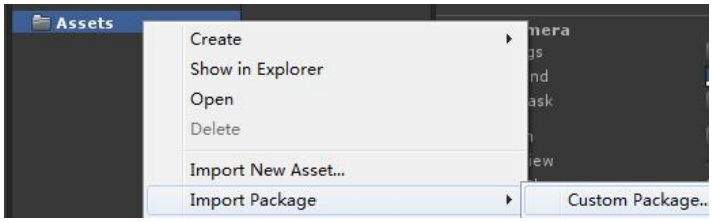
Summary

This tool is helpful to assist us build unity's asset bundle. We don't have to write script to do these things, instead of this, we just config a xml file, then click a button to let the command working automatic. It's Easy to use, enjoy it.

Usage

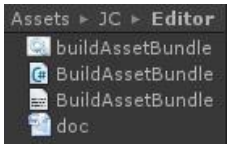
Install

1. Open your unity project
2. Import "jc.unitypackage"



3. Now your project window looks like this.

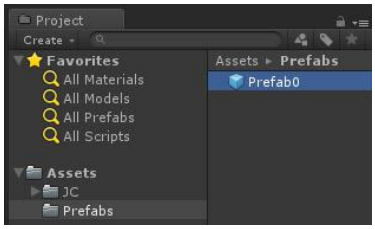
4. In the JC folder is some useful tool script, you can explore by yourself.
5. The core of build asset bundle tool is in the Editor folder.



The doc.docx and doc.pdf is the document.
The buildAssetBundle.bat is the command line tool.
The BuildAssetBundle.cs is the core script of this tool.
The BuildAssetBundle.xml is the config file of build asset bundle.
You can get the details of these files below.

A simplest example

1. First you nees a asset, and I use a prefab for example.



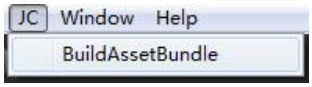
2. Open the BuildAssetBundle.xml file to type config.

   A thing in your mind that is the path of file or directory must be relative to Assets folder of your project.

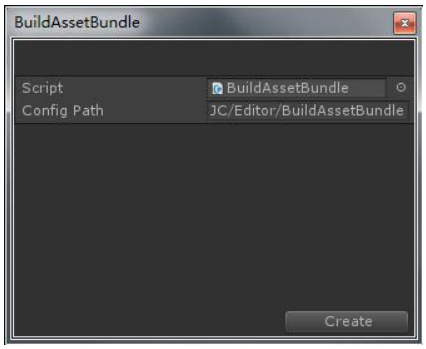
   And this is my config.

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <item type="file">
            <path><![CDATA[Prefabs/Prefab0.prefab]]></path>
        </item>
    </asset>

</buildAssetBundle>
```

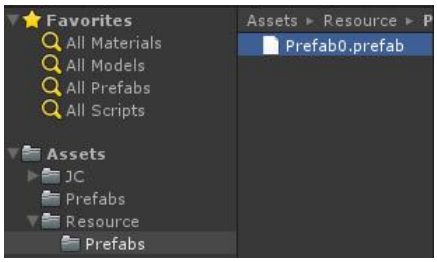
3. Run the build command.



Click create button.



Now the unity3d asset file is generated.



4. Let's write some script to load the asset

```
using UnityEngine;
using System.Collections;
using JC;

public class NewBehaviourScript : MonoBehaviour
{
    private void Start()
    {
        Loader.LoadSQ(new LoaderRequest(URL.file + URL.dataPath +
"/Resource/Prefabs/Prefab0.prefab.unity3d", LoaderType.AssetBundle), (response) => {
            Object.Instantiate(response.assetBundle.Load("Prefabs/Prefab0.prefab"));
        });
```

```
    }
}
```

5. That's all.

The details of the config file

1. Where the generated asset file save to?

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <item type="file">
            <path><![CDATA[Prefabs/Prefab0.prefab]]></path>
        </item>
    </asset>

</buildAssetBundle>
```

```
<saveRoot><![CDATA[Resource]]></saveRoot>
```

The saveRoot node is the root directory that all generated asset files will be save in it.

```
<path><![CDATA[Prefabs/Prefab0.prefab]]></path>
```

The path of a asset is spedfied in the node.

