HTML ELEMENT

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/html-element

The **HTML Element** plugin allows you to include custom HTML content in your project. The HTML content can also be styled with CSS to alter its appearance. This is a good way to display content with more complex layout, such as tables, grids, and menus.

Construct creates a single HTML Element for each instance of the object, and sizes and positions the element to match the object's size and position in the project. Beyond that the content is entirely determined by the HTML you provide.

This guide will not teach how HTML and CSS work. However there is lots of information on the web about them. For example take a look at the HTML basics article on the MDN Web Docs.

You can find several examples of using the HTML Element object in the Example Browser. Add the HTML Element plugin as a filter and the list will display projects using it.

Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IHTMLElementInstance script interface. This allows access in worker mode; when worker mode is disabled, you can also use all the browser standard DOM APIs.

Layering HTML objects

As the name suggests, this object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

CSS styling

If you only need a small number of custom CSS styles for HTML Element, these can be entered in to the *Style attribute* property. However it's often more convenient to use a CSS file to write styles in a more readable and easily editable way.

CSS styles can be added to the project by adding a stylesheet to the *Files* folder in the Project Bar. See Project files for more details.

The HTML Element object has *ID* and *Class* properties. These can be filled in and then CSS styles added for the corresponding ID or class, which will allow altering the appearance of the main HTML element created by Construct. For example if the class is entered as *myelement*, then CSS styles can be added for it with the rule `.myelement { ... }`.

Note that Construct sets the following styles on the HTML element via its style attribute. It is best not to override these as they may cause the element to work incorrectly.

- `position`
- `left`, `top`, `width` and `height`
- `box-sizing`
- `display` if made invisible
- `color` and `background-color` if *Set color* and *Set background color* are enabled
- `font-size` if *Auto font size* is enabled
- `user-select` if *Allow text selection* is disabled

One useful technique is to put a wrapper element inside the HTML element that is sized to fill its container. Then any custom styles or appearances can be used on that wrapper element without accidentally affecting the element created by Construct. This would mean the HTML content is something like this:

```
<div class="wrapper">
        <!-- Content goes here -->
</div>
```

This can then be styled to fill its container. There are several ways to do this in CSS; here is one approach using absolute positioning.

```
.wrapper {
        position: absolute;
        left: 0;
        top: 0;
        right: 0;
        bottom: 0;
}
```

Now further styles can be added to the wrapper, such as `font-size`, without conflicting with the styles added by Construct.

Scaling with Construct's canvas

Construct sets a CSS variable named `--construct-scale` on the root `html` element of the document, with a number representing the canvas scale as a multiplier. You can use this with `calc()` to scale CSS properties to match the displayed canvas size, e.g.:

```
.mybutton {
        height: calc(var(--construct-scale) * 2em);
}
```

This sets `.mybutton` to have a height of 2em at 100% scale, but also adjusts the height to follow Construct's fullscreen scaling.

Managing long HTML content

If you have a lot of HTML content to display, consider adding it as a HTML project file. Then use the AJAX object to fetch the project file, and set the content of the HTML Element to the content of the project file.

Scaling with the canvas

By default *Auto font size* is enabled. This means Construct sets a `font-size` style on the main HTML element to an `em` size that scales with the displayed canvas size. For example at 100% scale it will set `font-size: 1em`. Then if the window is resized to display twice as large, it will set `font-size: 2em`.

This is useful for scaling your HTML content with the displayed size of the game. The key is to **use `em` units for sizing**. Then everything will automatically scale with the canvas.

For example if you set `border-radius: 20px`, that will use a fixed 20px size regardless of how large or small the window size is, meaning it will look wrong at some sizes. However if you set `border-radius: 2em`, it will scale the size to match the canvas size, ensuring it looks consistent at all sizes.

If you want to change the `font-size` for your HTML content, note that Construct sets that property already. The best approach is to use a wrapper element as described above, and change the `font-size` of the wrapper instead.

Using BBCode

HTML Element also sets supporting its content using simple formatting tags known as "BBCode". This involves using square bracket tags such as [b] and [/b] around text to make bold, e.g. `[b]bold text[/b]`. The tags you can use for BBCode in HTML Element are listed below.

> *Note these BBCode tags are different to those supported by the Text or SpriteFont objects.*

Tag	Description
[b]...[/b]	**Bold text**
[i]...[/i]	*Italic text*
[s]...[/s]	~~Strikethrough text~~
[u]...[/u]	Underline text
[sub]...[/sub]	Subscript text
[sup]...[/sup]	Superscript text
[small]...[/small]	Smaller text
[mark]...[/mark]	Mark text with highlight
[code]...[/code]	Format as code snippet
[h1]...[/h1]	Header 1
[h2]...[/h2]	Header 2
[h3]...[/h3]	Header 3
[h4]...[/h4]	Header 4
[item]	Item bullet point: •

Security

Be careful when adding user-provided values in to HTML. Read this section carefully if you do this.

