

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 /// <summary>
7 /// PreviewStructureManager controls the primary logic involved with      ↗
8   creating custom circuits within editor scenes.
9 /// </summary>
10 public class PreviewStructureManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static PreviewStructureManager instance;
14
15     /// <summary>
16     /// Denotes whether each internal circuit within the custom circuit      ↗
17     has been reached.<br/><br/>
18     /// Functionally, this list is used to run the depth-first search      ↗
19     (DFS) algorithm to determine whether all circuits in an editor scene      ↗
20     are connected.
21     /// </summary>
22     private bool[] reachedCircuits;
23
24     /// <summary>
25     /// List of inputs with no connections and all inputs respectively.
26     /// </summary>
27     private List<Circuit.Input> emptyInputs,
28         inputs;
29
30     /// <summary>
31     /// List of outputs with no connections and all outputs respectively.
32     /// </summary>
33     private List<Circuit.Output> emptyOutputs,
34         outputs;
35
36     /// <summary>
37     /// The number of circuits that have been reached.<br/><br/>
38     /// Functionally, circuitCount is utilized alongside <seealso      ↗
39     cref="reachedCircuits"/> to determine whether all circuits in an      ↗
40     editor scene are connected.
41     /// </summary>
42     private int circuitCount;
43
44     /// <summary>
45     /// The prospective name for the current custom circuit.<br/><br/>
46     /// If all validation tests succeed, it will be utilized as the name      ↗
47     of the custom circuit.
48     /// </summary>
49     private string currentName;
```

```
43
44     // Enforces a singleton state pattern
45     private void Awake()
46     {
47         if (instance != null)
48         {
49             Destroy(this);
50             throw new Exception("PreviewStructureManager instance already
51                                     established; terminating.");
52         }
53         instance = this;
54     }
55
56     /// <summary>
57     /// Calls the coroutine that begins the circuit creation process,
58     /// namely its validation tests.
59     /// </summary>
60     /// <param name="name">The prospective name of the custom circuit to
61     use.</param>
62     public void VerifyPreviewStructure(string name) { StartCoroutine
63         (VerifyPreviewStructureCoroutine(name)); }
64
65     /// <summary>
66     /// Performs a series of tests to verify the validity of a prospective
67     custom circuit based on the current editor scene.
68     /// </summary>
69     /// <param name="name">The prospective name of the custom circuit to
70     use.</param>
71     private IEnumerator VerifyPreviewStructureCoroutine(string name)
72     {
73         // Skipping a frame ensures the UI dialog for verifying a custom
74         circuit will show.
75         yield return null;
76
77         // Validation test #1: non-empty name
78         if (name == "")
79         {
80             TaskbarManager.Instance.CircuitSaveError("The custom circuit
81                 must not have an empty name.");
82             yield break;
83         }
84
85         // Validation test #2: unique name
86         foreach (PreviewStructure previewStructure in
87             MenuSetupManager.Instance.PreviewStructures)
88         {
89             if (previewStructure.Name == name)
90             {
91                 TaskbarManager.Instance.CircuitSaveError("The custom circuit
92                     must not have a duplicate name.");
93                 yield break;
94             }
95         }
96     }
97 }
```

