

ASSETBUNDLEMAGIC  
GETTINGS STARTED

# SOMMARIO

Introduction .....	3
The first step .....	3
General section .....	4
Platforms section .....	4
Bundles section .....	5
Understanding the Workflow .....	6
Asset classification and organization into bundles .....	6
Bundle creation (and upload) .....	6
Bundles load strategy and management .....	7
Bundle runtime loading and unloading via AssetBundleMagic API .....	8
API: Load a local bundle in a synchronous mode .....	8
API: Load a local bundle in asynchronous mode .....	8
API: Download Version.txt file and update version information .....	8
API: Download a bundle from the network .....	9
API: Download a bundle from the network by checking versions first .....	9
API: Clean bundles cache .....	10
API: Unload a bundle .....	10
Set up for Testing bundle download .....	10
Chunk Manager .....	11
Demo scenes .....	13

# INTRODUCTION

Bundles are used in several circumstances in UNITY development. With bundles, you can postpone the asset download at startup time, you can update your game without re-publish it, you can manage asset variants to support low-level devices, you can manage dynamic load/unload of entire parts of your game, and so on.

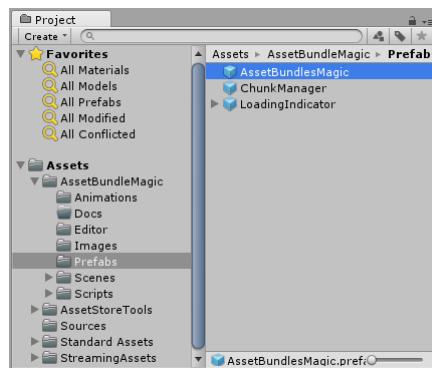
AssetBundleMagic is born with the goal of simplifying the workflow of asset bundles management, by driving the user in a well-defined process to support all possible use cases.

In a word, AssetBundleMagic is a Unity extension that let you manage AssetBundles with ease. With AssetBundleMagic you can use a very simple workflow that simplify the process and let you manage bundles from a complete different point of view. In addition, in the package, you will find some utilities addons, that help you to use asset bundles for common tasks.

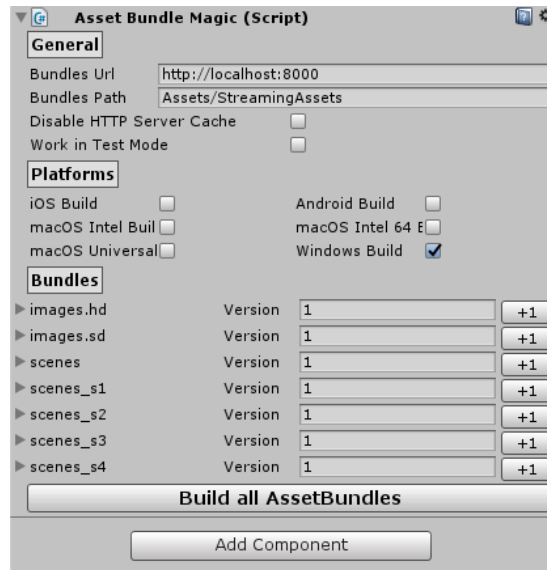
AssetBundleMagic exposes a very simple high level API that you have to learn, to manage the script-side of the process. In this document, you will find all you need to get started with the package.

## THE FIRST STEP

The first step you have to do, is the instantiation of the AssetBundleMagic prefab. To do that, you can browse the AssetBundleMagic package folder under Assets folder, the click on the Prefabs folder, and drag the “AssetBundleMagic” prefab on to the Hierarchy view.



Once you’ve dragged the prefab on the scene, you can take a look at the inspector.



The AssetBundleMagic prefab, lets you to configure the process of building and loading asset bundles. The inspector is organized in three sections. In the following paragraph, we will explain single item of the inspector.

