Asset Bundle Suite

ASSET BUNDLE SUITE

BY FISHERMAN PRODUCTION

QUICK SUMMARY

This product is made for handling your asset bundle needs. Starting with assigning asset bundles to your assets over building, uploading, downloading and assets loading during runtime. It provides configurations to define rules to auto assign assets or folders to asset bundles, to define upload and download locations and runtime behavior settings. It comes with a complete version handling for you bundles and its own cache system. You changed an asset, just build bundles with the suite, it will compare your latest live files against your new build and only uploads the changes. Your users do not need to download everything, just the changes. If download size matters ship bundles within your app as streaming assets but keep the possibilities to update them later with a download. This package contains also a complete asset loader implementation which cares about downloading and caching bundles, loading assets and keep references to avoid duplicated loadings. It utilizes the promise pattern for easy syntax. Just call AssetBundleService.Get<GameObject>(„assetName“).Then(OnLoadSuccess, OnLoadFailed); In OnLoadSuccess you will get the loaded GameObject as a parameter.

VERSION

1.0.0

HOW DO I SET IT UP?

Just import the package which you get over the asset store. It contains also examples files with configurations and example assets. The example will use a subfolder in your project folder to simulate the download location. If you do not want to have the example files just uncheck the example folder.

EDITOR SETUP

1. After the import you can create all needed configuration assets with clicking on Tools – Asset Bundle Suite – Create Default Configuration. It will create a folder called AssetBundles containing all configurations files. A more detailed explanation of these files can be found in chapter “Configurations”.
2. The AssetBundleBuildConfig will be created with a local file server connection. With this setup the asset bundle suite uses for up- and download a system which is just loading files from a local folder. If you have already a remote fileserver you can create a new connection setup by clicking “Create new connection settings”. Choose ftp or sftp. If you have created a

new connection configuration you can choose it from the drop down list. Afterwards check the created configuration file and enter the needed credentials.

3. Now enter a build location information (not needed for local file server). Select the AssetBundleBuildConfig. Here you can find Usable Locations. This is used to tell the build system the host url and port to upload with the before chosen file server connection. Additional you specify under Root Directory your wanted path to the subfolder on your fileserver. (Example: var/www/files/game/assetbundles/)
4. After this setup you also have to setup the same location for downloading which will be used at runtime. For that select AssetBundleRuntimeConfig and also enter a Usable Location. This time you have to define how to access the assets. The root directory could be just assetbundles if based on the previous example the folder var/www/files/game is already behind your official address game.mypage.com. The asset bundle suite will then try to download the bundles from game.mypage.com/assetbundles.

   NOTE: If you use the local file server to create the asset bundles this will not work as a download location at runtime. If you just want to test the system, ensure that the downloading of asset bundles is deactivated in editor and use the editor simulation instead. Or building only streaming assets only builds. To do so uncheck the option under: Tools – Asset Bundle Suite – Download Bundles in Editor
5. Assign your assets to bundles or use the auto assigner configuration to specify assigning rules. These rules will be applied just before building the bundles.
6. To run the asset bundle build pipeline, go to Tools – Asset Bundle Suite – Run Build Pipeline. The AB builder windows opens. Select your build configuration and click “Start asset bundle pipeline”.
7. If everything ran fine you have now your bundles uploaded to your remote location and a streaming asset folder was created to ship the version of the manifest and the version file for steaming assets with your app. Everything is ready to get downloaded.

QUICK RUNTIME SETUP WITH USAGE OF PREPARED COMPONENTS (NO CODE NEEDED)

1. Create a new game object or choose one which is your bootstrapper off you game. Add the component “InitAssetBundleSuite”. Link your AssetBundleRuntimeConfiguration to the field “Runtime Configuration”. By doing this, it’s not necessary anymore to have the AssetBundleRuntimeConfig in the Resources folder.
2. If you entered more than one location, set the index to the one you want to use.
3. Now you can listen to the success of the initialization. Afterwards you can use the system to load assets from asset bundles
4. For example, you can add a game object as a placeholder with a Load Prefab Component. Place it deactivate in the scene and trigger its activation from the Suite Initialized Event from Init Asset Bundle Suite Component. This will load a prefab from an asset bundle and instantiate it.
5. Let it run and have fun!

RUNTIME SETUP (CODE NEEDED)

This will describe the setup via code. Whenever you want to initialize the asset bundle suite call AssetBundleService.Initialize and pass a log setup to connect your logger and a coroutine setup which provides methods to start and stop coroutines.

```
1.  [SerializeField]
2.  private AssetBundleRuntimeConfiguration runtimeConfig;
3.
4.  void Start ()
5.  {
6.      AssetBundleService.Initialize(LogSetup.CreateUnitySetup(),
7.      new CoroutineSetup(StartCoroutine, StopCoroutine), runtimeConfig)
8.      .Then(OnServiceInitalized, OnServiceFailed);
9.  }
10.
11. private void OnServiceInitalized()
12. {
13.     Debug.Log("Asset Bundle Service ready for use.");
14. }
```

Instead of having the runtime configuration serialized here you can have it somewhere else like a global Configuration. Alternatively, you can load it from the Resources folder.

