Instructions for DebugXR

Ole Tobiesen

July 2020

1 About Me

My name is Ole Tobiesen, and I have been a professional AR/VR developer since 2016. My current workplace is Bouvet. You can find my LinkedIn profile **here**.

2 What is DebugXR?

DebugXR intercepts logs from the app you are developing. These logs can be stuff you write in Debug.Log() calls, exceptions, warnings or anything else you normally would have seen in the Unity console.

This is to make it easier to debug your app while you are testing it in a *HoloLens*

or an *Oculus Quest*. The script will work just as well on Android as it will on UWP. Furthermore, it will output the log messages regardless of which scripting back-end or .NET verison you use.

3 Installing DebugXR

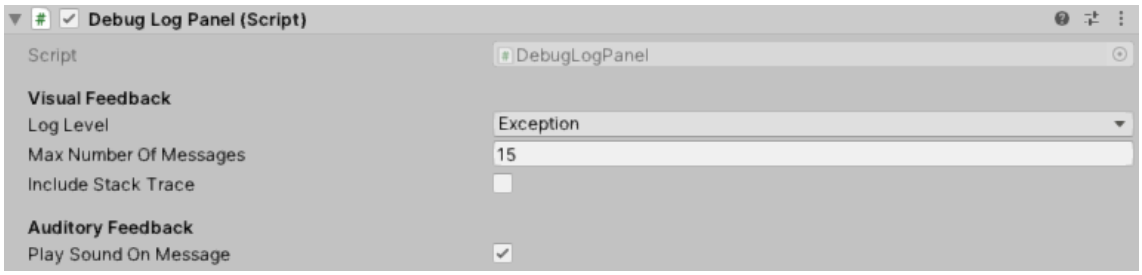
DebugXR is installed just like any other Unity Package. You can see the pre-requirements below.

3.1 Pre-requirements

- TextMesh Pro (you will be prompted about importing the recommended TextMesh Pro Settings once you import the package)
- Unity 2018.3.x or newer. It may work with older versions as well, but it hasn't been tested on anything older

4 How to use it

To get started, simply drag one of the prefabs into the scene. If you wish to configure it manually, you can create an empty gameobject and add the "Debug Log Panel" behaviour script to it.



If you add an AudioSource to this gameobject, you can also play a sound whenever a message is printed to the screen. If you wish to include the stack trace, the stack trace will be appended to the relevant message.

When you reach the maximum number of messages (the suggested number is 15), the oldest message will be deleted when a new message arrives.

4.1 The different log levels explained

The three prefabs in this package covers three log types. These are **Warning**, **Log** and **Exception**. You may, however, set the log type to **Error** and **Assert** as well. The levels are explained below:

- **Log** simply refers to anything that comes from a Debug.Log() call
- **Exception** will display exceptions, for example NullReferenceExceptions. Unlike errors, exceptions can be handled at runtime
- **Warning** will display warnings that normally will not impair the app directly, for example if a method you are using is deprecated
- **Error** are similar to exceptions, but cannot be handled at runtime. An error will usually signal the end of the app session (and Unity), and thus, logging this isn't very useful
- **Assert** refers to a condition that needs to be met for an app to proceed. If the condition fails, a message and a corresponding stacktrace will be outputted

5 License