Cheat Sheets

Version 1.0.0 (2022-06-15)

Official Documentation

Louis-Pierre Aubert | **Louis8257**

Abstract

Searching articles on the Internet can be time consuming, becoming frustrating in the long term. The forums might be a good place to find information but duplicates can exist and can contain confusing or outdated information.

With the request of several developers, this is why I created the project « Cheat Sheets ». The goal of this project is to make cheat sheets regrouping every confusing or « easy-to-forgot » information about the editor and the game engine, so Unity can become a welcoming place for every beginners and contribute to the quality of everyone games!

Thank you for downloading Cheat Sheets! Don't forget you can contribute to this project by sharing your knowledge to everyone!

Table of contents

Overview

Cheat Sheets uses mainly reflections to detect new sheets and Unity's UIElements to display the sheet.

Custom Attribute

It uses a custom attribute called CustomCheatSheetAttribute that must be attached in a serializable class extending from CheatSheetElement.

This design allows modularity by getting every classes directly and then scan the attributes, regardless of the namespace or assembly.

Every cheat sheets must be in the Resources folder! Otherwise it won't be able to find your UXML files.

UXML and USS

To display text and images or any VisualElements, Cheat Sheets relies on UXML files. You can link an USS (Unity Style Sheet) by creating a Root VisualElement and then attach the style sheet inside but every elements must be inside the Root.

Stack

The stack is refreshed every time you open the Cheat Sheets window. It will read and connect every pages to others and place them in a tree.

When searching, it will create another tree but only with the attributes containing the keywords.

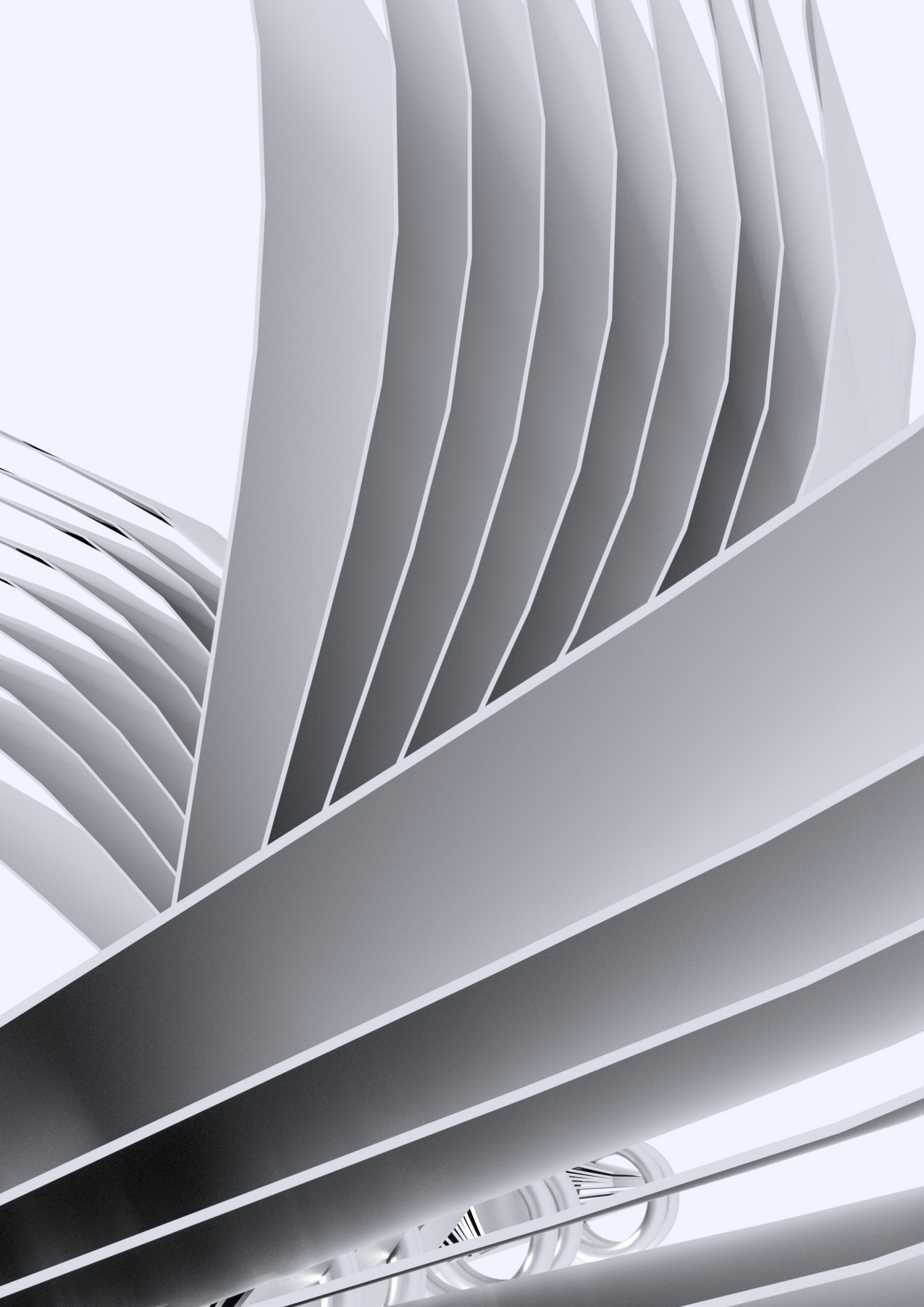
Selector

Self explanatory. You can open its window by clicking on the button at the top of the Cheat Sheets window. You can select any page from any attributes. They are sorted by alphabetical orders with a color mark.

Creator

To avoid to rewrite the same scripts and UXML resources again, you can use this tool! You can create several cheat sheets and pages. Pages are linked to the cheat sheet script automatically.

You can open the creator by clicking on the '+' button at the right top corner in the Selector window.



Implementation

1. Mindset

Before creating your cheat sheets, you should follow some common rules to keep consistency and efficiency so the users can read thousands of different cheat sheets with ease.

Size

"The simplest ones are often the best." Your cheat sheet should talk about a simple subject that talks about a spefic feature the most informative way in few lines.

Your cheat sheet should not exceed one scroll (the user should see the whole information in the first glance). **Don't forget you can use multiple pages!** You can divide the information into several pages. *You can include sources if you can't fit the whole information in one page.*

Text

The text must be the most informative and accurate as possible.

Sources gives credibility to your information. This allows to check the accuracy of this information. It also gives credit to the author if you have just copy-pasted.

Don't go off topic! You can talk about another subject in a separated page.

English is the prefered language, so a maximum of people can understand you.

The text must be readable! The color of the text must be different from the background's color.

Presentation

Your cheat sheets will be displayed inside the editor, you can adapt your text so it can fit in the window and use white color on grey background (the default background of the editor).

Act like you are writing on paper! Put a 10px margin on the left, 12px font for your text and a bigger one for your titles and spaces between paragraphs.

