NewBehaviourScript.cs

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour
{
// Start is called before the first frame update
void Start()
{
}
// Update is called once per frame
void Update()
{
}
}

ScriptCollectorToWord.cs

#if UNITY\_EDITOR
using UnityEditor;
using UnityEngine;
using System;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Collections.Generic;
public class ScriptCollectorToWord : EditorWindow
{
private const string NUGET\_GIT\_URL = "https://github.com/GlitchEnzo/NuGetForUnity.git#upm";
private const string OPEN\_XML\_PACKAGE\_ID = "DocumentFormat.OpenXml";
private const string OPEN\_XML\_TYPE\_NAME =
"DocumentFormat.OpenXml.Packaging.WordprocessingDocument, DocumentFormat.OpenXml";
private const float LERP\_SPEED = 10f;
private string \_rootFolder = Application.dataPath;
private string \_outputFolder = Application.dataPath;
private string \_outputFileName = "ScriptsBundle.docx";
private bool \_showAdvanced = false;
private bool \_includeAllSubfolders = true;
private bool \_excludeEditorFolders = false;
private string[] \_subfolders = Array.Empty<string>();
private bool[] \_subfolderSelected = Array.Empty<bool>();
private Vector2 \_subfoldersScroll = Vector2.zero;
private bool \_generationSucceeded = false;
private string \_generatedPath = string.Empty;
private bool \_openXmlLoaded = false;
private float \_pendingHeight = -1f;
private static bool s\_windowCentered = false;
[MenuItem("Tools/Generate Word from Scripts", priority = 250)]
private static void OpenWindow()
{
var window = GetWindow<ScriptCollectorToWord>("Scripts → Word");
window.minSize = new Vector2(580, 300);
if (!s\_windowCentered)
{
var resolution = UnityStats.screenRes.Split('x');
if (resolution.Length == 2 && int.TryParse(resolution[0], out int sw) &&
int.TryParse(resolution[1], out int sh))
{
var rect = window.position;
rect.x = (sw - rect.width) \* 0.5f;
rect.y = (sh - rect.height) \* 0.5f;
window.position = rect;
}
s\_windowCentered = true;
}
window.CheckDependencies();
window.RefreshSubfolders();
window.Repaint();
}
private void OnEnable()
{
CheckDependencies();
RefreshSubfolders();
}
private void Update()
{
if (\_pendingHeight < 0f)
{
return;
}
var rect = position;
rect.height = Mathf.Lerp(rect.height, \_pendingHeight, Time.deltaTime \* LERP\_SPEED);
if (Mathf.Abs(rect.height - \_pendingHeight) < 0.5f)
{
rect.height = \_pendingHeight;
\_pendingHeight = -1f;
}
position = rect;
}
private void OnGUI()
{
void Touch() => \_generationSucceeded = false;
bool nugetInstalled = IsNugetInstalled();
bool openXmlReady = \_openXmlLoaded;
if (!nugetInstalled || !openXmlReady)
{
EditorGUILayout.LabelField("Dependencies", EditorStyles.boldLabel);
EditorGUILayout.Space(2);
}
if (!nugetInstalled)
{
EditorGUILayout.HelpBox(
"NuGetForUnity is required. Install it via Package Manager → Add package from Git URL.",
MessageType.Warning);
DrawReadOnlyField("Git URL:", NUGET\_GIT\_URL);
if (GUILayout.Button("Open Package Manager", GUILayout.Height(22)))
{
EditorApplication.ExecuteMenuItem("Window/Package Manager");
}
EditorGUILayout.Space(4);
}
if (!openXmlReady)
{
EditorGUILayout.HelpBox(
$"Package \"{OPEN\_XML\_PACKAGE\_ID}\" not found. Install it via NuGetForUnity and press Refresh.",
MessageType.Info);
DrawReadOnlyField("Package:", OPEN\_XML\_PACKAGE\_ID);
EditorGUILayout.BeginHorizontal();
EditorGUI.BeginDisabledGroup(!nugetInstalled);
if (GUILayout.Button("Open NuGetForUnity", GUILayout.Width(160)))
{
EditorApplication.ExecuteMenuItem("NuGet/Manage NuGet Packages");
}
EditorGUI.EndDisabledGroup();
if (GUILayout.Button("Refresh", GUILayout.Width(80)))
{
AssetDatabase.Refresh();
CheckDependencies();
Touch();
}
EditorGUILayout.EndHorizontal();
EditorGUILayout.Space(6);
}
if (!nugetInstalled || !openXmlReady)
{
return;
}
EditorGUILayout.Space(6);
DrawRootFolderSelector(Touch);
bool previousAdvanced = \_showAdvanced;
\_showAdvanced = EditorGUILayout.ToggleLeft("Advanced options", \_showAdvanced, EditorStyles.boldLabel);
if (\_showAdvanced != previousAdvanced)
