# Developing On Scaffold

Copyright (c) 2020. Ascendant Design and Training, LLC.  
Last updated by Daniel Patterson, Thursday, October 22, 2020

Developers in the community are welcome to help out, test the runtime, or even use the beta versions for making your own educational projects (once we get to that point).

## Description

This project is developed in the .NET Framework, written in C#, and has a WinForms user interface. It currently runs on Windows and will be made cross-platform compatible after some of the first versions have been stabilized.

The current version is configured to work on Windows 64-bit desktop systems like the following.

* Windows 10.
* Windows 8 (64-bit).
* Windows 7 (64-bit).

## Distribution

The application currently contains the CefSharp HTML rendering subsystem for various variable display purposes. When distributing the application, that library is dependent upon the following[[1]](#footnote-1):

Minimum of .Net 4.5.2

Minimum of 'Visual C++ 2015 Redist' is installed (either 'x86' or 'x64' depending on your application). VC++ 2017/2019 are backwards compatible.

Please ensure your binaries directory contains these required dependencies:

* libcef.dll (CEF code)
* icudtl.dat (Unicode Support data)
* CefSharp.Core.dll, CefSharp.dll,
* CefSharp.BrowserSubprocess.exe, CefSharp.BrowserSubProcess.Core.dll
* These are required CefSharp binaries that are the common core logic binaries of CefSharp.
* One of the following UI presentation approaches:
  + CefSharp.WinForms.dll
  + CefSharp.Wpf.dll
  + CefSharp.OffScreen.dll
* Additional optional CEF files are described at: https://github.com/cefsharp/cef-binary/blob/master/README.txt#L82

NOTE: CefSharp does not currently support CEF sandboxing.

By default 'CEF' has it's own log file, 'Debug.log' which is located in your executing folder. e.g. 'bin'

More information can be found in the ReadMe.txt file at {GitHubProjects}/Scaffold/packages/CefSharp.WinForms.84.4.10

## Installation

To get a developer instance of Scaffold running, follow these steps.

* Verify that you have Microsoft Visual Studio 2019 or later installed on your PC.
* Open a command-line or terminal window.
* Issue the following commands on the command line.

<pre><code>git clone <https://github.com/ascendantdesign/Scaffold>  
cd Scaffold/Source/Scaffold</code></pre>

If Visual Studio 2019 or later is your default Visual Studio editor, you can type the following filename directly from the command-line to open the solution.

<pre><code>Scaffold.sln</code></pre>

Otherwise, you can open the project from Visual Studio 2019 or later from the folder {GitHubProjects}/Scaffold/Source/Scaffold/Scaffold.sln

Note that the redundancy in naming policy has to do with the fact that multiple projects are present in the Scaffold/Source folder, and Scaffold/Source/Scaffold contains the central solution of the project.

## Non-Linear Decision Tree Editor

The non-linear decision tree feature of Scaffold uses node network editing components to allow freeform expression of extremely complicated flows.

### Things You Can Do With Decision Tree Editor

A node editor for designing a flowing decision tree can be valuable in a number of cases.

* Designing branching scenario and non-linear course modules. This includes the following applications.
  + Adaptive course.
  + Changing stakes course.
  + Conversational session.
  + Escape-the-room game.
* Defining non-sequential pathways.

### Editor Make-Up

The editor offers multiple types of nodes. Each of the node types is specialized in what can possibly happen at a certain step.

### Start

The start node has no inputs and allows for an output fan-out condition. Each output can only be sent to one destination.

Following are some of the characteristics of the start node.

|  |  |
| --- | --- |
| **Name** | **Description** |
| Inputs | 0 |
| Input Source |  |
| Question | Mandatory |
| Answers | Multiple |
| Output Style | Per answer |
| Output Targets | Singular |

### Fork

A fork node has a single input capable of receiving signals from multiple nodes. The fork has a fanned output where each output node is only capable of being directed to a single target, similarly to the start node.

