Shader write for FUDGE applications

This tutorial allows you to use the FUDGE shader writer to convert your shaders written in GLSL files into TypeScript classes that can be used for the materials in your FUDGE project.

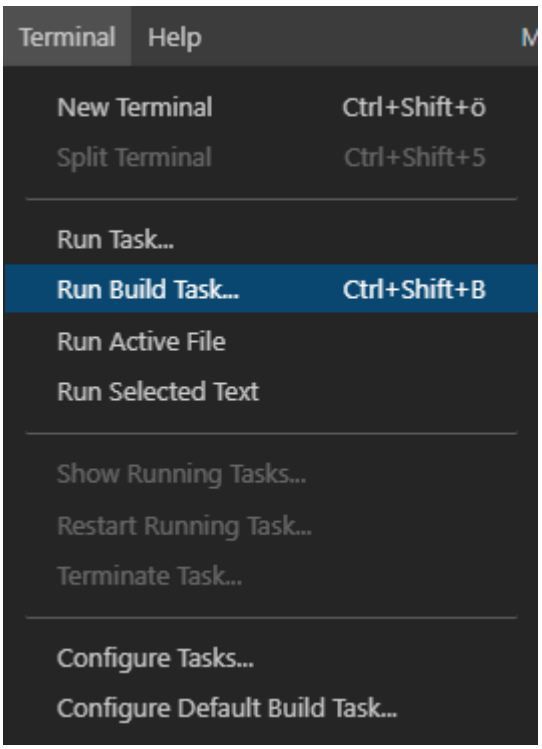
First you need the following files:

- ShaderWriter.js
- ShaderWriter.cmd
- (tasks.json)

Move them into the folder in your project where you plan to save your shader files. If your project already has a tasks.json you can edit it and add the following task:

```
{
 "type": "process",
 "label": "write shaders - ShaderWriter.cmd",
 "group": "build",
 "command": "ShaderWriter.cmd",
 "problemMatcher": []
}
```

VS Code should now allow you to execute a build task called “write shaders”. To do that you can click on “Terminal” and then “Run build task”. A list with all available build tasks will appear.



If your tasks.json is in another folder than ShaderWriter.js and ShaderWriter.cmd you will need to adjust the path in the build task under “command”.

Now that the build task is set up you can write your shaders. Every GLSL shader needs a vertex and a fragment shader in separate files. For the vertex shader use “*.vert” and for the fragment shader “*.frag”. The name of the two files for one shader needs to be identical and starts with “Shader” followed by the name (Shader[Name].vert, Shader[Name].frag).
For example ShaderCustom.vert and ShaderCustom.frag.
These two files need to be in the same folder as ShaderWriter.js and ShaderWriter.cmd.
When naming these files, avoid shader names that are already in use in the standard shader pool of FUDGE like ShaderFlat or ShaderPhong.
Now you can write your vertex and fragment shader in the .vert and .frag file.
These are the vertex and fragment shader for ShaderUniColor as an example:

```
#version 300 es
in vec3 a_position;
uniform mat4 u_projection;

void main() {
    gl_Position = u_projection * vec4(a_position, 1.0);
}
```

```
#version 300 es
precision mediump float;
uniform vec4 u_color;
out vec4 frag;

void main() {
  frag = u_color;
}
```

Now that the GLSL code is ready you can execute the build task “write shaders” and a new TypeScript file with the name of your shader will appear. In this example the files were named “ShaderCustom” and used the GLSL code shown above:

```
namespace NAMESPACE {

import ƒ = FudgeCore;

@ƒ.RenderInjectorShader.decorate
export abstract class ShaderCustom extends ƒ.Shader {

public static readonly iSubclass: number =
ƒ.Shader.registerSubclass(ShaderCustom);

public static vertexShaderSource: string = `#version 300 es
in vec3 a_position;
uniform mat4 u_projection;
```

```
void main() {
    gl_Position = u_projection * vec4(a_position, 1.0);
}`;

public static fragmentShaderSource: string = `#version 300 es
precision mediump float;
uniform vec4 u_color;
out vec4 frag;
//HALLO
void main() {
  frag = u_color;
}`;
}
}
```

To use this shader in your FUDGE project all that is left to do is to change the name of the name space (“NAMESPACE” on creation) to the one you use in this project. Now you can use this new TypeScript class (in this example “ShaderCustom”) the same way you would use a provided shader.