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

/// <summary>
/// EditorStructureManager accesses the current <see cref="EditorStructure"/> in use and directly manages its serialization/deserialization.
/// </summary>
public class EditorStructureManager : MonoBehaviour
{
    // Singleton state reference
    private static EditorStructureManager instance;

    /// <summary>
    /// List of instantiated circuits in the current editor scene.
    /// </summary>
    [HideInInspector]
    List<Circuit> circuits = new List<Circuit>();

    /// <summary>
    /// Index representation of all currently bookmarked circuits.
    /// </summary>
    [HideInInspector]
    List<int> bookmarks = new List<int>();

    /// <summary>
    /// List of instantiated connections in the current editor scene.
    /// </summary>
    [HideInInspector]
    List<CircuitConnector.Connection> connections = new List<CircuitConnector.Connection>();

    /// <summary>
    /// Whether a prompt to save the scene should appear or not when attempting to save.<br/><br/>
    /// False after a successful save and true if any important actions occur within a scene.
    /// </summary>
    private bool displaySavePrompt = false;

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

        instance = this;
    }

    // Begins the deserialization process as the scene is loaded.
    private void Start() { Deserialize(); }

    /// <summary>
    /// Begins the serialization process by calling its corresponding coroutine.
    /// </summary>
    public void Serialize()
    {
        displaySavePrompt = false;
        StartCoroutine(SerializeCoroutine());
    }

    /// <summary>
    /// Gathers all relevant information from the current scene and loads it into the current <see cref="EditorStructure"/> before saving it to the directory.
    /// </summary>
    public IEnumerator SerializeCoroutine()
    {
        // By skipping a frame, the UI loading screen is able to appear for more complex scenes.
        yield return null;

        // Obtains the current editor structure to override
        int sceneIndex = MenuLogicManager.Instance.CurrentSceneIndex;
        EditorStructure editorStructure = MenuSetupManager.Instance.EditorStructures[sceneIndex];

        List<bool> isPoweredInput = new List<bool>();
        List<CircuitIdentifier> circuitIdentifiers = new List<CircuitIdentifier>();

        // Iterates through all circuits within the scene to assign all circuit identifiers and powered inputs.
        foreach (Circuit circuit in circuits)
        {
            circuitIdentifiers.Add(new CircuitIdentifier(circuit));
            isPoweredInput.Add(circuit.GetType() == typeof(InputGate) && ((InputGate)circuit).Powered);
        }

        // After obtaining all relevant values, override the editor
```

```
          structure
        editorStructure.InGridMode = Coordinates.Instance.CurrentSnappingMode == Coordinates.SnappingMode.GRID;
        editorStructure.IsPoweredInput = isPoweredInput;
        editorStructure.Circuits = circuitIdentifiers;
        editorStructure.Bookmarks = bookmarks;
        editorStructure.BookmarkIDs = TaskbarManager.Instance.BookmarkIDs;
        editorStructure.CameraLocation = CameraMovement.Instance.PlayerCamera.transform.position;

        // Saves the new editor structure to the directory and begins serializing all connections within the scene.
        MenuSetupManager.Instance.UpdateEditorStructure(sceneIndex, editorStructure);
        MenuSetupManager.Instance.GenerateConnections(true, sceneIndex, connections);

        TaskbarManager.Instance.CloseMenu();
    }

    /// <summary>
    /// Opens an editor scene based on a specified <see cref="EditorStructure"/>, restoring components based on its save files.
    /// </summary>
    public void Deserialize()
    {
        // Obtains the current editor structure to reference
        int sceneIndex = MenuLogicManager.Instance.CurrentSceneIndex;
        EditorStructure editorStructure = MenuSetupManager.Instance.EditorStructures[sceneIndex];

        // If this is the first time the scene has been opened, set default values, save to directory, and return.
        if (MenuLogicManager.Instance.FirstOpen)
        {
            bool inGridMode = Coordinates.Instance.CurrentSnappingMode == Coordinates.SnappingMode.GRID;
            Vector3 cameraLocation = CameraMovement.Instance.PlayerCamera.transform.position;

            editorStructure.InGridMode = inGridMode;
            editorStructure.CameraLocation = cameraLocation;
            TaskbarManager.Instance.RestoreCustomCircuits(); // Ensures all custom circuits still appear in the add menu
            MenuSetupManager.Instance.UpdateEditorStructure(sceneIndex, editorStructure); // Saves to directory
            return;
        }

        Coordinates.Instance.CurrentSnappingMode = editorStructure.InGridMode ? Coordinates.SnappingMode.GRID : Coordinates.SnappingMode.NONE;
        CameraMovement.Instance.PlayerCamera.transform.position = editorStructure.CameraLocation;

        // Instantiates all circuits back to the scene
        foreach (CircuitIdentifier circuitIdentifier in editorStructure.Circuits)
        {
            circuits.Add(CircuitIdentifier.RestoreCircuit(circuitIdentifier));
        }

        // Restores all connections back to the scene
        MenuSetupManager.Instance.RestoreConnections(sceneIndex);

        int index = 0;

        // After all circuits have been established and connected, turn on any previously powered input gate.
        foreach (bool isPoweredInput in editorStructure.IsPoweredInput)
        {
            if (isPoweredInput) ((InputGate)circuits[index]).Powered = true;
            index++;
        }

        // Lastly, all bookmarks and custom circuits are restored to their respective menus.
        TaskbarManager.Instance.RestoreBookmarks(editorStructure.Bookmarks, editorStructure.BookmarkIDs);
        TaskbarManager.Instance.RestoreCustomCircuits();
    }

    // Getter methods
    public static EditorStructureManager Instance { get { return instance; } }

    public bool DisplaySavePrompt { get { return displaySavePrompt; } set { displaySavePrompt = value; } }

    public List<Circuit> Circuits { get { return circuits; } }

    public List<CircuitConnector.Connection> Connections { get { return connections; } }

    public List<int> Bookmarks { get { return bookmarks; } }
}
```