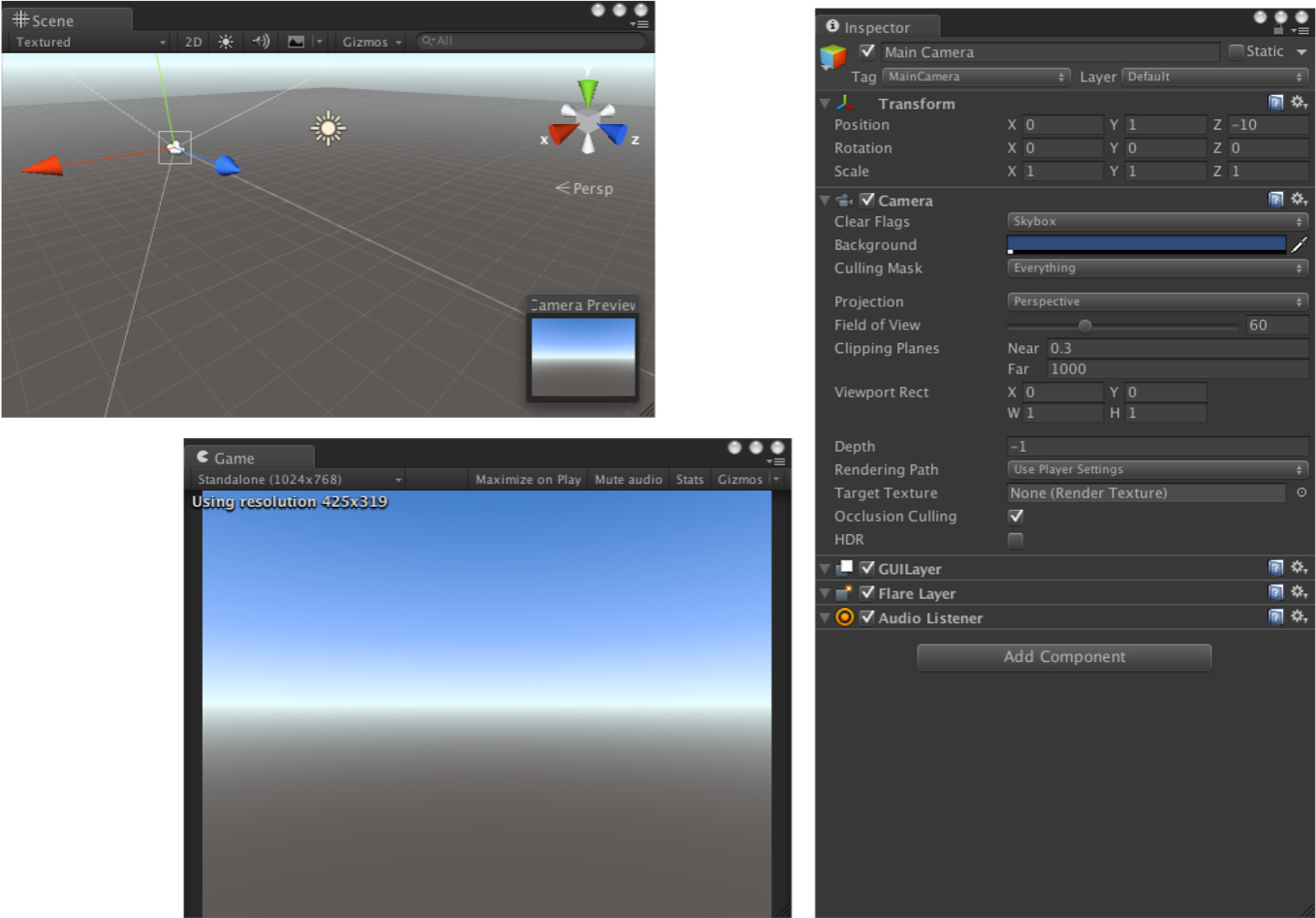
<http://anchan828.github.io/editor-manual/web/part1-editorwindow.html>

# **제 7장 EditorWindow**

에디터 확장에 처음으로 접할 때, 일단 윈도우를 표기하는 것으로부터 시작할 수도 있습니다. 이 장에서는 간단하게 EditorWindow를 표기하는 방법에서 목적에 맞는 EditorWindow의 선택, 이에 대한 특성에 대하여 해설합니다.

## **7.1 EditorWindow란**

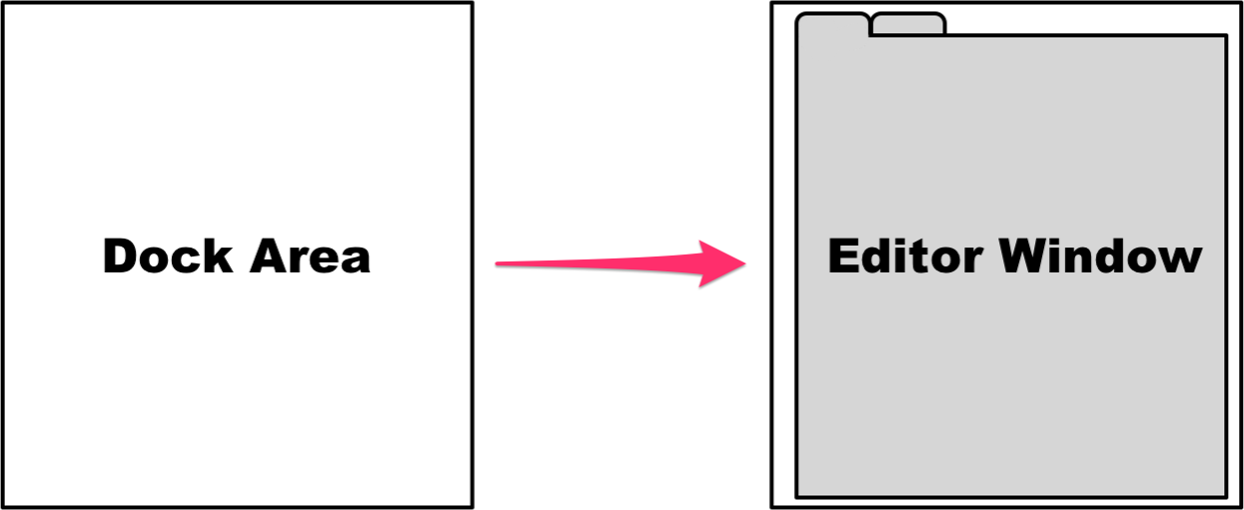
씬 윈도우, 게임 윈도우, 인스팩터 윈도우 등등, 이들을 모두 EditorWindow 입니다. 유니티 에디터는 여러 가지 기능을 가진 EditorWindow의 집합체로 되어있습니다.



이들 모두 EditorWindow

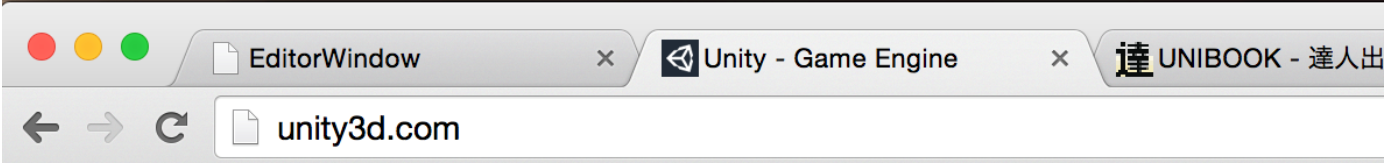
## **7.2 HostView 와 SplitView와 DockArea**

EditorWindow는 단체에서 묘화되는 것이 아닙니다. EditorWindow의 부모인 DockArea로부터 EditorWindow가 묘사됩니다.



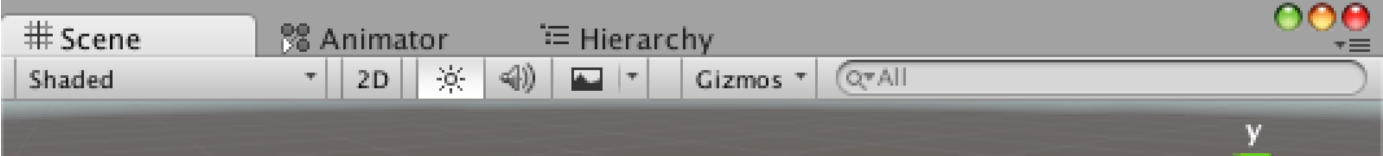
DockArea의 위에서 EditorWindow가 있는 이미지

예를 들면, 윈도우는 제 각각 독립한 3개의 윈도우로 되어있으며, 3개의 탭에 대하여 1개의 윈도우로 관리하는 것도 가능합니다.(인터넷 창을 이동할 때 탭 하나하나 이동할 수도 있지만 열려있는 탭들을 동시에 이동할 경우를 창의 상단(탭 영역을 벋어난 윗 부분)을 클릭해서 이동하는 것을 의미함.)



