## AssetOptimizationTool 使用说明

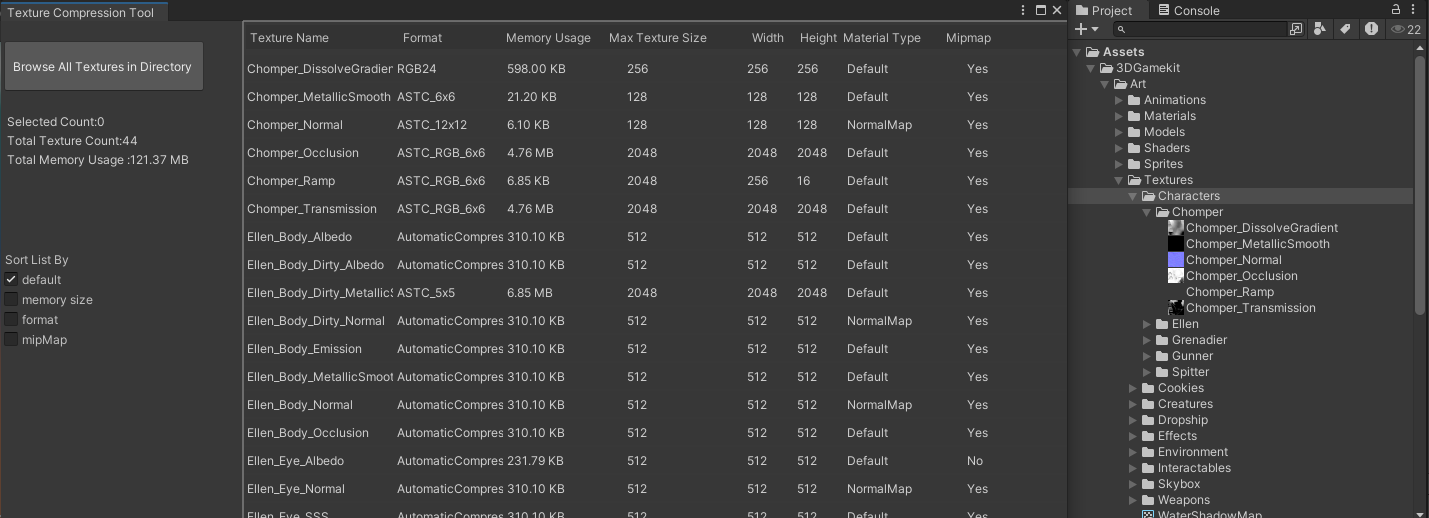
插件导入说明：

打开Windows -> PackageManager，点击左上角的+号，选择Add Package from disk，浏览本插件的根目录，选择package.json文件即可。使用时，点击菜单栏的AssetOptimization按钮。

1. Texture Compression Tool

TextureCompressionTool是一个可视化的贴图资源分类管理工具。本工具用于便捷地管理项目中的贴图资源，除了查看贴图的内存占用大小、压缩格式、尺寸等信息外，还支持批量修改贴图的压缩格式、最大尺寸、mipMap的开启或关闭。

首先，在Project面板中选择任意目录，然后点击BrowseAllTexturesInDirectory按钮。如果该目录包含贴图，则会出现列出该目录下的所有贴图，如下图所示：