```

...ets\Scripts\Editor Scripts\PreviewStructureManager.cs 3
83         TaskbarManager.Instance.CircuitSaveError("The custom  ↗
            circuit must have a unique name.");
84         yield break;
85     }
86 }
87
88 // Validation test #3: >= 1 circuits
89 if (EditorStructureManager.Instance.Circuits.Count == 0)
90 {
91     TaskbarManager.Instance.CircuitSaveError("The custom circuit  ↗
            must consist of (1) or more circuits.");
92     yield break;
93 }
94
95 // Validation test #4: no input/display gates
96 foreach (Circuit circuit in  ↗
    EditorStructureManager.Instance.Circuits)
97 {
98     Type type = circuit.GetType();
99
100     if (type == typeof(InputGate) || type == typeof(Display))
101     {
102         TaskbarManager.Instance.CircuitSaveError("The custom  ↗
            circuit must not consist of any input gates or  ↗
            displays.");
103         yield break;
104     }
105 }
106
107 // Validation test #5: all circuits are connected
108 reachedCircuits = new bool  ↗
    [EditorStructureManager.Instance.Circuits.Count];
109 emptyInputs = new List<Circuit.Input>(); inputs = new  ↗
    List<Circuit.Input>();
110 emptyOutputs = new List<Circuit.Output>(); outputs = new  ↗
    List<Circuit.Output>();
111 circuitCount = 0;
112 CircuitConnectionTest(EditorStructureManager.Instance.Circuits  ↗
    [0]); // Begins the DFS algorithm
113
114 if (circuitCount != reachedCircuits.Length)
115 {
116     TaskbarManager.Instance.CircuitSaveError("The custom circuit  ↗
            must be entirely connected.");
117     yield break;
118 }
119
120 // Validation test #6: >= 1 empty outputs
121 if (emptyOutputs.Count == 0)

```

```
122     {
123         TaskbarManager.Instance.CircuitSaveError("The custom circuit
124         must have (1) or more empty outputs.");
125         yield break;
126     }
127     /// All validation tests completed ///
128
129     currentName = name;
130     TaskbarManager.Instance.CloseMenu();
131     TaskbarManager.Instance.NullState();
132
133     // Begins the process in which the user assigns the order and
134     // labels of all empty inputs and outputs.
135     IOAssigner.Instance.Initialize(emptyInputs, emptyOutputs);
136 }
137
138 /// <summary>
139 /// Starts the coroutine involved in finally creating a custom
140 circuit.<br/><br/>
141 /// This method is specifically called by <see cref="IOAssigner"/>
142 after all empty inputs and outputs have been ordered by the user (as
143 well as any respective labling).
144 /// </summary>
145 /// <param name="orderedInputs"></param>
146 /// <param name="orderedOutputs"></param>
147 /// <param name="inputLabels"></param>
148 /// <param name="outputLabels"></param>
149 public void CreateCustomCircuit(List<Circuit.Input> orderedInputs,
150 List<Circuit.Output> orderedOutputs, List<string> inputLabels,
151 List<string> outputLabels)
152 {
153     StartCoroutine(CreatePreviewStructure(orderedInputs,
154     orderedOutputs, inputLabels, outputLabels));
155 }
156
157 /// <summary>
158 /// Serializes a custom circuit as well as its corresponding preview
159 structure.
160 /// </summary>
161 /// <param name="orderedInputs">The list of empty inputs, ordered.</
162 param>
163 /// <param name="orderedOutputs">The list of empty outputs, ordered.</
164 param>
165 /// <param name="inputLabels">Labels associated with each ordered
166 input.</param>
167 /// <param name="outputLabels">Labels associated with each ordered
168 output.</param>
169 private IEnumerator CreatePreviewStructure(List<Circuit.Input>
```

...	ets\Scripts\Editor Scripts\PreviewStructureManager.cs	5
-----	---	---