Use font styles! This will help the user to find information faster. For now you can assign a style on an VisualElement only. Rich text may be supported in the future. Each style have a meaning:

- Normal Style: The standard font. It represents the information.
- Bold Style: Means something to be careful of. If the text is red or a red background, it intensifies its importance.
- Italic style: Italic font means a complementary information or a quote. This information is not crucial, but still useful. Can also be used as a reminder.

Use a list to enumerate or decompose the information.

Images

"A picture is worth a thousand words." Use pictures to illustrate your information. Just like for the text, it must be readable and understandable.

When done correcty, it's possible to condensate the whole information in one picture only. You can also use sheets, diagrams or even drawing in the scene view if needed.

To not have blurry images, you have to set the render mode to “Editor GUI and Legacy GUI".

And last but not least: be yourself when writing your cheat sheet!

2. Tutorials

As you can see, there's several ways to make cheat sheets. You can make the resource files and scripts by hand, or use the built-in creator wizard.

2.1. Creator Wizard

In this tutorial, we will use the built-in Cheat Sheets Creator, so you can make the files quickly. We will also create three pages, so you learn how to create several cheat sheets and pages.

Cheat Sheets Creator Overview

To access the cheat sheet creator, click on the "+" in the top-right corner of the selector window, at the end of the search bar.

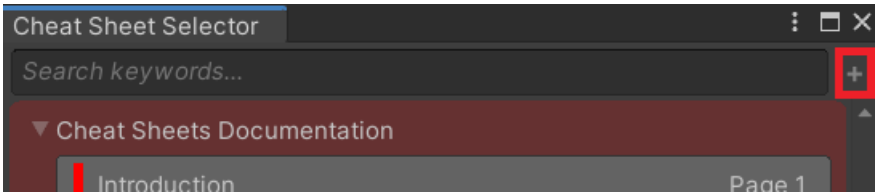


Fig. 2.1.1 *Creator wizard location*

Once you open the wizard, it might be confusing at first glance but it's actually simple once you decomposed it into layouts!

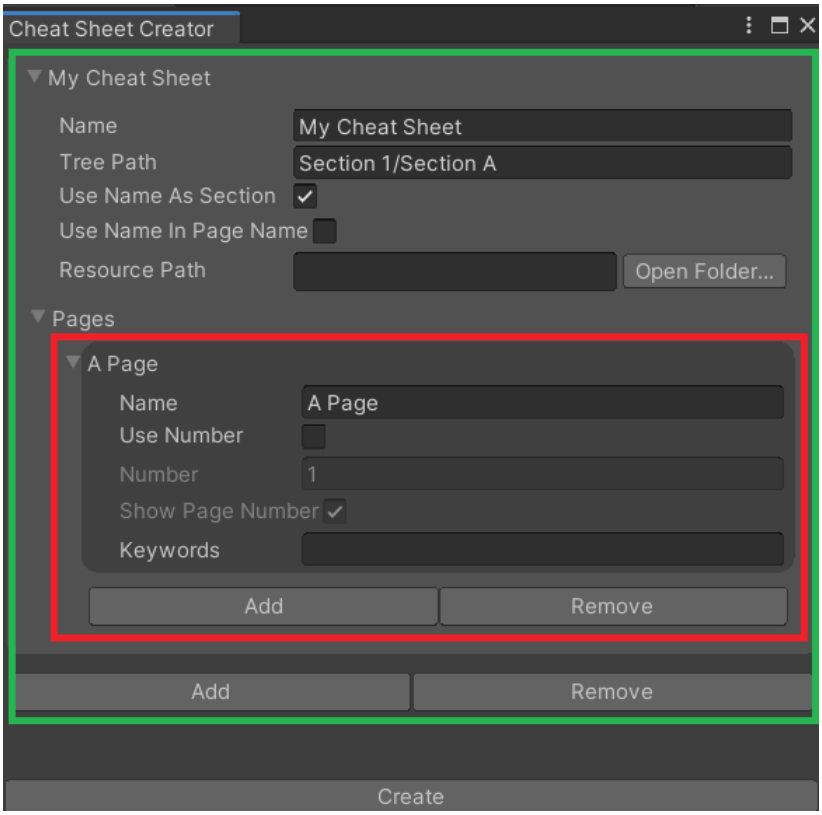


Fig. 2.1.2 *Creator wizard overview with layouts*

List Of Cheat Sheets

This will regroup the pages in the same section. You can also add several cheat sheets! It represents CheatSheetElement classes.

List Of Pages

This list is unique for each cheat sheet. It represents a CustomCheatSheet attributes.

ListView was considered during the development but you can't have individual item size and you can't make it reorderable anymore. I'm using ScrollView instead, for now.

Now we can start to make a cheat sheet! *Every fields have a tooltip, for more information.*

Configuring A Cheat Sheet

We will call this cheat sheet "Test". You can name it whatever you like.

The tree path will be "Test Section". If you want to create a subsection, you can put a '/' separator and then a new section name. You can also put no section name at all but I strongly recommand to regroup your pages under a section, unless you checked the checkbox bellow.

You can let these checkboxes as they are. It will add a subsection to the tree path with the name of this cheat sheet.

Now you have to choose a "Resources" folder. Try to create an empty "Resources" folder anywhere in your project. Click on "Open Folder..." to pick a folder.

Configuring A Page

Like for the cheat sheet, you can name this page whatever you like. Check the checkbox "Use Number" and don't change their values. *When we will create a new page, it will clone this one and increment the page number.*

Adding A New Page

Still in the cheat sheet layout, add a new page. It will clone the last page and increment the page number. Make one more. You should have three pages at this point. You are free to call them whatever you want.

Generate Resources

Now we can generate the resources! Click on the "Create" button at the bottom of the window. It will look for errors before generating the files.

If the window stays open, this mean there's a problem in a page. Look at the console to see the issues. No resource should have been generated. You can try again after fixing the pages or cheat sheets.

If it's not the case, this mean it's generating and import the files into the project. Unity will compile when this step is done. **If the compilation fails you can contact me, it should never happen.**

After the compilation, try to select it in the selector window. Your cheat sheet will show a generic message, that means it worked and you can modify the generated UXML file.

Congratulations! You have finished the tutorial! You should be able to make cheat sheets in no time. You can go further by adding Interactivity in your cheat sheets to improve the user experience (UX).

2.2. Creating Scripts And UXML Files By Hand

Every cheat sheets are represented as a class extending CheatSheetElement and each CustomCheatSheet attribute represents a page.

Making a cheat sheet is very straightforward and making pages can be simple or complex for specific usages.

Script

The script is just a class extending CheatSheetElement. It will not work without any attribute. It also regroups every interactions with the pages.

Just like in the Editor class, you can modify the methods OnEnable(), OnDisable() and OnGUI(). OnEnable and OnDisable is called when the page's visual element is created or destroyed, respectively. OnGUI is called from the main window and is refreshed every frame when an input is made in this window or you can override the method RequiresConstantRepaint() to return true to force a repaint on each frame.