Following are some of the identifiable features of the fork node.

|  |  |
| --- | --- |
| **Name** | **Description** |
| Inputs | 1 |
| Input Source | Multiple |
| Question | Optional |
| Answers | Multiple |
| Output Style | Per answer |
| Output Targets | Singular |

### Delay

The delay node is similar to the fork node in socket connections. It differs from the fork, however, in that none of the output sockets has a user answer, and all output sockets are all followed simultaneously.

Following are some of the characteristics of the delay node.

|  |  |
| --- | --- |
| **Name** | **Description** |
| Inputs | 1 |
| Input Source | Multiple |
| Question | None |
| Answers | None |
| Output Style | Multiple |
| Output Targets | Singular |

### Termination

The termination node has a single input capable of receiving signals but no output nodes.

Following are some of the notable traits of the termination node.

|  |  |
| --- | --- |
| **Name** | **Description** |
| Inputs | 1 |
| Input Source | Multiple |
| Question | None |
| Answers | Singular |
| Output Style | None |
| Output Targets |  |

## Windows and Views

In this discussion, a view control is normally similar to a canvas of potentially infinite size, where anything might be drawn or portrayed without hard boundaries. Conversely, a window control is typically an area that can be drawn upon but has a distinct limitation in position and size.

In many cases, view and window controls are used together. In those cases, the view sits behind the window sliding back and forth, while the window effectively crops the view into the form.

## Scrollbar Behavior

Scrollbars move the current view along the X and Y axis within the window. When the scrollbar is at the very top of its range, the top of the window is aligned with the top or left sides of the view. The opposite is also true. When the scrollbar is set to the bottom of its range, the bottom or right edges of the window are aligned with the bottom or right edges of the view, respectively.

## Undo Principles

Scaffold's undo system is declarative in nature. Undo storage is provided by a stack, to which items can be pushed during normal operation, then popped in reverse sequence if requested.

Each item in the undo stack is a loose reference to the object and its associated event. It contains the following information.

* **Item type**. The formal name of the item type. For example, "NodeType", "String", etc.
* **Item name**. The identification of the affected object.
* **Action type**. Name of the type of action that has taken place. For example, "Delete", "Add", "Edit", etc.
* **Properties**. A collection of notable property values. For each object and action, there might be a number of possible values useful for reestablishing the previous value. This collection contains name/value pairs to aid in the recovery process.

When using the undo system, you might notice that other than Properties[x].Value, all of the information found in an undo object is string-based. This serves the double purpose of allowing the undo system to be extensible to system plug-ins that will be implemented at a later date and keeping the overall data requirement low. Using this approach, basic analysis techniques can be utilized, first to determine whether the undo action is recognized or pass it to an external connection, then to reconstruct the object from details provided in the description.

In sharp contrast, previous experience tells us that the self-contained object/function/delegate form of undo management consumes several times more memory per object than each object being monitored, and that overhead analysis, the one thing to avoid during so-called self-contained management, tends to be required in the long run anyway, and also tends to be much more complicated due to the abstract delegate nature of its construction.

### Action Types

The current list of well-known edit action types follows.

|  |  |
| --- | --- |
| **Action Type** | **Remarks** |
| Add | An object has been added to the file. |
| Delete | An object has been deleted from the file. |
| Edit | A value of the object has been edited. |
| Move | The object has been moved. |

### Item Types

In this version, a few well-known item types are present.

| **Item Type** | **Item Name** | **Remarks** |
| --- | --- | --- |
| NodeItem | {Node.Ticket} | Information about a node or its properties. |
| SocketConnectionItem | {Node.Ticket},{Socket.Ticket} -> {Property[0].Value} | A socket to socket connection identification reference. |
| SocketItem | {Node.Ticket},{Socket.Ticket] | Information about a socket on a node, or that socket's properties.  Every socket is identified primarily by its parent node to optimize the search for that socket in the document. A single-layer search is performed for the parent node. If that item is found, only that node is searched for the specified socket. |