```

orderedInputs, List<Circuit.Output> orderedOutputs, List<string>
inputLabels, List<string> outputLabels)
158     {
159         TaskbarManager.Instance.OnSuccessfulPreviewVerification();
160
161         // Skipping a frame ensures the UI dialog for creating a custom
162         // circuit will show.
163         yield return null;
164
165         List<CircuitIdentifier> circuitIdentifiers = new
166             List<CircuitIdentifier>();
167         List<int> inputOrders = new List<int>(), outputOrders = new
168             List<int>();
169         PreviewStructure previewStructure = new PreviewStructure
170             (currentName);
171
172         // Serializes each circuit by instanting CircuitIdentifier
173         // references.
174         foreach (Circuit circuit in
175             EditorStructureManager.Instance.Circuits)
176         {
177             circuitIdentifiers.Add(new CircuitIdentifier(circuit));
178
179             foreach (Circuit.Input input in circuit.Inputs) { inputs.Add
180                 (input); inputOrders.Add(orderedInputs.IndexOf(input)); }
181
182             foreach (Circuit.Output output in circuit.Outputs)
183                 { outputs.Add(output); outputOrders.Add
184                     (orderedOutputs.IndexOf(output)); }
185         }
186
187         previewStructure.Circuits = circuitIdentifiers;
188         previewStructure.ID = UniqueID; // Assigns a unique ID to the
189         // preview structure.
190         previewStructure.InputOrders = inputOrders;
191         previewStructure.OutputOrders = outputOrders;
192         previewStructure.InputLabels = inputLabels;
193         previewStructure.OutputLabels = outputLabels;
194         previewStructure.CameraLocation =
195             CameraMovement.Instance.PlayerCamera.transform.position;
196
197         List<InternalConnection> internalConnections = new
198             List<InternalConnection>();
199
200         // Serializes each connection by assigning index values to each
201         // input/output pair within an InternalConnection instance.
202         foreach (CircuitConnector.Connection connection in
203             EditorStructureManager.Instance.Connections)
204         {

```

```
191         internalConnections.Add(new InternalConnection(
192             inputs.IndexOf(connection.Input),
193             outputs.IndexOf(connection.Output)
194         ));
195     }
196
197     previewStructure.Connections = internalConnections;
198
199     // Adds preview structure and its connections to the save directory and add menu.
200     MenuSetupManager.Instance.PreviewStructures.Add(previewStructure);
201     MenuSetupManager.Instance.GenerateConnections(false,
202         previewStructure.ID,
203         EditorStructureManager.Instance.Connections);
204     MenuSetupManager.Instance.UpdatePreviewStructure(previewStructure);
205     TaskbarManager.Instance.AddCustomCircuitPanel(previewStructure.ID, false);
206     TaskbarManager.Instance.OnSuccessfulPreviewStructure();
207 }
208
209 /// <summary>
210 /// Performs a depth-first search starting at the first placed circuit to determine whether the scene represents a complete graph. <br/>
211 /// At the same time, any circuit input or output without a connection is stored for the next test.
212 /// </summary>
213 private void CircuitConnectionTest(Circuit currentCircuit)
214 {
215     while (currentCircuit.customCircuit != null)
216     {
217         currentCircuit = currentCircuit.customCircuit;
218     }
219
220     int index = EditorStructureManager.Instance.Circuits.IndexOf(currentCircuit);
221
222     if (reachedCircuits[index]) return;
223
224     reachedCircuits[index] = true;
225     circuitCount++;
226
227     foreach (Circuit.Input input in currentCircuit.Inputs)
228     {
229         if (input.ParentOutput == null) { emptyInputs.Add(input); continue; }
230
231         CircuitConnectionTest(input.ParentOutput.ParentCircuit);
232     }
233 }
```

```
231
232     foreach (Circuit.Output output in currentCircuit.Outputs)
233     {
234         if (output.ChildInputs.Count == 0) { emptyOutputs.Add(output); }
235         continue; }
236
237     foreach (Circuit.Input input in output.ChildInputs)
238     {
239         CircuitConnectionTest(input.ParentCircuit);
240     }
241 }
242
243 /// <summary>
244 /// Returns a new unique ID for a new preview structure.<br/><br/>
245 /// A unique ID starts from 0 and increments onward.
246 /// </summary>
247 private int UniqueID
248 {
249     get
250     {
251         int currentID = 0;
252
253         // Keeps incrementing the current ID until it is unique
254         // This system ensures that if an ID that is not the largest
255         // is removed, it will be recycled in future custom circuit
256         // creations.
257         while (true)
258         {
259             if (!
260                 MenuSetupManager.Instance.PreviewStructureIDs.Contains
261                 (currentID))
262             {
263                 MenuSetupManager.Instance.PreviewStructureIDs.Add
264                 (currentID);
265                 return currentID;
266             }
267             currentID++;
268         }
269     }
270 }
271
272 // Getter method
273 public static PreviewStructureManager Instance { get { return
274     instance; } }
```