- LEVEL 3 -

ADDRESSABLES

INSIDE THIS GUIDE

ADDRESSABLES WINDOW
BUILD PIPELINE
PROFILING
NETWORK DELIVERY
STRATEGIES

The Gamedev Guru *Level up your skills!*

BY RUBEN TORRES BONET

03

ADDRESSABLES WINDOW OVERVIEW

all the options at your hand

05

THE BUILD PIPELINE

spend your precious time where it matters

07

PROFILING

don't let sneaky assets ruin your pay raise

08

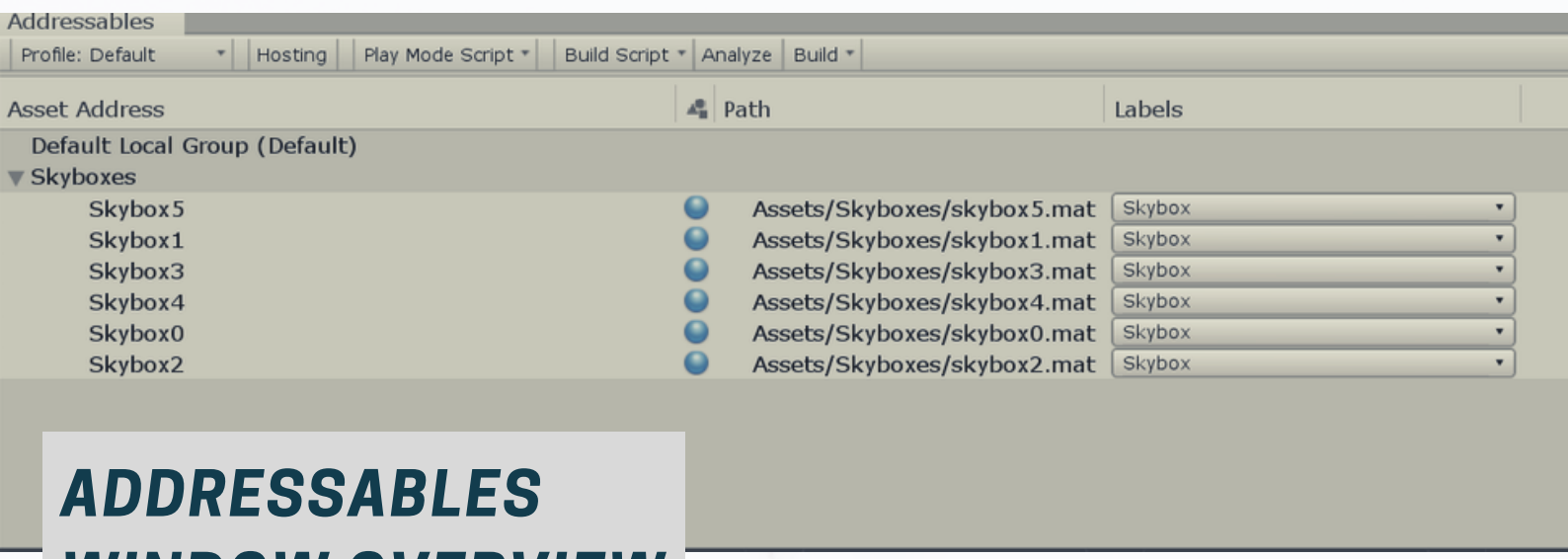
NETWORK DELIVERY

the highway to scalability

09

STRATEGIES

pragmatic tips between us



ADDRESSABLES WINDOW OVERVIEW

This is the heart of the system, your entry point to success.

Asset list

The main area contains the list of addressable assets — in our case, our skyboxes.
As a level 3 programmer, you will want to name them intuitively and group and label them in a way that is coherent with your asset access pattern.

Asset Names

The asset name or identifier is useful for loading in code, on top of pure readability in the editor. They can be renamed in each Material's inspector settings. You can offer hints such as *SkyboxBlue* to make your life easier when loading from code.

Asset Labels

The labels are great when you want to do batch operations such as *load all assets under the label Skybox*. Labels can easily be created in the scriptable object found under *Assets/AddressableAssetSettings*.

