Installation

There are currently three ways of obtaining Cubiquity for Unity3D. You can install it from the asset store if you are purchasing the commercial edition of Cubiquity, you can download a free-standing '.unitypackage' file if you are using non-commercial version of Cubiquity, or you can get the latest development version from our git repository.

Installing From The Asset Store

If you purchase the commercial version of Cubiquity for Unity3D then you can download and install it directly though the asset store, as with any other asset store purchase. You also have the option to download the .unitypackage file for later installation, in which case you should follow the instructions below.

Installing a Package File

If you choose to download the non-commercial version of Cubiquity for Unity3D from then you need to import the package file. You can do this by going to the main menu in the Unity3D editor and selecting `Assets -> Import Package -> Custom Package...` and then locating your downloaded package file. The unity package importer will then present you with a list of files which you can choose to import - for now the best option is to make sure everything is selected and then press `Import`.

The process we have described here is standard for all Unity3D packages, so if you have any difficulties you should consult the Unity3D documentation.

Installing from Git

Advanced users may be interested in using the latest development version of Cubiquity for Unity3D, rather than the stable releases. By doing this you may get access to functionality before it is officially released, but you can should expect that this is not yet fully tested. If you are interested in this then you can find our Git repository at **http://bitbucket.org/volumesoffun/cubiquity-for-unity3d** where further instructions are also provided.

Enabling 'unsafe' code

Cubiquity for Unity3D is built on top of a native code (C++) library which handles most of the heavy-lifting behind the scenes. In particular, it is responsible for generating the mesh representations of the voxels which are used for rendering and collision. These meshes need to be passed from the unmanaged native-code world of Cubiquity into the managed-code world of C# and Unity.

The C# language provides (at least) two ways to perform this communication. The first is is via *Marshalling*, in which the .NET runtime takes care of copying the arrays of indices and vertices into its own managed memory. From here we can perform some required decompression and then pass the data on to Unity. The second approach is to make use of the 'unsafe' keyword to enable pointers and direct memory access in our C# code, so that we can directly read from the memory owned by the

native code library.

The second approach is faster, cleaner, and is what we recommend. The Cubiquity library is already native-code and so managed-only targets such as the webplayer are already unsupported. However, there is a catch. Using unsafe code requires a special '-unsafe' flag to be passed to the C# compiler, and in Unity this requires editing some project files. We don't want to tamper with users projects and so by default this feature is switched off unless you enable it.

Passing custom compile flags is done by placing them in '.rsp' files in the Assets folder. The exact name of the .rsp file depends on the compiler being used as described at the bottom of the **Platform Dependent Compilation** section of the Unity manual. In our case we are only concerned with the C# compilers used by the editor and standalone player. If you *do not* already have .rsp files in your Assets folder then the easiest approach is to copy the sample files which we provide in the Assets/Cubiquity/Unsafe folder (only copy the .rsp files - corresponding .meta files will be generated automatically). If you *do* already have conflicting .rsp files then you will need to manually add the required compiler switches to them. These switches are:

```
-unsafe -define:CUBIQUITY_USE_UNSAFE
```

The first of these enables unsafe code for the compiler, and the second tells Cubiquity for Unity3D that this mode is available so that it can use the correct code path.

Note

You will need to trigger a rebuild for these changes to take effect. The easiest way to do this is to modify (or touch) one of your .cs files.