And the directory hierarchy will be mapping to the saveRoot directory.

2. How to build all assets in the directory?

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <item type="directory">
            <path><![CDATA[Prefabs]]></path>
        </item>
    </asset>

</buildAssetBundle>
```

```
<item type="directory">
```

The key is set the type attribute to directory, and specify a directory path into the path node.

3. How to build all assets in the directory recursively?

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <item type="directory" recursion="true">
            <path><![CDATA[Prefabs]]></path>
        </item>
    </asset>

</buildAssetBundle>
```

```
<item type="directory" recursion="true">
```

You may see the key is the recursion attribute, that's all.

4. How to build all assets into one bundle?

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <bind savePath="Test/test.unity3d">
            <item type="file">
                <path><![CDATA[Prefabs/Prefab0.prefab]]></path>
            </item>
            <item type="directory" recursion="true">
                <path><![CDATA[Prefabs]]></path>
            </item>
        </bind>
    </asset>

</buildAssetBundle>
```

All type of item node can be wrapped in bind node. Then all assets wrapped in bind node will generated into one bundle that name is specified in savePath attribute.

5. How to build dependent asset?

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <bind savePath="Test/test0.unity3d">
            <item type="file">
                <path><![CDATA[Prefabs/Prefab0.prefab]]></path>
            </item>
```

```
        </bind>
    </asset>

    <asset id="theSecondAsset" dependence="theFirstAsset">
        <bind savePath="Test/test1.unity3d">
            <item type="file">
                <path><![CDATA[Prefabs/Prefab1.prefab]]></path>
            </item>
        </bind>
    </asset>

</buildAssetBundle>
```

```
<asset id="theSecondAsset" dependence="theFirstAsset">
```

Set the asset id to the dependence attribute.

6. House to build a scene?

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <bind savePath="Test/test0.unity3d">
            <item type="file">
                <path><![CDATA[Scene/Scene0.unity]]></path>
            </item>
        </bind>
    </asset>

</buildAssetBundle>
```

It's simply like a common asset.

You can load the scene as usual, then use Application.LoadLevel api.

7. Build scene by prefab.

Sometimes we want put all objects of a scene into one prefab, and build the prefab assets.

Everything is well but the lightmap of scene.

Now you can type config like this.

```
<?xml version="1.0" encoding="utf-8"?>
<buildAssetBundle ignoreExtention="svn;meta" platform="StandaloneWindows">

    <saveRoot><![CDATA[Resource]]></saveRoot>

    <asset id="theFirstAsset">
        <bind savePath="Test/test0.unity3d">
            <item type="file">
                <path><![CDATA[Scene/Scene0.prefab]]></path>
                <lightmap><![CDATA[Scene/Scene0.scene]]></lightmap>
            </item>
```

```
        </bind>
    </asset>

</buildAssetBundle>
```

You can see a new node named lightmap. Specify the scene file at this node.

And after you load the asset file, set the lightmap data before instantiate scene prefab.

```
BuildAssetBundleData.SetupLightmap(response.assetBundle.Load("Scene/Scene0.prefab.lightmap"));
Object.Instantiate(response.assetBundle.Load("Scene/Scene0.prefab"))
```

Command line tool

Edit the buildAssetBundle.bat file with notepad.

```
echo off

:: set your unity path here
set unityPath=D:\Program Files\Unity\Editor

:: set your unity project path here
set projectPath=F:\XWorld\UI

:: invoke this method to build
:: don't change this manually
set command=JC.BuildAssetBundle.BuildDo

echo startup unity and build asset bundle
echo this step may take several minutes depends on the size of your asset bundle
echo please wait...

:: start up unity and build
"%unityPath%"\Unity.exe -projectPath %projectPath% -quit -batchmode
-executeMethod %command%

echo build asset bundle complete

pause
```

Then run this file. And now you don't have to open unity to build asset bundle, the work will run in background automatic.

Access config in your program runtime.

You also can access the config data in your progame runtime.

```
// Load the config file as string
// Assign to the variable
```

```
string configString;
BuildAssetBundleData.AssetBundleData assetBundleData = null;
if (!BuildAssetBundleData.AssetBundleData.ParseAssetBundleData(configString, out
assetBundleData))
{
    return;
}
```