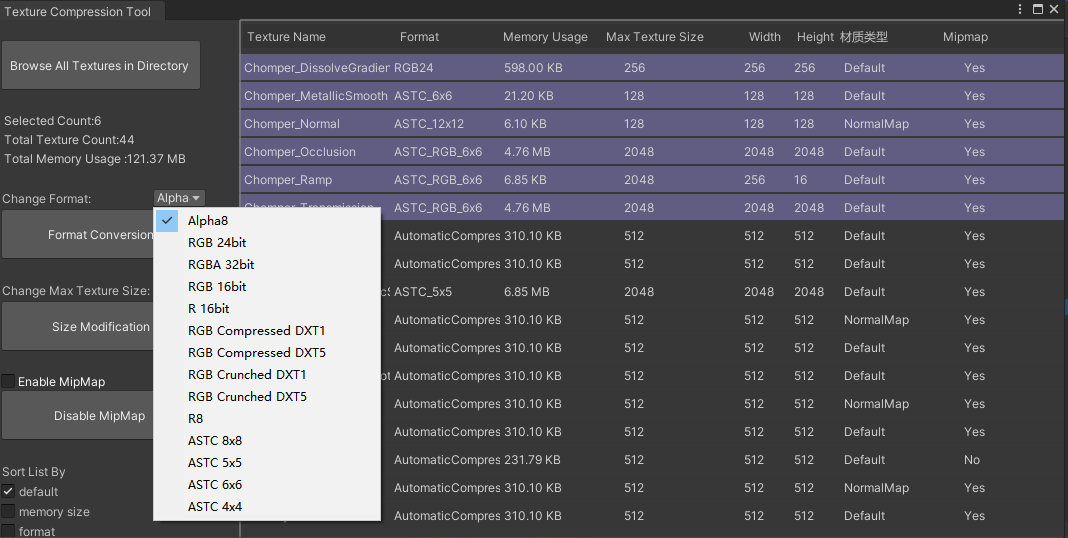


如果在未选择任何目录的情况下点击BrowseAllTexturesInDirectory按钮，则会列出项目中所有的贴图。如果项目中贴图数量很多，此步骤会消耗一定的时间。

列表左侧显示了该列表中贴图的总体信息。包括选择数量、总贴图数量、总贴图占用内存大小。列表还支持排序以方便用户查找，可以根据默认排序（即文件夹顺序）、贴图的内存占用排序、压缩格式排序、mipMap排序。

列表共分八个列，分别列出了贴图名称、压缩格式、内存占用、最大尺寸、宽、高、贴图类型、mipMap是否开启等信息。对于资源优化而言，最需要关注的三个信息是内存占用、压缩格式以及最大尺寸。贴图的压缩格式和最大尺寸决定了其在内存中的占用大小。

当选中列表中的任意一项时，列表左侧会出现三个新按钮：Change Format、Change Max Texture Size、Enable MipMap。它们的作用分别是改变压缩格式、改变最大尺寸、开启或关闭mipMap。列表支持使用Shift或Ctrl键多选以批量操作。如下图所示：



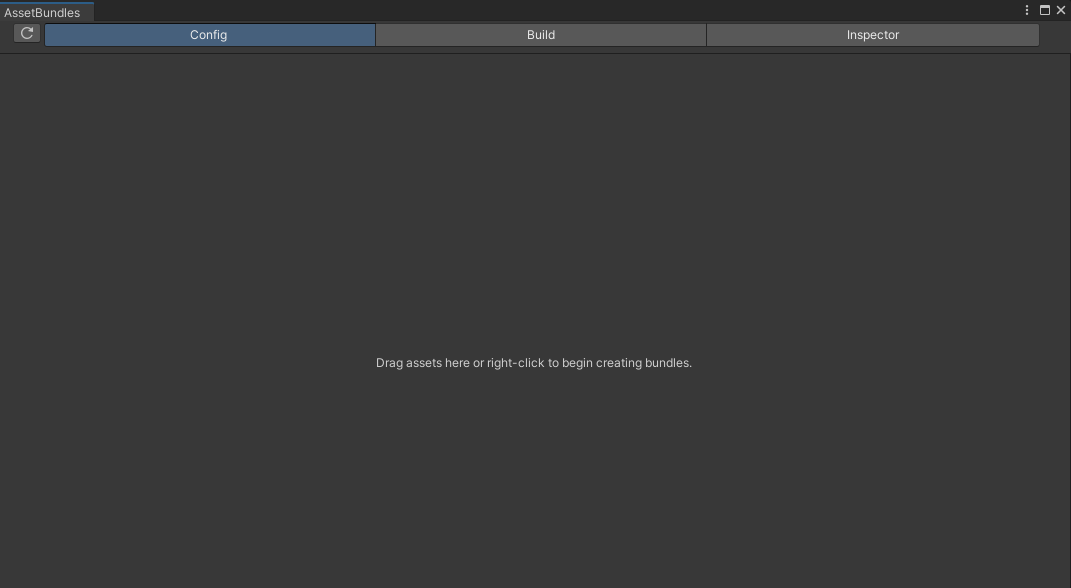
本插件支持多种常用贴图压缩格式的选择。对于webgl移动平台，通常推荐使用ASCT。ASTC是一种更先进的压缩格式，支持多种不同的压缩比率，可以在质量和文件大小之间进行良好的平衡。本插件也支持修改贴图的最大尺寸，从4096到32都可修改。由于webgl平台最大只支持2048x2048，为了兼容性和性能，建议使用1024x1024或512x512的贴图，这样可以提高加载速度并降低内存占用（对于部分贴图，可以使用256x256或更小，视项目实际情况而定）。本插件还支持一键开启或关闭贴图的mipMap，它是一种优化纹理图像的技术，主要用于图形渲染中的纹理映射。它通过为同一纹理提供多个分辨率版本来提高渲染性能和视觉质量。开启mipMap后，在渲染时，Unity根据物体与摄像机的距离选择最合适的mipMap级别。如果物体距离摄像机较远，会自动使用较低分辨率的mipMap，这样可以减少GPU计算和内存带宽的使用，从而提高渲染性能。但是，我们仅推荐分辨率较高、包含细节丰富的纹理开启此选项，需要在近处和远处都保持视觉质量。