## General section

The General section lets you to configure how AssetBundleMagic manages and stores asset bundles. In this section, there are some fields:

- **Bundles Url** – this field specify the Url from which AssetBundleMagic will connect to, to download bundles when requested via API. To use this feature, you will upload your bundles (created with AssetBundleMagic) on the server at this Url.
- **Bundles Path** – This is the path in which asset bundles are stored when created, and where are loaded when you ask AssetBundleMagic to load a bundle from file. If you plan to load bundles locally, you have to put “Assets/StreamingAssets” in this field.
- **Disable HTTP Server Cache** – This field tells AssetBundleMagic to try to disable HTTP Server caching, to always obtain a fresh copy of the element being downloaded.
- **Work in Test Mode** – When this flag is set, AssetBundleMagic load all requested bundles locally, without any network connection, also if the user calls “DownloadBundle” like methods. This is a useful feature to help programmers to test bundles before upload them to the server.

AssetBundleMagic can manage asset bundles locally or remotely. The creation part of the process is always local, in the path you’ve specified in **Bundles** Path field. You will select via API how to load a specified bundle.

## Platforms section

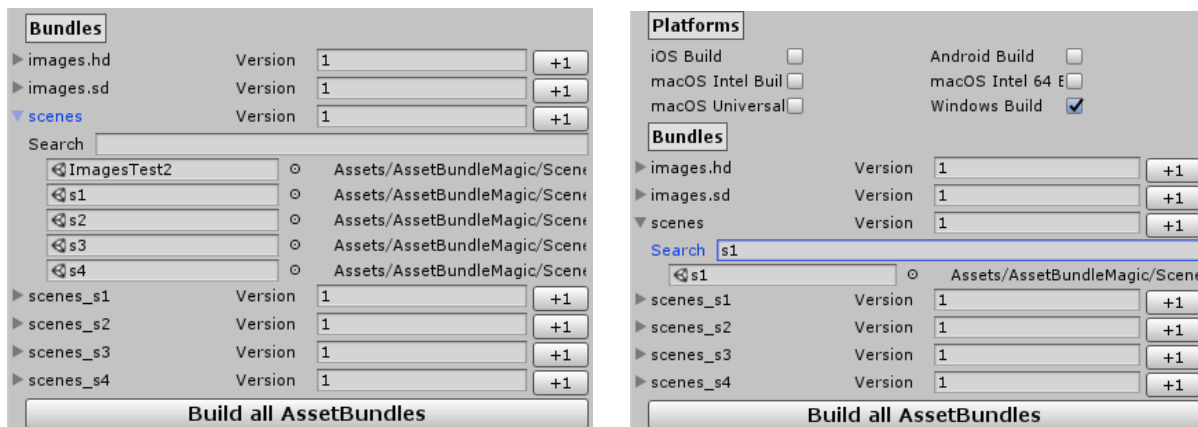
The Platforms section lets you specify for which platforms generate bundles. For development purposes, you can enable only the platform you are using (e.g. Windows, macOS), when you have to deploy the game, you can build all platform you need. AssetBundleMagic will organize all bundles in a directory structure based on platform’s name, under the base path you specify in the General section.

## Bundles section

The Bundles section lists all bundles defined for your assets. For every bundle, you can put the version number, that you have to increment every time you make a change on assets that belongs to that bundle.

AssetBundleMagic will use this version number by matching it with the version number on the server (or in the local path) to choose to load the bundle or reuse the cached one.

Each bundle item also let you to search within a specified bundle, to check all asset in it. Simply expand the bundle name, and put a search string in the proper field, to narrow the asset list.



The button “Build Asset Bundles”, creates the bundles for selected platforms, at the specified build path. The general structure of the create bundles will follow the schema:

- **[BASE PATH]**
  - **[PLATFORM1 NAME]**
    - **[BUNDLE1 NAME]**
    - **[BUNDLE1 MANIFEST]**
    - **[BUNDLE2 NAME]**
    - **[BUNDLE2 MANIFEST]**
    - ...
    - **Versions.txt**
  - **[PLATFORM2 NAME]**
    - **[BUNDLE1 NAME]**
    - **[BUNDLE1 MANIFEST]**
    - **[BUNDLE2 NAME]**
    - **[BUNDLE2 MANIFEST]**
    - ...
    - **Versions.txt**
  - ...