Groups

Groups become interesting when you decide it's time to cook certain assets the same way. You might want to compress them with LZMA, set specific request timeouts, pack them separately, mark them as static content, etc..
Groups can be created through a simple but elegant right click on top of the asset list.

Profiles

The global settings for your project's addressables system can be tweaked in the profiles you create.

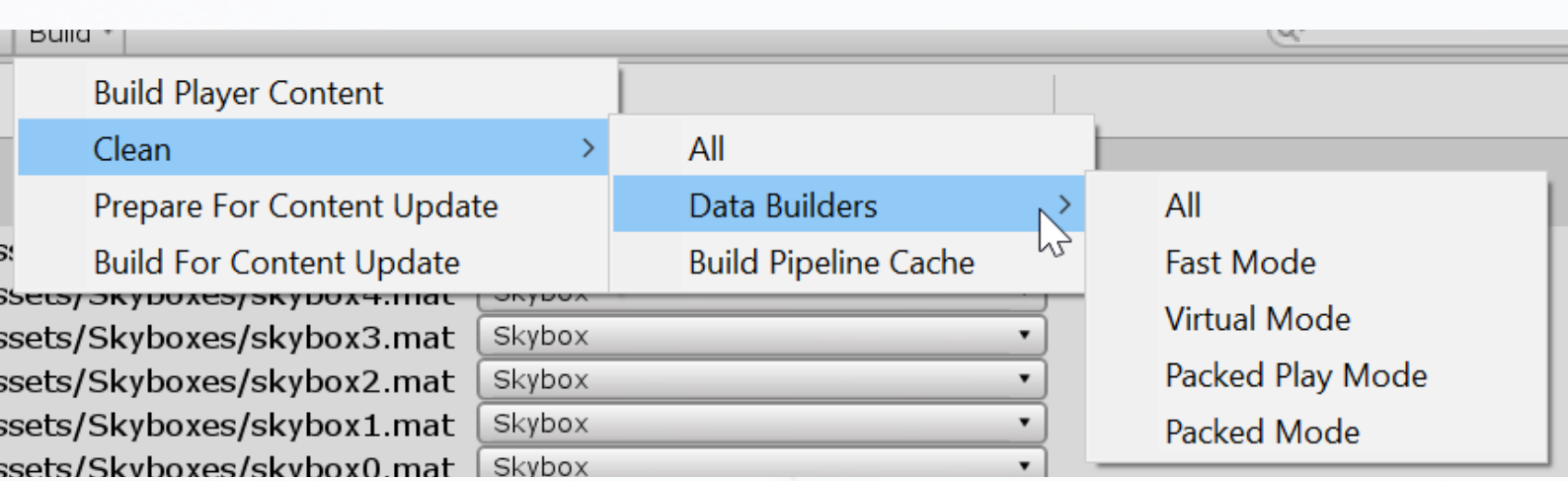
You can visit the default profile or create your own to handle different scenarios. Visiting the default profile will bring up some points worth mentioning.

Under *Send profiler events* you can enable profiling addressable events, useful for debugging purposes such as detecting possible memory leaks. You'll want to have it turned off the rest of the time though.

Each profile allows you to set different paths for each build target (Android, Standalone, etc.). Paths are defined both locally — relative to your project directory — and remotely — addresses players will access to to retrieve the content.

Hosting

Related to the asset delivery system. This will be explained in the Network Delivery chapter.



Play Mode Script
How the Unity editor loads the assets when pressing the play button is decided through the mode you select here.

There are three modes available:

- **Fast mode**: traditional way — Unity will load the assets from the AssetDatabase. Recommended for the typical developer day.
- **Virtual mode**: brings you closer to the actual final behavior, skipping the cooking process. Recommended when working on addressables.
- **Packed mode**: represents the player behavior, loading assets on demand from the packages that require prior cooking. Recommended for testing the final product.

Build Script
How assets will be cooked; the packed mode will cover most cases.

Analyze
Detect potential problems during development. I recommend you to use it frequently from the start.

Build
Indeed the most interesting action. Consider cooking as a synonym of building, term often used in other engines.

