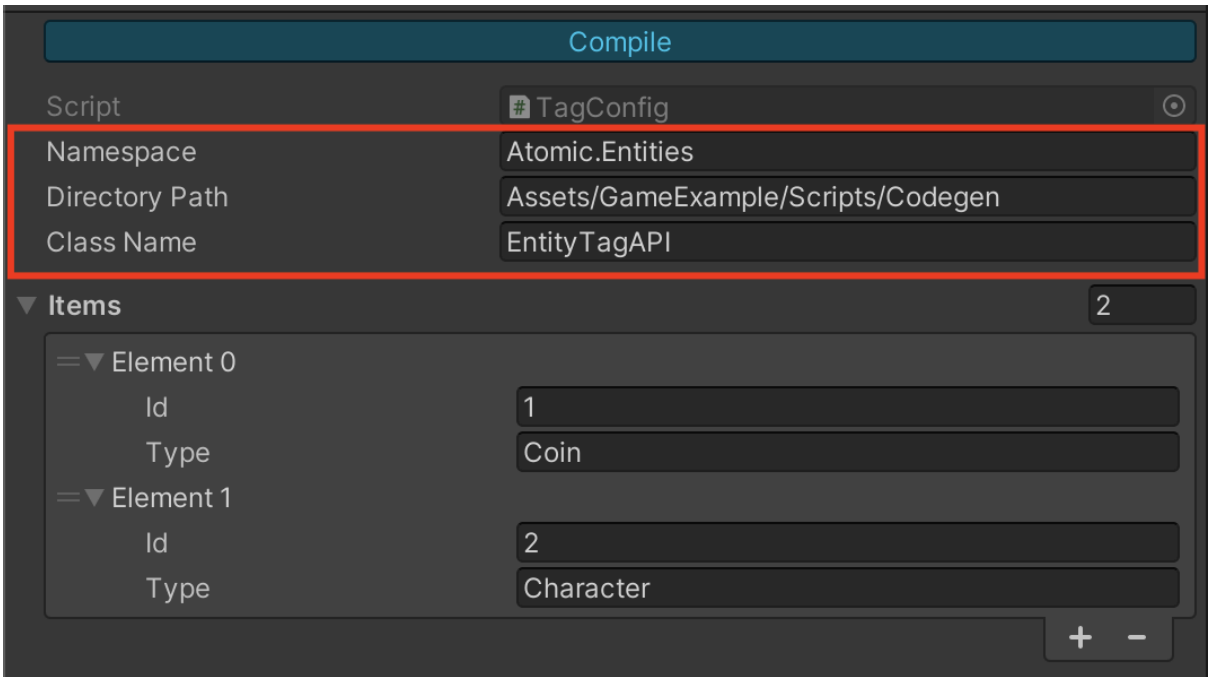


Similarly to the EntityValues settings, the developer can configure code generation for tags. To do this, he will need to click Tools→Entities→Select Tag Settings…

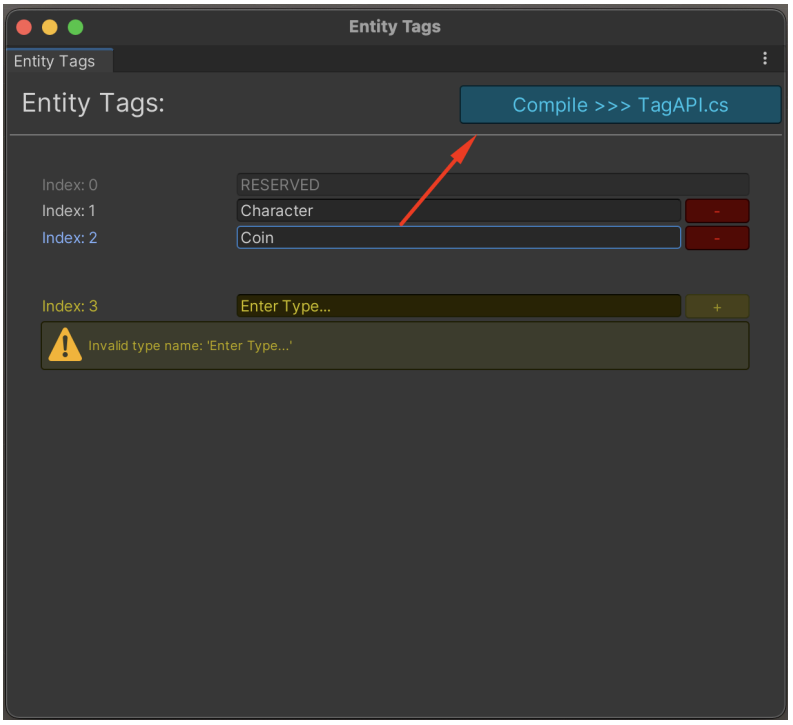


In the asset inspector, you can configure:

- namespace
- file generation path
- class name



After setting up, go back to the “Entity Tags” window and click the “Compile” button



Then the developer can use the generated extension methods, which will make development more convenient and faster:

```
public static class EntityTagAPI
{
    ///Keys
    public const int Coin = 1;
    public const int Character = 2;


    ///Extensions
    public static bool HasCoinTag(this IEntity obj) => obj.HasTag(Coin);
    public static bool AddCoinTag(this IEntity obj) => obj.AddTag(Coin);
    public static bool DelCoinTag(this IEntity obj) => obj.DelTag(Coin);

    public static bool HasCharacterTag(this IEntity obj) => obj.HasTag(Character);
    public static bool AddCharacterTag(this IEntity obj) => obj.AddTag(Character);
    public static bool DelCharacterTag(this IEntity obj) => obj.DelTag(Character);
}
```

Example of using Entity Tags…

```
entity.AddCoinTag();

if (entity.HasCoinTag())
{
    Debug.Log($"Entity {entity.Name} is Coin!");
}
```