{
Touch();
}
if (\_showAdvanced)
{
EditorGUILayout.Space(4);
DrawAdvancedPanel(Touch);
}
EditorGUILayout.Space(6);
DrawOutputSelector(Touch);
EditorGUILayout.Space(10);
if (\_generationSucceeded)
{
var style = new GUIStyle(EditorStyles.label)
{ normal = { textColor = Color.green }, fontStyle = FontStyle.Bold };
EditorGUILayout.BeginHorizontal(EditorStyles.helpBox);
GUILayout.Label("✔ Word document saved", style);
GUILayout.FlexibleSpace();
if (GUILayout.Button("Open Folder", GUILayout.Width(100)))
{
EditorUtility.RevealInFinder(\_generatedPath);
}
EditorGUILayout.EndHorizontal();
}
GUI.enabled = !string.IsNullOrWhiteSpace(\_rootFolder) && !string.IsNullOrWhiteSpace(\_outputFolder);
if (GUILayout.Button("Generate Word file", GUILayout.Height(32)))
{
GenerateDocx();
}
GUI.enabled = true;
if (Event.current.type == EventType.Repaint)
{
float required = GUILayoutUtility.GetLastRect().yMax + 10f;
required = Mathf.Max(required, minSize.y);
if (Mathf.Abs(position.height - required) > 0.5f)
{
\_pendingHeight = required;
}
}
}
private static void DrawReadOnlyField(string label, string value)
{
EditorGUILayout.BeginHorizontal();
EditorGUILayout.LabelField(label, GUILayout.Width(60));
EditorGUILayout.SelectableLabel(value, EditorStyles.textField,
GUILayout.Height(EditorGUIUtility.singleLineHeight));
if (GUILayout.Button("Copy", GUILayout.Width(50)))
{
EditorGUIUtility.systemCopyBuffer = value;
}
EditorGUILayout.EndHorizontal();
}
private void CheckDependencies() => \_openXmlLoaded = Type.GetType(OPEN\_XML\_TYPE\_NAME) != null;
private static bool IsNugetInstalled() => AppDomain.CurrentDomain.GetAssemblies().Any(a =>
a.GetName().Name.IndexOf("NuGetForUnity", StringComparison.OrdinalIgnoreCase) >= 0);
private void DrawRootFolderSelector(Action touch)
{
EditorGUILayout.LabelField("Root folder with scripts", EditorStyles.boldLabel);
EditorGUILayout.BeginHorizontal();
string newPath = EditorGUILayout.TextField(\_rootFolder);
if (newPath != \_rootFolder)
{
\_rootFolder = newPath;
RefreshSubfolders();
touch();
}
if (GUILayout.Button("…", GUILayout.Width(28)))
{
string selected = EditorUtility.OpenFolderPanel("Select root folder", \_rootFolder, "");
if (!string.IsNullOrEmpty(selected))
{
\_rootFolder = selected;
RefreshSubfolders();
touch();
}
}
EditorGUILayout.EndHorizontal();
}
private void DrawAdvancedPanel(Action touch)
{
\_includeAllSubfolders = EditorGUILayout.ToggleLeft("Include all sub-folders", \_includeAllSubfolders);
\_excludeEditorFolders = EditorGUILayout.ToggleLeft("Exclude folders named ‘Editor’", \_excludeEditorFolders);
if (!\_includeAllSubfolders)
{
EditorGUILayout.LabelField("Select sub-folders to scan", EditorStyles.boldLabel);
EditorGUILayout.BeginVertical(EditorStyles.helpBox);
\_subfoldersScroll = EditorGUILayout.BeginScrollView(\_subfoldersScroll, GUILayout.Height(120));
foreach (int i in Enumerable.Range(0, \_subfolders.Length))
{
bool selected = EditorGUILayout.ToggleLeft(\_subfolders[i], \_subfolderSelected[i]);
if (selected != \_subfolderSelected[i])
{
\_subfolderSelected[i] = selected;
touch();
}
}
EditorGUILayout.EndScrollView();
EditorGUILayout.EndVertical();
}
}
private void DrawOutputSelector(Action touch)
{
EditorGUILayout.LabelField("Output folder", EditorStyles.boldLabel);
EditorGUILayout.BeginHorizontal();
string newOutput = EditorGUILayout.TextField(\_outputFolder);
if (newOutput != \_outputFolder)
{
\_outputFolder = newOutput;
touch();
}
if (GUILayout.Button("…", GUILayout.Width(28)))
{
string selected = EditorUtility.OpenFolderPanel("Select output folder", \_outputFolder, "");
if (!string.IsNullOrEmpty(selected))
{
\_outputFolder = selected;
touch();
}
}
EditorGUILayout.EndHorizontal();
EditorGUILayout.BeginHorizontal();
GUILayout.Label("File name", GUILayout.Width(70));
string newFile = EditorGUILayout.TextField(\_outputFileName);
if (newFile != \_outputFileName)
{
\_outputFileName = newFile;
touch();
}
EditorGUILayout.EndHorizontal();
}
private void RefreshSubfolders()
{
if (!Directory.Exists(\_rootFolder))
{