By default the HTML Element object interprets content as HTML. This is safe with content you've written yourself. However it's common to insert user-provided values, such as the user's name, in to HTML. **This is a potential security risk** and must be handled carefully.

Inserting a string like `"<p>Your name is " & PlayerName & "!</p>"` is dangerous, because the player could enter HTML content for their name - which can even include script tags. For example the player could enter their name as `<script>runDangerousCode()</script>`. Then in the previous example the inserted HTML content will be `<p>Your name is <script>runDangerousCode()</script>!</p>` which will execute the given JavaScript code, which may be malicious.

The solution is to make sure all user-provided values are *escaped*. This makes sure that HTML tags are displayed as text, rather than interpreted as HTML, replacing characters like `<` and `>` with `<` and `>`. This is done with the *EscapeHTML* expression. For example the following expression is now safe as it escapes the player name:

```
"<p>Your name is " & HTMLElement.EscapeHTML(PlayerName) & "!</p>"
```

Another alternative is to use BBCode or plain text as the content type. In both cases it is impossible for users to insert unexpected HTML tags, so they are fundamentally safe.

HTML Element properties

Tag

The name of the HTML tag that Construct creates. By default this is `div` for a `<div>` element, but it can be changed to any other HTML element.

Content

The text to use as the initial content of the HTML Element. The way this is interpreted is set by the *Content type* property, which defaults to HTML. If you have a lot of content consider using a project file instead - see *Managing long HTML content* above.

Content type

Choose how the initial content text is interpreted. The default is *HTML*. It can also be set to *BBCode* (see the section on *Using BBCode* above), or *Plain text* to avoid interpreting any tags on the text at all.

Initially visible

Set whether the HTML Element is visible upon creation. The element is made invisible by setting the style `display: none`.

ID

The ID to set for the HTML Element.

Class

The class to set for the HTML Element. Like with the HTML *class* attribute, multiple classes can be specified separated by spaces.

Allow context menu

Whether the browser's default context menu should be allowed on the HTML Element. Typically this should be allowed for input elements, but often disallowing it is appropriate for games, which may use controls like right-clicking for other purposes. If disallowed, Construct will call `preventDefault()` on the `"contextmenu"` event.

Stop input events

Construct sometimes stops input events on HTML elements from reaching the rest of the project. For example using arrow keys to move the caret inside a text input should not also move the player if they are controlled by arrow keys. This also applies to mouse or touch input. This is optional with the HTML Element object and has three modes:

- **No** - no input events are stopped. All mouse, touch or keyboard input on the element will reach the rest of the project.
- **Child elements only** - input events reaching child elements of the main HTML element will be stopped from reaching the rest of the project. However the main HTML element will not stop any input.
- **Entire element** - all input events on the entire HTML element will be stopped from reaching the rest of the project.

Origin

Choose the position of the origin of the object relative to its unrotated bounding rectangle.

Set color

Default color

Enable *Set color* to set the `color` CSS style on the HTML element to the given color. This changes the text color. Usually this is desirable as otherwise the text color may not be visible

against the background.

Set background color

Default background color

Enable *Set background color* to set the `background-color` style on the HTML element to the given color.

Auto font size

Automatically set a `font-size` CSS style on the HTML element according to the display scale of the canvas. This provides a convenient way to size HTML content with the canvas by using `em` units. See the section *Scaling with the canvas* above for more details.

Allow text selection

Set whether dragging over text is allowed to create a selection. By default this is disabled, which adds the CSS style `user-select: none`.

Style attribute

Additional CSS styles for the main HTML element can be added here, separated by semicolons. This will be set via the `style` attribute of the HTML element. Consider using CSS files if there are more than a couple of simple styles - see the section *CSS styling* above for more details.

HTML Element conditions

On clicked

Triggered when any part of the HTML element is clicked. The *TargetID* and *TargetClass* expressions are set to the ID and class of the clicked element.

On clicked class

Triggered when an element with a given list of CSS classes is clicked. This includes a click on any child elements contained by the element with the given CSS classes. A single class name can be specified, or multiple classes given separated by spaces, in which case the clicked element must match all the given classes.

> *This does not use a CSS selector, so don't prefix class names with a dot.*

On clicked ID

Triggered when an element with a given ID is clicked. This includes a click on any child elements contained by the element with the given ID.

> *This does not use a CSS selector, so don't prefix the ID with* `#`.

On CSS animation ended

Triggered when a CSS animation for any contained element finishes (i.e. the `"animationend"` event). The animation name is specified by the `@keyframes` rule. The *TargetID* and *TargetClass* expressions are set to the ID and class of the element that finished a CSS animation. This trigger is useful for things like destroying the HTML element after a fade out CSS animation has finished.

