TEXT

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

The **Text** object can display text using a font in your project. Note that there are not many built-in fonts common to all computers. Instead you can import **web fonts** for use with the Text object.

Note the Text object is used for displaying text only. Don't confuse it with the Text input object, which is a form control used for entering text in to.

Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the ITextInstance script interface.

Using web fonts

Follow these steps to use a custom web font in the Text object.

1. Locate a web font to use, in WOFF or WOFF2 format. There are some web services that list web fonts. (Be sure to check the web font license to ensure you use it correctly.)

   Be sure to use WOFF or WOFF2 format fonts, as these are the only formats that work consistently across all platforms. If you use other formats like TTF or OTF, they may appear to work, but then stop working when you export your project to a different platform.

2. Download the **.woff** or **.woff2** file for the web font.
3. In the Project Bar, **right-click** the **Fonts** folder and select Import fonts.
4. Import the .woff or .woff2 file you downloaded previously. This will add the web font file as a project file.
5. Select a Text object in the Layout View, and click the button next to the *Font* property in the Properties Bar.
6. In the Font Picker dialog, pick the web font you imported from the second drop-down list (under *Or pick a web font from this project*), and click OK.

The Text object will now be displaying the custom web font in the Layout View. Since the web font is bundled with your project, it will be available on any platform.

Note: the Free Edition of Construct is limited to only importing one web font.

Text rendering

The Text object does not display anything if its bounding rectangle is too small to fit a single letter of text. If text objects appear to go invisible, try resizing them larger.

Different browsers render text in different ways. This means you should expect the appearance of the Text object to vary slightly across browsers. You should test your project in a range of browsers to ensure text objects display how you intend for all users. For more information see Best practices.

Using BBCode

By default the Text object allows the use of BBCode, a simple way of marking up text for formatting like **bold** and *italic*. If you don't want such tags to affect the formatting of the text, you can opt-out of it by unchecking the *Enable BBCode* property.

BBCode uses "tags" in square brackets to mark the start and end of formatting. For example to make a word bold, wrap it in `[b]` and `[/b]`, e.g. `[b]Hello[/b]`. Some tags take a parameter, such as the font name to use, which is specified after an equals sign in the opening tag, e.g. `[font=Arial]Hello[/font]`.

The following tags are supported:

- `[b]bold text[/b]`
- `[i]italic text[/i]`
- `[u]underline text[/u]`
- `[s]strikethrough text[/s]`
- `[size=20]change font size (in pt)[/size]`
- `[font=Arial]change font face[/font]` - you can also use any web font imported to the project.
- `[color=#ff0000]change text color[/color]` - the color can be specified in the same way CSS colors are specified, e.g. hexadecimal, using `rgb()`, etc.
- `[opacity=50]change text opacity[/opacity]`
- `[hide]invisible text[/hide]` - this is useful for flashing effects, since the text still takes up the same width while invisible
- `[background=#ff0000]change background color[/background]`
- `[offsetx=10]offset X[/offsetx]` and `[offsety=10]offset Y[/offsety]` - move text by a number of pixels on each axis, useful for animated effects. The offset can optionally also be specified as a percentage of the line height, e.g. `[offsety=50%]` means offset downwards by half the line height.
- `[stroke]stroke text[/stroke]`, drawing an outline rather than a solid fill
- `[outline=#ff0000]outlined text (on top)[/outline]`, which adds an outline with a different color drawn on top of the text (as opposed to stroke, which removes the fill)

- `[outlineback=#ff0000]outlined text (underneath)[/outline]`, which adds an outline with a different color drawn behind the text. This variant can look better with very thick outlines.
- `[lineThickness=2]change line thickness[/lineThickness]`, affecting the line thickness used for stroke, outline, strikethrough and underline
- `[icon=0]` or `[icon=tag]` - insert an icon to the text. The icons are taken from the animation frames of a Sprite object set in the *Icon set* property. The icon can be referred to by its zero-based frame index, or by the *Tag* property of the animation frame.
- `[iconoffsety=50%]icon offset Y[/iconoffsety]` - change the vertical alignment of any icons between the tags. A percentage uses an amount relative to the line height. This is relative to the alphabetic baseline, so a value of 0 will align the bottom of the icon with the bottom of the character 'x'. The default icon offset Y is 20% so icons are normally aligned slightly below the alphabetic baseline.
- `[tag=mytag]tag a range of text[/tag]`, assigns the tag "mytag" to a range of text, which can then be referred to in events (e.g. the *Has tag at position* condition, or expressions to get its size and position). Note see the section *Tagged range fragmentation* below for more details.
- `[insert]inserted text[/insert]` does not change the text style, but inserts the given text separately to the rest of the string. This is useful when inserting right-to-left (RTL) text in to left-to-right (LTR) strings: normally RTL text inside an LTR string may change the direction of other text in the string, but if you put the RTL text inside these tags, it will ensure it does not change the direction of any text outside of the tags. It will also prevent other text merging features, such as kerning and ligatures, across the tag boundary.