I recommand to create a method or class for each page and using a switch for each page number with enums.

Attribute

The attribute define a page following this format bellow:

```
[CustomCheatSheet("Cheat Sheets Documentation/#3 Script And Custom Attribute",
                  "CheatSheets/Core/Home/Home_3",
 keywords = new string[] { "Cheat Sheet", "Pages", "Tree", "Sections"})]
```

Fig. 2.2.1 *CustomCheatSheet attributes decomposed in lines*

- **Tree Path**: Tells to the stack in which section this page will be placed. Must contain a '/' separator for each section.

  You can also define the page number and name by adding '#' at the end followed by and integer number and a name. There must be a space between this number and the name.
- **Resource Path**: The location of the UXML file in the resource folder. Must contain a '/' separator for each folder and contain an unique file name at the end. *Keep in mind that you must place your UXML file in any 'Resources' folder.*
- **Keywords**: To make your page more possible and easier to find, you can describe its content here. *It's optional but I strongly recommand to do so.*

  I also recommand to add plural words instead of singular words to maximize the chance of the user finding this page. If the plural word is written different than the

singular word (like tooth and teeth or mouse and mice for example), use both.

2.3. Using UI Builder

After creating the script file and the UXML file, you can use the UI Builder already provided in the newest versions of Unity!

For versions prior to Unity 2021, you need to download a package. *Click here for the official documentation*.

Overview

The UI Builder's window is divided into three columns from left to right, respectively:

- **StyleSheets, Hierarchy and Library**: Regroups every UI elements related data. The style sheet can be applied to any individual elements. The hierarchy works the same way as the scene hierarchy. The library lists every loaded UI elements, including custom ones.
- **Viewport and Code Preview**: Displays the content of the opened UXML file as UI elements. You can interact with them in the Preview mode You can also modify the source UXML file and this view will update with the new data. It also show a preview of the UXML and USS code below the viewport. Useful if you want to experiment.
- **Inspector**: Just like the classic Inspector, it will show modifiable properties of the selected element. Multiselection is not supported yet.

Modify UXML file

To modify your UXML file, click on the 'File' button in the upper-left corner of the viewport or click on your file in the 'Project' tab. You should see the rendered visual tree with the UI elements now. Once you opened the file, you can place any UI elements in the viewport.

Limitations And Bugs

For long UI elements, the flex may clip through other VisualElement underneath it. Be sure to expand the preview window to fix this problem.

The interactive part is not a standard Unity feature and, therefore, not functional in the UI Builder's preview mode.

2.4. Add Interactivity

In every ".cs" files, you can find methods that allows your cheat sheets to interact with the editor or the game. You can change the selected page from the window or open a link, for example. Everything you can do in Unity Editor is also doable here!

You can take a look at the script example, that is the source code of this plugin home's main menu.

CheatSheetElement contains familiar method names and signatures like *OnEnable()*, *OnDisable()* and *OnGUI()*. The most important variable is *rootVisualElement* that contains the visual tree so you can *querry* for any visual elements in your page. **Keep in mind that every pages *(as CustomCheatSheetAttribute)* can use this same class.** Here's some suggestions on how to use these methods correctly with several pages :

- **Use enums** :enums are *int* values that uses names as *keys*. You can name this key like a page name and use this in a *switch* anywhere in your code.
- **Add custom methods** : I recommand to create a method for each pages because you have to access the rootVisualElement to register an event in a button, for example. You will be able to use them with ease in a *switch*, with your *enums*.
- **OnEnable()** : *This method is called when one of your page is shown in CheatSheetWindow.*

  You should get your buttons or labels inside this method. Like *serialized properties*, visual elements can be accessed through reference fields that can be modified. *Therefore it doesn't contains any form of serialization. To get this behaviour, you can use OnGui() and update the display with the value of your serializable variables.*
- **OnDisable()** : *This method is called when one of your page is no longer shown in CheatSheetWindow.*

  You can destroy callbacks or delete some GameObjects or windows here.
- **OnGUI()** : *This method is called when one of your page is currently shown in CheatSheetWindow To call it every frame, the method RequiresConstantRepaint() must return true.*

  You can catch input events *(with UnityEngine.Event)* and update visual elements here.

You can see an example in the next page.