The Version.txt file is automatically generated by the build process, and contains data about bundles name, versions and CRCs). At load time, AssetBundleMagic, will use this file to choose the appropriate action. This file has the JSON format, and the following structure:

```
{
  "bundles": [
    { "bundleName": "[name1]", "version": [version1], "crc": [CRC1] },
    { "bundleName": "[name2]", "version": [version2], "crc": [CRC2] },
    { "bundleName": "[name3]", "version": [version3], "crc": [CRC3] },
    { "bundleName": "[name4]", "version": [version4], "crc": [CRC4] }
  ]
}
```

## UNDERSTANDING THE WORKFLOW

The general workflow using AssetBundleMagic, has the following schema:

1. Asset classification and organization into bundles
2. Bundle creation (and upload)
3. Bundles load strategy and management
4. Bundle runtime loading and unloading via AssetBundleMagic API
5. Testing

### Asset classification and organization into bundles

It is the crucial part of the workflow. This step must take into account the main goals that have made you choose asset bundles as a development strategy. For example, if you chose to adopt asset bundles to dynamically load and unload pieces of a particular scene, you have to put in the same bundle all assets that belongs to the part that you want load or unload at runtime. If you chose to use bundles to create asset variants based on the specific class of device capabilities, you have to put in different (variants of) bundles, different version of the same assets, at different resolution (think about meshes, textures, media files). Finally, if you plan to update the game via bundles, you should organize bundles by updatable parts. You might also have to implement several of these use cases simultaneously.

This part of the process is supported directly by UNITY editor. To assign a particular asset to a bundle, you can use the standard editor. Please refer to [official documentation](#) for details.

### Bundle creation (and upload)

Once you've classified all asset into bundles, you can create bundles for platform that you want. To create bundles, you simply press the "Build All AssetBundles" button, and AssetBundleMagic do the rest. By calling the UNITY API, the process will create bundles that really need to be rebuilt.

When you build bundles, AssetBundleMagic creates a new version of “Versions.txt” file, that contains all data needed at load time, including the bundles version. If you want some bundles to detect as new version, you have to increment the version number of that bundle, prior to press the “Build All AssetBundles” button.

At runtime, AssetBundleMagic will reload (re-download) only bundles for which the last version is older than the new version in Versions.txt file.

If you plan to download asset bundles from internet, you have to upload all bundles (of all platforms) on a public accessible WEB server directory. You will copy the entire base path specified in General section. The address of the server must be equal what you specified in base path URL in General section.

## Bundles load strategy and management

When bundles are created and available, there are several methods to manage them. The main choice you have to do is:

1. Maintain and distribute bundles inside the game, and distribute them with the game too?
2. Separate bundles from the main game core, and download them as soon as the game needs them?

Furthermore,

1. You manage variants, so you have several versions of some assets?
2. Different bundles contain different assets, so load a bundle only when an asset inside it is needed?

The choices are very game specific, AssetBundleMagic give you a simple high level API to manage these use cases. In the following paragraph, we introduce the complete API, but AssetBundleMagic uses a particular strategy for bundle management that we have to introduce, prior to explain the API: the Version.txt file.

This file is only used when you work with remote asset bundles, and represents the complete bundles state on the server. For each bundle it stores the name, the version number, and the CRC for error free download check.

The main idea is that when your game boots up, it downloads the Version.txt file, to update the version state of the bundles on the server; when you request AssetBundleMagic to download a bundle, for example to dynamically loading a level in additive way, AssetBundleMagic check if the cached version of the bundle (if there is one) matches with the version on the server. If so, uses the cached version, else proceed to download.

If you don't update the bundle versions, the download will always take place. You can choose to download the versions of bundles only once (for example at game start), in this case, if you update bundles during a play, the game will not be updated. If you download Version.txt each time you request an asset bundle, you download it each time it is updated on the server, also during a play. There is a specific API that do the work for you.

Refer to the next paragraph for a complete description of the API.

# Bundle runtime loading and unloading via AssetBundleMagic API

To working with AssetBundleMagic, there is a specific simple high level API, so you have to do all via scripting. AssetBundleMagic implements the “Singleton” design pattern, so the access to all API methods is made by a call of type:

```
AssetBundleMagic.SomeMethod();
```

## API: LOAD A LOCAL BUNDLE IN A SYNCHRONOUS MODE

### Method signature:

```
public static AssetBundle LoadBundle (string bundleName);
```

This is the simplest method to load a bundle. You specify the name of the bundle (bundleName) and AssetBundleMagic will return the bundle as method result, synchronously. This method is not the better way to load an asset bundle, except for very particular situations.

