Link to Original post (https://forum.unity.com/threads/serilization-objects-with-ref-to-assetineditor-and-build.1332996/)

This text based on created set of examples, which I uploaded to GitHub (https://github.com/DSivtsov/Git_Serialization)
Introduction (TLDR, skip till “demanded “targets”)
The standard serialization is the basis of Unity Editor works. Also, we can use the standard JsonUtility class (namespace UnityEngine) to serialize object before storing it on disk (commonly transforming UnityObject states into String). The standard serialization methods from this class have some limitations e.g., they do not give a possibility to store the reference to assets (include any ScriptableObject), because:

1. Unity serializes the reference to "instanceID", which will be a constant only in the one session of Unity Editor.
2. Unity uses different format to refer to reference in Editor and Build.

Therefore, various workarounds are used to solve this limitation (the most simple - convert the data from initial class to simple C# plain class and use it to serialization and back in backward order, other variant was described by other “forum mates” in **“post”** (there I also placed the “bad results" of use standard Unity methods).

Because there are options in C# binary serialization to control the serialization process (SurrogateSelector class, etc.), I decided to test the possibilities of Odin OdinSerializer “is an opensource serializer built for and used by Odin - Inspector & Serializer (https://odininspector.com/odinserializer) and combine it possibilities with proposals of “forum mates” to receive a solution which give to achieve **the demanded “targets”**

1. Store C# plain class (and ScriptableObject) which contains the members with reference to assets (I tested with ScriptableObject)
2. The stored reference must be workable constantly in Editor (in different sessions).
3. The format of reference must be the same in the Editor and in the Build.

In tests I use next classes:
ScriptableObject:
· ComplexitySO and LevelSO (for simplicity they are empty), used as reference.
· GameSettingsSpecialSO (contains data members and member with ref to asset, and methods to support serialization also), used for storage
· GameSettingsSimpleSO (contains only data members and member with ref to asset, except methods for testing), used for storage

C# Plain class:
· GameSettings (contains only data members and member with ref to asset, except methods for testing), used for storage
· GameSettingsComplex (contains data members and many (two) members with ref to asset, except methods for testing), used for storage

All examples I divided between different scenes:

1. **ByOdinClassicEditorOnly**
Store object like GameSettings class
2. **BySpecialSO_Unity_Odin**
Store object like GameSettingsSpecialSO class

3. **ByOdinPlainClass**
Store object like GameSettings class
4. **ByOdinUnityObject**
Store object like GameSettingsSimpleSO class
5. **ByOdinByPhaseCompexClass**
Store object like GameSettingsSimpleSO class

Note.
Except from 1 Example (ByOdinClassicEditorOnly) all other variants can work in Editor and in Build. The 2 Example can work with Unity and Odin Serializer, all other use the Odin Serializer only.

The resolving of a reference to the asset in the 1 Example was realized by Odin, based on "documentation" and used the AssetDatabase class (in Editor only).

In other variants was used the different solution, which based on store of the SO name in serialized data and on restore the reference to SO by use this name and the Dictionay<string, typeof(SO)>, which is created from Resources.LoadAll< typeof(SO)>.

The all details of realization of all variants are described in the separated file (**https://github.com/DSivtsov/Git_Serialization/blob/master/Additional information to Examples (draft).pdf**).

All Examples was separated between different Scenes. Output made to logs and to display.

I hope this information will be useful for someone and help to make own solution to store data in Unity or other related tasks with serialization data in Unity with use Odin Serializer or others.

Thanks to everyone who helped me create these solutions, especially to @Kurt-Dekker and to @spiney199 , and also to the **Tor** from the Odin support channel on Discord.