### Supported Actions

In this version, the following undo actions are supported.

| **Item Type** | **Action Type** | **Properties** | **Remarks** |
| --- | --- | --- | --- |
| NodeItem | Add | (none) | The ticket of new item is present, and the Properties collection is empty. |
| NodeItem | Edit | (changed) | Only the value or values changed during this event.  In the Properties list, Properties[0] contains the name of the property being changed. Properties[1,2] contain {Name}Before and {Name}After values. |
| NodeItem | Delete | (current) | All node properties at the time of deletion, including all sockets and their properties.  When restoring this value, restoration of any previous connection references is attempted. If the connection can not be restored, the reference is discarded. |
| NodeItem | Move | X, Y |  |
| SocketItem | Add | (none) | A new socket has been added to a node.  ItemName contains {Node.Ticket},{Socket.Ticket}. The Properties collection is empty. |
| SocketConnectionItem | Add | Connection | A new connection has been added.  ItemName contains {Node.Ticket},{Socket.Ticket}  Properties[0] contains the ticket value of the added connection. |
| SocketConnectionItem | Delete | Connection | A connection has been deleted.  ItemName contains {Node.Ticket},{Socket.Ticket}  Properties[0] contains the ticket value of the deleted connection. |
| SocketItem | Edit | (changed) | Only the value or values changed during this event.  ItemName contains {Node.Ticket},{Socket.Ticket}  Properties[0] contains the property name.  Properties[1, 2] are labeled {PropertyName}Before and {PropertyName}After, and contain the before and after values. |
| SocketItem | Delete | (current) | All socket properties at the time of deletion.  When restoring this value, restoration of any previous connection references is attempted. If the connection can not be restored, the reference is discarded.  ItemName contains {Node.Ticket},{Socket.Ticket}  Properties collection contains full references to operational Properties and Connections collections, which are reattached to a newly created socket. |

### Undo Object Structure

The following time-sensitive undo object model and general strategy is currently in use. The approach greatly reduces the number of items the user has to act upon when undoing recent actions, and mostly contextualizes multiple actions into transient temporal domains.

* UndoPack. Container of multiple undo stacks. When used in a time-sensitive scenario, all of the items of a single stack can be undone, and that stack can be popped. This allows multi-property activities to be condensed into individual user actions.
  + UndoStack. Container of multiple undo items. The stack has a time reference that can be used to partition sets of undo actions into time-sensitive single actions.
    - UndoItem. An individual undo item.

## Node And Socket Media Support

Nodes and sockets both have support for multiple types of media and support for basic interaction. In general, nodes and each of a node's sockets have support for the following elements.

These additions allow the node and its sockets to present a self-contained card-like personality during runtime that has multiple aspects available for interactive purposes. In industry terms, this effect produces that of the Hero Card.

|  |  |  |
| --- | --- | --- |
| **General Element** | **Property Name** | **Description** |
| Audio | MediaAudio | On nodes, this is a URI referring to an audio to play when the node is loaded. On sockets, the URI is played when the option is selected and before the connection is followed to the next node. |
| Image | MediaImage | A URI referring to an image to display in the card. base64 data URI is allowed. |
| Link | MediaLink | A URL the user can click to view more information about the current context. Standard href link syntax. |
| Video | MediaVideo | On nodes, this is a URI referring to a video to play when the node is loaded. On sockets, the URI is played when the option is selected, and before the connection is followed to the next node. |

Notice that in this version, if audio and video URIs are both specified, the video URI will take precedence, and the audio will only be utilized on non-video systems that provide audio support, such as a telephone system or radio, etc.

1. From CefSharp NuGet Readme.txt at <https://github.com/cefsharp/CefSharp/blob/master/NuGet/Readme.txt> [↑](#footnote-ref-1)