## API: LOAD A LOCAL BUNDLE IN ASYNCHRONOUS MODE

### Method signature:

```
public static Progress LoadBundle (string bundleName, LoadBundleFinisehdDelegate finished)
```

### Delegate:

```
public delegate void LoadBundleFinisehdDelegate (AssetBundle ab);
```

This is the asynchronous version of the previous method, and let you supply a delegate to receive the bundle when it is ready to use. In the meantime, you can use the **AssetBundleMagic.Progress** instance returned from the method, to check the progress status of the download. The Progress class exposes the **GetProgress()** method to check it.

**Tip:** Use this method whenever you want to load an asset bundle locally.

## API: DOWNLOAD VERSION.TXT FILE AND UPDATE VERSION INFORMATION

### Method signature:

```
public static void DownloadVersions (DownloadVersionsFinisehdDelegate finished,  
DownloadVersionsErrorDelegate error)
```

### Delegates:

```
public delegate void DownloadVersionsFinisehdDelegate (string versions);
```

```
public delegate void DownloadVersionsErrorDelegate (string error);
```

This is the call you have to made, for update versions information inside AssetBundleMagic engine. Typically, this is the first call to the AssetBundleMagic package, and if you aren't interested in hot update of bundles, this call is



made only once per run of the game. If you don't call this method, AssetBundleMagic doesn't know versions on the server, and then it always proceeds to download bundles when requested.

The success delegate is called once the versions file is downloaded and processed. The content of the Versions.txt file is passed to the delegate only for debugging purposes; when the delegate is called, its content has already been processed, and internal versions state of AssetBundleMagic package has already been updated.

**Tip:** Call this method at least one time, at the game start.

## API: DOWNLOAD A BUNDLE FROM THE NETWORK

### Method signature:

```
public static Progress DownloadBundle (string bundleName, LoadBundleFinishedDelegate finished, LoadBundleErrorDelegate error)
```

### Delegates:

```
public delegate void LoadBundleFinishedDelegate (AssetBundle ab);  
public delegate void LoadBundleErrorDelegate (string error);
```

This is the main method to download a bundle. You specify the bundle name, and AssetBundleMagic downloads the bundle giving you the chance to monitor the download process via AssetBundleMagic.Process instance, immediately returned by this method.

When the download has finished, the LoadBundleFinishedDelegate is called, passing the downloaded asset bundle. In case of error, the LoadBundleErrorDelegate is called, passing the error string.

**Tip:** Call this method to download bundles from a server, without the need to check last-minute updates of the downloaded bundle.

## API: DOWNLOAD A BUNDLE FROM THE NETWORK BY CHECKING VERSIONS FIRST

### Method signature:

```
public static void DownloadUpdatedBundle (string bundleName, LoadBundleStartedDelegate started, LoadBundleFinishedDelegate finished, LoadBundleErrorDelegate error)
```

### Delegates:

```
public delegate void LoadBundleStartedDelegate (Progress p);  
public delegate void LoadBundleFinishedDelegate (AssetBundle ab);  
public delegate void LoadBundleErrorDelegate (string error);
```

This method is equivalent to the previous one, except for the fact that AssetBundleMagic first checks versions of bundles by downloading a fresh copy of Version.txt file. For that reason, this method doesn't start immediately the download of the bundle (because it must download Versions.txt first), and returns nothing (void). When the download of the bundle starts, the LoadBundleStartedDelegate is called, and the Progress is passed to it, so you

can monitor the download progress. As in the previous version of the method, delegate methods are called to signal the download finished or download error situations.

**Tip:** Use this method if you want to be sure to download the most updated version of a bundle.

## API: CLEAN BUNDLES CACHE

### Method signature:

```
public static void CleanBundlesCache ()
```

This method cleans all bundles from the cache. After calling this method, all bundles will be downloaded again.

## API: UNLOAD A BUNDLE

### Method signature:

```
public static void UnloadBundle (string bundleName, bool unloadAssets)
```

Unload a specific bundle if loaded, and removes the bundle name from the list of current loaded bundles inside AssetBundleMagic package.

