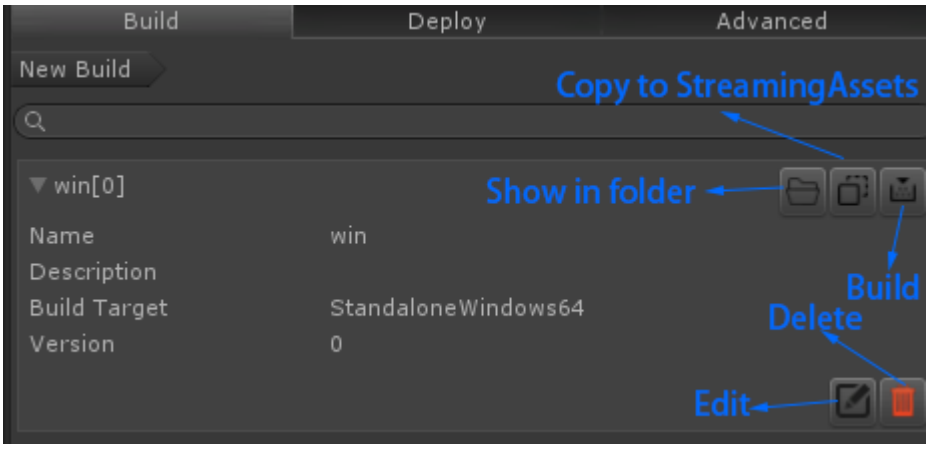
FinalPatch

1. Build



1.1 New Build

Click this button to create a new build. The build refers to the game package for a certain platform. For example, if your game needs to support Windows, Mac, Android, and IOS, you **AT LEAST** need to establish these 4 platforms separately. If you modify the C# code, you will not be able to update this part of the code on the IOS platform or using the IL2CPP function. You need to release a new game package. In order for online players to

download the new version of the client, you also need Add a new build here, and then set the version of the corresponding channel to this new version in the `Deploy` tab. For more details, see New Channel.

1.2 Show in folder

Click this button to browse all the assets of the current build in the folder.

1.3 Copy to StreamingAssets

Click this button to copy the latest assets of the current version to StreamingAssets. This means that the game package you built at this time already contains the latest assets. If you want players to download this game package and no longer need to update additional assets, you can click this button after the build step is completed. However, this will lead to an increase in the size of the game package you are building. Some publishing platforms may limit the size of the game package, in this case, you may consider not using this way, and let players to download game assets through hot updates.

Note: If you want to include assets in the package, be sure to use only this button, as this will generate some necessary data. FinalPatch does not directly use the assets you manually put into the StreamingAssets directory.

1.4 Build

Click this button build assets. The generated assets is located at: `Application.dataPath/../AssetBundles/`, the assets contains two parts: `Full` and `Package`. `Full` contains all assets. `Package` is based on the difference between the current version and the previous version of the `Full` assets. When uploading a CDN, you only need to care about the assets in `Package`.

Note: Although you only need to care about the `Package` part, `Full` can't be deleted, because the calculation process of `Package` depends on the previous version of the `Full` assets. So please at least maintain the last version in `Full`.

2. Deploy

When a patch is ready, in order for online players to update these assets, you need to publish the assets. The `Deploy` tab is where assets are released..

2.1 Deploy Data

Each game has a 'deploy data' which records all the channel information supported by the game. When a client request hot update, it will download the data first, and then according to its own channel to determine which address to download assets from. When you open the editor for the first time, you need to create or select a deployment data.

Note: The depoy data needs to be uploaded to the CDN for client downloading.

2.2 New Channel

When you add a new channel, you need to fill in some necessary parameters.

- Channel Name: The client will find the channel with its own name to compare the version and download assets.

- URL: Assets will be downloaded from this address during hot updates.
- Build: The client does not allow cross-build updates. for example, if the current build of the client is `android_1.0`, if the build is set to `android_1.1` here, clients will be prompted to download the new game package. So don't modify the version easily.
- Version: The client will compare the version number to determine if it needs to be updated. When the version number is higher than the client, the client will upgrade to the version. When the client is lower than the client, the client will be downgraded to the version.

Note: The actual URL for the hot update is: `URL/Build/Version/`. For example, the URL is: http://localhost:8000/AssetBundles. The Build is: demo. The version is: 3. The actual resource download address is: http://localhost:8000/AssetBundles/demo/3/

2.3 Edit Channel

If a patch is released, click the Edit button and modify `Version`.

2.4 Delete Channel

Please do not delete a channel easily, which will cause online players can't enter game.

3. Settings

There are some settings in the `Advanced` tab. All the names with `(?)` indicate that there is a tooltip. You can hover the name to see it.

3.1 Build

- Asset Path: All assets located under this path will be built. For details, see Build
- Extract Duplicate Dependencies: Assume that CubeA and CubeB both depend on the same Material. Suppose the size of CubeA is 1KB, size of CubeB is 1KB, and the size of Material is 1KB. Assuming we only package CubeA and CubeB, the result is: CubeA(1KB+1KB), CubeB (1KB+1KB). The content of Material exists in the asset bundle of CubeA and CubeB. After enable this option, it is detected that the Material is dependent on multiple AssetBundles when building, so the Material is forced to be built into AssetBundle. The result is: CubeA (1KB), CubeB(1KB), Material(1KB). It is highly recommended to enable this option to reduce the total size of the assets.
- Bundle Offset: A certain number of empty bytes will be written before the AssetBundle for simple encryption. This is a very simple way and can not completely prevent the crack from happening. Therefore, please set this value according to your actual needs. If the value is too high, the size of each AssetBundle will become larger.
- Build Callback: You can implement `BBGo.FinalPatch.IBuildCallback` to write your own build callback . After implementing your own callback, you need to select your class here. You can use this to copy all assets in `Package` after building to CDN directory. For more details, see: `BBGo.FinalPatch.BuildEditorCallback.cs`
- Path Converter: The assets needs to provide a full name path when loading, for example: `Assets/Res/Cube.prefab` (including extension), but we can't know the asset bundle name according

to it. The path converter will do this for you. You can just use `BBGo.FinalPatch.DefaultPathConverter.cs` if you don't have special requirements.