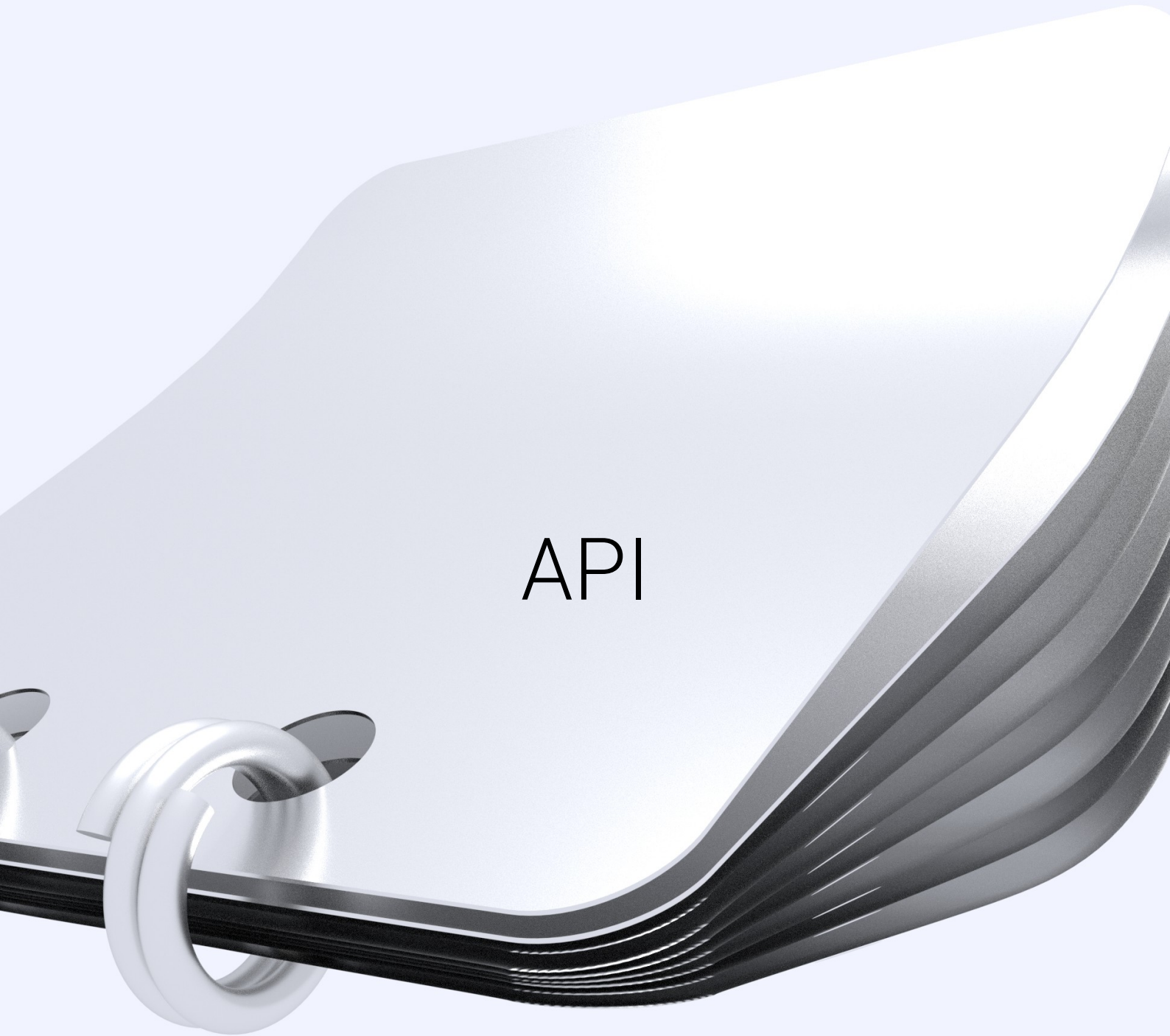
3. Script Example

```
using UnityEngine;
using UnityEngine.UIElements;

namespace CheatSheets {

    [CustomCheatSheet("Cheat Sheets Documentation/#1 Introduction",
"CheatSheets/Core/Home/Home_1",
        keywords = new string[]{"Welcome", "Pages", "Buttons"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#2 How does it works",
"CheatSheets/Core/Home/Home_2",
        keywords = new string[] {"Pages", "Life Cycle", "Tree", "Buttons", "Custom
Attributes", "Serialization", "Keywords", "Styles", "UXML"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#3 Script & Custom Attribute",
"CheatSheets/Core/Home/Home_3",
        keywords = new string[] { "Cheat Sheet", "Pages", "Tree", "Sections",
"Serialization", "Keywords", "UXML"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#4 Mindset",
"CheatSheets/Core/Home/Home_4",
        keywords = new string[] { "Share", "Open source", "Documentation" })]
    [CustomCheatSheet("Cheat Sheets Documentation/#5 Create A Cheat Sheet",
"CheatSheets/Core/Home/Home_5",
        keywords = new string[] {"Pages", "Cheat Sheet", "Custom Attributes", "Scripts",
"Documentation", "Styles", "UXML"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#6 Using UI Builder",
"CheatSheets/Core/Home/Home_6",
        keywords = new string[] {"Window", "User Friendly", "UXML"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#7 Share your cheat sheets",
"CheatSheets/Core/Home/Home_7",
        keywords = new string[] {"Open Source", "Share", "Mindset", "Documentation",
"GitHub", "Asset Store"})]
    [CustomCheatSheet("Cheat Sheets Documentation/#8 Credits & Contacts",
"CheatSheets/Core/Home/Home_8",
        keywords = new string[] {"Louis8257", "Louis-Pierre Aubert",
"louis.8257@hotmail.com"})]
    public class HomeCheatSheet : CheatSheetElement {

        Button testBtn;

        enum Page {
            Introduction        = 1,
            HowDoesItWorks      = 2,
```

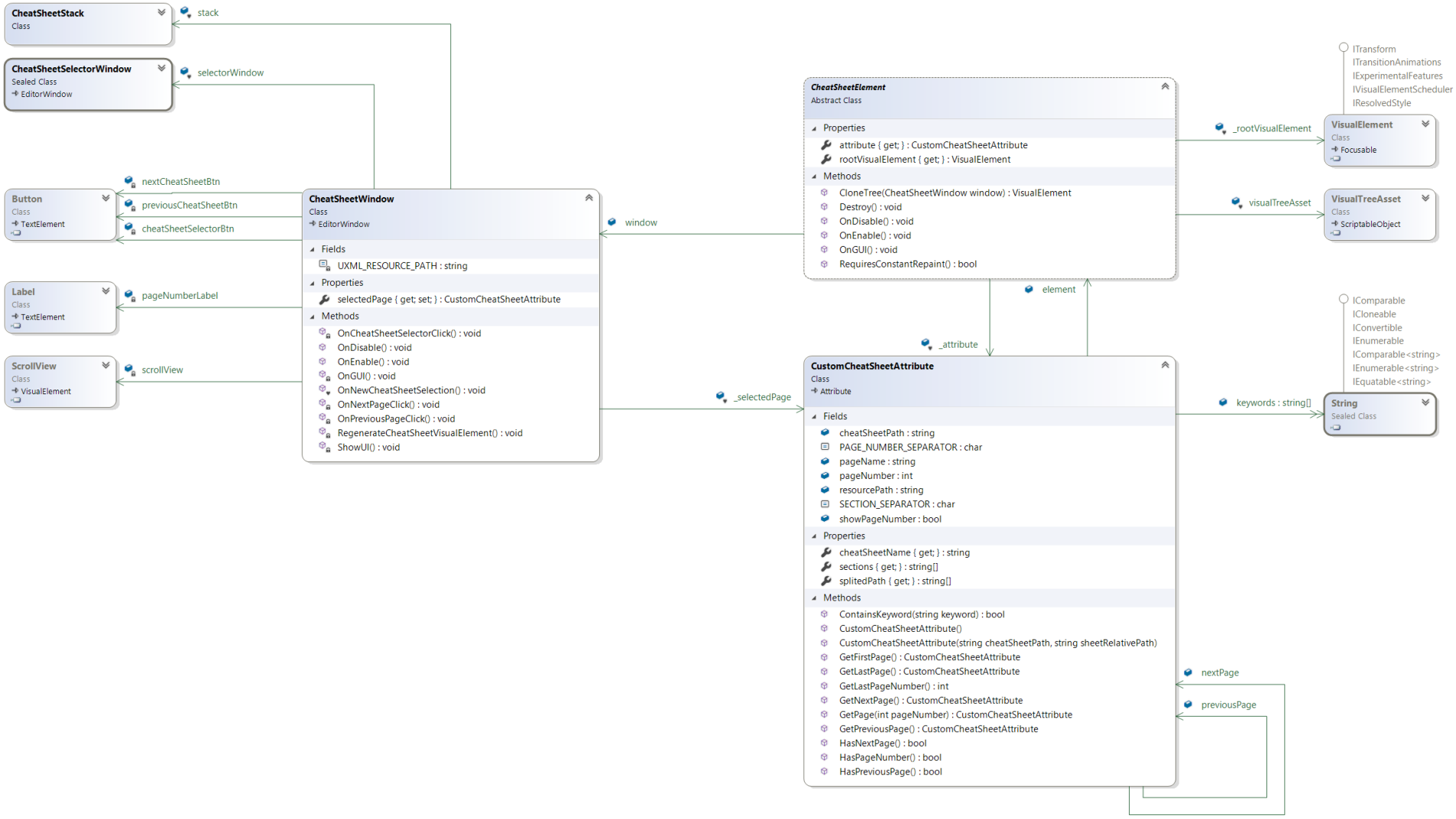
```
            ScriptAndAttributes = 3,
            Mindset             = 4,
            CreateCheatSheet    = 5,
            UIBuilder           = 6,
            Share               = 7,
            Credits             = 8
        }

        public override void OnEnable () {
            this.testBtn = this.rootVisualElement.Q<Button>("TestBtn");
            if ( this.testBtn != null ) {
                this.testBtn.clicked += () => { this.window.selectedPage =
this.window.stack.loadedAttributes[0]; };
            }

            Page index = (Page)this.attribute.pageNumber;
            switch ( index ) {
                case Page.CreateCheatSheet: OnEnablePage5(); break;
                case Page.UIBuilder:        OnEnablePage6(); break;
                case Page.Share:            OnEnablePage7(); break;
            }
        }

        #region "Page 5"
        void OnEnablePage5 () {
            Button goToPage3Btn = this.rootVisualElement.Q<Button>("GoToPage3Btn");

            goToPage3Btn.clicked += GoToPage3Btn_clicked;
        }

        void GoToPage3Btn_clicked () {
            this.window.selectedPage = this.attribute.GetPage(3);
        }
        #endregion

        #region "Page 6"
        void OnEnablePage6 () {
            Button uiBuilderInstallDocBtn =
this.rootVisualElement.Q<Button>("UiBuilderInstallDocBtn");
```

```
            uiBuilderInstallDocBtn.clicked += UiBuilderInstallDocBtn_clicked;
        }

        private void UiBuilderInstallDocBtn_clicked () {
            Application.OpenURL("https://docs.unity3d.com/Packages/com.unity.ui.builder@1.0/
manual/index.html");
        }
        #endregion

        #region "Page 7"
        void OnEnablePage7 () {
            Button gitHubLinkBtn = this.rootVisualElement.Q<Button>("GitHubLinkBtn");
            Button ccZeroLinkBtn = this.rootVisualElement.Q<Button>("CcZeroLinkBtn");

            gitHubLinkBtn.clicked += GitHubLinkBtn_clicked;
            ccZeroLinkBtn.clicked += CcZeroLinkBtn_clicked;
        }

        private void GitHubLinkBtn_clicked () {
            //Application.OpenURL();
        }

        private void CcZeroLinkBtn_clicked () {
            Application.OpenURL("https://creativecommons.org/publicdomain/zero/1.0/");
        }
        #endregion
    }
}
```