1. AssetBundle Manager

AssetBundleManager是一个可视化的分包工具，支持资源包的创建、管理、打包、预览等一系列实用功能，帮助用户更方便的管理分包流程。

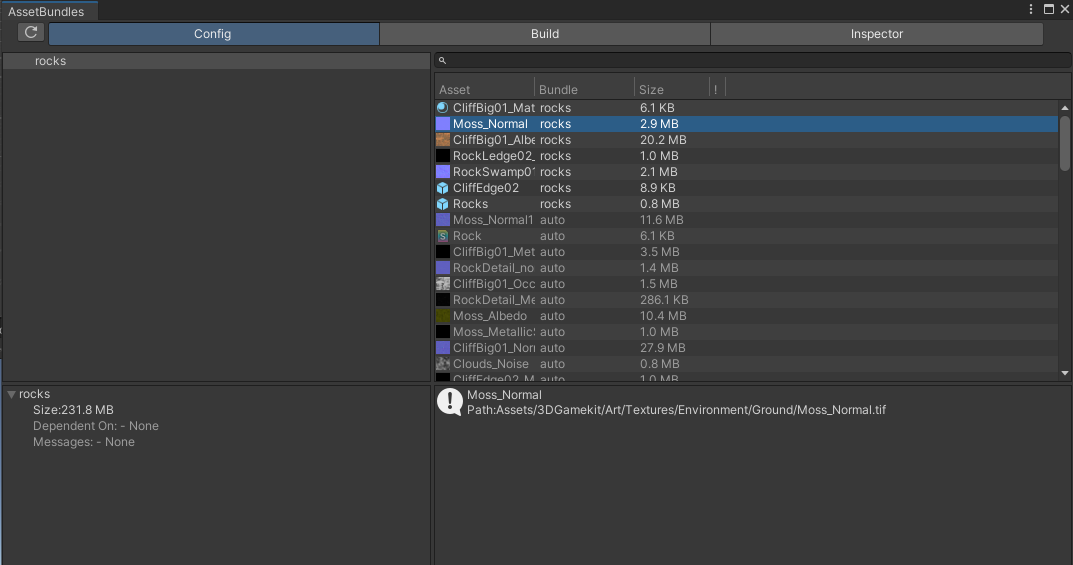
本工具的主菜单位于最上方，分为三个大类：配置、生成、检阅，对应三个页面。

1. 配置页面：



还未配置任何资源包时，该页面会提示用户从Project拖拽资源来生成资源包，或使用鼠标右键创建新的资源包。拖拽资源来生成资源包的方式与AssetBundle或Addressable标记资源为资源包的原理相同，资源可以是预制体、贴图、材质、模型、场景等。我们这里以预制体”rocks”为例，将其拖拽到Config面板。

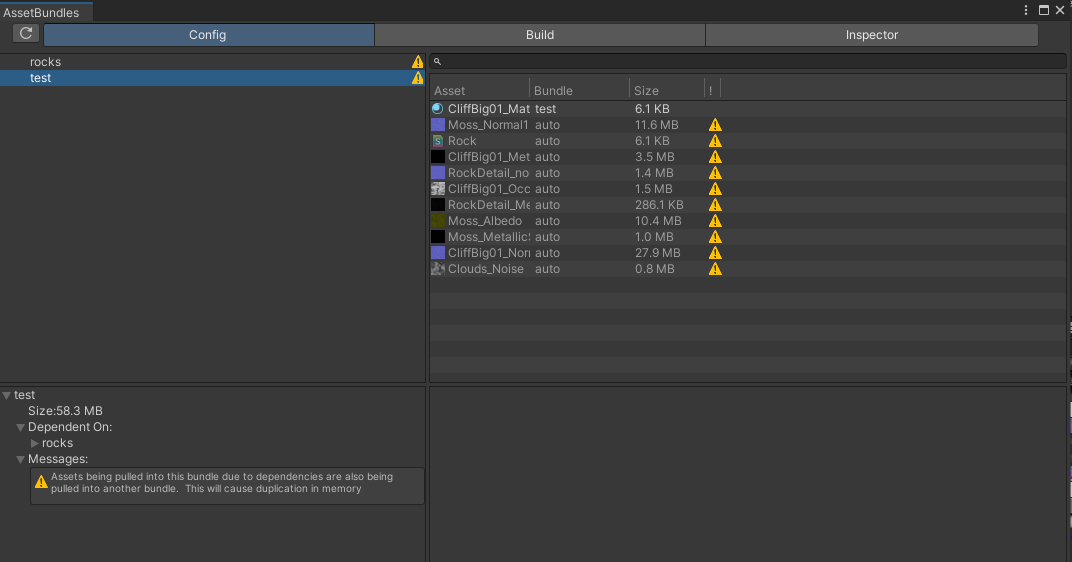
此时Config面板分为四个模块，左上角是AssetBundle的列表，在该列表选中任意资源包时，左下角会列出该资源包的细节、依赖信息，右上角列出该资源包所包含的资源内容列表（比如一个预制体可能包含多种资源，包括材质、贴图、模型、音频等）。当选中右上角列表里的任意对象时，右下角会展示选中对象的属性。如下图所示：