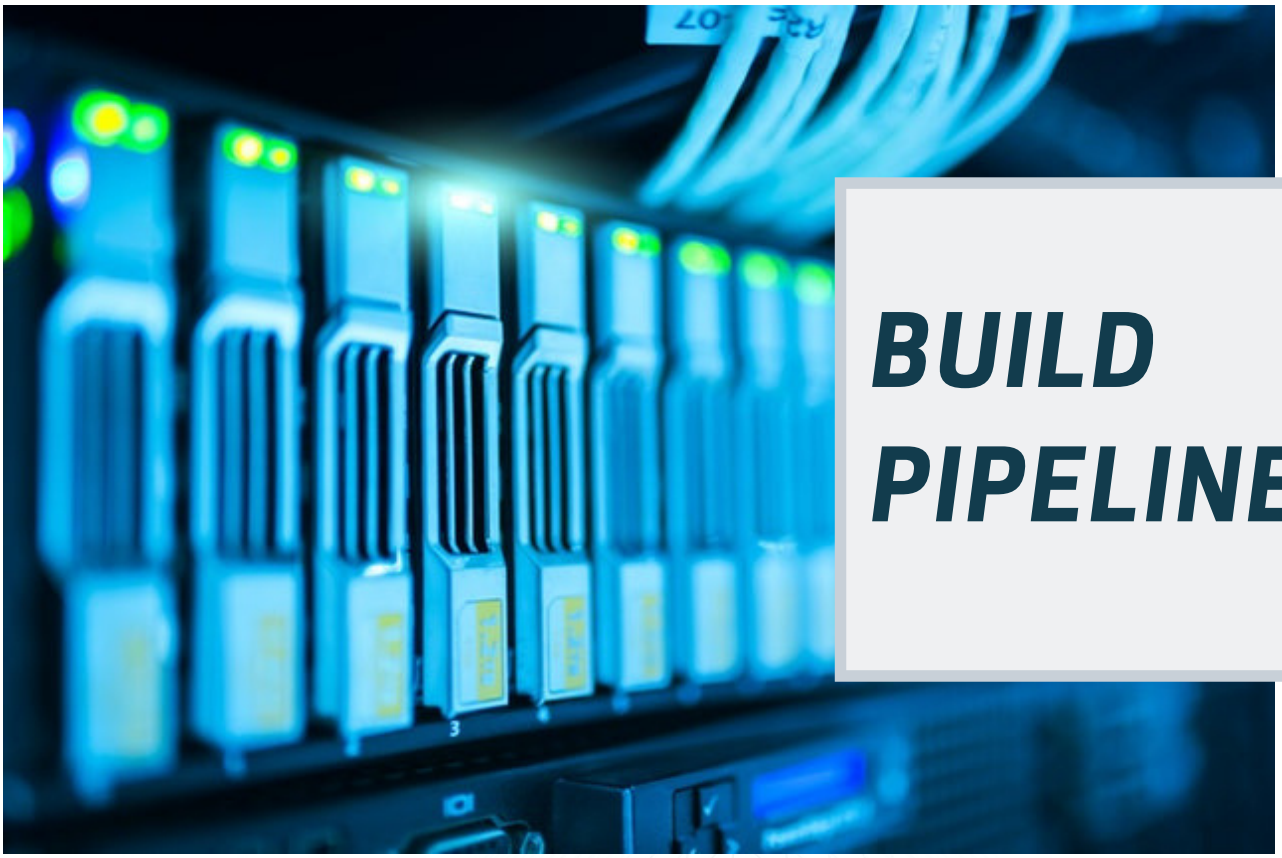
You will want to **cook player content** in the following situations:

- You are using the *Packed play mode* and assets changed between play sessions.
- You are building a player

You do not have to build player content manually, you can automate this step in your build scripts.

Often enough you will want to **clean** the cooked content to make sure that the process remains stable. This is useful in the *but it works on my machine* situations, as well as when setting new build nodes.

With Unity you will also be able to perform **content upgrades**. Rarely will your game be a one-shot release, it is common to update it over time with new content. In those cases, you will want to structure your data in a way that minimizes update times and network bandwidth wasted.



BUILD PIPELINE

Remember when I said you have to build the content player before building the player itself?

Well, I never said you had to do it manually. In fact, the process of automating it is so simple that you will have saved net time by the second time you build them.

