**// 1.0. LOADING ASSETS DURING RUNTIME**

To summarise, the application being requested is a place for users to establish personal portfolios that they can pick up and pass around to others in augmented reality (AR). Given this description, the application itself will require a method of importing assets in a way that does not involve going through the developers or product owner directly, nor through the Unity integrated development environment (IDE). As such, it will be necessary to load these assets during runtime in some manner; understanding how to go about that will be the purpose of this piece.

**// 1.1. RESOURCE FOLDER**

The Unity manual (version 2019.3) specifies that assets found in the project’s *Resource Folders* do not get loaded until directly requested, so immediately it can be deduced that it is possible to create such a thing with Unity. It’s possible to force load resources through:

public static T Load(string path);

// part of the Resources class   
// found at https://docs.unity3d.com/ScriptReference/Resources.Load.html

Where T is the type of object being loaded. If nothing can be found at the specified path, it will return null. It should be mentioned that anything loaded through this method must exist in the existing Resources folder.

**// 1.2. ASSET BUNDLES**

*Asset Bundles* will be an important thing to explore, as it stores assets externally and each project can have multiple. Asset bundles are created via the function:

public static AssetBundleManifest BuildAssetBundles(string outputPath, BuildAssetBundleOptions assetBundleOptions, BuildTarget targetPlatform);

// above is a part of the BuildPipeline class  
// found at https://docs.unity3d.com/ScriptReference/BuildPipeline.BuildAssetBundles.html

Objects to be added will need to be specified in an array that is passed through as one of the arguments above. Once established, these can be loaded during runtime with:

public Object LoadAsset(string name);   
public Object LoadAsset(string name, Type type);  
public T LoadAsset(string name);

// above are part of the AssetBundle class  
// found at https://docs.unity3d.com/ScriptReference/AssetBundle.LoadAsset.html

If the Asset Bundle being loaded has a specific type, it can be loaded with the second or third function, where T refers to the Type. Likewise, and to save on memory, you can unload Asset Bundles by using:

public void Unload(bool unloadAllLoadedObjects);

// part of the AssetBundle class  
 // found at https://docs.unity3d.com/ScriptReference/AssetBundle.Unload.html

When unloadAllLoadedObjects is true, all objects that were loaded from the bundle in question will be destroyed at the same time, though when set to false any objects loaded already will remain in the form they are in. By doing this, you can effectively load a bundle, instantiate a single object from it and then clean up memory use.

**Personal Thoughts:** May be able to establish a database of personal Asset Bundles hosted on a separate server. Would be necessary to give the ability to create Bundles through the application or web server and then push them onto the server. If we could get Bundles downloaded directly to the project’s Resource Folders, it could be as simple as loading that Bundle during runtime.

**// 1.3. SIMILAR THIRD-PARTY TOOLS**

It should be no surprise that others have worked on similar projects that involved asset loading during runtime. One example of this comes from Marc Kusters who has provided a free script online that imports Blender .obj files.

This can be found at: http://wiki.unity3d.com/index.php/FastObjImporter

Another is the Simple .OBJ tool found on the Unity Asset Store, created by Orbcreation, again only importing Blender .obj files.

This can be found at: https://assetstore.unity.com/packages/tools/input-management/simple-obj-22195

**// 1.4. RESOURCES**

<https://docs.unity3d.com/ScriptReference/Resources.Load.html>

<https://docs.unity3d.com/ScriptReference/BuildPipeline.BuildAssetBundles.html>

<https://docs.unity3d.com/ScriptReference/AssetBundle.Unload.html>

<https://docs.unity3d.com/Manual/LoadingResourcesatRuntime.html>

<https://docs.unity3d.com/ScriptReference/AssetBundle.html>

**// 2.0. SHARING GALLERY WITH OTHERS**

If we are going to pass around portfolios, it will first need to be stored onto an external location. Each asset bundle would be created locally on the user’s device, so an option to import directly from the hard-drive is possible given a bundle is added to a specific location. However, a much more productive method of trading and sharing portfolios would be to upload and download to a server host.

**// 2.1. ADDING & RETRIEVING ASSET BUNDLES FROM A SERVER**

To do this, we require the ability to add and retrieve each bundle from whichever server we use. In order to download a bundle, we can use:

UnityWebRequest.GetAssetBundle(string link);

This will establish the target URL. You can additionally provide a checksum parameter in the above’s arguments in an attempt to validate the integrity of the downloaded data. The bundle can then be extracted once downloaded with:

DownloadHandlerAssetBundle.GetContent(UnityWebRequest W);

To upload asset bundles however seems significantly more complicated, potentially even impossible to do efficiently. It’s likely a better idea to have creator’s establish their portfolio’s online, say on our website, then forward that data to a database that they can then access through their email and password through the application itself.

**// 2.2. APPEARING IN A USEABLE FORM**

Uploading assets and downloading them for the application to use will likely not store the assets in a useable form, being an actual Asset Bundle, and would probably store assets individually in a data structure of sorts. Having a check the first time the creator views the portfolio that compiles it as a bundle could be a viable solution. Refer to   
*1.2. ASSET BUNDLES*.

The database would then refer to this compiled bundle from then on, thus if someone was to download it elsewhere it would immediately work. Small edits to things such as rotation, scale and position should be possible once the bundle is compiled and stored in that database.

**// 2.3. KEY ACCESS**

In order to pass portfolios around, we may be able to use a randomly generated key as a substitute for file transfers and passing around account details. When the key is generated, it would establish a link between itself and the location of the asset bundle in our database, then a simple check for the key would give the ability to retrieve the download. Furthermore, we may be able to give read-only or write permissions during key creation.

**// 2.4. IN-APP “STORE”**

Alternatively, a store could be set up where creators could provide downloads to their portfolios. The only downside would be a potential need for moderation. This could possibly be done simply by cycling through a portion of the available portfolios that opt in for download in a dedicated menu.

**// 2.5. RESOURCES**

<https://docs.unity3d.com/Manual/UnityWebRequest-DownloadingAssetBundle.html>