新建资源包：在左上角空白处右击时，可以添加一个新的空包或文件夹，用于创建你的自定义资源包。你可以将任意资源从Project面板中拖到自定义的包中，也可以在选中某个资源包时，从右上角的列表中将其包含的一个或多个资源拖到自定义的包中。在移动资源的过程中，插件会自动查找该资源的所有依赖项，将其一起打包进去。

查看资源详情：右侧列表共有四个列，分别是资源名称、所属的资源包、尺寸、提示信息。如果某一行是灰色的，且其所属的资源包为auto，说明这个资源是一个依赖项，被自动打包进了某个资源包中。比如一个材质资源，其包含的所有贴图资源都是依赖项。如果向资源包A添加该材质，那么其包含的所有贴图资源都会被自动打包到A中，并以灰色显示。

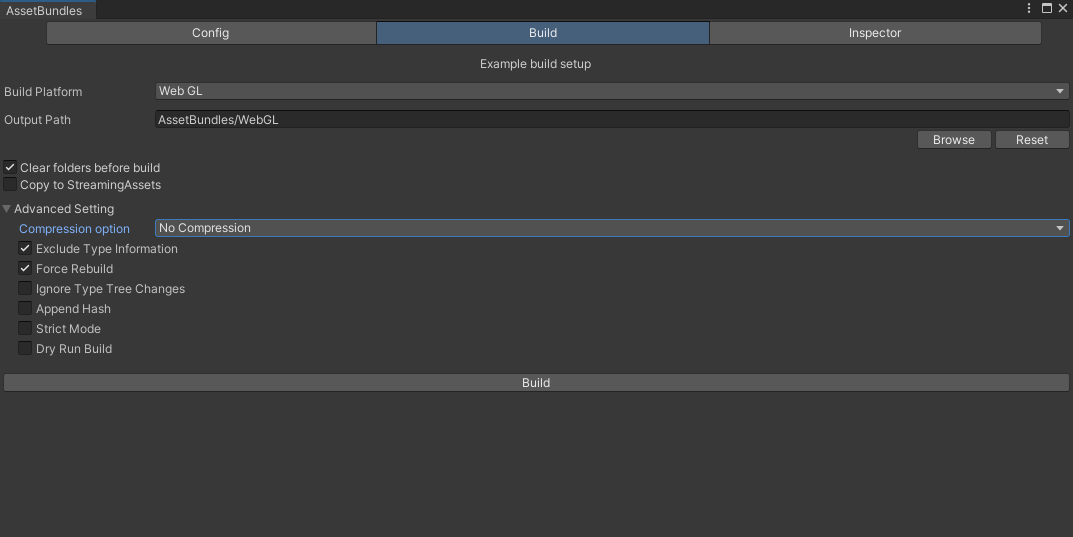
重复资源的检测与修复：本工具会自动检测依赖资源是否重复在多个资源中。仍然以材质为例，可能存在多个材质共用一张贴图的情况，如果将这些材质放入不同的资源包，那么就会重复将同一张贴图打包几次，造成不必要的内存浪费。接下来做一个测试，我们将rock中的一个材质拖入到新建的资源包”test”中，可以发现其右侧列表的资源多了一些黄色警告，如下图所示：