What is more important, you will spare considerable frustration levels when you find out that you once forgot to cook the content and you see that things look pinkish on a device.

This is our goal here. To make it rock-solid and to focus on the things that matter.

Did you ever stumble upon the UnityEditor.Build namespace? If so, it might have been one of your first considerations if you had to automate the asset cooking process.

Well, that will not work for us for certain technical reasons, but worry no further because I have other solutions. As of now, we have two options depending on how you build your project:

A) Build & Run
B) BuildPipeline.BuildPlayer API

If you are using Unity's UI to build, you will need to patch the building process. Otherwise, if you are using your own scripts to build your packages (option B), then you just have to insert two lines of code before you call *BuildPlayer*. Check both options in the next page.

And that, my friend, that will be magic.

A) Using Unity's User Interface "Build & Run":
Assets/Scripts/Editor/PreBuildPlayerContent.cs

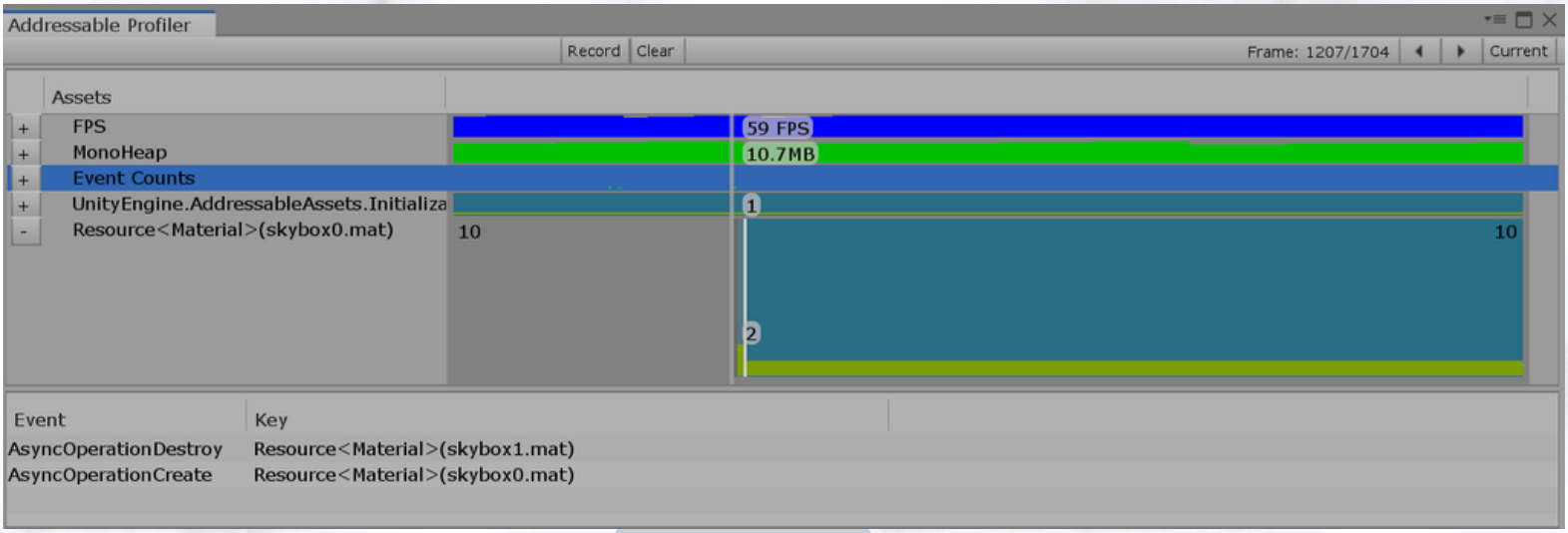
```
public class PreBuildPlayerContent
{
    [InitializeOnLoadMethod]
    private static void Initialize()
    {
        BuildPlayerWindow.RegisterBuildPlayerHandler(BuildPlayerHandler);
    }

    private static void BuildPlayerHandler(BuildPlayerOptions buildPlayerOptions)
    {
        AddressableAssetSettings.CleanPlayerContent();
        AddressableAssetSettings.BuildPlayerContent();
        BuildPlayerWindow.DefaultBuildMethods.BuildPlayer(buildPlayerOptions);
    }
}
```

AND NOW...