HOW TO USE IT?

LOAD ASSETS

When the asset bundle service in initialized you can call from everywhere you want the get method to load an asset from a bundle.

```
1.  private void OnServiceInitalized()
2.  {
3.      IPromise<GameObject> assetPromise = AssetBundleService.Get<GameObject>("Triangle")
4.      .Then(OnPrefabLoaded);
5.  }
6.
7.  private void OnPrefabLoaded(GameObject loadedPrefab)
8.  {
9.      GameObject newInstance = Instantiate(loadedPrefab);
10.     AssetBundleService.Release(loadedPrefab);
11. }
```

As type you can use every type which inherits from Unity.Object like ScriptableObject, Sprite, Material, Texture, etc.

UNLOADING

If you want that the asset bundle service can keep track of the usage of loaded assets it will be recommend to release an asset if it will not be used anymore. In the first instance release will just

decrease internally the reference count. But if this reaches 0 it will be marked as unused and can be unloaded on UnloadUnusedAssets or UnloadUnusedBundles. It's up to you if you want to use it. Keep track of the usage is not easy in all cases.

```
1.  private void OnDestroy(GameObject instance)
2.  {
3.      AssetBundleService.Release("Triangle", true);
4.  }
5.
6.  private void ReleaseSprite(Sprite sprite)
7.  {
8.      AssetBundleService.Release(sprite);
9.  }
```

ADVISE: TAKE CARE OF ASYNCHRONOUSLY LOADING

Always test with delayed loading time and you will see miss behavior if you don't have the async loading in mind. Easy example: Imagine you want to load a random sprite on every mouse click. It would be easy for the user to click faster than the loading of the sprite will happen. This can lead to showing the wrong image if the asset of the second click is loaded before the first one, because the onLoadedCallback from image 1 will overwrite the set of image 2 again. To avoid that save your current wanted asset and compare against it afterwards.

```
1.  [SerializeField]
2.  private SpriteRenderer spriteRenderer;
3.
4.  // save the currently wanted asset
5.  private string requestAssetName;
6.
7.  private void LoadAsset(string assetName)
8.  {
9.      requestAssetName = assetName;
10.     AssetBundleService.Get<Sprite>(assetName).Then(OnAssetLoaded);
11. }
12.
13. private void OnAssetLoaded(Sprite sprite)
14. {
15.     if(sprite.name != requestAssetName)
16.     {
17.         AssetBundleService.Release(sprite);
18.         return;
19.     }
20.
21.     spriteRenderer.sprite = sprite;
22. }
```

ADVISE: WRITE YOUR OWN APPLICATION BUILDER

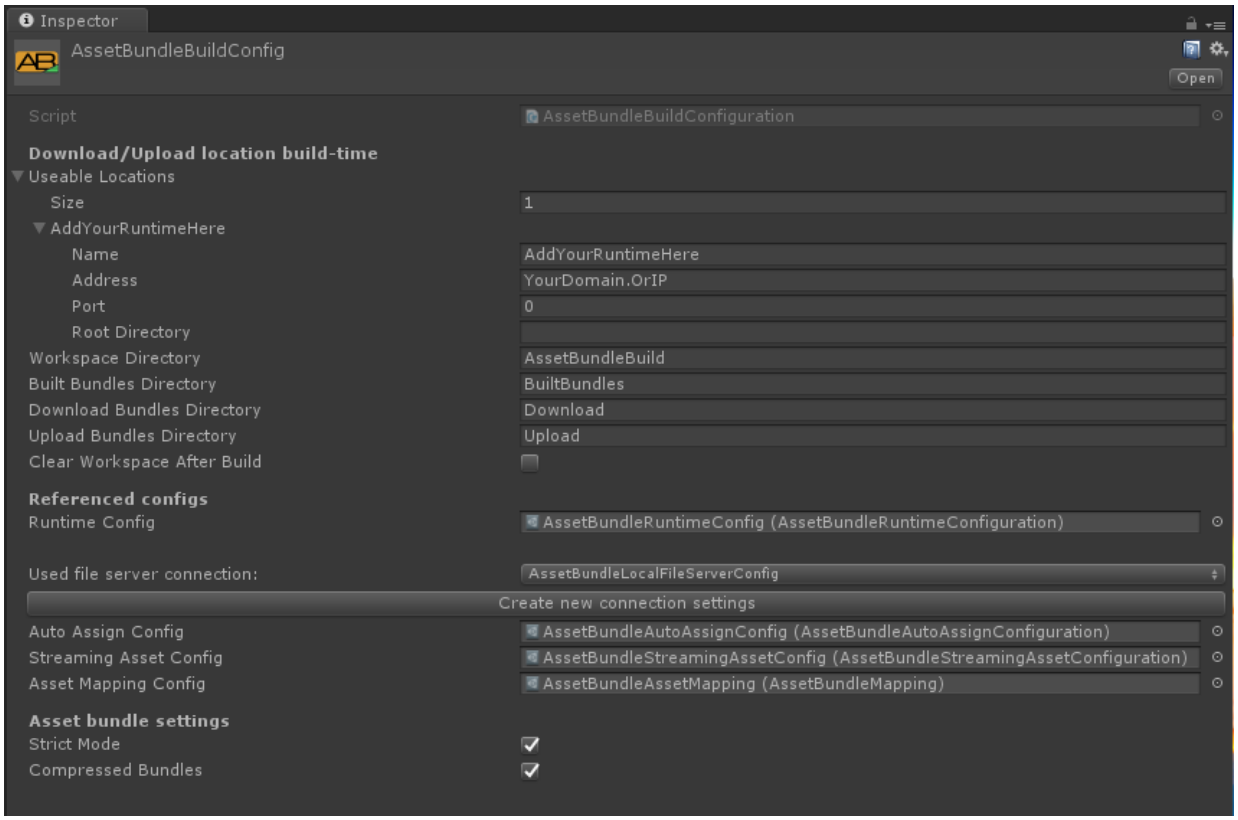
Since you have to configure some values in you AssetBundleRuntimeConfig and AssetBundleBuildConfig I recommend to have an own executing method to build you app instead of just using the Build button in the Build Settings. Potentially have an own build windows for that as well. That you can be sure that the correct version and settings about