## Set up for Testing bundle download

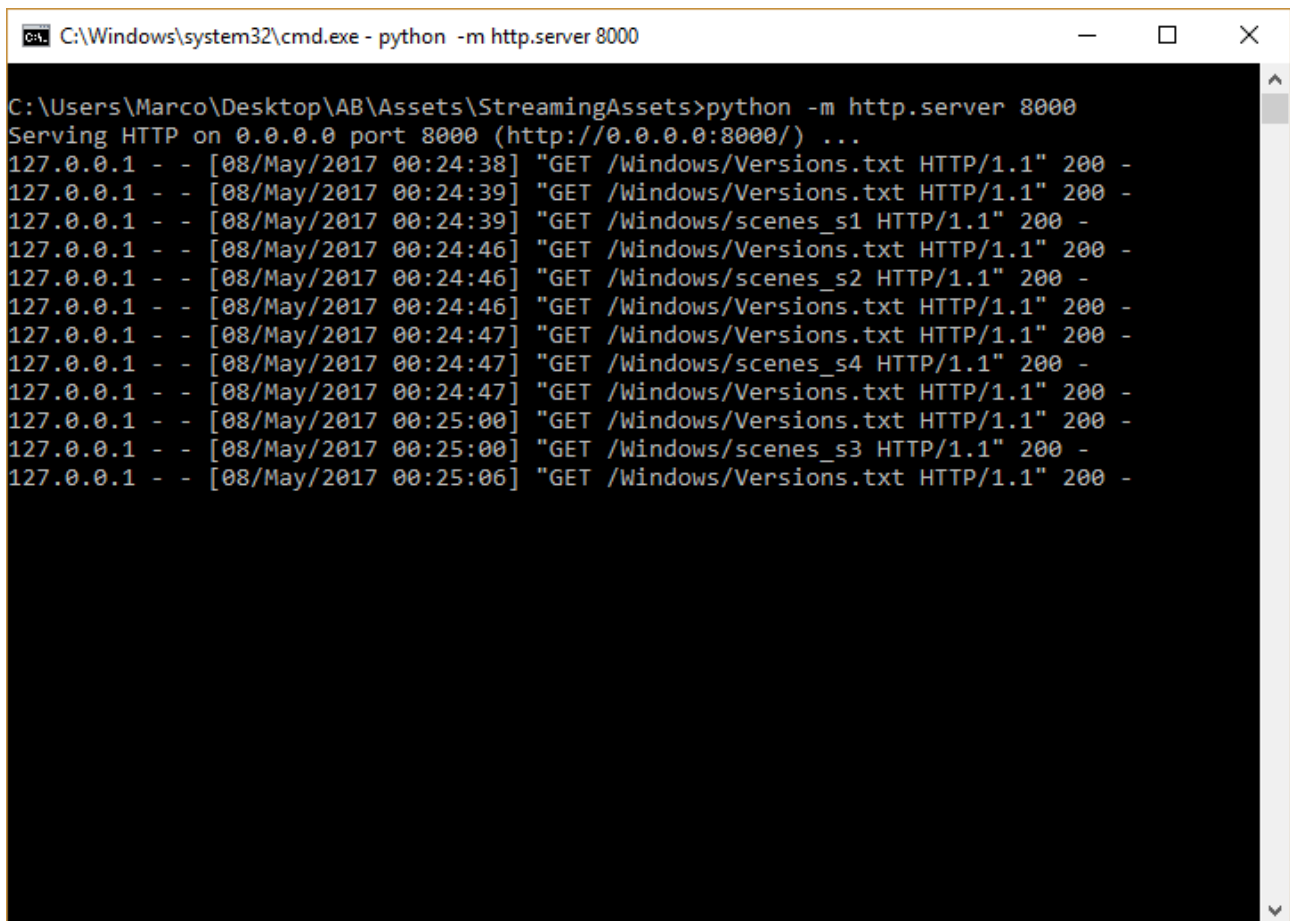
During the development stage, you can enable “Work in test mode”, in general settings of inspector. When this flag is set, AssetBundleMagic will load bundles locally, even if you use methods for download (DownloadBundle and DownloadUpdateBundle).

If you want to test the effective download, for example to test download time and general user experience, you can download a simple basic web server in the local machine, and set up it to serve the StreamingAssets local directory in the Assets project folder (or what you’ve configured as local bundle path).