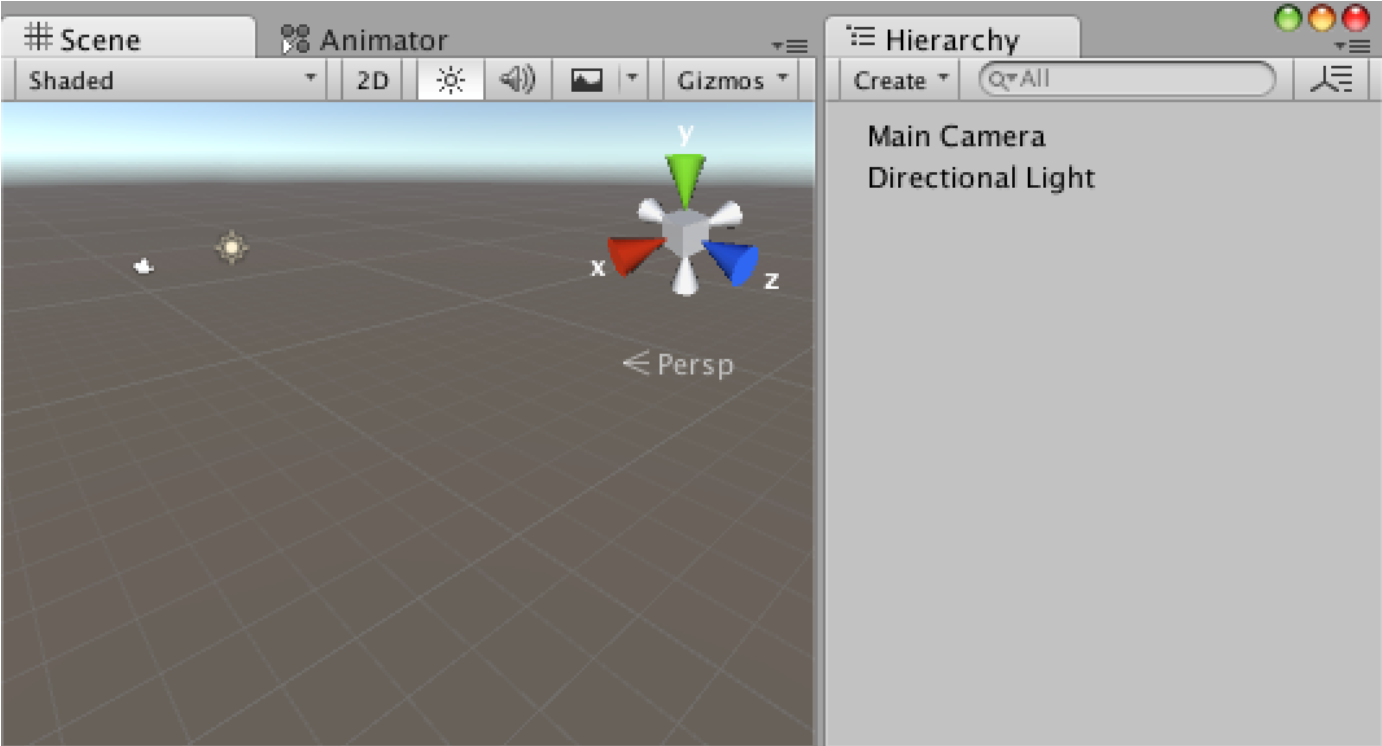
Chrome의 탭 기능

모양세도 닮으며, 탭의 기능 역시, 탭을 잡아서 별도의 윈도우로 나누는 것도 가능합니다.



DockArea의 탭 기능

이와 같이, DockArea에는 1개 이상의 EditorWindow를 묘사하기 위한 기능이 준비되어있습니다. 예를 들면, 2개 이상의 EditorWindow가 DockArea에 존재할 경우, 탭 기능을 사용하여 각 각의 EditorWindow를 표시하던지 Split Window로 DockArea의 영역을 분할하여 표시할 수 있습니다.



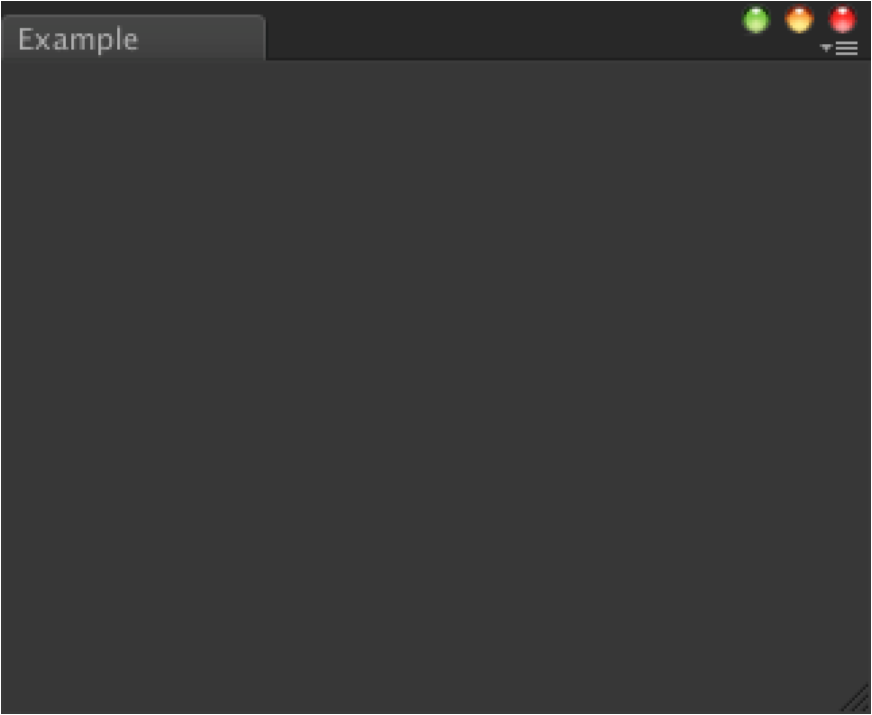
DockArea에 씬 뷰와 계층 뷰를 분할하여 표기

더욱이 DockArea는 Host View의 역할도 가지고 있습니다. Host View는, 여러 가지 객체, 이벤트와의 교신을 진행하기 위한 기능이 있습니다. 윈도우의 Update 관계나 OnSelectionChange 관계 등을 실행하기 위한 기능을 포함하고 있습니다.

3개의 윈도우 [Host View], [Split View], [DockArea]를 소개했습니다. 이들의 영역(클래스)에는 아쉽게도 일반적으로 접근할 수 없습니다. 하지만 이러한 EditorWindow의 구조를 기억해두면 이에 관련된 기능들을 보다 빠르게 이해할 수도 있습니다.

## **7.3 EditorWindow의 작성**

일단은 일반적인 EditorWindow를 작성해봅시다.

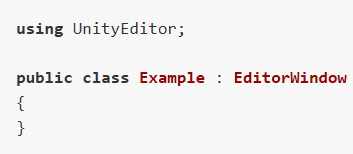


표시하기만 하고 아무것도 없는 EditorWindow

### **EditorWindow를 표시하기 전**

EditorWindow를 표시하기 전까지 기본적인 흐름은 3개의 스탭으로 구성되어있습니다. 먼저 이 3개의 스탭을 기억해야 합니다.

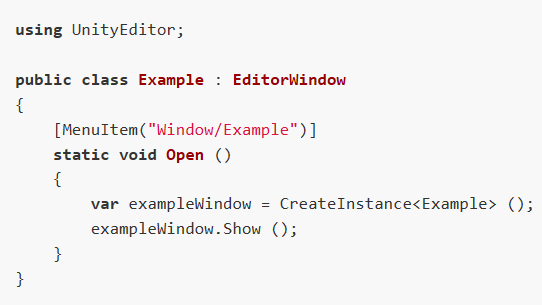
1. EditorWindow를 작성할려면 EditorWindow를 계승한 클래스를 작성합니다.

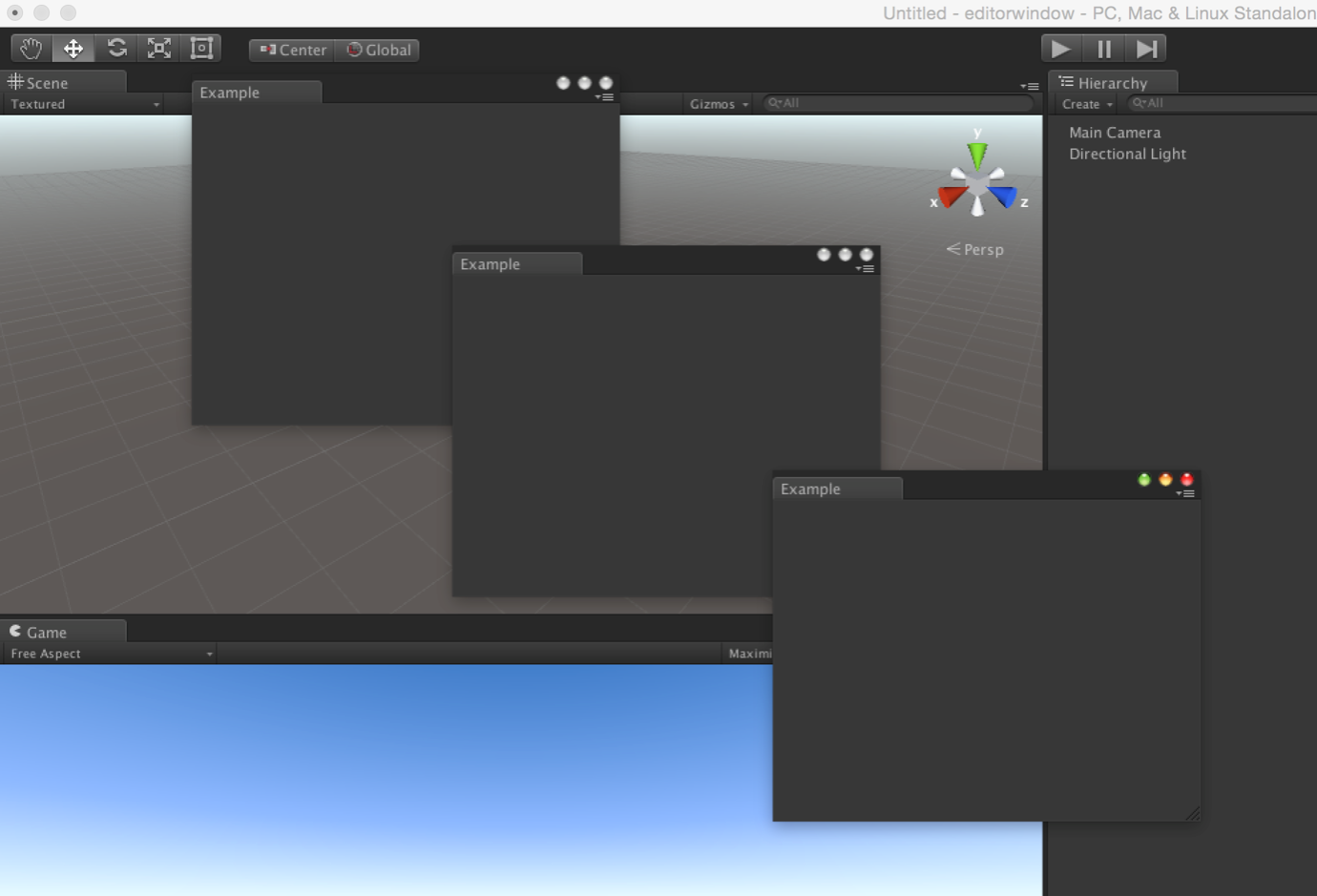


1. 다음은 EditorWindow를 표시하기 위한 트리거로써 메뉴를 추가합니다.