3.2 Patch

- Automatic Update: Enable it to let FinalPatch automatically check for updates when the game starts. If you have special requirements, you can disable and manually update the assets through API.
- Number of retries: Try several times if the automatic update fails.
- Retry interval: The interval between each attempt.

3.3 Pool

FinalPatch provides a basic GameObjectPool to help you manage GameObjects. See FinalPool for details.

- Maximum Cache: The maximum number of caches allowed in the pool. Resources that exceed this amount will be destroyed directly when they are recycled.
- Auto Release: If the pool is idle (without any references), it is automatically released to reduce memory usage. It is strongly recommended to turn this feature on.
- Release Delay: Automatically releases the pool when the pool is idle.

3.4 Other

- Apply Editor Mode: In the editor mode, the hot update phase will be skipped directly, and assets will not be loaded from the AssetBundle. It is recommended to turn this feature on to avoid frequent build during development. Note: Even if you turn this feature on, any version of the client you build will not be affected. So you can always open it.
- Persistent Path: Hot-updated assets will be downloaded in this directory, you can manually delete the resources here to update and test repeatedly.

4. Patch

In order to properly compare and update resources, the client needs `FinalPatchClient`. Click `Tools->BBGo->Add Final Patch Client` to add. You need to fill in some necessary parameters:

- Deployment data address: The client will download the deployment data from this address. For details, see: Deploy Data.
- Deployment channel: The client will find the matching channel from 'Deployment Data` to compare the version and download resources. For details, please refer to: New Channel

4.1 Automatic Patch

If you have Auto Patch turned on and the game is running, FinalPatch will automatically check the version for you and update the resources. Can refer to: `FinalPatch.AutoPatch()`.

4.2 Manual Patch

If you have special needs, you can manually update the resources.
`FinalPatcher.Collect()` will collect information about the required update resources. The return value is `PatchReport`, which contains the resources to be downloaded, total bytes, and patch status.

`FinalPatcher.Patch(PatchReport report)` is used to update resources. This is an asynchronous method. For specific update methods, please refer to: `FinalPatcher.AutoPatch()`.

Note: Regardless of which method is used, the update step is essential. If automatic update is not turned on and no manual update is performed, the resource load will fail.

5. Assets Loading

5.1 FinalLoader

FinalLoader provides an interface for assets loading.
`FinalLoader.LoadAssetAsync<T>(string assetName)` is used to load resources.
`FinalLoader.UnloadAsset(string assetName)` is used to unload resources.
`FinalLoader.LoadSceneAsync(string sceneName, LoadSceneMode mode)` is used to load the scene.
`FinalLoader.UnloadSceneAsync(string sceneName)` is used to unload the scene.

```
public async void Start()
{
    //load a material
    Material mat = await FinalLoader.LoadAssetAsync<Material>
("Assets/Res/Red.mat");

    //unload asset
    FinalLoader.UnloadAsset(mat);

    //load a scene
    await FinalLoader.LoadSceneAsync("Assets/Scenes/demo.unity",
LoadSceneMode.Additive);

    //unload scene
    await FinalLoader.UnloadSceneAsync("Assets/Scenes/demo.unity");
}
```

5.2 FinalPool

The FinalPool interface can be called when you need to load a GameObject. Internally, FinalPool also loads resources through FinalLoader, and FinalPool uses pools to help you cache these GameObjects to reduce the performance overhead caused by frequent instantiations. So if you need GameObject, strongly recommended to use FinalPool. `FinalPool.GetGameObjectAsync(string assetName)` is used to get a GameObject.
`FinalPool.ReleaseGameObject(GameObject obj)` is used to release a GameObject.
`FinalPool.DestroyUnusedPools()` is used to destroy all unused pools.

```
public async void Start()
{
    //get a cube from pool. FinalPool will load this cube via FinalLoader
at first time you request and create a pool named "Assets/Res/Cube.prefab"
```

```
    GameObject cube1 = await
FinalPool.GetGameObjectAsync("Assets/Res/Cube.prefab");

    //if you request a gameobject which alred cached, final pool will
instantiate a copy for you instead of loading from asset bundles.
    GameObject cube2 = await
FinalPool.GetGameObjectAsync("Assets/Res/Cube.prefab");

    //release cube1 to pool. FinalPool just hide cube1. there is 1
Assets/Res/Cube.prefab in FinalPool after releasing.
    FinalPool.ReleaseGameObject(cube1);

    //FinalPool will show the cube you just released in last step. there
is 0 Assets/Res/Cube.prefab in FinalPool now.
    GameObject cube3 = await
FinalPool.GetGameObjectAsync("Assets/Res/Cube.prefab");

    //release cube2 to pool
    FinalPool.ReleaseGameObject(cube2);

    //Nothing happend because cube3 is in use.
    FinalPool.DestroyUnusedPools();

    //release cube3 to pool
    FinalPool.ReleaseGameObject(cube3);

    //the pool Asset/Res/Cube.prefab will release immediately. if you
enabled AutoRelease in advanced tab, you don't need to call this function
manually.
    FinalPool.DestroyUnusedPools();
}
```

Contact Me

If you have any questions or suggestions, please contact me at teddyzhang29@gmail.com, I will contact you ASAP.