A simple solution for this is to download Python 3.6.x (<https://www.python.org/downloads/release/python-360/>) available for many platforms. With python installed, and python executable reachable from the command line, you can go in the StreamingAssets folder and type:

```
python -m http.server <port number>
```

this will start a python process that serves the local directory via HTTP at the specified port. In the following screenshot, you can see the HTTP server in action.



```
C:\Windows\system32\cmd.exe - python -m http.server 8000

C:\Users\Marco\Desktop\AB\Assets\StreamingAssets>python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [08/May/2017 00:24:38] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:39] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:39] "GET /Windows/scenes_s1 HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:46] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:46] "GET /Windows/scenes_s2 HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:46] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:47] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:47] "GET /Windows/scenes_s4 HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:24:47] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:25:00] "GET /Windows/Versions.txt HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:25:00] "GET /Windows/scenes_s3 HTTP/1.1" 200 -
127.0.0.1 - - [08/May/2017 00:25:06] "GET /Windows/Versions.txt HTTP/1.1" 200 -
```

## CHUNK MANAGER

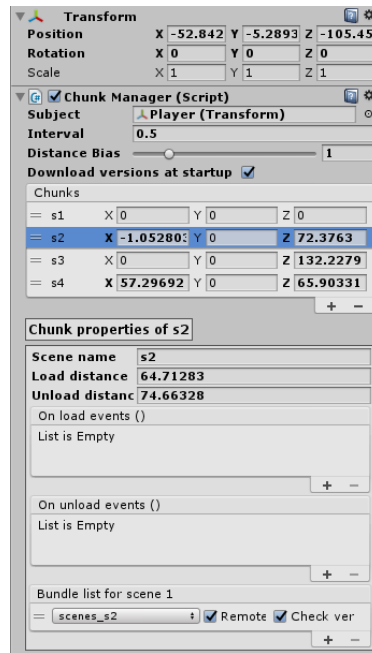
Inside the AssetBundleMagic package, there is another great UNITY extension, to manage dynamic loading/unloading of pieces of a big game level.

ChunkManager allows to define chunks, pieces of the main level, organized as UNITY scenes, that ChunkManager load/unload via SceneManager.LoadScene in additive mode. Each chunk has a reference scene, a center in the 3D space, and two zones: the enter zone and the exit zone. Each "n" seconds ChunkManager check if the player is within a new enter zone of a chunk, and if so, load the reference scene for the chunk.

When the player exit from the exit zone of a chunk, the ChunkManager unload its reference scene.

The ChunkManager can work with local scenes (scenes included in the target UNITY executable) and with scenes included in asset bundles (local or remote), via AssetBundleMagic mechanism.

