(Note: More updated docs and samples available at onejs.com)

What is OneJS?

OneJS is a Scripting Engine specifically designed for Unity. It is lightweight, performant, has first-class Typescript support, and works everywhere. It allows you to directly use things like Preact and Tailwind with UI Toolkit.

Feature Highlights

- Use **Typescript & JSX** for UI Development in Unity
- **Fast Iteration speed** with Live Reload (even across multiple devices)
- **Awesome Performance** due to 1-to-1 interop between Preact and UI Toolkit
- **Tailwind!** Utility Classes, Arbitrary Values, and Responsive Breakpoints are all supported.
- **TS Definitions** for tons of UnityEngine and UIElements types. We also provide an C#-to-TS Type Converter that will make your Typing life much easier.
- **Works Everywhere** (Mac, Windows, iOS, Android, Editor, Standalone, Mono, Il2cpp)
- **Built-in Security** when you need it. Should you choose to give your players scripting capabilities, you can set many security settings such as memory limit, call depth, script timeout, among many others, courtesy of Jint.

Requirements

- Unity.Mathematics
- Unity Version 2021.3+ (for stable UI Toolkit)
- Unity Version 2022.1+ (if you need to use UI Toolkit's Vector API)

Quick Start

After downloading and importing OneJS from the Asset Store. You can just

- Drag and drop the `ScriptEngine` prefab onto a new scene.
- Enter Play mode.

In the console, if you see `[index.js]: OneJS is good to go.`, then OneJS is all set. Refer to the included sample scene to see how Preact and UI Toolkit work together. The script(s) responsible for the sample scene are under `{ProjectDir}/OneJS/Addons/Sample`.

More Info

OneJS uses `{ProjectDir}/OneJS` as its working directory (NOTE: `{ProjectDir}` is not your `Assets` folder; it is one level above the `Assets` folder). So, you can safely check the `OneJS` folder into Version Control. When building for standalone, the scripts from `{ProjectDir}/OneJS` will be automatically bundled up and be extracted to `{persistentDataPath}/OneJS` at runtime. Refer to the Deployment page for more details on that.

The first time `ScriptEngine` runs, it will set up a few things automatically under `{ProjectDir}/OneJS`.

These are:

- A default `tsconfig.json`
- A default `.vscode/settings.json`
- A default `index.js` script (that just logs something to the console)
- `ScriptLib` folder containing all the Javascript library files (and TS definitions) that are used by OneJS.
- `Addons` folder containing some sample code you can look at.

VSCode

`{ProjectDir}/OneJS` is the folder to open when using VSCode. Typescript (.ts and .tsx) is the recommended language to use with OneJS. To have VSCode continuously transpile TS to JS in watch mode, use `Ctrl + Shift + B` or `Cmd + Shift + B` and choose `tsc: watch - tsconfig.json`.

You can, of course, just use plain .js and .jsx files as well. But do note that by default OneJS only support CommonJS modules (i.e. `require()` and `module.exports`). So if you want to use ES modules (i.e. `import` / `export` statements), Typescript is the way to go.

The default `.vscode/settings.json` will enable Explorer File Nesting for you, as well as some PowerShell settings for better usage on Windows.

ScriptEngine

The `ScriptEngine` component is the core of OneJS. It manages interop between C# and JS (via Jint) and provides the DOM implementations needed by Preact. Below are the settings you can set on the component, divided into 3 categories: Interop, Assets, and Security.

Interop

Settings under Interop are generally about what features from C#/.Net you want to expose to Javascript.

- **Assemblies**: List of Assembly names you want to access from Javascript. (i.e. `"UnityEngine.CoreModule"` and `"Unity.Mathematics"`)
- **Extensions**: List of Extension names you want to access from Javascript. (i.e. `"UnityEngine.UIElements.PointerCaptureHelper"`)
- **Namespaces**: You can map C# namespaces to JS module here. (i.e. `"UnityEngine.UIElements" =>; "UnityEngine/UIElements"`)
- **Static Classes**: Map C# static classes to JS module. (i.e. `"Unity.Mathematics.math" => "math"`)
- **Objects**: Map a MonoBehaviour component to JS module. Note you can drag a Component to the slot, but you can probably only do so by locking the `ScriptEngine` Inspector and opening a new Inspector Tab for the target MonoBahaviour. (i.e. `"MaterialManager" => "matman"`)

Assets

The settings under Assets mostly contain templates or default files for OneJS to use, such as `tsconfig.json` and `settings.json`. So You don't really need to touch any of that. The only thing you may want to add are the Style Sheets (USS).

Security

`ScriptEngine` provides the following security settings for you to set in the Inspector.

- **Allow Reflection**
- **Allow** `GetType()`
- **Memory Limit**
- **Timeout**
- **Recursion Depth**

These are some of the security settings exposed directly from Jint. To set more granular security measures such as Member Accessor & TypeResolver, you can do so during the `OnPostInit` event (refer to the event API below).

ScriptEngine APIs

`ScriptEngine` exposes some public APIs for you to use from code.