streaming assets and so on is set correctly. But you can also still do manually adjustments and use afterwards the build pipeline from the asset bundle suite and afterwards building you app.

CONFIGURATIONS

There are several configuration files used in the asset bundle suite. The following list is explaining the settings of each of them.

BUILD CONFIGURATION



In the AssetBundleBuildConfig files you can specify everything for the build process of asset bundles. It also holds links to all related configuration files as well. This is needed to guarantee the correct usage if you have more than one setup in your project.

- **Useable Locations** – Here you can define different remote settings for download and upload during build time. In address only enter the main domain or ip address. Setting up the path to a subfolder on your machines will be done in Root Directory. Be aware that the path to your desired directory is depending on your chosen connection. Setting up more than on location can be useful if you have different environments like develop, staging, production. Then you can choose the correct one on buildtime via SetUsedLocationIndex.

- **Workspace Directory** – Name of the folder on your local machine to store all files during the asset bundle building. It is relative to your project folder.

- **Built Bundles Directory** – Name of subfolder in workspace where all built bundles are stored

- **Download Bundles Directory** – Name of subfolder in workspace where all downloaded bundles are stored. Bundles will be downloaded from the remote location to compare against the built ones to detect changes.

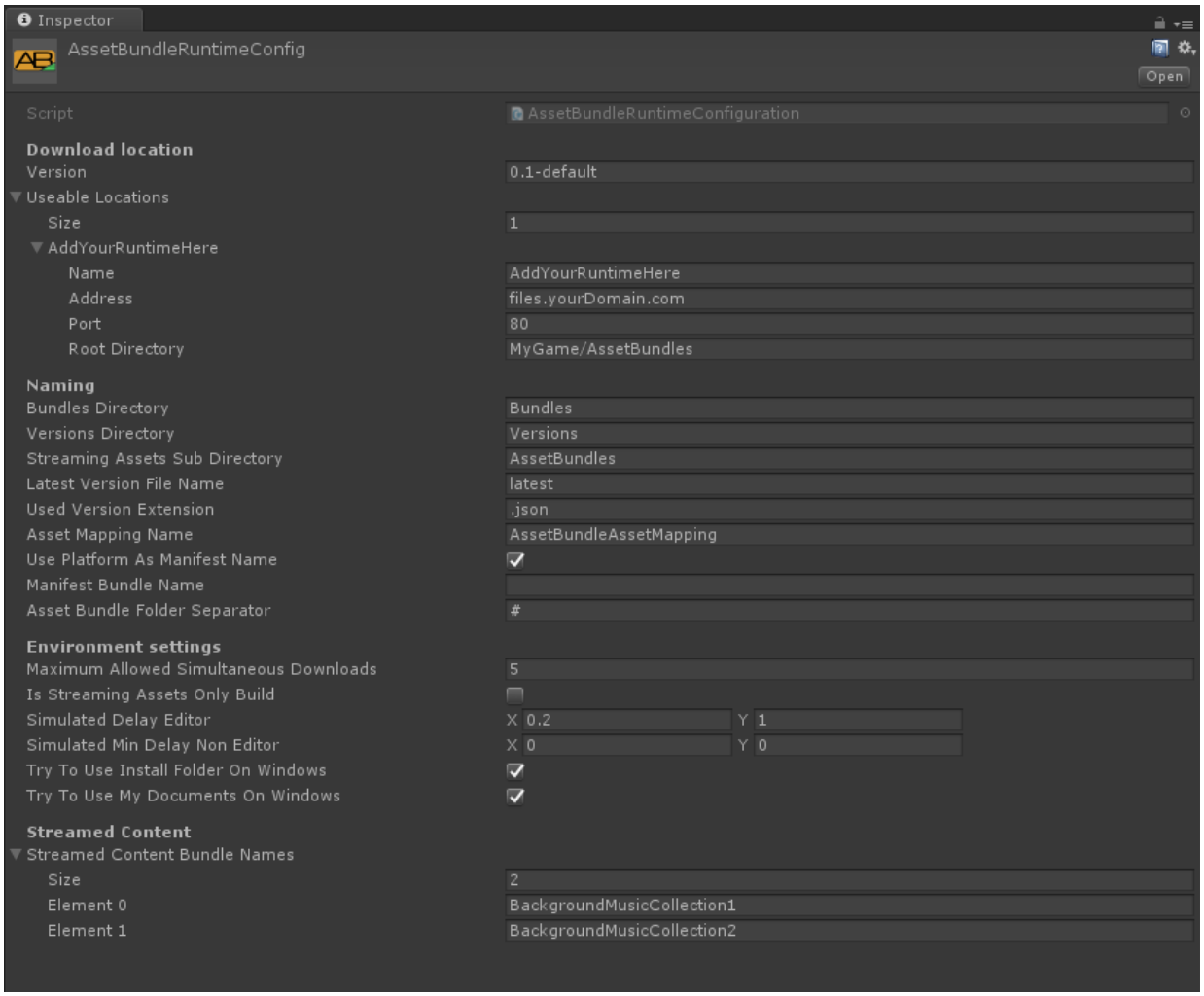
- **Upload Bundles Directory** – Name of subfolder in workspace where all uploaded bundles are stored. In this folder all bundles are located which were uploaded during the last asset bundle build. Only changed asset bundles will be uploaded.