See the Text formatting example for a demonstration of the various formatting tags, and the Icons in text example for a demonstration of using icon tags.

Tagged range fragmentation

This section also applies to the Sprite Font object.

As noted above, certain ranges of text can be tagged, e.g.:

```
The [tag=mytag]quick brown fox[/tag] jumps over the lazy dog
```

This will assign the tag "mytag" to the words "quick brown fox". This works straightforwardly with the *Has tag at position* condition and *TagAtPosition* expression. However when accessing the size and position of the tag with the *TagCount*, *TagX* etc. expressions, the tagged range can in fact be split in to multiple *fragments*. There are two reasons this can happen:

1. Word wrap - if the tagged range is broken across two or more lines
2. Changing styles within the tagged range

In the normal case you may expect the above tagged range to count as a single tag with a single size and position. However it may be broken across lines by word wrap, such as shown below:

```
The quick brown
fox jumps over
the lazy dog
```

Notice how the tagged range is now in two separate places across two lines: the first line having "quick brown", and the second line having "fox". Each of the two parts is referred to as a *fragment*. There are now two positions and two sizes for the two fragments that the tagged range was broken in to. Therefore *TagCount("mytag")* will return 2. The other expressions to retrieve the size and position of the tagged range will return each of the fragments separately, identified by the index parameter. Note that long tagged ranges could be broken across three or even more lines, further increasing the number of fragments.

Changing the style of text also fragments the text. For example consider the following BBcode: `The [tag=mytag]quick [b]brown[/b] fox[/tag] jumps over the lazy dog`

Now inside the tagged range, just the word "brown" is made bold. This also fragments the tagged range, this time in to three fragments: the first with "quick " (including a space), the second "brown" in bold, and the third " fox" (including a space). This time *TagCount("mytag")* will return 3 and each fragment's size and position will be returned separately based on the index parameter.

Text properties

Text

The text for the object to initially be showing.

Enable BBCode

Whether to enable the use of BBCode formatting in the text. See above for a list of allowed tags. If disabled, any BBCode tags will simply be displayed as plain text.

Font

The font the text object uses to display its text. Click the button to the right of the font name to open a font picker dialog.

> *You may see a permission prompt to access the full list of fonts installed on your system.*

Remember local fonts may not be available on other devices - consider using a web font, as described in *Using web fonts* above.

Size

The size of the text to display, in points (pt).

Line height

Amount to change the space between each line of text, in pixels. Use 0 for the default amount, -5 for 5 pixels shorter than default, 10 for 10 pixels taller than default, and so on.

Bold

Whether to use the **bold** variant of the font, if available.

Italic

Whether to use the *italic* variant of the font, if available.

Color

Choose the color of the text object's text.

Horizontal alignment

Choose whether the text displays left, center or right aligned within its bounding rectangle.

Vertical alignment

Choose whether the text displays top, center or bottom aligned within its bounding rectangle.

Wrapping

Choose how text wraps at the end of a line. *Word* will wrap entire words separated by spaces or hyphens. *Character* will wrap to the next line on any character, even punctuation. *CJK* works similarly to *Character*, but has special handling for Chinese, Japanese and Korean punctuation characters, intended to ensure punctuation wraps appropriately.

Text direction

Set the direction of the text flow. The default is left-to-right (LTR). For some languages the right-to-left (RTL) direction is more appropriate, such as Arabic and Hebrew. The text direction is particularly significant when using BBcode formatting, as it affects the order of formatted fragments.

