```csharp
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// PreviewStructureManager controls the primary logic involved with creating custom circuits within editor scenes.
/// </summary>
public class PreviewStructureManager : MonoBehaviour
{
    // Singleton state reference
    private static PreviewStructureManager instance;

    /// <summary>
    /// Denotes whether each internal circuit within the custom circuit has been reached.<br/><br/>
    /// Functionally, this list is used to run the depth-first search (DFS) algorithm to determine whether all circuits in an editor scene are connected.
    /// </summary>
    private bool[] reachedCircuits;

    /// <summary>
    /// List of inputs with no connections and all inputs respectively.
    /// </summary>
    private List<Circuit.Input> emptyInputs,
        inputs;

    /// <summary>
    /// List of outputs with no connections and all outputs respectively.
    /// </summary>
    private List<Circuit.Output> emptyOutputs,
        outputs;

    /// <summary>
    /// The number of circuits that have been reached.<br/><br/>
    /// Functionally, circuitCount is utilized alongside <seealso cref="reachedCircuits"/> to determine whether all circuits in an editor scene are connected.
    /// </summary>
    private int circuitCount;

    /// <summary>
    /// The prospective name for the current custom circuit.<br/><br/>
    /// If all validation tests succeed, it will be utilized as the name of the custom circuit.
    /// </summary>
    private string currentName;

    // Enforces a singleton state pattern
    private void Awake()
    {
        if (instance != null)
        {
            Destroy(this);
            throw new Exception("PreviewStructureManager instance already established; terminating.");
        }

        instance = this;
    }

    /// <summary>
    /// Calls the coroutine that begins the circuit creation process, namely its validation tests.
    /// </summary>
    /// <param name="name">The prospective name of the custom circuit to use.</param>
    public void VerifyPreviewStructure(string name) { StartCoroutine(VerifyPreviewStructureCoroutine(name)); }

    /// <summary>
    /// Performs a series of tests to verify the validity of a prospective custom circuit based on the current editor scene.
    /// </summary>
    /// <param name="name">The prospective name of the custom circuit to use.</param>
    private IEnumerator VerifyPreviewStructureCoroutine(string name)
    {
        // Skipping a frame ensures the UI dialog for verifying a custom circuit will show.
        yield return null;

        // Validation test #1: non-empty name
        if (name == "")
        {
            TaskbarManager.Instance.CircuitSaveError("The custom circuit must not have an empty name.");
            yield break;
        }

        // Validation test #2: unique name
        foreach (PreviewStructure previewStructure in MenuSetupManager.Instance.PreviewStructures)
        {
            if (previewStructure.Name == name)
            {
                TaskbarManager.Instance.CircuitSaveError("The custom circuit must have a unique name.");
                yield break;
            }
        }

        // Validation test #3: >= 1 circuits
        if (EditorStructureManager.Instance.Circuits.Count == 0)
        {
            TaskbarManager.Instance.CircuitSaveError("The custom circuit must consist of (1) or more circuits.");
            yield break;
        }

        // Validation test #4: no input/display gates
        foreach (Circuit circuit in EditorStructureManager.Instance.Circuits)
        {
            Type type = circuit.GetType();

            if (type == typeof(InputGate) || type == typeof(Display))
            {
                TaskbarManager.Instance.CircuitSaveError("The custom circuit must not consist of any input gates or displays.");
                yield break;
            }
        }

        // Validation test #5: all circuits are connected
        reachedCircuits = new bool[EditorStructureManager.Instance.Circuits.Count];
        emptyInputs = new List<Circuit.Input>(); inputs = new List<Circuit.Input>();
        emptyOutputs = new List<Circuit.Output>(); outputs = new List<Circuit.Output>();
        circuitCount = 0;
        CircuitConnectionTest(EditorStructureManager.Instance.Circuits[0]); // Begins the DFS algorithm

        if (circuitCount != reachedCircuits.Length)
        {
            TaskbarManager.Instance.CircuitSaveError("The custom circuit must be entirely connected.");
            yield break;
        }

        // Validation test #6: >= 1 empty outputs
        if (emptyOutputs.Count == 0)
        {
            TaskbarManager.Instance.CircuitSaveError("The custom circuit must have (1) or more empty outputs.");
            yield break;
        }

        /// All validation tests completed ///

        currentName = name;
        TaskbarManager.Instance.CloseMenu();
        TaskbarManager.Instance.NullState();

        // Begins the process in which the user assigns the order and labels of all empty inputs and outputs.
        IOAssigner.Instance.Initialize(emptyInputs, emptyOutputs);
    }

    /// <summary>
    /// Starts the coroutine involved in finally creating a custom circuit.<br/><br/>
    /// This method is specifically called by <see cref="IOAssigner"/> after all empty inputs and outputs have been ordered by the user (as well as any respective labling).
    /// </summary>
    /// <param name="orderedInputs"></param>
    /// <param name="orderedOutputs"></param>
    /// <param name="inputLabels"></param>
    /// <param name="outputLabels"></param>
    public void CreateCustomCircuit(List<Circuit.Input> orderedInputs, List<Circuit.Output> orderedOutputs, List<string> inputLabels, List<string> outputLabels)
    {
        StartCoroutine(CreatePreviewStructure(orderedInputs, orderedOutputs, inputLabels, outputLabels));
    }

    /// <summary>
    /// Serializes a custom circuit as well as its corresponding preview structure.