1. 마지막으로 EditorWindow를 표시합니다. EditorWindow는 ScriptableObject를 계승하고 있으므로, EditorWindow.CreateInstance로 EditorWindow의 객체를 생성할 수 있습니다. 그리고 Show를 호출함으로써 EditorWindow를 표시할 수 있습니다.





“Window/Example” 메뉴를 실행해서 표시할 수 있습니다.

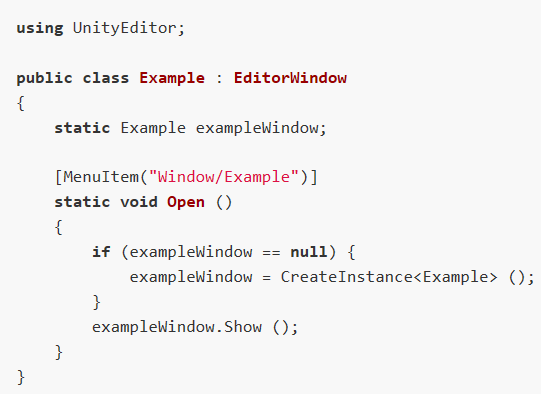
실행하기 위하여 EditorWindow를 신규 작성합니다.

## **7.4 EditorWindow.GetWindow**

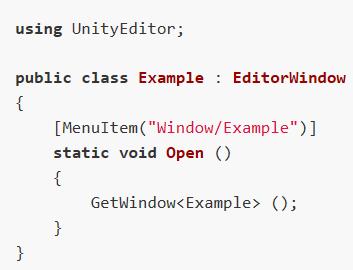
EditorWindow를 작성할 경우, [복수의 존재를 허가할 EditorWindow], [단수만을 허가할 EditorWindow]의 2종류를 생각할 수 있습니다. 복수의 존재를 허가할 EditorWindow는 앞서 설명한 EditorWindow.CreateInstance를 사용해서 EditorWindow를 작성해서 표시하고 있습니다.

### **단수만을 허가할 EditorWindow**

단수만으로 할 경우 [이미 EditorWindow가 존재할 경우 생성하지 않는다]라는 체크를 통하여 구현 방식을 결정하면 됩니다. 이 체크를 추가한 로직은 다음 코드와 같습니다.

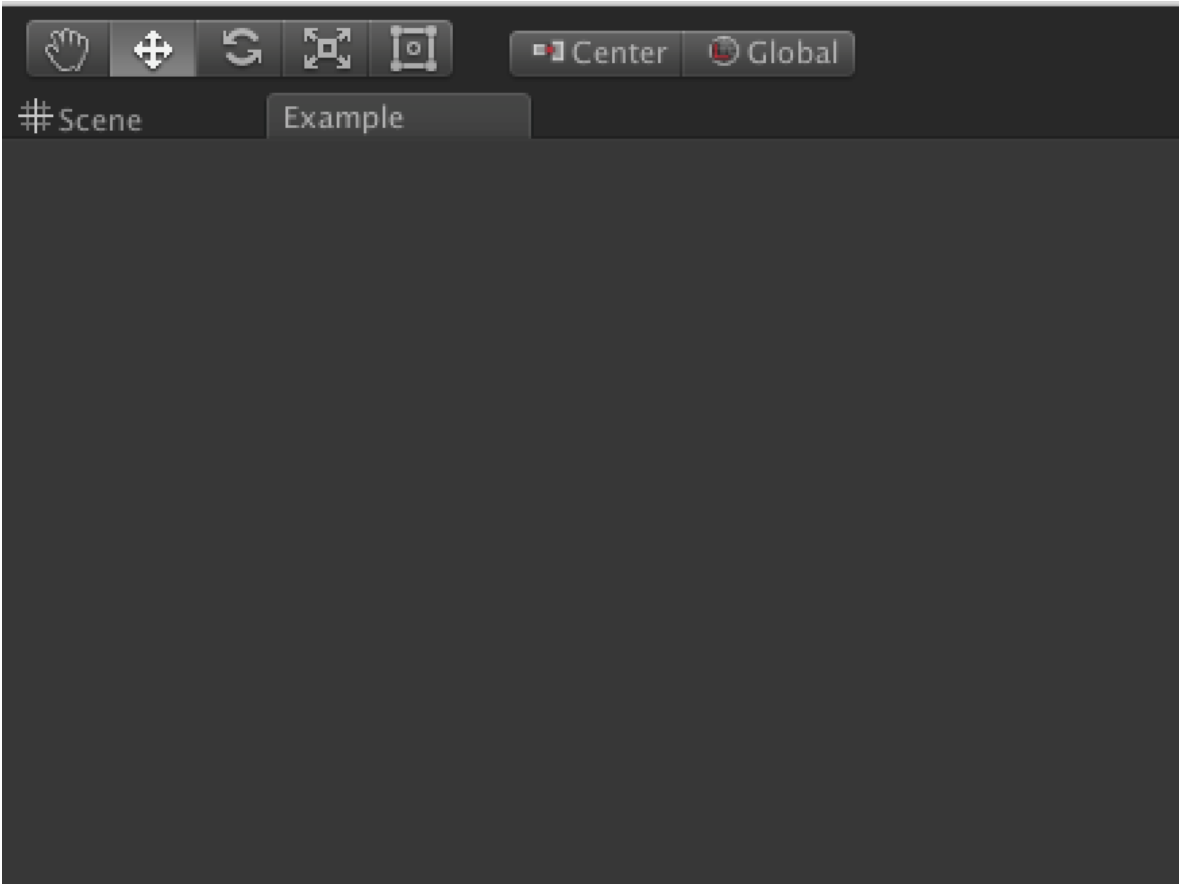


이 방식으로 작성해도 괜찮지만, [이미 EditorWindow가 존재한다면 해당 인스턴스를 취득한다. 그렇지 않으면 생성한다. 마지막에는 Show함수를 실행한다]라는 이와 같은 복수의 기능을 하나로 묶은 API가 존재합니다. 바로 EditorWindow.GetWindow 입니다.

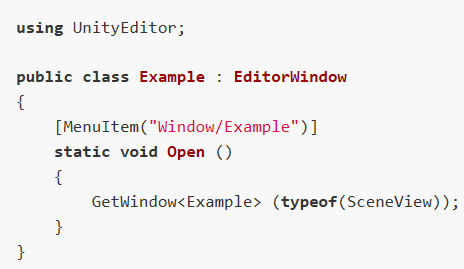


GetWindow를 실행하면 내부에서 인스턴스가 캐쉬되어집니다.

더욱이 GetWindow 함수에는 편리한 기능이 있는데, 특정 EditorWindow에 탭 윈도우로써 표시하는 것이 가능합니다.(DockArea에 EditorWindow를 추가)



씬 윈도우에 탭 윈도우로써 추가할 수 있음.



## **7.5 어떤 Show 함수를 호출함에 따라 변화는 특수한 EditorWindow**

지금까지 이 장에서 다룬 EditorWindow는 디폴트 상태로 사용되는 윈도우입니다. 이외에도 EditorWindow는 여러 가지 종류의 윈도우를 작성할 수 있습니다.

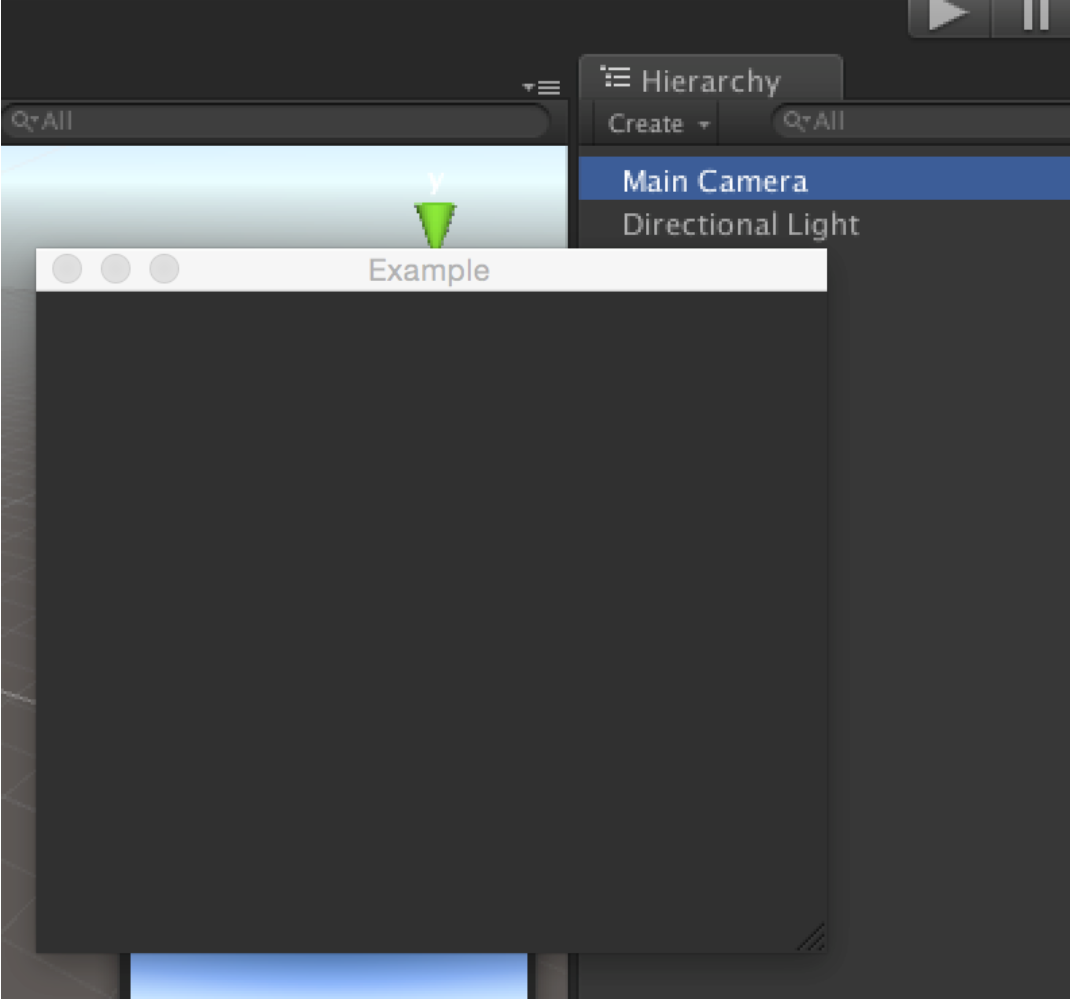
Show 함수는 복수용으로도 존재하며, 어떤 Show 함수를 호출함에 따라 표시되는 EditorWindow가 변화합니다.