Fig. 5.1. *Source code of Cheat Sheets home*


API

1. CheatSheets



1.1. CheatSheetWindow

→ Public

→ Extends UnityEditor.EditorWindow

Constants	
UXML_RESOURCE_PATH : string	
Fields	
_selectedPage : CustomCheatSheetAttribute	→ Internal *For serialization.*
cheatSheetSelectorBtn : Button	
nextCheatSheetBtn : Button	
pageNumberLabel : Label	
previousCheatSheetBtn : Button	
scrollView : ScrollView	
selectorWindow : CheatSheetSelectorWindow	→ Internal

stack : CheatSheetStack	→ Public You can read the stack if you want to get a specific page. **You should never need to write or modify data in this variable.**
Properties	
selectedPage : CustomCheatSheetAttribute	→ Public Once changed, will display the newly selected page.
Methods	
OnCheatSheetSelectorClick : void	
OnDisable : void	→ Public *Override.*
OnEnable : void	→ Public *Override.*
OnGUI : void	
OnNewCheatSheetSelection : void	
OnNextPageClick : void	
OnPreviousPageClick : void	
RegenerateCheatSheetVisualElement : void	
ShowUI : void	

1.2. CheatSheetElement

→ Public

→ Abstract

The base class for every cheat sheet.

You can even use this class to add interactivity in your cheat sheets!

Must be paired with, atleast, one *CustomCheatSheetAttribute*.

Must be serializable or it will break the stack system!

Fields	
_attribute : CustomCheatSheetAttribute	→ Internal
_rootVisualElement : VisualElement	→ Internal
visualTreeAsset : VisualTreeAsset	→ Internal
window : CheatSheetWindow	→ Public
Properties	
attribute : CustomCheatSheetAttribute	→ Public
rootVisualElement : VisualElement	→ Public

Methods	
CloneTree window: CheatSheetWindow : VisualElement	→ *Public*
Destroy : void	→ Public
OnDisable : void	→ Public Is called each time the *visualTreeAsset* is destroyed. *Call Destroy instead if you want to destroy this element's rootVisualElement.*
OnEnable : void	→ *Public* Is called each time the *visualTreeAsset* is cloned.
OnGUI : void	→ *Public* Is called in *CheatSheetWindow.OnGUI*.
RequiresConstantRepaint : bool	→ *Public*

1.3. CustomCheatSheetAttribute

→ Public

→ Extends System.Attribute

→ Flagged System.Serializable

An attribute to define cheat sheets linked to this class.

You can also create multiple CustomCheatSheetAttribute.

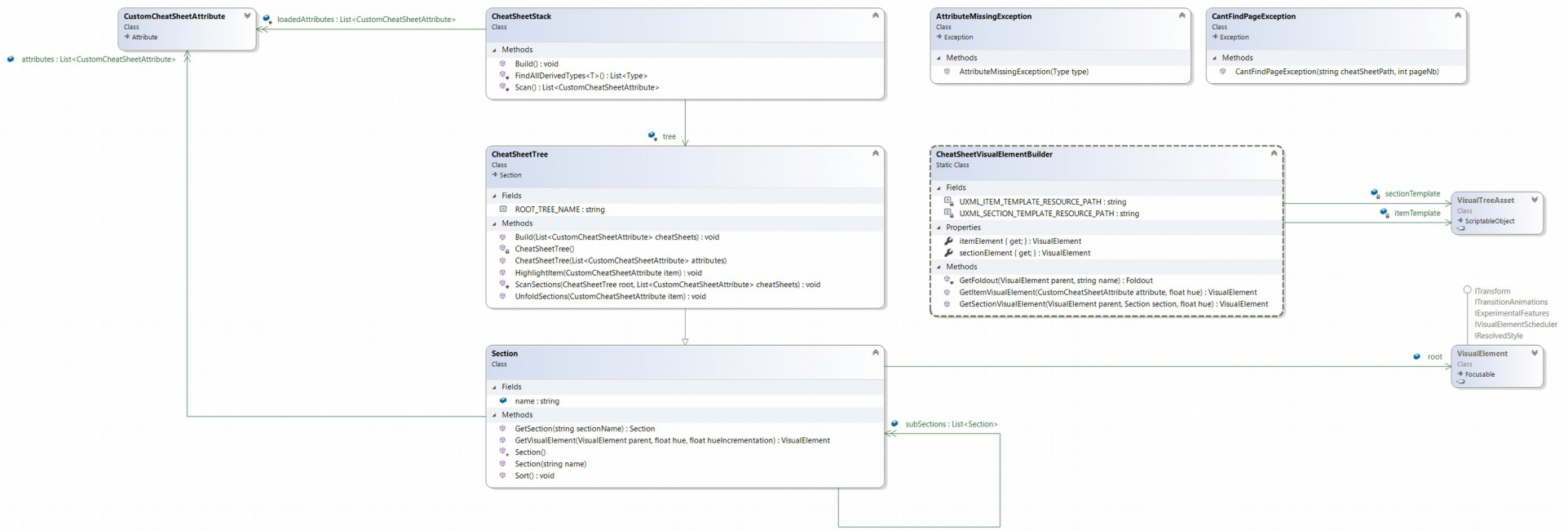
Constants	
PAGE_NUMBER_SEPARATOR : char	→ Public
SECTION_SEPARATOR : char	→ Public
Fields	
cheatSheetPath : string	→ Public The path in the *CheatSheetTree*. **Doesn't contain the page number and page name!**
element : CheatSheetElement	→ Public
keywords : string[ ]	→ Public Should describe this cheat sheet or explicitly say the conserned topics.
nextPage : CustomCheatSheetAttribute	→ Public