这些黄色警告所标记的资源，即为重复打包的资源。可以选择将这些资源移除或移动到新的资源包中。在左上角选中任意资源包并右键点击：Move duplicates to new bundle，即可自动将该资源包的重复资源移动到新的资源包中。右键菜单还包括了以下常用功能：添加同级、转换为变体、重命名包、删除包。

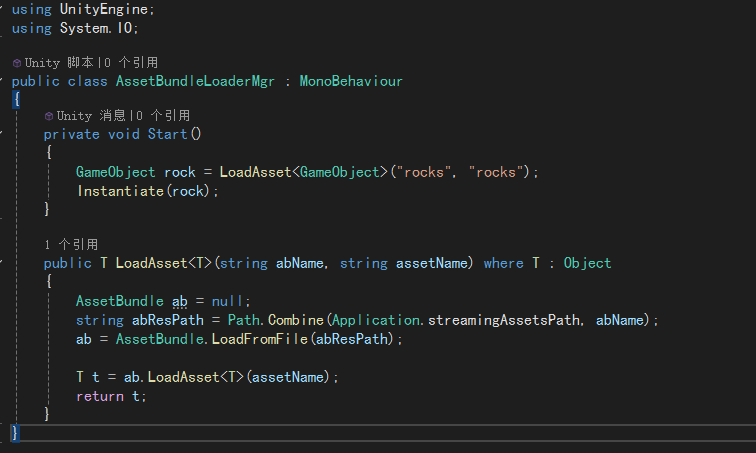
如果资源包有任何错误资源，右侧列表会出现错误图标，将鼠标放上去可以查看错误详情。

1. Build页面：

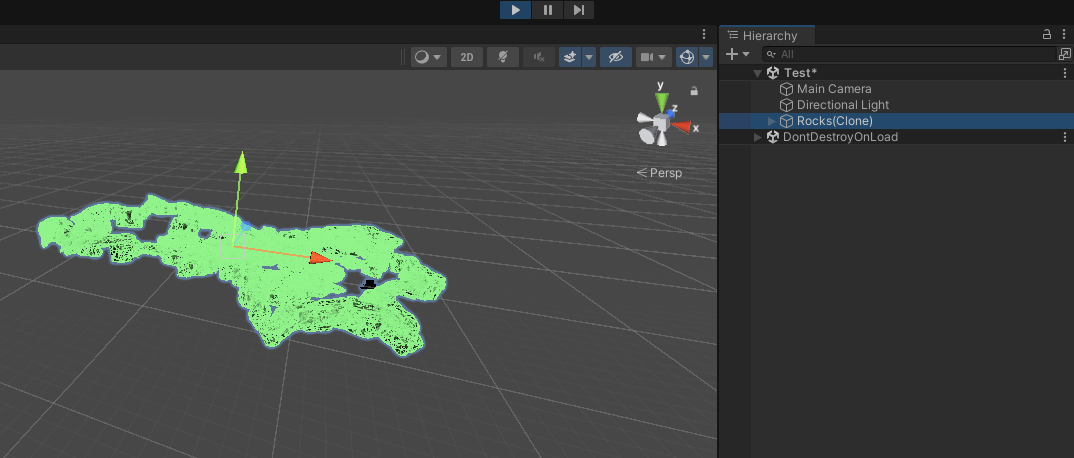


在此页面可以选择目标构建平台（默认是webgl）以及输出路径（默认为 AssetBundles/）。有两个可选项可供用户选择，分别是清除文件夹内的内容重新生成、复制到StreamingAssets文件夹中，方便测试时加载调用。

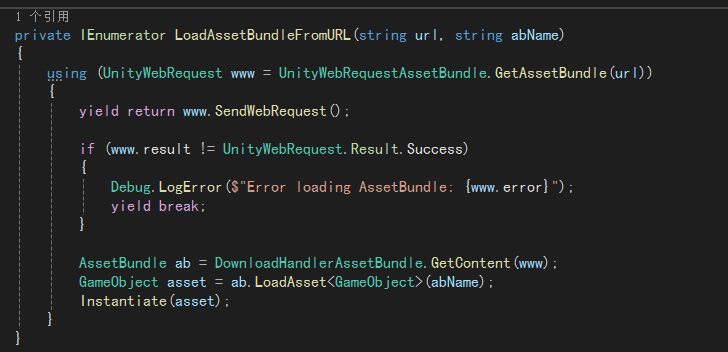
Ab包本地加载测试：点击Build按钮，开始生成AssetBundle文件。检查生成的文件夹，会发现除了ab包文件，还包括了对应的manifest文件。这些文件是加载时需要的配置文件。以加载StreamingAssets文件夹中的AssetBundle为例，核心代码如下：