### **Show**

디폴트 상태의 탭 윈도우로써 다뤄집니다. EditorWindow.GetWindow를 사용할 경우 내부에서 Show함수가 호출되어집니다.

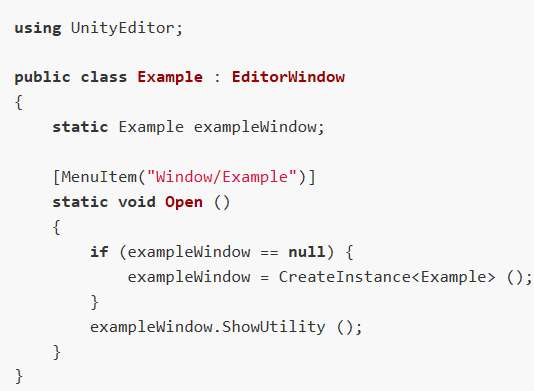
### **ShowUtility**

탭 윈도우로써 다뤄지지 않고, 보통 앞에 표기하여 진행하는 윈도우입니다. 설령 다른 윈도우에 포커스를 둔다고 해서 해당 Window 자체가 뒷면으로 빠지는 경우가 생기지 않습니다. 설정 윈도우와 같이 빈번하게 다른 윈도우를 조작하더라도 최 전면에 표시하고 싶은 경우 사용하면 됩니다.



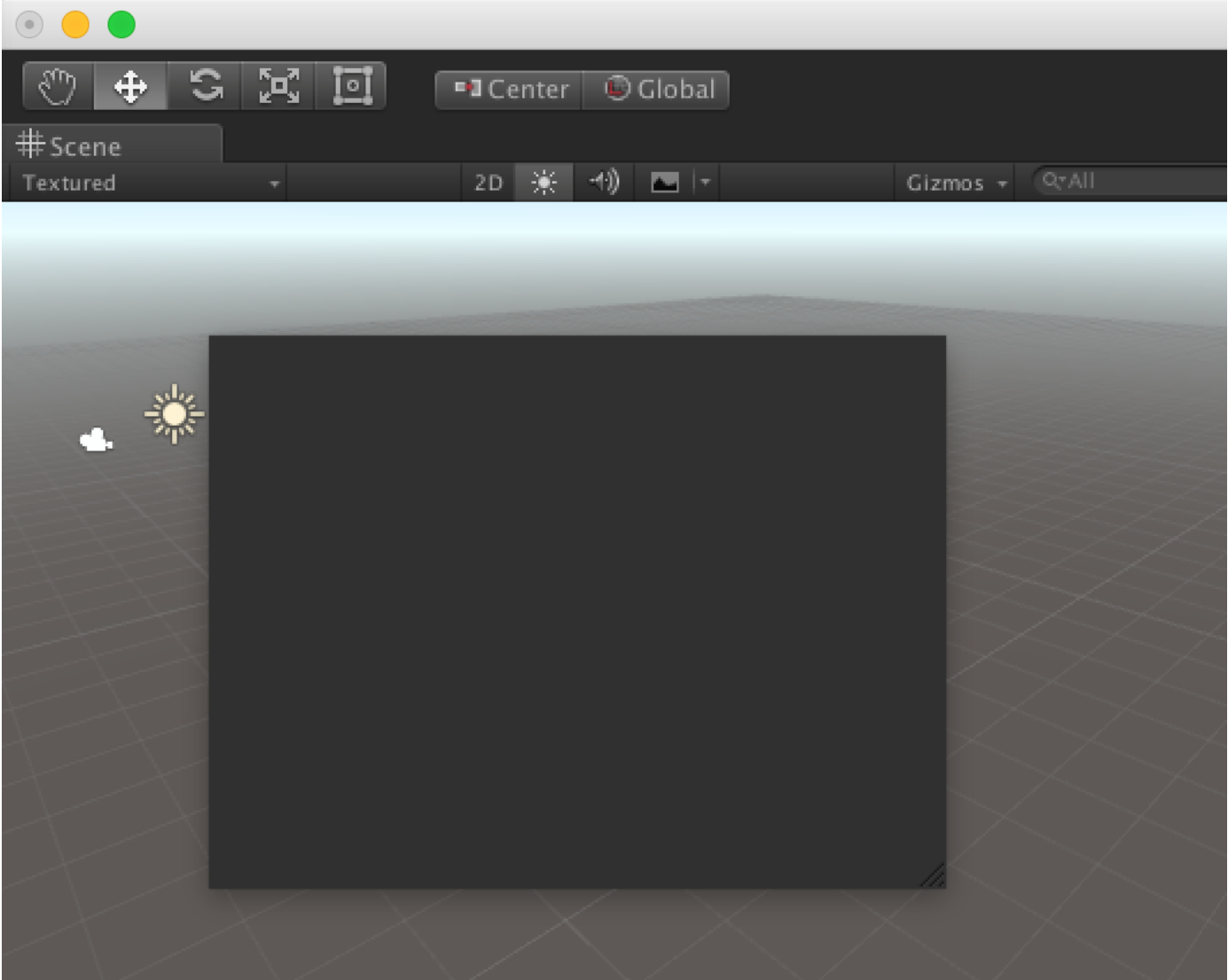
계층 뷰의 Main Camera를 선택해도 윈도우(Example)는 최 전면에 표시됨.

GetWindow는 사용할 수 없으므로 그 대신 CreateInstance를 사용합니다.

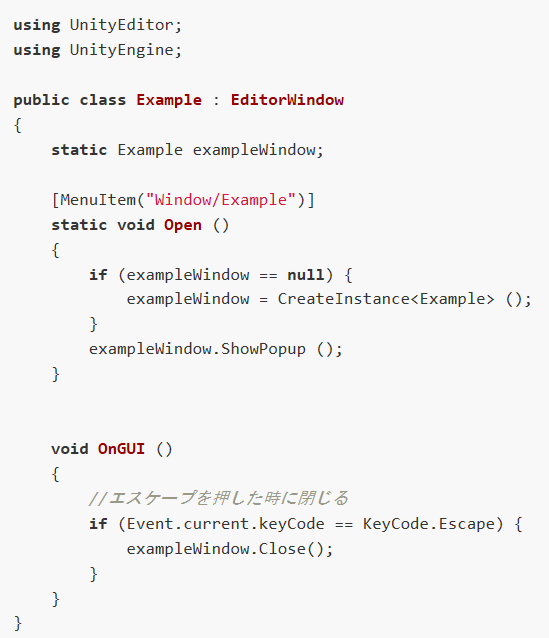


### **ShowPopup**

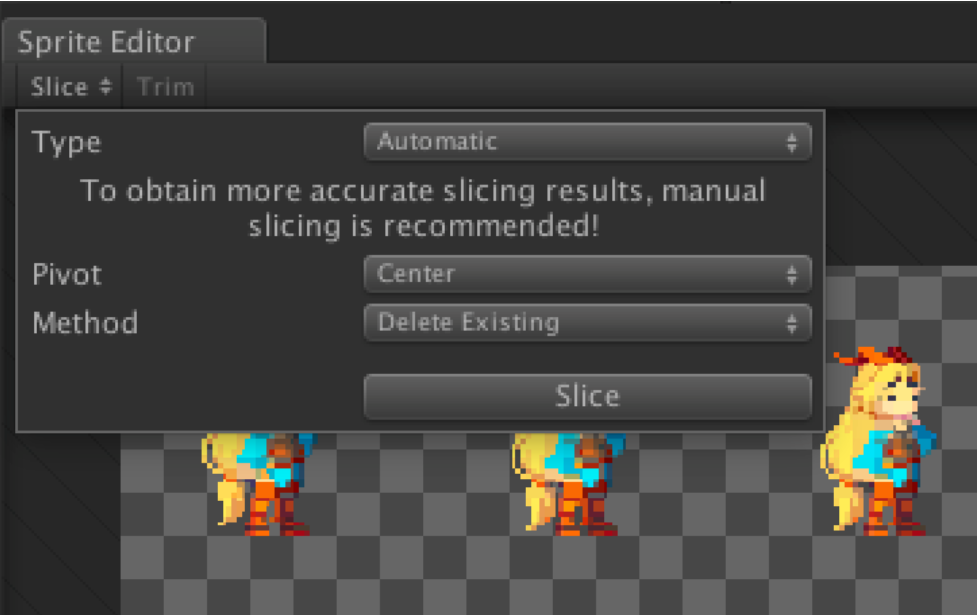
윈도우 타이틀과 닫는 버튼이 없는 윈도우입니다. 설령 다른 윈도우에 포커스를 둔다고 해도 해당 Window 자체는 뒷면으로 빠지는 경우가 생기지 않습니다. 닫는 버튼이 없기에 윈도우를 닫는 처리를 사전에 실장해야 할 필요가 있습니다.



씬 윈도우 위에 표시되어있음



사용되는 곳에 대한 예시는 Sprite Editor와 같은 슬라이스 메뉴 버튼으로부터 Popup을 표시하는 부분입니다.