pageName : string	→ Public Is used when the *pageNumber* is superior than 0.
pageNumber : int	→ Public Will be shown in the *cheatSheetName* if *showPageNumber* is enabled. *The sorting system also uses this variable.*
previousPage : CustomCheatSheetAttribute	→ Public
resourcePath : string	→ Public
showPageNumber : bool	→ Public Will show the page number on the right of the *Button*. *True by default.*
Properties	
cheatSheetName : string	→ Public
sections : string[ ]	→ Public
splitedPath : string[ ]	→ Public

Constructors	
[no parameter]	→ Public
cheatSheetPath : string sheetRelativePath : string	→ Public cheatSheetPath: Can contain the character '/' for sections like "A/B/C" and etc. Must contain the name of the cheat sheet at the end. sheetRelativePath: Must contain the name of the UXML file, excluding its extension.
Methods	
ContainsKeyword keyword : string : bool	→ Public Will look at *cheatSheetName* and *keywords* for any string containing the *keyword*.
GetFirstPage : CustomCheatSheetAttribute	→ Public
GetLastPage : CustomCheatSheetAttribute	→ Public
GetLastPageNumber : int	→ Public
GetNextPage : CustomCheatSheetAttribute	→ Public
GetPage pageNumber : int : CustomCheatSheetAttribute	→ Public
GetPreviousPage : CustomCheatSheetAttribute	→ Public

HasNextPage : bool	→ Public
HasPageNumber : bool	→ Public
HasPreviousPage : bool	→ Public

2.2. CheatSheets.Stack



2.1. CheatSheetStack

→ Internal

→ Flagged System.Serializable

Class to find a specific *DiagramObjectEditor* for a *DiagramObject*.

Inspired from this source: https://stackoverflow.com/questions/2362580/discovering-derived-types-using-reflection

Also inspired from Conceptiode's reflector, one of my plugins ;)

Fields	
loadedAttributes : List<CustomCheatSheetAttribute>	→ Public List of *CustomCheatSheetAttribute* for fast search.
tree : CheatSheetTree	→ Public Stores any known *CheatSheetElement* with their sections.
Methods	
Build : void	→ Public Will rebuild the *tree* and rescan for every *CustomCheatSheetAttribute*.
FindAllDerivedTypes<T> : List<Type>	→ Internal
Scan : void	→ Internal

2.2. CheatSheetVisualElementBuilder

→ Internal

→ Static class

Constraints	
UXML_ITEM_TEMPLATE_RESOURCE_PATH : string	
UXML_SECTION_TEMPLATE_RESOURCE_PATH : string	
Fields	
itemTemplate : VisualTreeAsset	→ Public
sectionTemplate : VisualTreeAsset	→ Public
Properties	
itemElement : VisualElement	→ Public
sectionElement : VisualElement	→ Public
Methods	
GetFoldout parent : VisualElement name : string : Foldout	→ Internal

GetItemVisualElement attribute : CustomCheatSheetAttribute hue : float : VisualElement	→ Public
GetSectionVisualElement attribute : CustomCheatSheetAttribute section : Section hue : float : VisualElement	→ Public

2.3. CheatSheetTree

→ Internal

→ Extends Section

Constants	
ROOT_TREE_NAME : string	→ Public
Constructors	
[no parameter]	→ Private
attributes : List<CustomCheatSheetAttribute>	→ Public
Methods	
Build cheatSheets : List<CustomCheatSheetAttribute> : float	→ Public
HighlightItem item: CustomCheatSheetAttribute : bool	→ Public Will highlight the choosen *item*. *Can check if item is null.*
ScanSections root : CheatSheetTree cheatSheets : List<CustomCheatSheetAttribute> : void	→ Internal

UnfoldSections item: CustomCheatSheetAttribute : void	→ Public Will unfold every section specified in *item*. *Can check if item is null.*