Properties

```
public Engine JintEngine; // Internal Jint Engine

public Dom DocumentBody; // Dom for document.body
```

Events

```
public event Action OnPostInit; // Happens after every ScriptEngine reload

public event Action OnReload; // Happens when ScriptEngine is just about to reload
```

Methods

```
public void RunScript(string scriptPath); // Run a script as is

public void ReloadAndRunScript(string scriptPath); // Reloads the ScriptEngine and then
```

Live Reload

The `LiveReload` MonoBehaviour component watches your `{ProjectDir}/OneJS` directory and will reload the `ScriptEngine` (and your entry script) when code changes are detected. Settings:

- **Run On Start** (default On):
- **Entry Script** (default "index.js"): Which file to run on `ScriptEngine` reload. Note this should always be a .js file.
- **Watch Filter** (default "*.js"): What type of files to watch

Multi-Device Live Reload

There is a 4th setting (Net Sync) that you can set to enable Live Reload across different devices. For example, you can make code changes in VSCode on your Desktop and have the change live reloaded on your deployed mobile app. Devices will self discover as long as they are on the same network (i.e. same wifi or lan).

After toggling on Net Sync, you can pick 3 modes:

- **Auto** defaults to Server for Desktop, and Client for Mobile
- **Server** will be responsible for broadcasting code changes to clients
- **Client** will be listening for code changes from Server

Janitor

A `JanitorSpawner` should be used along side `LiveReload`. It'll spawn a Janitor GameObject that will help cleaning up previous GameObjects and Console logs upon every Reload.

C# to TS Definition

Out of the box, OneJS provides tons of TS definitions for the Unity ecosystem. `UnityEngine`, `UIElements`, `Mathematics`, to name a few. But there will be times that you'll want to make your own TS definitions for things we haven't covered.

So to make things easier for you, we include an auto converter that can extract TS definitions out of any C# type. You can access it from Unity menu (OneJS -> C# to TSDef Converter). The name you use should be the fully qualified type name. Remember you can use syntax like `Foo`1` for generics and `Foo+Bar` for nested types.

Please note that no such converter is perfect. Ours works better than all the other ones we've tried. It'll get 90% of the work done for you, but you'll still need to make adjustments here and there.

Tailwind

Just as how UI Toolkit supports a subset of CSS, OneJS implements a subset of Tailwind's utility classes. To enable Tailwind, just drag the `Tailwind` component onto a ScriptEngine GameObject and setup the stylesheets. Please refer to the included `ScriptEngine` prefab for default setup of the `Tailwind` component.

Supported Categories:

- **Backgrounds**: Background Color, Unity Specific (`bg-crop`, `bg-fit`, `bg-fill`)
- **Borders**: Border Radius, Border Width, Border Color
- **Effects**: Opacity
- **Flexbox**: Basis, Direction, Wrap, Grow, Shrink, Justify Content, Align Content, Align Items, Align Self
- **Layout**: Container, Display, Overflow, Position, Top, Right, Bottom, Left
- **Sizing**: Width, Min-Width, Max-Width, Height, Min-Height, Max-Height
- **Spacing**: Padding, Margin

- **Transforms**: Translate, Rotate, Scale
- **Transitions**: (arbitrary values only) `transition`, `ease`, `delay`, `duration`
- **Typography**: Font Size, Italic & Bold, Text Alignment, Text Color, Whitespace

Negative values are supported for Margin, Top, Right, Bottom, Left (i.e. `-mt-8 -left-4 -top-4`). Arbitrary values and responsive breakpoints also work as you'd expect.

Notable Caveats

- Currently you cannot use Pseudo-classes and arbitrary values together. In other words, `hover:text-[#FFFFFF]` is not supported yet. `hover:text-rose-200` and `text-[#ABCDEF]` work fine individually.

VSCode Extension support

You can use the official Tailwind VSCode extension. Just remember to `npm install -D tailwindcss` and include a default `tailwind.config.js` first at your working directory (`persistentDataPath`):

```
// tailwind.config.js
module.exports = {
  content: [],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Note the resulting `node_module` directory and `tailwind.config.js` are not actually needed/used by OneJS. They are just required for the Tailwind VSCode extension to activate.

Deployment

When building standalone apps, the scripts from `{ProjectDir}/OneJS` will be automatically bundled up and be extracted to `{persistentDataPath}/OneJS` at runtime.

This process is fairly automatic so you don't need to worry about it too much. But here's a bit more info. The `OneJSBuildProcessor` ScriptableObject hooks into your build process and zips up all the scripts under `{ProjectDir}/OneJS`. The Bundler component (on ScriptEngine) takes care of extracting the zip during runtime.

Live Reload, on or off

For production deployment where you don't need Live Reload, remember to turn it off by unchecking the "Turn On For Standalone" option on the Live Reload component. By default, this option is on because during development you may still find Live Reload useful for Standalone builds/apps.

link.xml for AOT Platforms and IL2CPP

AOT Platforms and IL2CPP builds will strip all your unused C# code. So for all the classes you'd like to call dynamically from Javascript, you'd need to preserve them. link.xml will do the job. Here's an example:

```
<linker>
    <assembly fullname="mscorlib" preserve="all" />
    <assembly fullname="OneJS" preserve="all" />
    <assembly fullname="UnityEngine.CoreModule" preserve="all" />
    <assembly fullname="UnityEngine.PhysicsModule" preserve="all" />
    <assembly fullname="UnityEngine.TextRenderingModule" preserve="all" />
    <assembly fullname="UnityEngine.UIElementsModule" preserve="all" />
    <assembly fullname="UnityEngine.IMGUIModule" preserve="all" />
    <assembly fullname="Unity.Mathematics" preserve="all" />
</linker>
```