- **Clear Workspace After Build** – If checked, the workspace directory and all subfolders will be deleted. If you want to see what happens in the last build pipeline run, you should let it unchecked

- **Used file server connection** – Defines which one of you created file server connection will be used

- **Strict Mode** – Check if you want Unity builds asset bundles in Strict Mode (https://docs.unity3d.com/ScriptReference/BuildAssetBundleOptions.StrictMode.html)

- **Compressed Bundles** – Check if you want Unity builds compressed asset bundles (https://docs.unity3d.com/ScriptReference/BuildAssetBundleOptions.UncompressedAssetBundle.html)

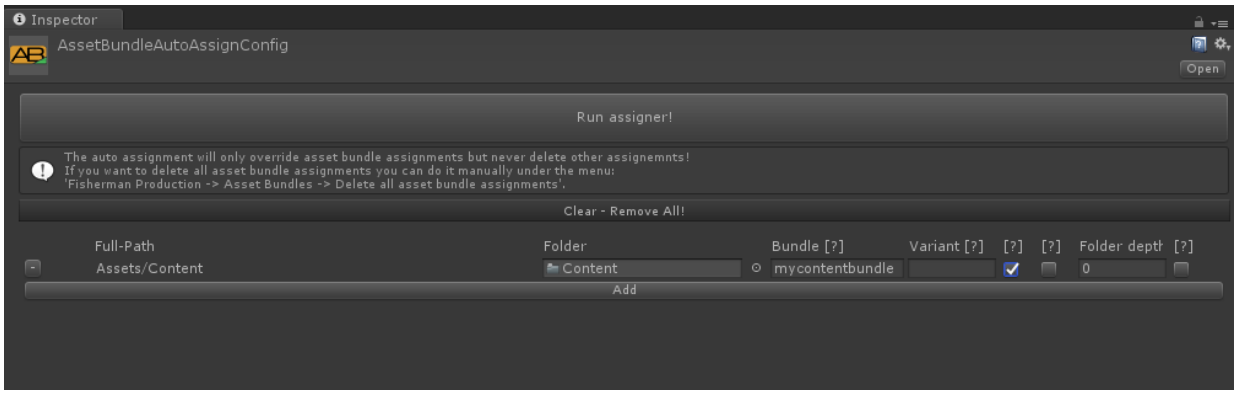


The asset bundle runtime configuration has to be delivered with you game and contains all needed information for runtime.

- **Version** – This fields specifies the version of the asset bundles. Depending on that field the version file will be loaded which contains all version settings of the bundles. Be sure you set the correct version before releasing your game.
- **Useable Locations** – Here you can define different remote settings for downloading bundles during runtime. In address only enter the main domain or ip address. Setting up the path to a subfolder will be done under Root Directory. Be aware that the path is like if you would call the address in a browser. Setting up more than on location can be useful if you have different environments like develop, staging, production. Then you can choose the correct one on runtime via SetUsedLocationIndex.
- **Bundles Directory** – Name of subfolder for asset bundles
- **Versions Directory** – Name of subfolder for version files
- **Streaming Assets Sub Directory** – Name of subfolder in StreamingAssets folder which will be used to place all streaming asset bundles to ship them with the game.