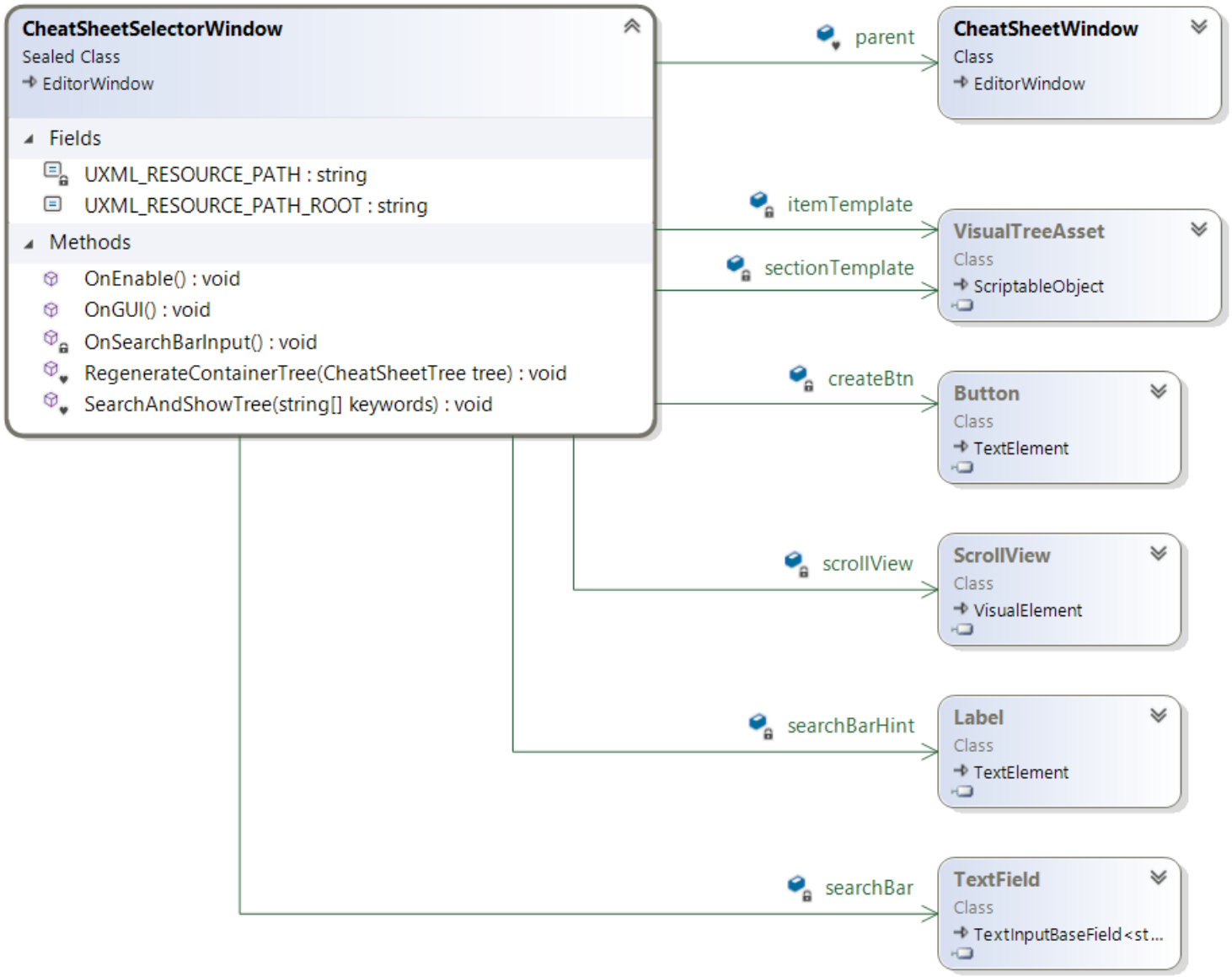
2.2.4. Section

→ Internal

Fields	
attributes : List<CustomCheatSheetAttribute>	→ Public
name : string	→ Public
root : VisualElement	→ Public
subSections : List<Section>	→ Public
Constructors	
[no parameter]	→ Protected
name : string	→ Public
Methods	
GetSection sectionName : string : Section	→ Public

GetVisualElement parent : VisualElement hue : float hueIncrementation : float : VisualElement	→ Public Recursive.
Sort : void	→ Public Will sort this section in alphabetical order.

3. CheatSheets.Selector



3.1. CheatSheetSelectorWindow

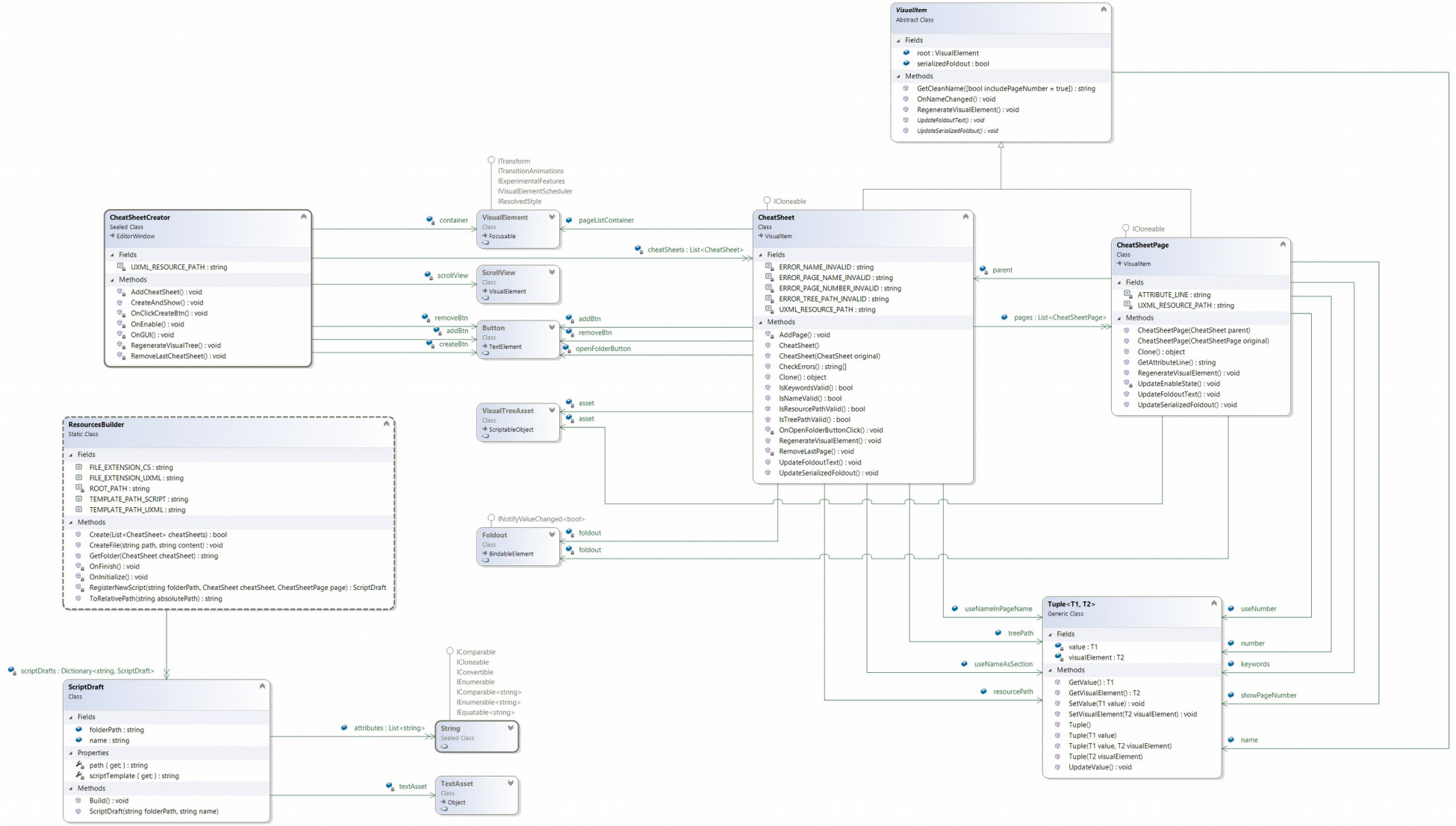
→ Internal, sealed

→ Extends UnityEditor.EditorWindow

Constants	
UXML_RESOURCE_PATH : string	→ Public
UXML_RESOURCE_PATH_ROOT : string	
Fields	
createBtn : Button	
itemTemplate : VisualTreeAsset	
parent : CheatSheetWindow	→ Internal
scrollView : ScrollView	
searchBar : TextField	
searchBarHint : Label	
sectionTemplate : VisualTreeAsset	

Methods	
OnEnable : void	→ Public Override.
OnGUI : void	→ Public Override.
OnSearchBarInput : void	
RegenerateContainerTree tree : CheatSheetTree : void	→ Internal
SearchAndShowTree keywords : string[ ] : void	→ Internal

