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

/// <summary>
/// Logical reprentation of a custom circuit consisting of a variable number/type of other circuits.
/// </summary>
public class CustomCircuit : Circuit
{
    /// <summary>
    /// The current custom circuit that is being rendered.<br/><br/>
    ///
    /// This value is utilized to differentiate between external and internal (part of a custom circuit) custom circuits.
    /// </summary>
    private static CustomCircuit currentCustomCircuit;

    /// <summary>
    /// Whether or not the custom circuit has been removed and therefore deferenced by its child circuits.
    /// </summary>
    private bool shouldDereference;

    /// <summary>
    /// The list of all internal circuits within the custom circuit.
    /// </summary>
    private List<Circuit> circuitList = new List<Circuit>();

    /// <summary>
    /// The parent GameObject under which all internal connections are attached.
    /// </summary>
    private GameObject connections;

    /// <summary>
    /// The list of all inputs within the custom circuit that have no connections.<br/><br/>
    /// All empty inputs are rendered by <see cref="CircuitVisualizer"/>, meaning they can be externally connected to other circuits within a scene.
    /// </summary>
    private List<Input> emptyInputs = new List<Input>();

    /// <summary>
    /// The list of all inputs within the custom circuit.
    /// </summary>
    private List<Input> inputs = new List<Input>();

    /// <summary>
    /// The list of all outputs within the custom circuit that have no connections.<br/><br/>
    /// All empty outputs are rendered by <see cref="CircuitVisualizer"/>, meaning they can be externally connected to other circuits within a scene.
    /// </summary>
    private List<Output> emptyOutputs = new List<Output>();

    /// <summary>
    /// The list of all empty outputs yet to have received an update call.<br/><br/>
    /// This list is utilized to ensure that any placed custom circuit is properly updated by allowing for update call overrides that would otherwise not occur.
    /// </summary>
    public List<Output> finalOutputs;

    /// <summary>
    /// The list of all outputs within the custom circuit.
    /// </summary>
    private List<Output> outputs = new List<Output>();

    /// <summary>
    /// The preview structure the custom circuit is referring to.
    /// </summary>
    private PreviewStructure previewStructure;

    /// <summary>
    /// Alternate signature intended for creating custom circuits that is not inside a custom circuit, i.e. external.
    /// </summary>
    /// <param name="previewStructure"></param>
    public CustomCircuit(PreviewStructure previewStructure) : this(previewStructure, Vector2.zero, true) {}

    /// <summary>
    /// Primary constructor for instantiating any custom circuit.
    /// </summary>
    /// <param name="previewStructure">The preview structure the custom circuit is referring to.</param>
    /// <param name="startingPos">The in-scene position of the circuit (not applicable if the custom circuit is not visible).</param>
    /// <param name="isFirst">Whether the custom circuit is external, in which case it will be visibly rendered.</param>
    public CustomCircuit(PreviewStructure previewStructure, Vector2 startingPos, bool isFirst) : base(previewStructure.Name, Vector2.positiveInfinity)
    {
        // If this custom circuit is external, it should be marked as the
```

```csharp
        current custom circuit to be built as well as visible.
        if (isFirst) { shouldDereference = false; currentCustomCircuit = this; Visible = true; }

        CircuitName = previewStructure.Name;
        this.previewStructure = previewStructure;
        CreateCircuit(startingPos);
    }

    private void CreateCircuit(Vector2 startingPos)
    {
        connections = new GameObject("Connections [CUSTOM CIRCUIT]");

        // Intantiates each internal circuit within the custom circuit
        foreach (CircuitIdentifier circuitIdentifier in previewStructure.Circuits)
        {
            Circuit circuit = CircuitIdentifier.RestoreCircuit(circuitIdentifier, false);

            // All non-custom circuits are designated as the child of the current custom circuit
            if (circuit.GetType() != typeof(CustomCircuit)) circuit.customCircuit = this;

            circuitList.Add(circuit);

            foreach (Input input in circuit.Inputs) inputs.Add(input);

            foreach (Output output in circuit.Outputs) outputs.Add(output);
        }

        int inputAmount = previewStructure.InputLabels.Count;

        // Restores all empty inputs as designated by the assigned preview structure.
        for (int i = 0; i < inputAmount; i++) emptyInputs.Add(inputs[previewStructure.InputOrders.IndexOf(i)]);

        int outputAmount = previewStructure.OutputLabels.Count;

        // Restores all empty outputs as designated by the assigned preview structure.
        for (int i = 0; i < outputAmount; i++) emptyOutputs.Add(outputs[previewStructure.OutputOrders.IndexOf(i)]);

        // Sets the inputs and outputs as ONLY the empty inputs and outputs.
        Inputs = emptyInputs.ToArray(); Outputs = emptyOutputs.ToArray();

        int index = 0;

        finalOutputs = new List<Output>(emptyOutputs);

        // If the custom circuit is external/visible (synonymous with one another), render it into the scene.
        if (Visible) CircuitVisualizer.Instance.VisualizeCustomCircuit(this, startingPos);

        List<UpdateCall> updateCalls = new List<UpdateCall>();

        // Within the custom circuit, reinstate every connection.
        foreach (InternalConnection internalConnection in previewStructure.Connections)
        {
            CircuitConnector.Connection connection = connections.AddComponent<CircuitConnector.Connection>();
            Input input = inputs[internalConnection.InputIndex];
            Output output = outputs[internalConnection.OutputIndex];

            // Sets all values of the current connection
            connection.Input = input;
            connection.Output = output;
            input.Connection = connection;
            input.ParentOutput = output;
            output.Connections.Add(connection);
            output.ChildInputs.Add(input);
            updateCalls.Add(new UpdateCall(output.Powered, input, output));
            index++;
        }

        // Begins to call each connection.
        CircuitCaller.InitiateUpdateCalls(updateCalls);

        // Begins the chain reaction to inevitably update the outputs.
        UpdateOutputs();

        /* Implies that the current custom circuit is a part of another custom circuit.
         * As such, it points its custom circuit to the external custom circuit (parent).
         * Furthermore, the GameObject holding its connection information becomes the child of the parent's connection GameObject.
         */
        if (!Visible)
        {
            customCircuit = currentCustomCircuit;
            connections.transform.SetParent(customCircuit.Connections.transform);
        }

        // Implies the current custom circuit IS the external custom curcuit (i.e. currentCustomCircuit == null --> parent custom circuit).
        else currentCustomCircuit = null;
    }

    /// <summary>
    /// Utilized after the instantiation of a custom circuit to update its logic to default status.<br/><br/>
    /// Since a custom circuit does not store the exact predicate that controls the output, this method aims to bring about a chain reaction from the known inputs to eventually update the outputs in variable time.<br/><br/>
    /// Furthermore, a custom circuit never has its UpdateOutputs() method accessed; as such, the return value is not necessary and thus yields null.
    /// </summary>
    protected override List<Output> UpdateOutputs()
    {
        foreach (Input input in emptyInputs) UpdateCircuit(false, input, null);

        return null;
    }

    // Getter and setter method
    public bool ShouldDereference { get { return shouldDereference; } set { shouldDereference = value; } }

    // Getter methods
    public GameObject Connections { get { return connections; } }

    public PreviewStructure PreviewStructure { get { return previewStructure; } }
}
```