HTML Element actions

> *Note that some actions have the same name. Actions in the HTML element group are common to all HTML-based plugins like Button and Text Input, and will only affect the main HTML element. Actions in the HTML content group are unique to the HTML Element plugin and can update the content of the object.*

Create sprite image element

Creates an `<img>` element with the content of a given Sprite object's current image, and inserts it to the HTML element. The location to insert is set by a CSS selector. This can be left blank to insert in to the main HTML element. Alternatively a selector like `".myclass"` will mean to insert content to the child element with the class *myclass*. The element can be set to inserted at the start or the end of the given element, or alternatively replacing all the content of the given element with the image. The inserted image element can optionally also be given an ID and class, which helps make it easy to style the inserted image with CSS.

> *This action provides a simple way to show a Sprite image on top of a HTML element, since normally HTML elements always show on top of Sprites.*

Insert content

Insert the given string of content inside the HTML element. The string is interpreted according to the content type (HTML, BBCode or plain text). The position indicates whether to insert at the start or the end of the given location. The location to insert is set by a CSS selector. This can be left blank to insert in to the main HTML element. Alternatively a selector like `".myclass"` will mean to insert content to the child element with the class *myclass*; if the *Type* is set to *all*, it will insert the same content to all elements matching the selector.

Be careful when inserting user-provided values as HTML. See the section on Security above.

Position object at element

Sets the position and size of a given object to match the position and size of a specific HTML element. The element is chosen by a CSS selector, e.g. `".myclass"` to position an object at an element with the class *myclass*.

This action provides a way to use invisible HTML and CSS for complex layouts, while displaying the actual content with other objects, allowing for full use of Z order, effects and so on.

Remove content

Either removes elements entirely, or clears their content to make them empty, according to the given mode. The content to remove or clear is given by a CSS selector. This can be left blank to clear the main HTML element. (Note the main HTML element cannot be removed, so in this case it will be cleared instead.) Alternatively a selector like `".myclass"` will mean an element with the class *myclass* will be removed or cleared; if the *Type* is set to *all*, this will remove or clear all elements matching the selector.

Set attribute

Either sets or removes a named attribute on a HTML element, according to the given mode. The value is ignored if removing. The HTML element to alter attributes for is given by a CSS selector. This can be left blank to alter attributes for the main HTML element. Alternatively a selector like `".myclass"` will update attributes for an element with the class *myclass*; if the *Type* is set to *all*, this will update attributes for all elements matching the selector.

Set class

Either adds, toggles or removes classes on a HTML element, according to the given mode. Multiple classes can be updated at once by separating their names with spaces. The HTML element to alter classes for is given by a CSS selector. This can be left blank to alter classes for the main HTML element. Alternatively a selector like `".myclass"` will update the classes for an element with the class *myclass*; if the *Type* is set to *all*, this will update classes for all elements matching the selector.

Note that the Class parameter is not a selector so does not need class names to be prefixed with a dot; however the Selector parameter is a CSS selector and so needs class names to be prefixed with a dot.

Set content

Replaces some content inside the HTML element with the given string. The string is interpreted according to the content type (HTML, BBCode or plain text). The location to replace content is set by a CSS selector. This can be left blank to replace the content of the entire main HTML element. Alternatively a selector like `".myclass"` will mean to replace the content of the child element with the class *myclass*; if the *Type* is set to *all*, it will replace the content of all elements matching the selector.

> *Be careful when inserting user-provided values as HTML. See the section on Security above.*

Set CSS style

Set a single CSS style on the style attribute inside the HTML element, based on a CSS property name and a string for its value. Setting the value to an empty string will remove the property from the style attribute. The element to change the style for is set by a CSS selector. This can be left blank to update the style attribute of the main HTML element. Alternatively a selector like `".myclass"` will mean to update the CSS style of the child element with the class *myclass*; if the *Type* is set to *all*, it will update the style of all elements matching the selector.

Set scroll position

Set the horizontal or vertical scroll position of an element. The HTML element to scroll is given by a CSS selector. This can be left blank to scroll the main HTML element, or use a CSS selector string like `".myclass"` to scroll an element with the class *myclass*. This action only scrolls one element matching the selector. The *Direction* specifies whether to set the scroll top (vertical) or left (horizontal) position, and the *Position* value is the scroll position to set in CSS pixels.

HTML element expressions

EscapeHTML(string)

Escapes any characters with meaning in HTML from the given string, such as replacing `<` with `<`. This is important for securely inserting user content in to HTML strings. See the *Security* section above for more details.

HTMLContent

TextContent

A string with the complete content of the HTML element, either as a full HTML string, or plain text only (with HTML tags removed).

Note that this does not update immediately after actions that change the HTML content - use the Wait for previous actions to complete system action to ensure any modifying actions have completed before reading updated content from these expressions.

TargetClass

TargetID

In a trigger like *On clicked*, these contain the class and ID of the affected HTML element. Note that *TargetClass* can be a space-separated list of multiple classes, as per the *class* HTML attribute.