이 것은 EditorWindow의 하나의 종류

### **PopupWindow**

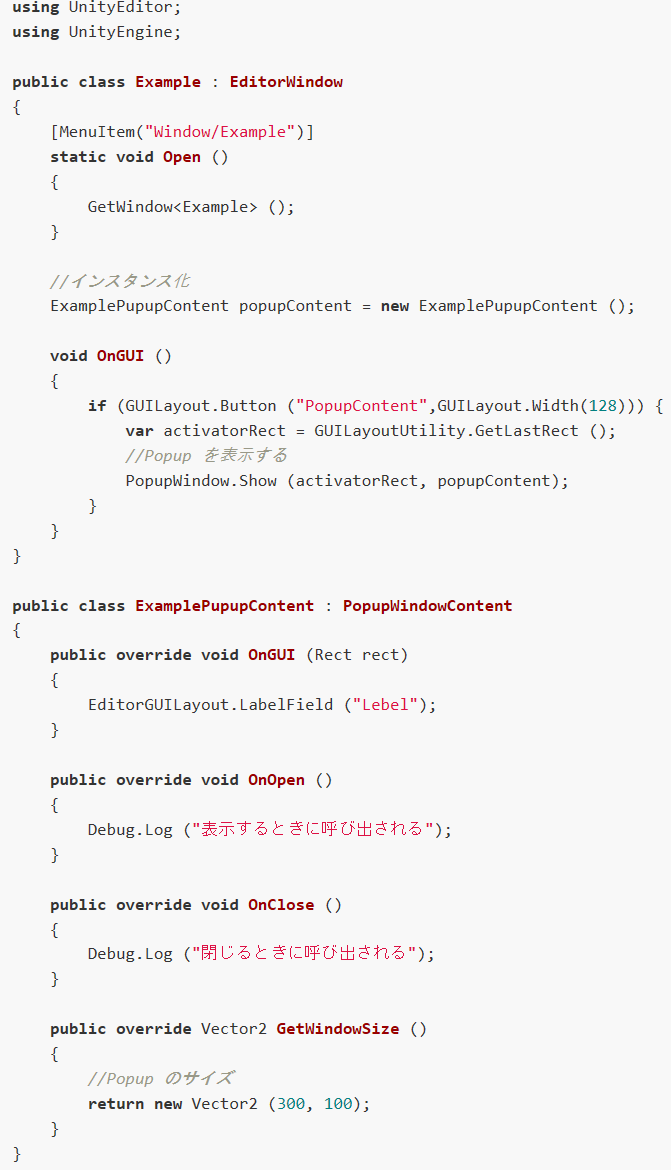
직전에 소개한 ShowPopup과 거의 같은 Popup을 표시하기 위한 기능입니다.

PopupWindow는 Popup을 범용적으로 다루기 위한 것으로 생각하시면 됩니다.



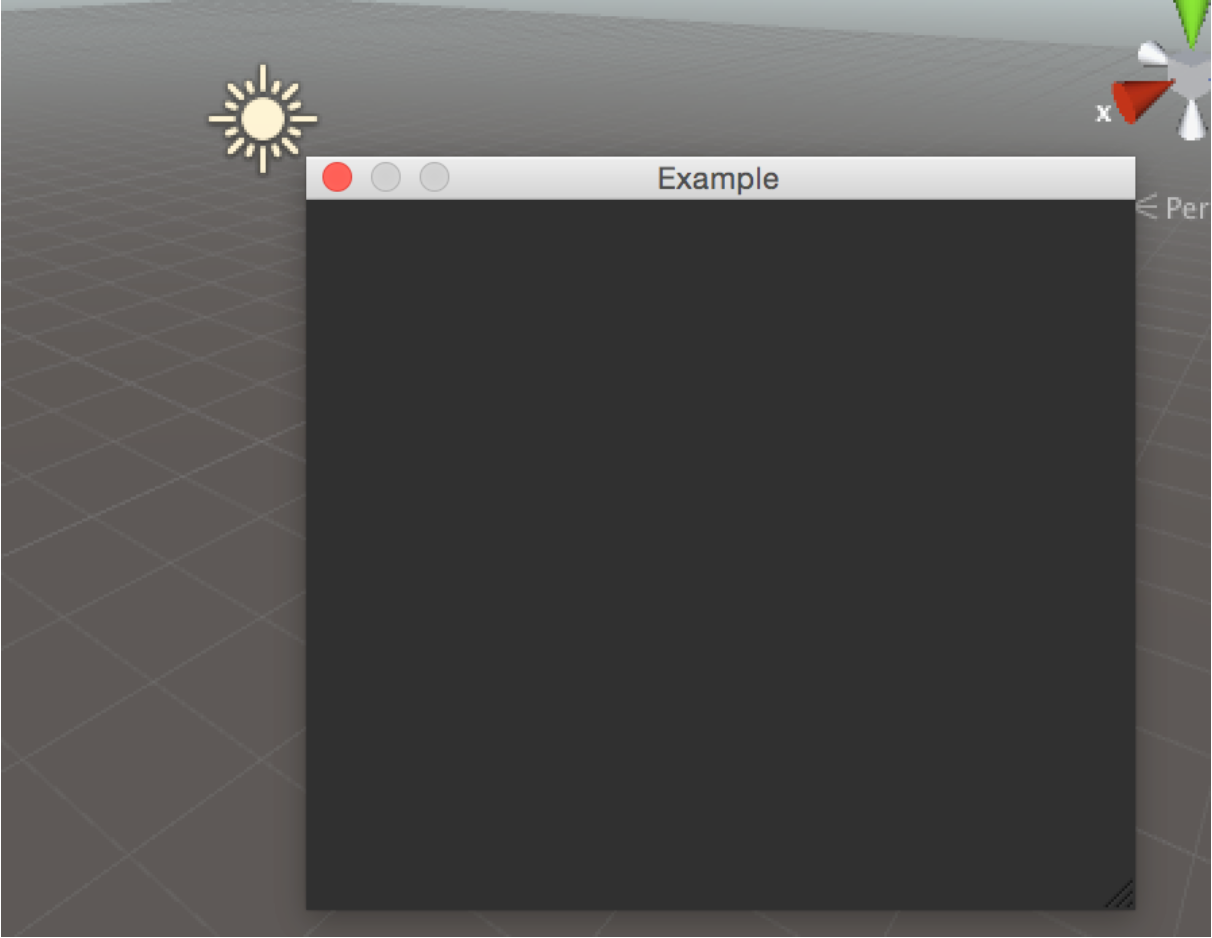
버튼을 누르면 직하에 Popup이 표시됨.

사용하는 방법은 매우 간단합니다. 일단 PopupWindowContent를 계승한 클래스를 작성합니다. 그리고 PopupWindow.Show로 Popup을 표시합니다.



### **ShowAuxWindow**

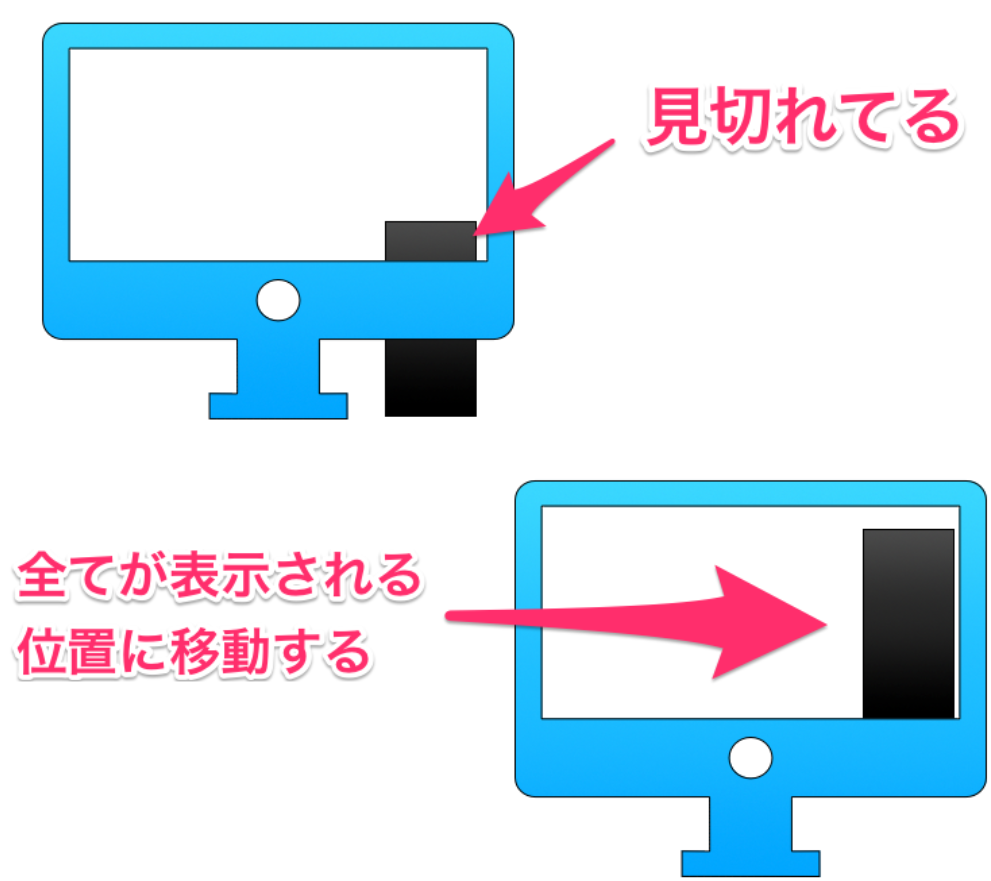
탭 윈도우로써 다뤄지지 않는 윈도우를 작성합니다. 보기에는 ShowUtility와 같지만 다른 윈도우에 포커스를 두면 윈도우는 삭제됩니다. 윈도우의 삭제하는 것을 잊어버리는 현상이 잃어나지 않기에 개수가 증가하는 걱정을 할 필요가 없으며 간단한 설정 및 조작으로 사용하는 용도로 추천합니다.