将此代码挂载到场景任意物体上，点击运行，可以在编辑器里看到加载出的预制体：

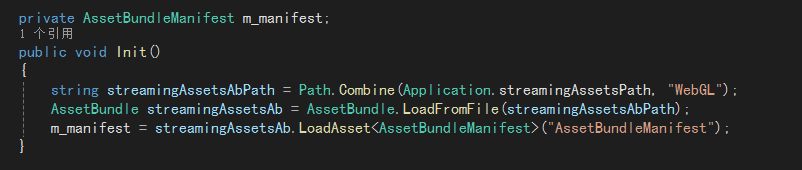


如果想要使用网络加载在线的AB包，只需使用下面的方法即可（url填入AB包所在的地址）：



加载依赖测试：上述的加载测试中，我们只是加载了 AssetBundle文件（没有后缀名的文件），并没有加载它的依赖文件（即同名的，后缀为Manifest的文件）。前面提到，在实际项目中我们会遇到多个预制体有相同的部分，所以我们可以把共同的部分单独作为一个AssetBundle进行打包，这样其他不同的部分就会跟这个共同部分产生一个依赖，我们要加载的时候，也必须把依赖加载上，不然就会造成资源丢失，不是原本的预制体的效果。

首先，需要新增一个方法，用于生成AssetBundleManifest，方便后面查找依赖：



该方法应该在程序开始时调用。它的目的是查找目录下名为WebGL的文件（具体名称取决于你的打包平台，它是插件自动生成的）并加载，然后生成AssetBundleManifest。

接下来，补充LoadAsset方法，加载目标包的所有依赖包：



本插件还提供了一些打包时可能用到的高级设置，内容如下：  
\*压缩：在不压缩、标准 LZMA 或基于块的 LZ4 压缩之间进行选择。

\*排除类型信息：不在资源包中包含类型信息。

\*强制重建：重建需要构建的包。这与“清除文件夹”不同，因为该选项不会删除不再存在的包。

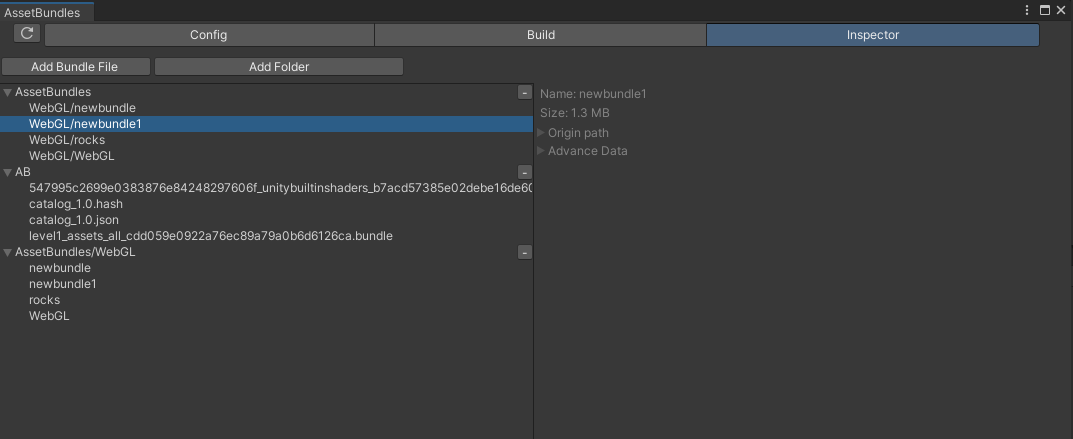
\*忽略类型树更改：在进行增量构建检查时忽略类型树更改。

\*附加哈希：在资源包名称后附加哈希。

\*严格模式：如果在构建过程中报告任何错误，则不允许构建成功。

\*干运行构建：进行一次干运行构建。

1. Inspector页面



你之前通过Build页面构建的路径将会自动添加到此处。该页面功能如下

\*点击“添加文件”或“添加文件夹”以添加要检查的包。

\*点击每行旁边的“-”以移除该文件或文件夹。注意，你无法移除通过添加文件夹而添加的单个文件。

\*选择任何列出的包以查看详细信息，包括：

1. 名称

2. 磁盘上的大小

3. 源资产路径：显式添加到此包的资产。请注意，此列表对于场景包是不完整的。

4. 高级数据：包括预加载表、容器（显式资产）和依赖项的信息。