- **Latest Version File Name** – Name of file which will be used to save the latest created asset bundle version. This file will be in every asset bundle folder and is used for correct version increments.
- **Used Version Extension** – Extension which will be used for the version file. You can leave it empty if you want. Maybe you want that the file has a specific extension to setup rules in your cdn for it. *Note: version files should not be cached on cdns.*
- **Asset Mapping Name** – Custom name for the asset mapping configuration and its bundle
- **Use Platform As Manifest Name** – If checked the created manifest file will be called as the platform which was used to build asset bundles.
- **Manifest Bundle Name** – If the platform is not used as manifest name then you can specify here the name for the manifest file.
- **Asset Bundle Folder Separator** – '/' will not be used during runtime to avoid problems with directory separator on file server requests, so it will be replaced by the specified sign. Use a sign which you definitely not use as part of you bundle names.
- **Maximum Allowed Simultaneous Downloads** – Limits the amount of parallel running downloads. If the maximum is reached the download request will be queued. Too much parallel downloads can lead to problems on some platforms.
- **Is Streaming Assets Only Build** – Defines if the current build has all bundles in the streaming assets folder. This will basically deactivate any download for bundles from the internet, it only uses the bundles in streaming assets. If activate on build-time the pipeline will ensure that all bundles are packed into streaming assets folder and will ignore the streaming assets configuration.
- **Simulated Delay Editor** – Range of delay in seconds when playing in editor with not downloading live assets. (Default behavior in editor). This will be delay any asset load by a random value between the ranges. Good to see issues which only occur on longer loading times.
- **Simulated Min Delay Non Editor** – Range of minimum delay when playing in editor but with downloading assets from remote. If the download is faster that the set delay, then it will be hold back from the asset bundle loader until the desired delay is reached.
- **Try To Use Install Folder On Windows** – Only affects windows builds. If active, it will try to cache downloaded bundles in the install folder of the game instead of a folder in AppData system folder.
- **Try To Use My Documents On Windows** – Only affects windows builds. If active, it will try to cache downloaded bundles in a subfolder of my documents instead of a folder in AppData. If Try To Use Install Folder On Windows is active, it first try to save files into the install folder and if that fails it tries to use the my documents folder. This this fails as well, it is using the AppData folder.
- **Streamed Content Bundle Name** – List of bundle names which containing streamed content like audio which are set for streaming. Needed to avoid unloading the bundle after the asset was loaded to keep the streaming alive.

AUTO ASSIGN CONFIGURATION



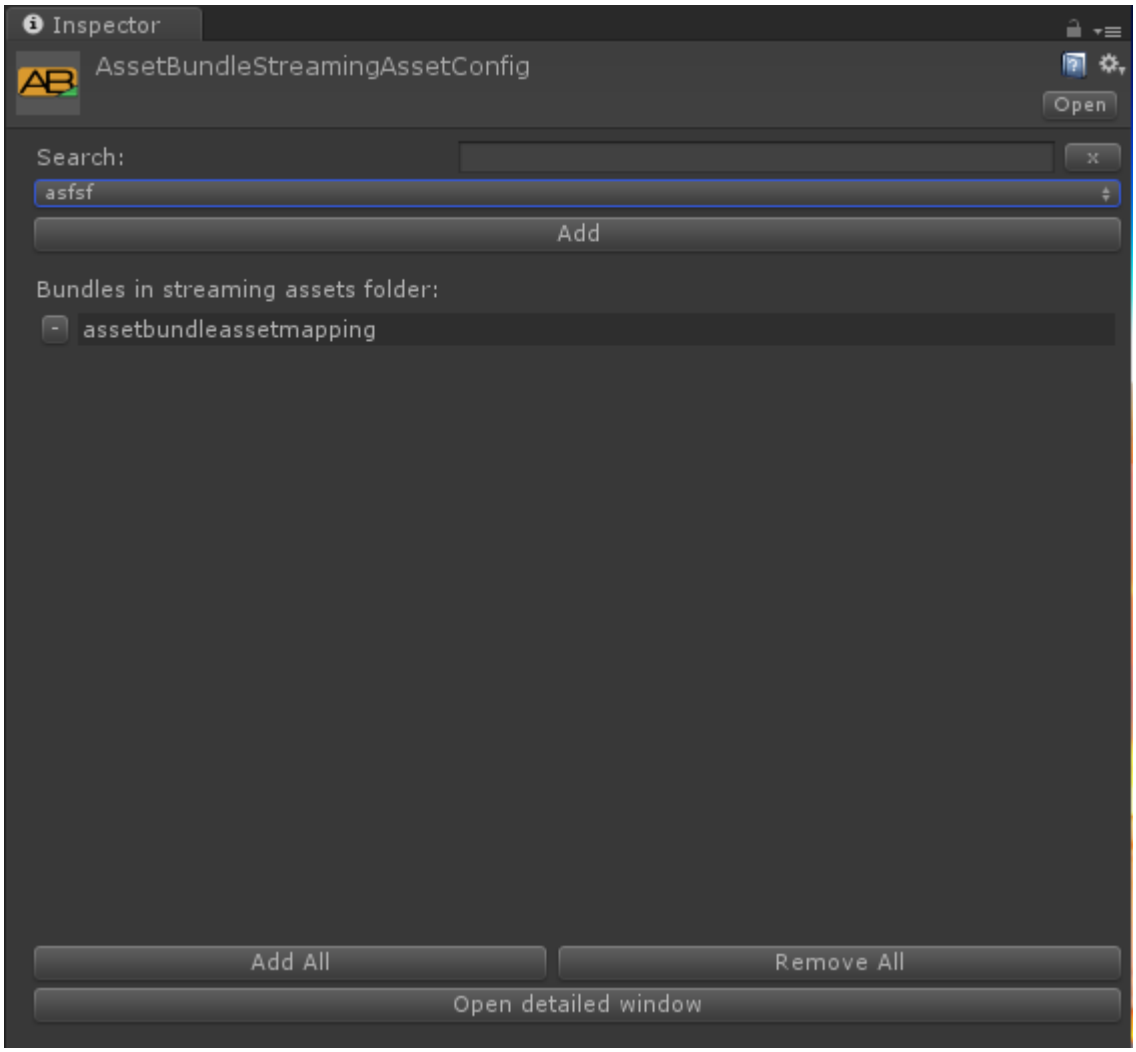
In this configuration you can setup rules to automatically assign asset bundle names to your assets or folders. The auto assignment will run always before the pipeline calls unity to build the asset bundles. The auto assigner will not delete asset bundle assignments but it will override them. With that you can still have only manual assigned bundles if you want.