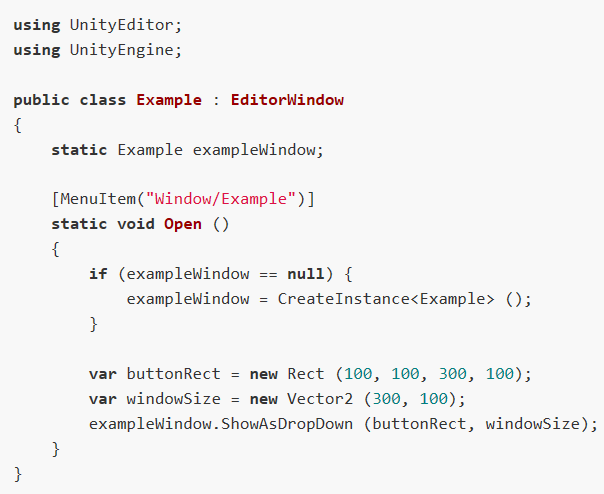
겉보기에는 ShowUtility인지 ShowAuxWindow인지 판단하기가 어려움.

### **ShowAsDropDown**

Popup과 같으며, [윈도우 타이틀], [닫는 버튼]이 없는 윈도우입니다. 단, PC 화면 크기를 고려해서 윈도우를 표시할 위치를 충분한 영역을 확보하지 못했을 때, 윈도우를 표시할 영역 내에 포함하기 위해 X/Y축의 위치를 자동적으로 보정하도록 합니다. 바꿔 말하면, 화면의 구석에 윈도우를 내보내도 반드시 PC의 표시영역에 모두 표시됩니다.



검은 색이 ShowAsDropDown에 표시한 윈도우라고 가정



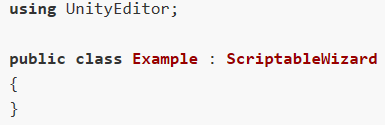
이 것 이외에는 Popup과 같은 기능입니다.

### **ScriptableWizard**

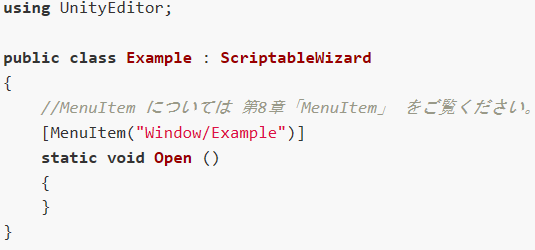
[GameObject를 만들다], [Prefab을 만들다], [에셋을 만들다], 와 같이 무언가를 [만들다] 때에 사용하는 윈도우입니다. ScriptableWizard는 지금까지 소개해 온 윈도우와 조금 다릅니다.

**ScriptableWizard의 사용법**

1. ScriptableWizard을 계승한 클래스를 작성합니다.

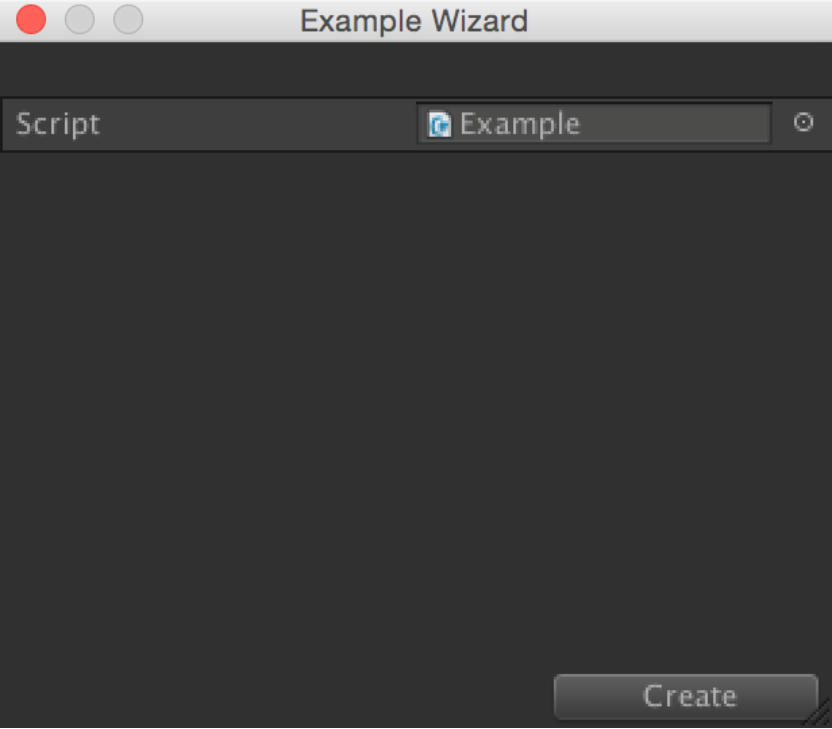


1. 다음은 ScriptableWizard을 표시하기 위하여 트리거로써 메뉴를 추가합니다.



1. ScriptableWizard를 표시합니다. 표시는 ScriptableWizard.DisplayWizard로 진행합니다.

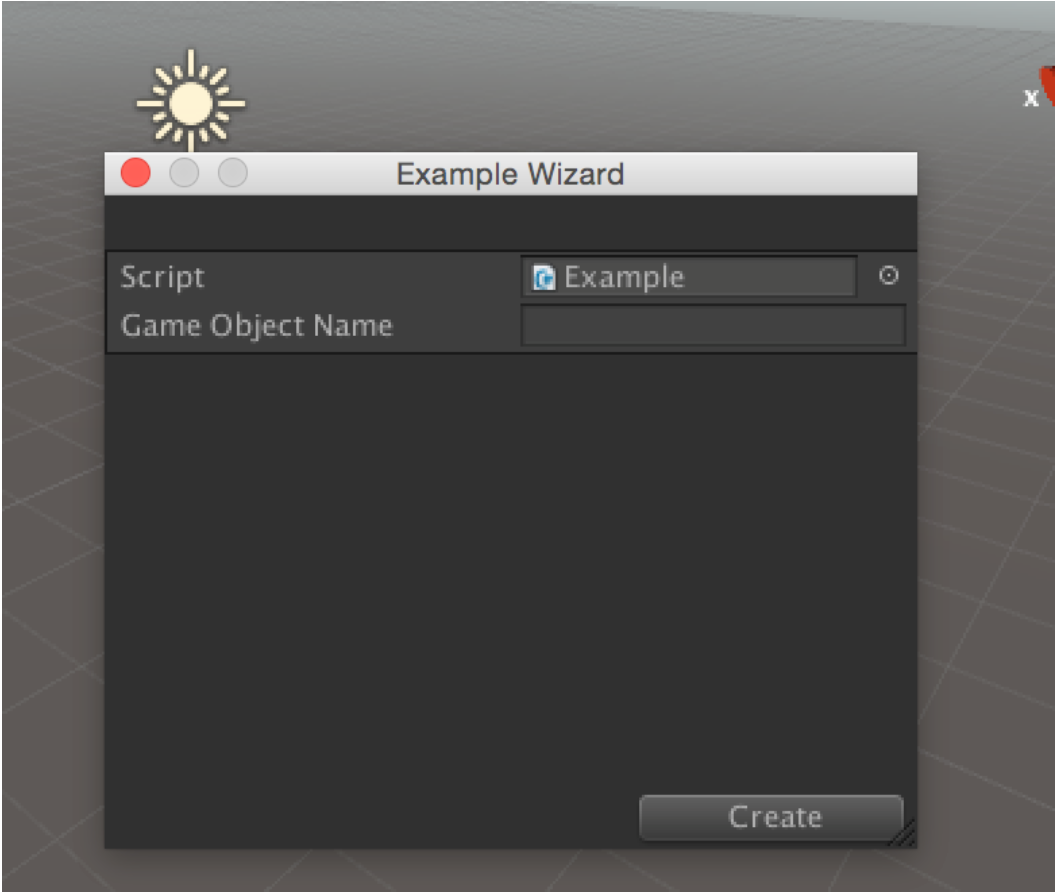




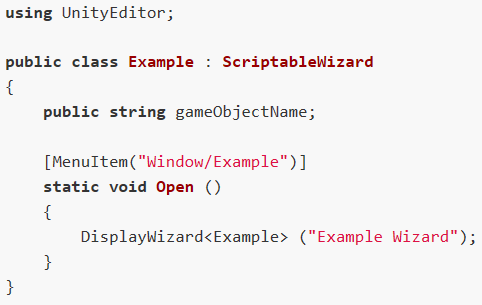
표준으로 왼쪽 아래의 Create버튼이 표시됩니다.

**ScriptableWizard에는 클래스의 필드가 표시됩니다.**

다른 EditorWindow에는 GUI의 표시에 EditorGUI클래스를 사용하지만 ScriptableWizard에는 사용할 수 없습니다. ScriptableWizard에는 인스팩터에 표시 되어 질 수 있는 [public같은 필드][시리얼라이즈 같은 필드]와 같은 것들이 윈도우에 표시됩니다.

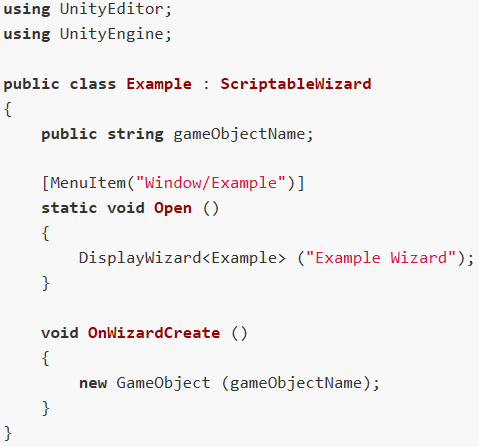


인스팩터에 표시 되어지는 것들이 그 대로 표시됩니다.



#### OnWizardCreate

ScriptableWizard의 왼쪽 아래에 있는 Create 보턴을 눌렀을 때 호출되는 함수입니다.



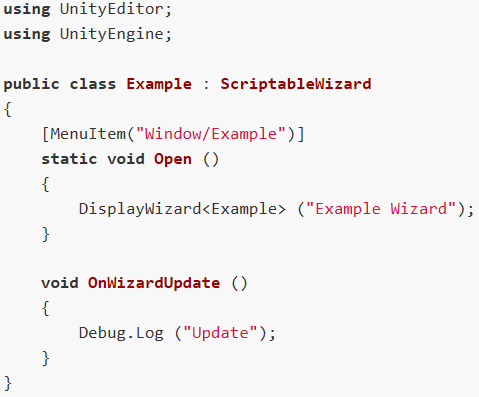
#### OnWizardOtherButton

Create 버튼 외에 또 다른 1개의 버튼을 추가할 수 있습니다. 작성에 관련된 2번째의 버튼을 만들고 싶을 경우 밑의 코드를 참고하여 사용해주세요. 버튼을 추가하기 위해서 사용되는 방법은 ScriptableWizard.DisplayWizard의 제 3인수에 버튼 명을 지정할 필요가 있습니다.