MAKE YOUR CHOICE

B) Using custom build pipeline:
Assets/Scripts/Editor/BuildPlayer.cs

```
public class BuildPlayer
{
    [MenuItem("Build pipeline/Build")]
    private static void Build()
    {
        // .....
        AddressableAssetSettings.CleanPlayerContent();
        AddressableAssetSettings.BuildPlayerContent();
        // .....
        BuildReport report = BuildPipeline.BuildPlayer(buildPlayerOptions);
        // ....
    }
}
```



PROFILING

Sometimes, no matter how skilled we might be, we make mistakes. And there are many types of mistakes, but those who pass unnoticed are the worst.

Luckily, in Unity you and I can rely on some tools to minimize the risk of those happening. One of these tools is the **Profiler**.

You can access it through *Window → Asset management → Addressables profiler*. Feel free to have a look around, but before using it, we do have to enable support for it. The quick way is going to *AddressableAssetSettings.asset* inside *Assets/AddressableAssetsData*. There, just enable *Send Profiler Events* and we are ready to rock.

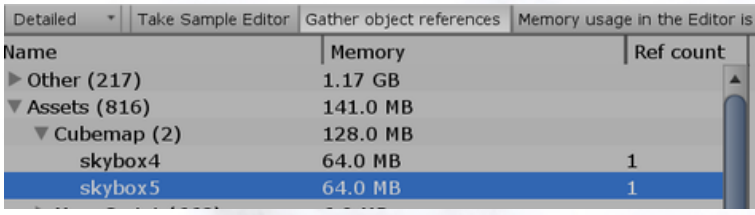
Well, what are we waiting for? Start a play session and then press clear and record to start profiling. You will end up with something like the screenshot displayed at the top of this page.

In such profiling session, you will see a few interesting metrics. Among the most useful are the events (load/unload/instantiate assets) and the current living objects with their reference counts.

So, this might be your first stop when something is not meeting your standards in your application.

No matter what happens, do not ever forget Unity has you covered with tools. Combine the addressables profiler with the detailed memory profiler and there're little things you will not be able to diagnose.

You will want to track the reference count. Skyboxes are likely to need only a maximum of a single reference count. Enemies will likely have a reference per instanced object.



Do you know what helped me in my projects?

Setting up expectations.

Expectations will give you standards about whether the results are appropriate or not. Then, just check with the profiler that they are met at different points of the game.

This is how you create rock-solid games.

NETWORK DELIVERY

One of my all-time favorites!

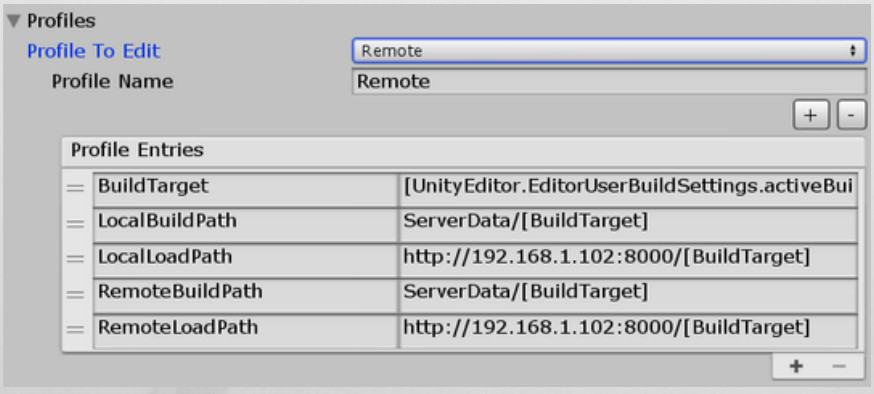
Why deliver great content over the network? And how? The curious, pragmatic mind will find answers here.

Some of the reasons to consider network delivery are:

- Reduce installation size and time
- Reduce unity editor play times
- Update content, not updating the client builds
- It's fun

Now the how. You and I will need to 1) tweak settings and produce the remote content and 2) deliver it over the network. Let us tweak the settings first.