You can add a rule by clicking on add. Afterwards drop a folder or select one with the object picker. Next to the folder name you find some settings.

- **Bundle** – bundle name which gets assigned to that folder
- **Variant** – variant name which gets assigned to that folder
- **1st checkbox** – If active not the folder will be assigned to a bundle, instead every asset in that folder gets its own bundle assigned. (Name = Bundle/AssetName)
- **2nd checkbox** – If not active the assignment will go through all subfolder by default. If active you can specify in the filed next to it, how deep you the assignment should go. If set depth to 0, just the main folder itself will get assigned.
- **3rd checkbox** – If active the rule will not affect the main folder itself, just the subfolders. Have this active plus the 2nd checkbox active with a depth of 0 will lead to a rule which is doing nothing, because it ignores main and subfolders.

The big button “Run assigner” in the top will just start the automatically assignment. It can be necessary to run it manually if you testing in editor, because the editor runtime loader will report errors if you want to load an asset which is not assigned to an asset bundle.

Note: There is a plan to have a nicer window for these settings in the future.

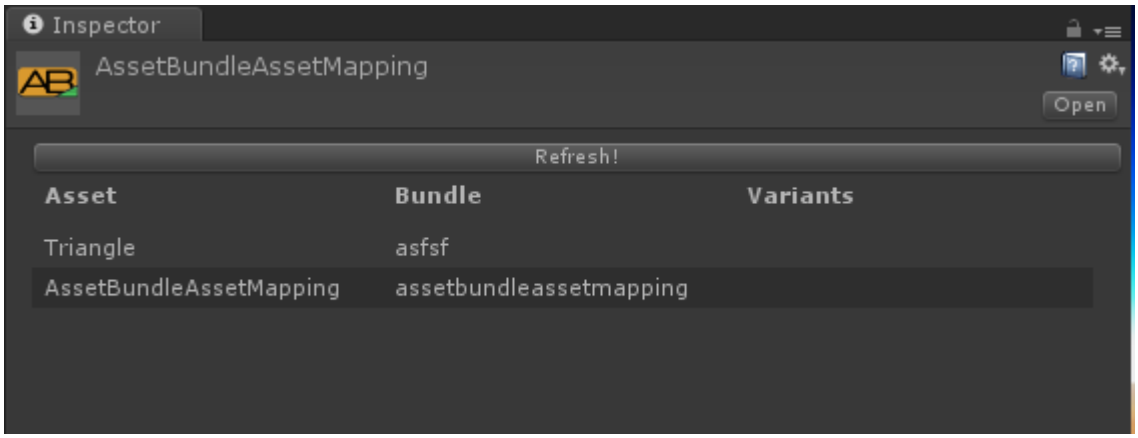


In the AssetBundleStreamingConfiguration you can specify which of your bundles you want to deliver with your game inside the streaming asset folders. Every bundle added to that list will be automatically added to the streaming asset folder. Beside that it will be still uploaded to your remote location.

Adding files to the streaming assets folder has the adventure that your users does not have to download these assets from the internet. But you still have the chance to update these assets if you upload newer versions to the remote location the asset bundle service will detected it and starts downloading the new version on the next need. So you can deliver your assets with an update but still have the possibility to change things without a full app update.