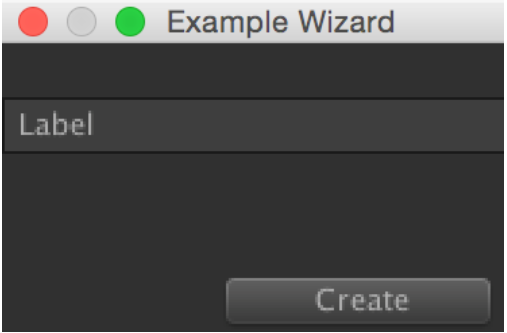
#### OnWizardUpdate

모든 필드의 값에 대하여 값의 변경이 있을 경우 호출되는 함수입니다.



#### DrawWizardGUI

마법사(Wizard) 내의 GUI를 묘화하기 위한 함수입니다. 이 함수를 오버라이딩 하는 것으로 GUI를 커스텀 마이징을 할 수 있습니다. 단, 반환 값으로써 true를 반환하도록 해주세요. True를 반환하지 않으면, OnWizardUpdate가 호출되지 않게 됩니다.

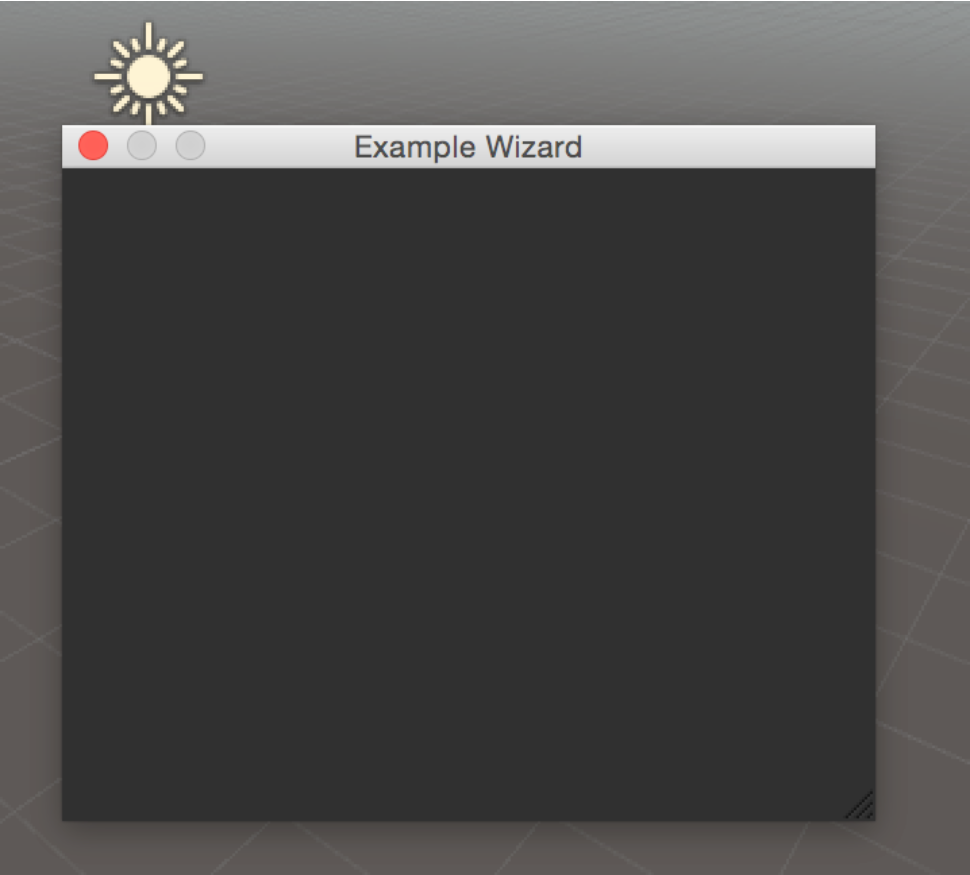


지금까지 표시 되어져 있던 프로퍼티가 없어졌고 Label가 표시되었습니다.



#### OnGUI 함수를 사용하지 않는 것

ScriptableWizard 클래스는 EditorWindow를 계승하고 있습니다. 그렇기에 OnGUI 함수를 사용하면 통상 EditorWindow로써 표시 되어지므로 필드의 값이나 Create 버튼이 표시되지 않게 됩니다.

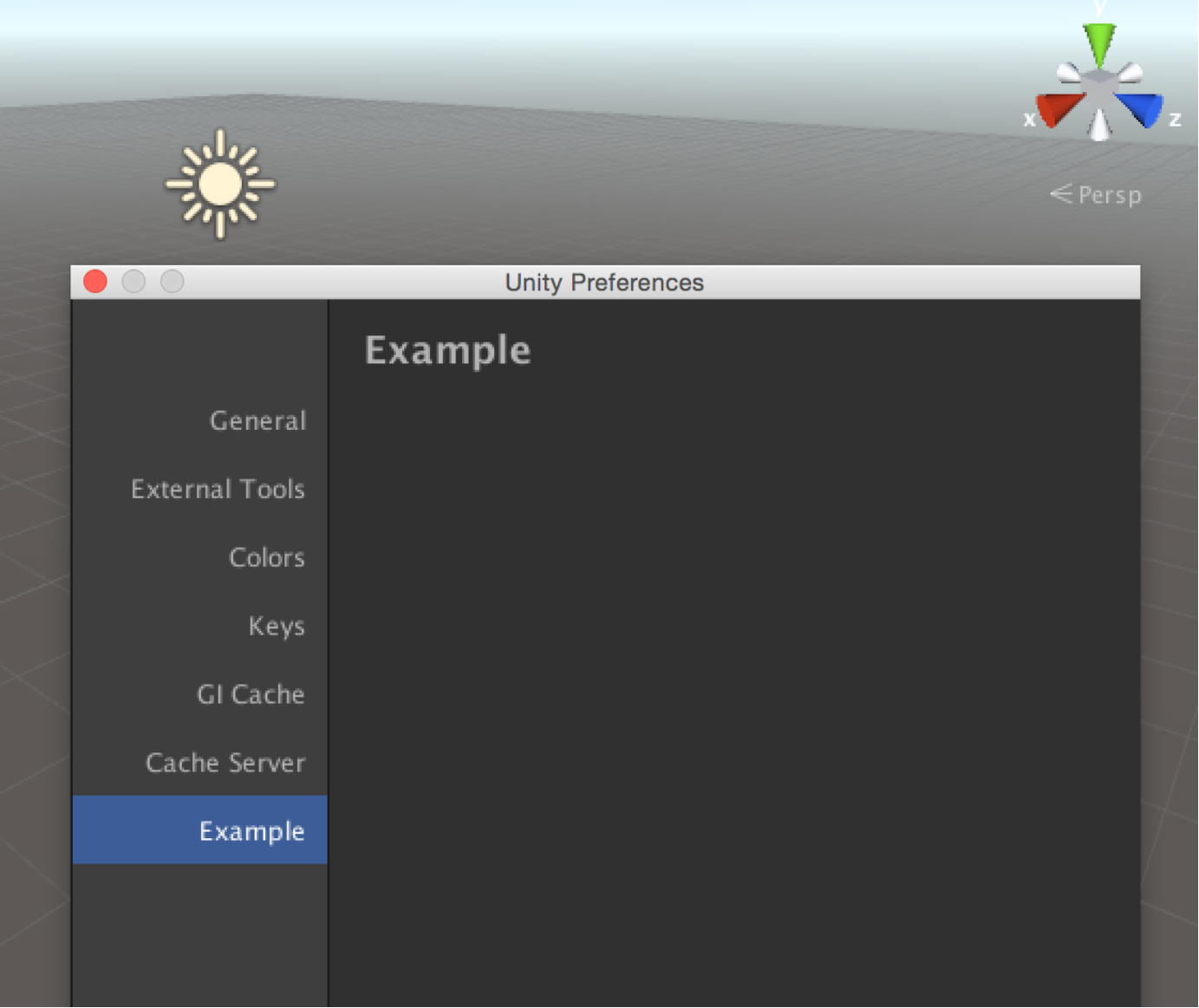


OnGUI 함수를 기술하면 Create 버튼이 사라집니다.

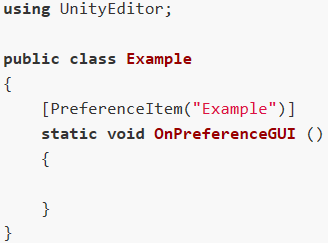
#### PreferenceItem

PreferenceItem은 Unity Preferences에 메뉴를 추가할 수 있는 기능입니다.

Unity Preferences에는 Unity에디터 전체에 영향이 있는 설정을 하기 위하여 존재합니다.