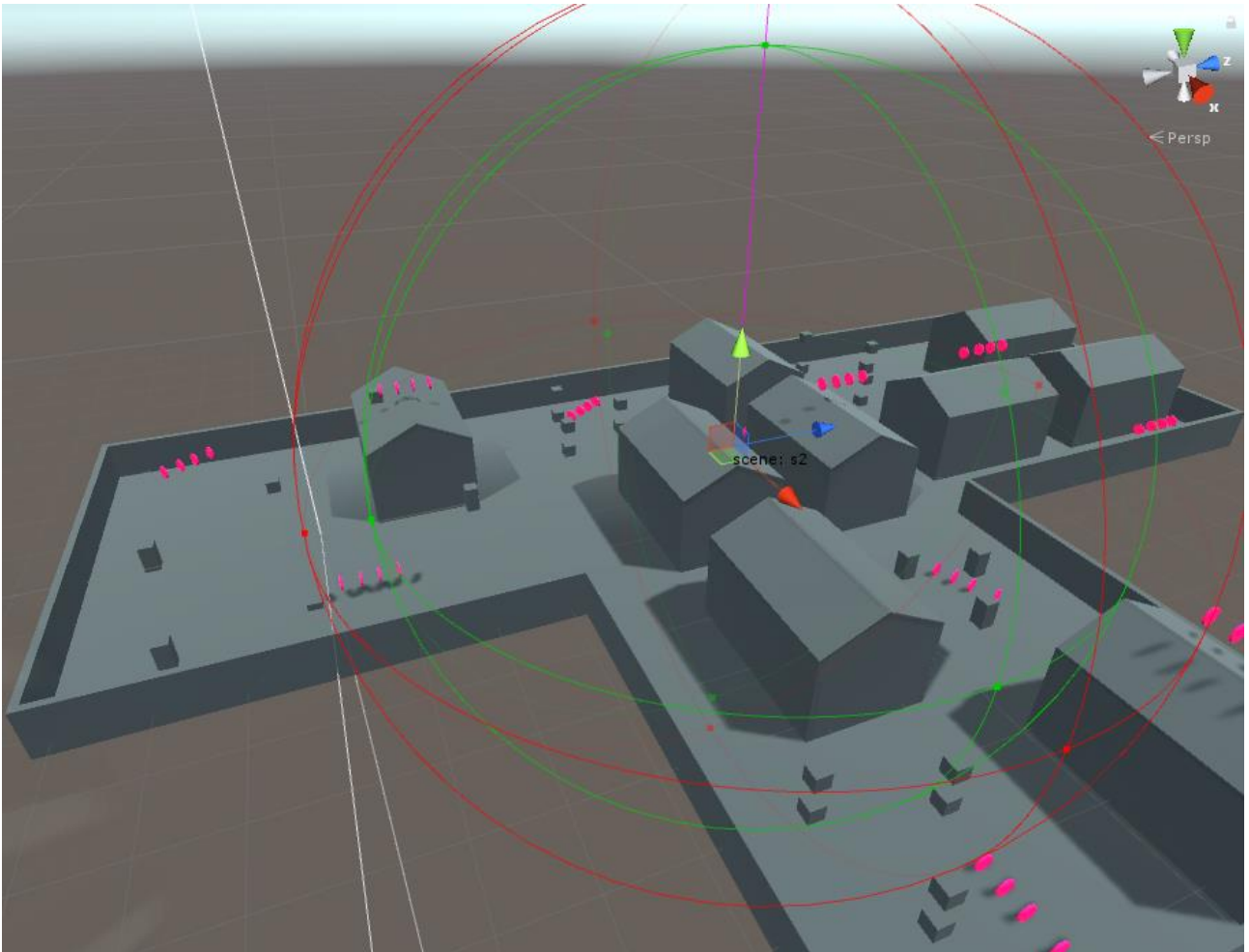
To use ChunkManager, you must drag the ChunkManager prefab (in Assets/AssetBundleMagic/Prefabs) on the Hierarchy view. As in the following picture.



The inspector has following fields:

- Subject – this is the player that ChunkManager will follow to check if load/unload scenes
- Interval – this is the time between one check and another
- DistanceBias – With this float value, you can adjust areas of coverage of chunks
- Download versions at startup – if checked, ChunkManager as first step, update the Versions.txt file of the AssetBundleMagic instance
- Chunks – a list of user defined chunks, for each of them
  - X, Y, Z – this is the absolute position of the chunk in 3D space
  - Scene name – this is the scene that is loaded/unloaded if the player enters or exits the enter/exit zones
  - Load distance – if the distance from player to the chunk is less or equal this field, ChunkManager try to load the scene
  - UnloadDistance – if the distance from player to the chunks is greater this field, ChunkManager unload the reference scene
  - On Load Events – a list of events that are fired when this chunk is loaded
  - On Unload Events – a list of events that are fired when this chunk is unloaded
  - Bundle list for the scene – A list of bundles that have to be loaded before the reference scene is loaded, each bundle specification includes
    - A combo box in which you can select the bundle
    - Remote flag – if this flag is set, ChunkManager will download this particular bundle from the network via AssetBundleMagic.DownloadBundle
    - Check Ver – if this flag is set, ChunkManager will download this particular bundle from the network via AssetBundleMagic.DownloadUpdatedBundle, so it first checks if the bundle to be loaded is update

There is a graphical representation of chunks in scene view:



The green sphere represents the enter zone, and can be dragged to expand or contract the zone itself, the red sphere represents the exit zone, and can be dragged too.

With position handlers, you can move your chunks to adjust load/unload zones. If no chunk is selected in the inspector, the scene view shows all spheres from all chunks. There is an interesting demo scene that demonstrates the classic use of ChunkManager: ChunksTest.unity.

## DEMO SCENES

To help the programmer to take full advantages from the package, we made two demo scenes in it. The first (ImageTest.unity) demonstrates how to manage asset bundle variants (in this case images.hd and images.sd). The second one, demonstrates the power of ChunkManager using AssetBundleMagic.

For any further question, please refer to the address [marco@jacovone.com](mailto:marco@jacovone.com)

Good luck!