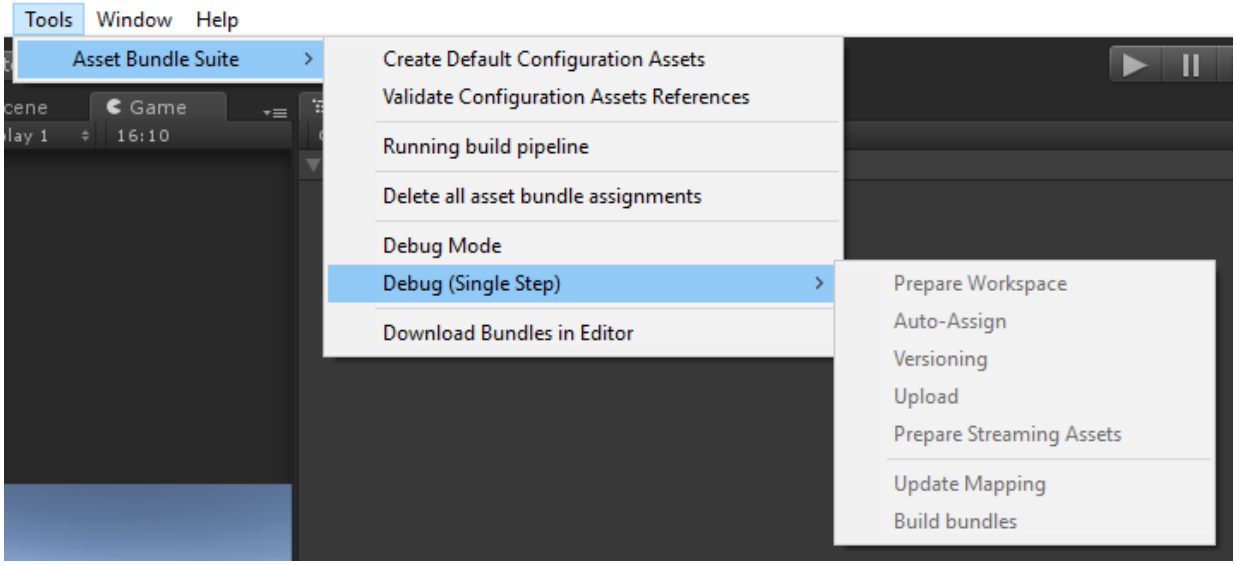
With the search filed you can filter the results in the drop down menu.

ASSET MAPPING CONFIGURATION



This is not a configuration where you have to setup anything. But you can use it to control the mappings after an asset bundle build. At runtime this mapping will be used to know which bundle the system has to load if you request a certain asset.

EDITOR MENU OPTIONS



CREATE DEFAULT CONFIGURATION ASSETS

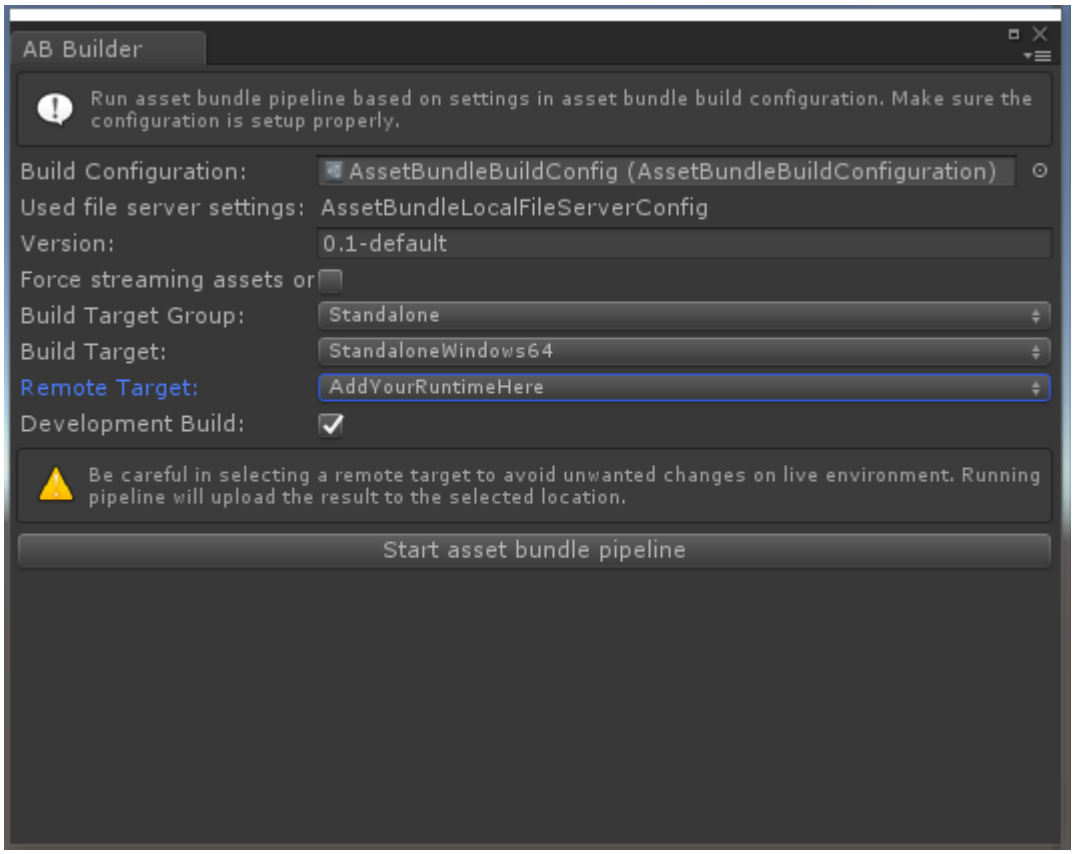
This will create all needed configuration assets to run the asset bundle suite. Normally you will only use this once.