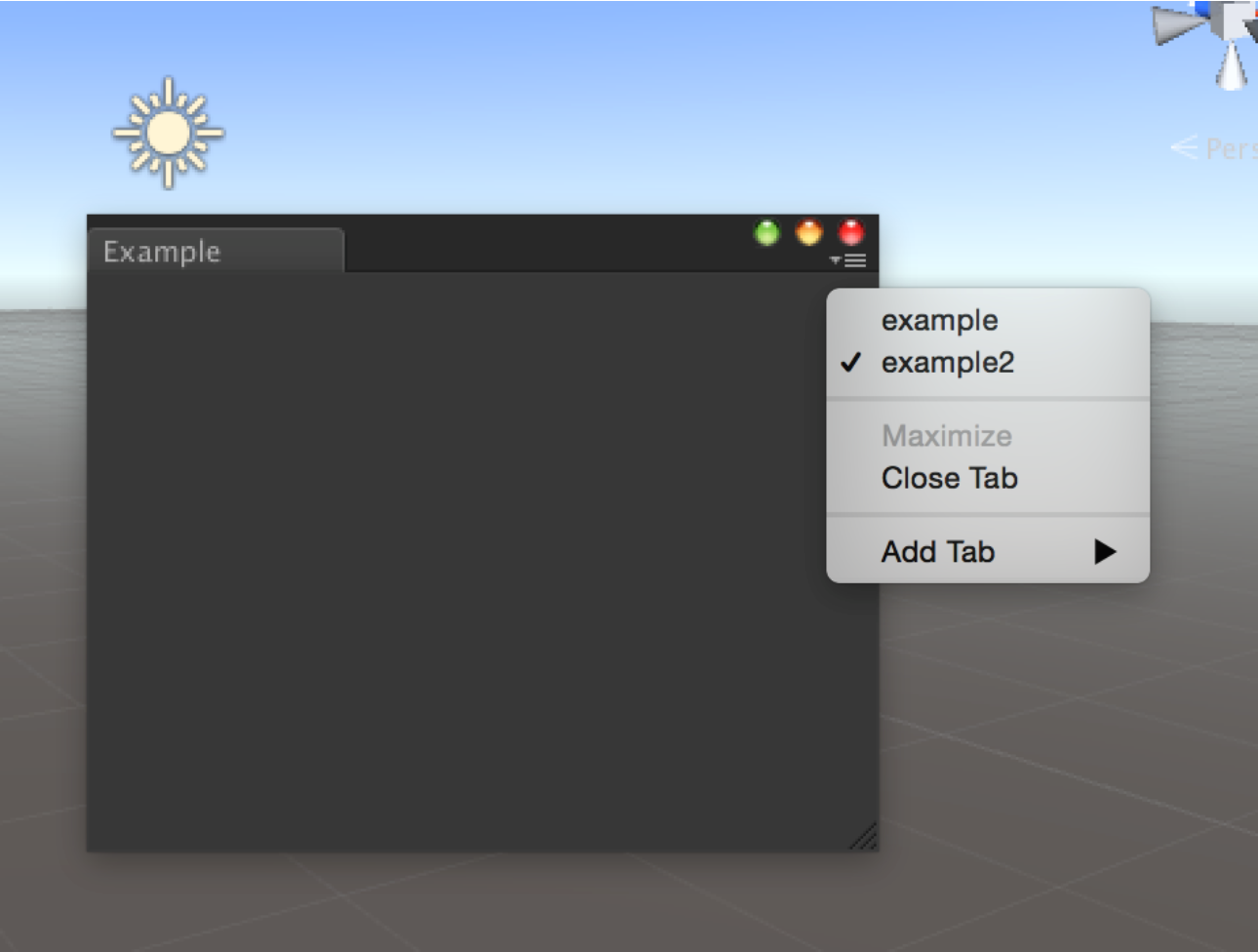


추가된 메뉴는 가장 맨 뒤의 위치에 추가됩니다.



## **7.6 메뉴를 추가하는 IHasCustomMenu**

탭 위에 오른쪽 클릭, 혹은 三을 클릭하는 것으로 표시되는 컨텍스트 메뉴에 메뉴를 추가합니다.

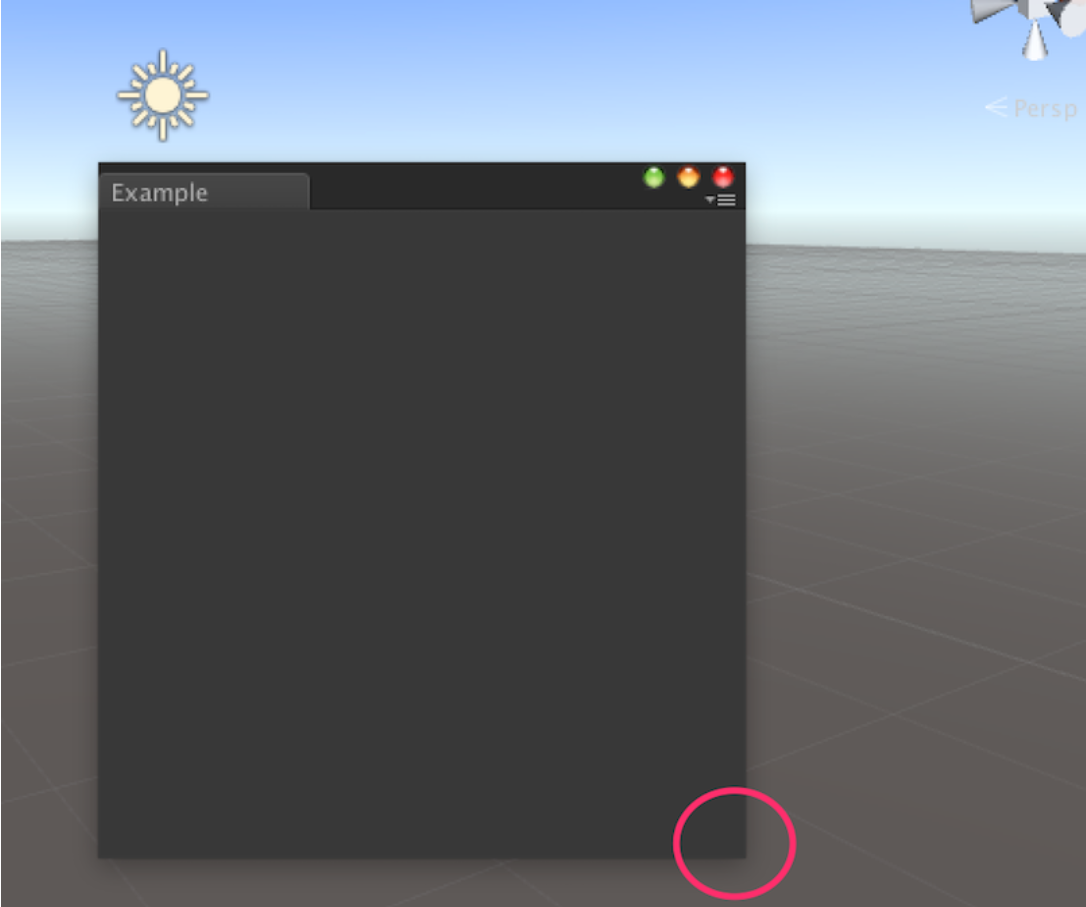


example과 example2가 추가됩니다.

IHasCustomMenu는 인터페이스로써 실장됩니다.



## **7.7 EditorWindow의 크기를 변경하지 못하도록 하기**



오른쪽 하단에 있는 재조정하기 위한 삼각마크가 사라져있음

EditorWindow.minSize 와 EditorWindow.maxSize에 의해 EditorWindow의 크기를 제한을 할 수 있습니다. 최소값과 최대값이 같으면 EditorWindow의 크기를 변경할 필요가 없다고 판단하여 오른쪽 하단의 표시되어있는 삼각 마크가 비 표시됩니다.



## **7.8 윈도우에 아이콘을 추가하기**

아이콘을 추가하려면 EditorWindow.titleContent에 아이콘을 처리하는 GUIContent에 대입합니다.



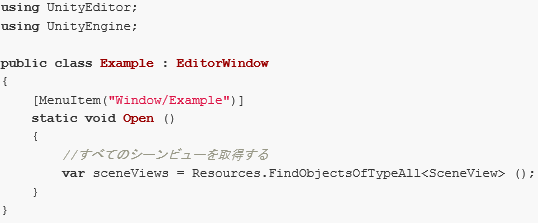
아이콘은 무척 작기 때문에 알기 쉬운 아이콘을 사용해야 함



## **7.9 GetWindow를 사용하지 않고 이미 있는 EditorWindow를 취득**

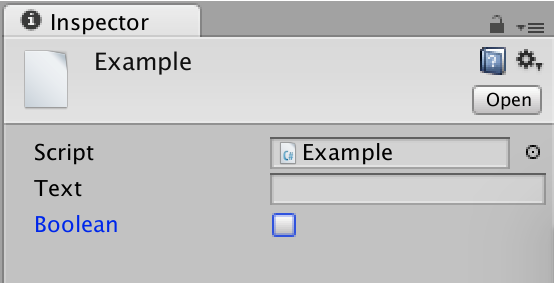
사전에 싱클톤을 실장할지, GetWindow로 내부 캐쉬를 사용을 할지 등등의 방법으로 EditorWindow에 엑세스할 수 있습니다. 하지만 앞서 위에서 언급한 2개의 방법을 쓰지 않는 상황도 있습니다. 그럴 때에는 Resources 클래스 안에 존재하는 정적으로 구현된 함수인 Resources.FindObjectsOfTypeAll 를 사용합니다.

FindObjectsOfTypeAll는 현재 불러져 있는 모든 객체로부터 특정 객체를 검색하고 취득할 수 있습니다. 이 것은 런타임에 사용할 객체 뿐만 아니라, 에디터에서 사용할 객체도 검색대상이 됩니다.

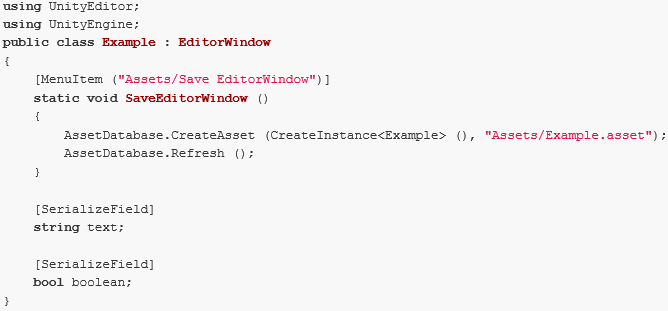


## **7.10 EditorWindow도 ScriptableObject의 하나입니다.**

EditorWindow는 ScriptableObject의 동작에 따르며, EditorWindow 객체를 엑세스해서 보존하는 것이 가능합니다. 그럴 때, 인스팩터에 시리얼라이즈 된 프로퍼티를 표시합니다.



7.24 Example 윈도우를 엑세스해서 보존했을 때의 인스팩터



윈도우 위치나 사이즈 등을 보존합니다. 이들에 대한 데이터는 보려면 직접 YAML형식의 파일을 텍스트 에디터로 보면 확인할 수 있습니다.