4. CheatSheets.Creator



4.1. CheatSheetCreator

→ Internal, sealed

→ Extends UnityEditor.EditorWindow

Constants	
UXML_RESOURCE_PATH : string	
Fields	
addBtn : Button	
cheatSheets : List<CheatSheet>	
container : VisualElement	
createBtn : Button	
removeBtn : Button	
scrollView : ScrollView	

Methods	
AddCheatSheet : void	
CreateAndShow : void	→ Public
OnClickCreateBtn : void	
OnEnable : void	*Override.*
OnGUI : void	*Override.*
RegenerateVisualTree : void	
RemoveLastCheatSheet : void	

4.2. VisualItem

→ Internal

→ Abstract class

Fields	
root : VisualElement	→ Public
serializedFoldout : bool	→ Public
Methods	
GetCleanName includePageNumber : bool : string	→ Public
OnNameChanged : void	→ Public
RegenerateVisualElement : CheatSheet	→ Public
UpdateFoldoutText : bool	→ Public
UpdateSerializedFoldout : bool	→ Public

4.3. CheatSheet

→ Internal

→ Extends VisualItem, implements System.ICloneable

→ Flagged System.Serializable

Constants	
ERROR_NAME_INVALID : string	
ERROR_PAGE_NAME_INVALID : string	
ERROR_PAGE_NUMBER_INVALID : string	
ERROR_TREE_PATH_INVALID : string	
UXML_RESOURCE_PATH : string	
Fields	
addBtn : Button	
asset : VisualTreeAsset	
foldout : Foldout	
openFolderBtn : Button	

pageListContainer : VisualElement	
pages : List<CheatSheetPage>	→ Public
removeBtn : Button	
resourcePath : Tuple<string, TextField>	→ Public
treePath : Tuple<string, TextField>	→ Public
useNameAsSection : Tuple<bool, Toggle>	→ Public
useNameInPage : Tuple<bool, Toggle>	→ Public
Constructors	
[no parameter]	→ Public
original : CheatSheet	→ Public
Methods	
AddPage : void	
CheckErrors : string[ ]	→ Public
Clone : CheatSheet	→ Public *Override.*
IsKeywordsValid : bool	→ Public

IsNameValid : bool	→ Public
IsResourcePathValid : bool	→ Public
IsTreePathValid : bool	→ Public
OnOpenFolderButtonClick : void	
RegenerateVisualElement : void	→ Public *Override.*
RemoveLastPage : void	
UpdateFoldoutText : void	→ Public *Override.*
UpdateSerializedFoldout : void	→ Public *Override.*

4.4. CheatSheetPage

→ Internal

→ Extends VisualItem, implements System.ICloneable

→ Flagged System.Serializable

Constants	
ATTRIBUTE_LINE : string	Will use the two parameters constructor of *CustomCheatSheetAttribute*. The first represents its tree path with the page number if wanted. The second one represents the resource path. The third are extra options such as showing the page number and other fields.
UXML_RESOURCE_PATH : string	
Fields	
asset : VisualTreeAsset	
foldout : Foldout	
keywords : Tuple<string, TextField>	→ Public
number : Tuple<int, BaseField<int>>	→ Public
parent : CheatSheet	

showPageNumber : Tuple<bool, Toggle>	→ Public
useNumber : Tuple<bool, Toggle>	→ Public
Constructors	
parent : CheatSheet	→ Public
original : CheatSheetPage	→ Public
Methods	
Clone : CheatSheet	→ Public *Override.*
GetAttributeLine : string	
RegenerateVisualElement : void	→ Public *Override.*
UpdateEnableState : void	
UpdateFoldoutText : void	→ Public *Override.*
UpdateSerializedFoldout : void	→ Public *Override.*

4.5. Tuple<T1, T2>

→ Where T2 : BaseField<T1>

→ Internal

Fields	
value : T1	When no *visualElement* is set, this field can set the other one later.
visualElement : T2	
Constructors	
[no parameter]	→ Public
value : T1	→ Public
value : T1 visualElement : T2	→ Public
visualElement : T2	→ Public
Methods	
GetValue : T1	→ Public
GetVisualElement : T2	→ Public

SetValue value : T1 : void	→ Public
SetVisualElement visualElement : T2 : void	→ Public
UpdateValue : void	→ Public

4.6. ResourcesBuilder

→ Internal

→ Static class

Constants	
FILE_EXTENSION_CS : string	→ Public
FILE_EXTENSION_UXML : string	→ Public
ROOT_PATH : string	
TEMPLATE_PATH_SCRIPT : string	→ Public As text file.
TEMPLATE_PATH_UXML : string	→ Public *As text file.*
Fields	
scriptDrafts : Dictionary<string, ScriptDraft>	
Methods	
Create cheatSheets : List<CheatSheet> : bool	→ Public **Will always perform errors checks for protecting stored files!** Returns true if the operation was successfull.

CreateFile path : string content : string : string	→ Public
GetFolder cheatSheet : CheatSheet : string	→ Public Will get the folder path with the cheat sheet name. Will create a folder if can't be found.
OnFinish : void	
OnInitialize : void	
RegisterNewScript folderPath : string cheatSheet : CheatSheet page : CheatSheetPage : ScriptDraft	
ToRelativePath absolutePath : string : string	→ Public

4.7. ScriptDraft

→ Internal

Fields	
attributes : List<string>	→ Public **Must have no whitespaces.**
folderPath : string	→ Public Path of the folder.
name : string	→ Public
textAsset : TextAsset	
Properties	
path : string	Path of the file.
scriptTemplate : string	
Constructor	
folderPath : string name : string	→ Public

Methods	
Build cheatSheets : List<CheatSheet> : bool	→ Public

Conclusion

The main goal of this plugin is to allow users to access information freely and fast. Don't hesitate to share your cheat sheets on the Internet! You may need to use the GitHub link of this plugin so your readers can read it.

This plugin is published under the Creative Commons Zero v1.0 Universal licence. You are allowed to share it or sell your cheat sheets with your plugins with any modifications, even without my authorization or credits!

Thank you very much for spending your time reading the documentation! Please keep in mind that it was created by one person. Feel free to give your opinion by talking about this project in social medias or by sending an e-mail!

Legal Notice

Cheat Sheets is made possible by Louis-Pierre Aubert (www.louis8257.com).

Contact

For any information, please contact me by e-mail at louis.8257@hotmail.com. You will receive an answer under 2 business days. Feel free to give your feedback!

Credits

Fonts Used

The Roboto font is designed by Christian Robertson. Not modified. Used in the documentation, not in the software.

Softwares Used

Screenshots taken from the Unity Editor 2019.4.8.f1 (unity.com).

Documentation wroten with LibreOffice 6.2.3.2 (libreoffice.org).

Illustrations made with Blender 2.78 (blender.org).

Diagram classes and code made with Microsoft's Visual Studio Community 2019 (visualstudio.microsoft.com).