    /// </summary>
    /// <param name="orderedInputs">The list of empty inputs, ordered.</param>
    /// <param name="orderedOutputs">The list of empty outputs, ordered.</param>
    /// <param name="inputLabels">Labels associated with each ordered input.</param>
    /// <param name="outputLabels">Labels associated with each ordered output.</param>
    private IEnumerator CreatePreviewStructure(List<Circuit.Input>
      orderedInputs, List<Circuit.Output> orderedOutputs, List<string> inputLabels, List<string> outputLabels)
    {
        TaskbarManager.Instance.OnSuccessfulPreviewVerification();

        // Skipping a frame ensures the UI dialog for creating a custom circuit will show.
        yield return null;

        List<CircuitIdentifier> circuitIdentifiers = new List<CircuitIdentifier>();
        List<int> inputOrders = new List<int>(), outputOrders = new List<int>();
        PreviewStructure previewStructure = new PreviewStructure(currentName);

        // Serializes each circuit by instanting CircuitIdentifier references.
        foreach (Circuit circuit in EditorStructureManager.Instance.Circuits)
        {
            circuitIdentifiers.Add(new CircuitIdentifier(circuit));

            foreach (Circuit.Input input in circuit.Inputs) { inputs.Add(input); inputOrders.Add(orderedInputs.IndexOf(input)); }

            foreach (Circuit.Output output in circuit.Outputs) { outputs.Add(output); outputOrders.Add(orderedOutputs.IndexOf(output)); }
        }

        previewStructure.Circuits = circuitIdentifiers;
        previewStructure.ID = UniqueID; // Assigns a unique ID to the preview structure.
        previewStructure.InputOrders = inputOrders;
        previewStructure.OutputOrders = outputOrders;
        previewStructure.InputLabels = inputLabels;
        previewStructure.OutputLabels = outputLabels;
        previewStructure.CameraLocation = CameraMovement.Instance.PlayerCamera.transform.position;

          List<InternalConnection> internalConnections = new List<InternalConnection>();

        // Serializes each connection by assigning index values to each input/output pair within an InternalConnection instance.
        foreach (CircuitConnector.Connection connection in EditorStructureManager.Instance.Connections)
        {
```

```csharp
            internalConnections.Add(new InternalConnection(
                inputs.IndexOf(connection.Input),
                outputs.IndexOf(connection.Output)
                ));
        }

        previewStructure.Connections = internalConnections;

        // Adds preview structure and its connections to the save directory and add menu.
        MenuSetupManager.Instance.PreviewStructures.Add(previewStructure);
        MenuSetupManager.Instance.GenerateConnections(false, previewStructure.ID, EditorStructureManager.Instance.Connections);
        MenuSetupManager.Instance.UpdatePreviewStructure(previewStructure);
        TaskbarManager.Instance.AddCustomCircuitPanel(previewStructure.ID, false);
        TaskbarManager.Instance.OnSuccessfulPreviewStructure();
    }

    /// <summary>
    /// Performs a depth-first search starting at the first placed circuit to determine whether the scene represents a complete graph. <br/>
    /// At the same time, any circuit input or output without a connection is stored for the next test.
    /// </summary>
    private void CircuitConnectionTest(Circuit currentCircuit)
    {
        while (currentCircuit.customCircuit != null)
        {
            currentCircuit = currentCircuit.customCircuit;
        }

        int index = EditorStructureManager.Instance.Circuits.IndexOf(currentCircuit);

        if (reachedCircuits[index]) return;

        reachedCircuits[index] = true;
        circuitCount++;

        foreach (Circuit.Input input in currentCircuit.Inputs)
        {
            if (input.ParentOutput == null) { emptyInputs.Add(input); continue; }

            CircuitConnectionTest(input.ParentOutput.ParentCircuit);
        }

        foreach (Circuit.Output output in currentCircuit.Outputs)
        {
            if (output.ChildInputs.Count == 0) { emptyOutputs.Add(output); continue; }

            foreach (Circuit.Input input in output.ChildInputs)
            {
                CircuitConnectionTest(input.ParentCircuit);
            }
        }
    }

    /// <summary>
    /// Returns a new unique ID for a new preview structure.<br/><br/>
    /// A unique ID starts from 0 and increments onward.
    /// </summary>
    private int UniqueID
    {
        get
        {
            int currentID = 0;

            // Keeps incrementing the current ID until it is unique
            // This system ensures that if an ID that is not the largest is removed, it will be recycled in future custom circuit creations.
            while (true)
            {
                if (!MenuSetupManager.Instance.PreviewStructureIDs.Contains(currentID))
                {
                    MenuSetupManager.Instance.PreviewStructureIDs.Add(currentID);
                    return currentID;
                }

                currentID++;
            }
        }
    }

    // Getter method
    public static PreviewStructureManager Instance { get { return instance; } }
}
```