\_subfolders = Array.Empty<string>();
\_subfolderSelected = Array.Empty<bool>();
return;
}
\_subfolders = Directory.GetDirectories(\_rootFolder, "\*", SearchOption.TopDirectoryOnly).Select(Path.GetFileName)
.ToArray();
\_subfolderSelected = \_subfolders.Select(\_ => true).ToArray();
}
private void GenerateDocx()
{
\_generationSucceeded = false;
var allFiles = Directory.GetFiles(\_rootFolder, "\*.cs", SearchOption.AllDirectories);
var filteredFiles = \_includeAllSubfolders ? allFiles : allFiles.Where(IsAllowed).ToArray();
if (\_excludeEditorFolders)
{
filteredFiles = filteredFiles.Where(f => !f.Replace("\\", "/").Contains("/Editor/")).ToArray();
}
if (filteredFiles.Length == 0)
{
EditorUtility.DisplayDialog("No scripts", "No C# scripts matched your selection.", "OK");
return;
}
var wordprocessingType = Type.GetType(OPEN\_XML\_TYPE\_NAME);
if (wordprocessingType == null)
{
Debug.LogError("OpenXML SDK not available");
return;
}
var assembly = wordprocessingType.Assembly;
var bodyType = assembly.GetType("DocumentFormat.OpenXml.Wordprocessing.Body");
var paragraphType = assembly.GetType("DocumentFormat.OpenXml.Wordprocessing.Paragraph");
var runType = assembly.GetType("DocumentFormat.OpenXml.Wordprocessing.Run");
var textType = assembly.GetType("DocumentFormat.OpenXml.Wordprocessing.Text");
var documentType = assembly.GetType("DocumentFormat.OpenXml.Wordprocessing.Document");
var mainPartType = assembly.GetType("DocumentFormat.OpenXml.Packaging.MainDocumentPart");
var enumType = assembly.GetType("DocumentFormat.OpenXml.WordprocessingDocumentType");
var openXmlElementType = assembly.GetType("DocumentFormat.OpenXml.OpenXmlElement");
var compositeType = assembly.GetType("DocumentFormat.OpenXml.OpenXmlCompositeElement");
var appendChildMethod = compositeType.GetMethods()
.First(m => m.Name == "AppendChild" && m.IsGenericMethodDefinition && m.GetParameters().Length == 1)
.MakeGenericMethod(openXmlElementType);
string docxPath = Path.Combine(\_outputFolder, \_outputFileName);
if (File.Exists(docxPath))
{
try
{
File.Delete(docxPath);
}
catch (IOException)
{
if (!EditorUtility.DisplayDialog("File in use", "Close it and retry.", "Retry", "Cancel"))
{
return;
}
File.Delete(docxPath);
}
}
object docx = wordprocessingType.GetMethod("Create", new[] { typeof(string), enumType, typeof(bool) })
.Invoke(null, new object[] { docxPath, Enum.Parse(enumType, "Document"), false });
object mainPart = wordprocessingType.GetMethod("AddMainDocumentPart").Invoke(docx, null);
var docProperty = mainPartType.GetProperty("Document");
object document = Activator.CreateInstance(documentType);
docProperty.SetValue(mainPart, document);
object body = Activator.CreateInstance(bodyType);
appendChildMethod.Invoke(document, new[] { body });
foreach (var file in filteredFiles)
{
AddParagraph(body, paragraphType, runType, textType, appendChildMethod, Path.GetFileName(file));
AddParagraph(body, paragraphType, runType, textType, appendChildMethod, File.ReadAllText(file));
}
wordprocessingType.GetMethod("Save").Invoke(docx, null);
wordprocessingType.GetMethod("Dispose").Invoke(docx, null);
AssetDatabase.Refresh();
\_generationSucceeded = true;
\_generatedPath = docxPath;
Debug.Log($"Collected {filteredFiles.Length} scripts → '{docxPath}'");
}
private bool IsAllowed(string path)
{
if (\_includeAllSubfolders)
{
return true;
}
var allowed = new HashSet<string>(\_subfolders.Where((s, i) => \_subfolderSelected[i]));
return allowed.Contains(Path.GetFileName(Path.GetDirectoryName(path)));
}
private static void AddParagraph(object body, Type paragraphType, Type runType, Type textType,
MethodInfo appendChildMethod, string content)
{
var paragraph = Activator.CreateInstance(paragraphType);
var run = Activator.CreateInstance(runType);
var text = Activator.CreateInstance(textType);
textType.GetProperty("Text")?.SetValue(text, content);
appendChildMethod.Invoke(run, new[] { text });
appendChildMethod.Invoke(paragraph, new[] { run });
appendChildMethod.Invoke(body, new[] { paragraph });
}
}
#endif