VALIDATE CONFIGUATION ASSETS REFERENCES

This will check all asset bundle configurations. Start point for every check are the asset bundle build configurations. Results will be printed in the log.

RUN BUILD PIPELINE

This will open the asset bundle builder window.



Select your build configuration with which you want to start the pipeline. Here you can also set another version as currently configured in your runtime config. You can also force a streaming asset only build here, select your target platform and the used remote target settings.

DELETE ALL ASSET BUNDLE ASSIGNMENTS

This will remove all asset bundle assignments in your project. Does not matter if it was set from the auto assignment or manual by you.

DEBUG MODE

If active, it will allow to execute all entries in the Debug (Single Step) submenu. It's kind of a safety feature to prevent accidently clicking on single steps of the asset bundle pipeline.

DEBUG MODE (SINGLE STEPS)

Here you can find all steps which will be executed from the asset bundle build pipeline. You can trigger every step separately. Mainly needed for debugging and normally no need to use that.

DOWNLOAD BUNDLES IN EDITOR

If active, even in editor the normal asset bundle loader will be used instead of the in-editor version. Make sure you have the correct bundles already uploaded to the remote location.

UTILS

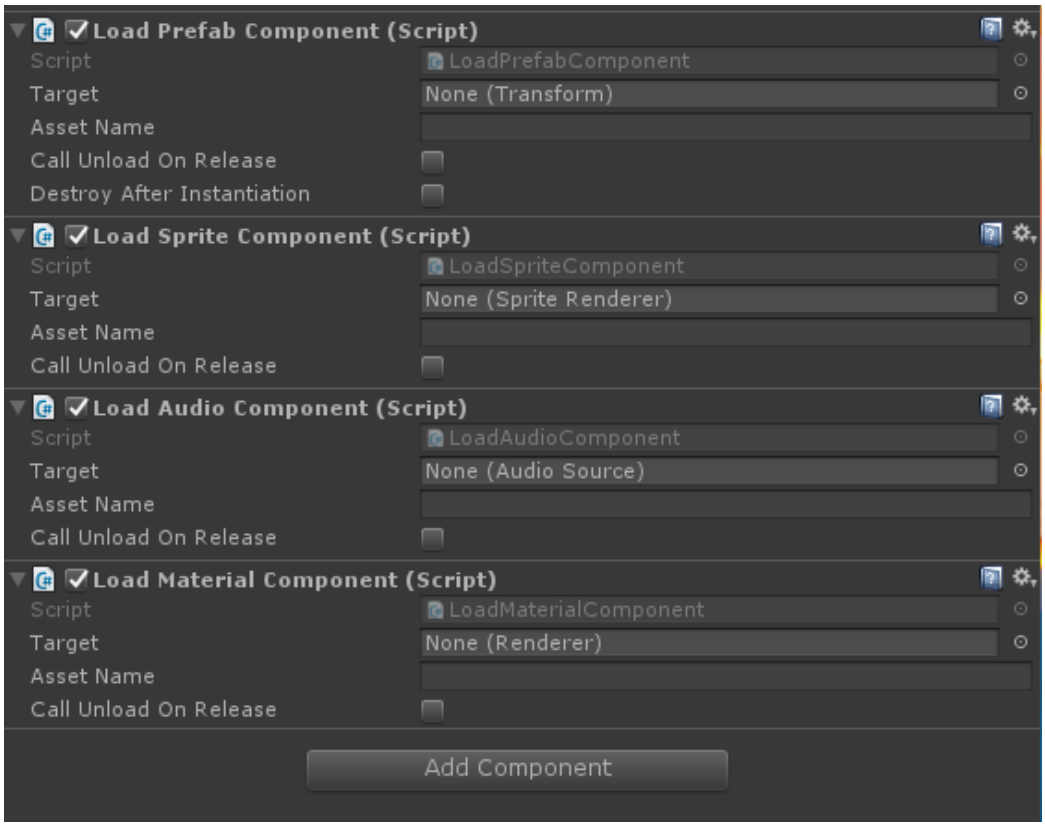
ASSET REFERENCE AND ASSET REFERENCE GROUP

Wrapper object which hold the loaded asset and will release the assets as soon as the asset reference object gets garbage collected due to a lack of reference. This is a try to keep the reference up to date when you just set you loaded asset to null. The asset reference group is doing the same but holds several assets instead of just one.

LOAD ASSET COMPONENTS

Components which are using the Asset Bundle Service to load assets and assign them after successful loading. You can easily write your own load components but you do not have to use them.

- Target – Defines the target where the loaded asset gets applied to
- Asset Name – Name of the requested asset
- Call Unload On Release – When assets gets release because the component gets destroyed then it will call on release with unload asset



COROUTINE SETUP

Wrapper class to pass your own setup of methods to start and stop coroutines. If you use external libraries or pass the methods from one of you monobehaviours is up to you.

LOG SETUP

Wrapper class to pass your own setup of methods for logging. You can connect your own logger or just the Unity debug methods. It's up to you.

WHO DO I TALK TO?

If you have feedback or any questions, please just write me. I will try to react as fast as I can and will be happy to hear from you.

Contact via Mail: unity@fisherman-production.de