In our *AddressableAssetSettings*, create a new profile for our remote configuration as depicted. Replace the IP address/port with your target development/production http server.



Once the profile has been created, select it in the main Addressables Window instead of the default one. Then, build the content as usual. This time, the files will not be in the Library directory but in the one you specified. A catalog will be generated so clients know where to download the assets from.

Go ahead: make a build and test it.

- *Spoiler*: it will not work.
- *Reason*: you do not have a http server running.
- *Solution*:
  - Use Unity's integrated http server. I do not suggest you this, as you would be dependent on keeping the editor running.
  - Use another http server, e.g. AWS or the integrated python module. If you have python 3 installed, navigate to ServerData directory in your command line and run: *python -m http.server*

There will be a subdirectory for each platform you target. Keep in mind: each addressables content build will create a **catalog file** with a different version of the bundled assets.
You might not want to cook the content on every player build, but only when the assets do really change.

strategies

Level up your skills!

1 - Faster editor iteration times

How many times do you press the play button per day? Chances are, many. My figure is above 35.

I do really hope for you that you are not waiting longer than 7 seconds for booting up in the scene you are working on.

If you do, I have a tip for you: make assets addressable, adapt your systems to work with them, cook the assets and set the editor play mode to packed. You will see shorter iteration times in your project.

Which assets should you mark as addressables? Preferably, the biggest ones you are not actively working on. You will benefit from this strategy in two ways:

- The contents will be loaded only when they are really needed. Previously, direct references forced full pre-loading.
- The cooked assets are pre-processed, therefore less effort is required to load them.

Are faster iteration times worth this investment? You bet they are!

2 - Faster player iteration times

This is the younger sibling of the editor version.

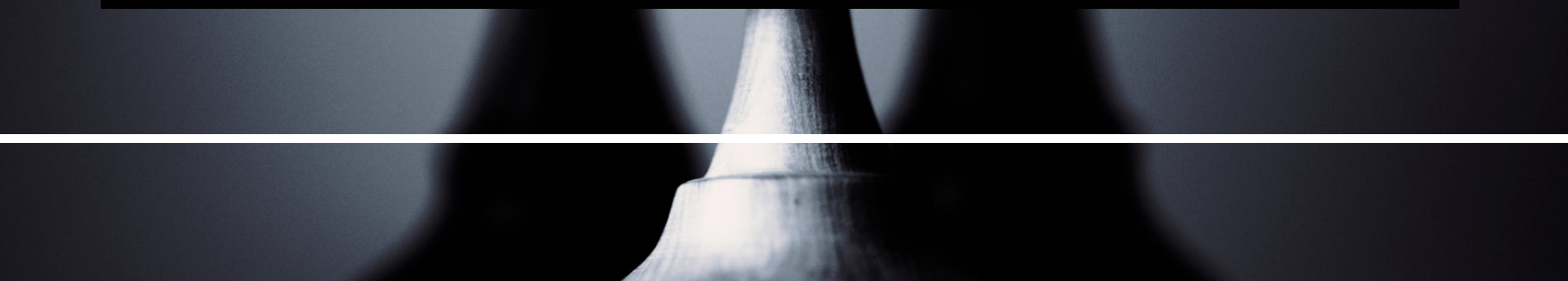
You will want to profit from this as well. Consider two scenarios: Person A installs a 3 GB package on Android 5 times a day, while person B installs a 1 GB 15 times a day. Now spot the two differences.

Here's the solution: smaller deployment packages lead to faster iteration times. You are likely to iterate on your project quite several times a day, you will want to reduce this to a minimum to get the most of your day.

How to achieve this?

Make heavy assets addressable, adapt your system to add support to this asynchronous behavior, set them to be remotely loaded, cook your content and profit!

If you are working in a medium-sized company, you are likely to have a few 24/7 servers behind a 1-Gb Ethernet switch. This is perfect for your new plans. Grab it and make it yours.



ADDRESSABLES

THAT WAS IT?

NO.

There is yet so much to uncover.
What's next?
Get in touch. Pass your feedback. You will get more of this.

The Gamedev Guru *Level up your skills!*

BY RUBEN TORRES BONET