Icon set

Choose a Sprite object to use for BBcode icon tags. Each animation frame of the Sprite object is treated as an individual icon. Icons can then be referred to by the animation frame index, or the animation frame *Tag* property. See the section *Using BBcode* above for more details.

Initially visibile

Whether or not the object is shown (visible) or hidden (invisible) when the layout starts.

Origin

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

Read aloud

Check to indicate to screen readers that the contents of this text object ought to be read aloud when it appears or changes. By default text objects are not read out by screen readers as they are rendered in to a canvas, which is essentially a large image and so not accessible to screen readers. Further, Text objects are not all automatically read aloud as this can provide a poor screen reader experience, such as constantly reading out a changing score instead of more helpful information. Checking this option for important text objects improves the accessibility of projects to ensure the contents of text objects can be understood by people who cannot necessarily read them visually. The text object does not need to be on-screen, so a dedicated text object for screen readers with this option checked can also be used. See also the Speech Synthesis plugin which can be used for similar purposes.

Text conditions

For conditions common to other objects, see common conditions.

Compare text

Test whether the text object is currently displaying a certain string of text. The comparison can be either case sensitive ("TEXT" is different to "text") or case insensitive ("TEXT" is considered the same as "text"). To test if the text object is not showing some text, invert the condition.

Has tag at position

Test if there is text with a specific tag at the given position (case insensitive). For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then testing if the tag "mytag" is at a given position will check if that position is over just the part of the text that says "world".

Is running typewriter text

True while text is being written out using the *Typewriter text* action.

On typewriter text finished

Triggered when text being written out using the *Typewriter text* action finishes writing out all the text.

Text actions

For actions common to other objects, see common actions.

Set font color

Set the color of the text. Use an expression in the form `rgb(red, green, blue)`.

Set font face

Change the font used to display the text. This must be the name of a web font imported to the project, or a local font that is pre-installed on the user's device.

Set font size

Set the size of the text in points (pt).

Set horizontal alignment

Set vertical alignment

Set line height

Set read aloud

Set text direction

Set wrapping

Change the corresponding properties. See *Text properties* above for more information.

Set resolution mode

By default Text objects use *Automatic* resolution mode, which means the resolution of the text adjusts according to the 2D display scale. This produces the best quality display, but only works with 2D display, and can also cause the text to constantly re-render when being smoothly scaled, which can sometimes have a significant performance overhead. *Fixed* resolution mode causes the text to render at a fixed resolution according to a provided scale factor, and ignore the display scale. This usually results in a reduced display quality but better performance.

Append text

Add some text to the end of the current text. For example, if the text object contains *Hello* and has *World* appended, the text object then contains *HelloWorld*.

Set text

Set the text the object is currently displaying. Use the **&** operator to combine text and numbers. For more information, see expressions.

Change icon set

Changes the *Icon set* property, replacing the Sprite used for BBcode icons. This can be used to change the set of icons displayed by the Text object. Note if the new Sprite object does not have the same number of animation frames, or the same animation frame tags, then some icons may disappear.

Typewriter text

Set the text over time by starting with an empty string and gradually adding characters until the full text is written out, over a duration specified in seconds. Once the full text is written out, *On typewriter text finished* triggers. Note using *Set text* or *Append text* while text is being written out will cancel the effect.

> *You can use a speed in characters per second instead of an overall time by using an expression like* `len(Self.PlainText) / 10` *for the time. In this case it will write out 10 characters per second regardless of the length of the string.*

Finish typewriter

If text is being written out with the *Typewriter text* action, force it to finish immediately.

Update HTML

This action converts the contents of the Text object, including any icons, in to a string of HTML. This can then be displayed in the HTML Element object. This action is *asynchronous*: it takes a moment complete, so you need to use it with the *Wait for previous actions to complete* system action before you can use the result in the *AsHTML* expression. See the Text icons to HTML example for a demonstration.

Text expressions

For expressions common to other objects, see common expressions.

FaceName

FaceSize

LineHeight

Return the corresponding object's properties. See *Text properties* above for more details.

Text

Return a string containing the object's current text.

PlainText

Return a string containing the object's current text, with any BBCode tags stripped out. For example if the text is `[b]Hello[/b]`, the *Text* expression will return that (with BBCode tags included), but the *PlainText* expression will return just `Hello`.

AsHTML

Returns a string of HTML code that represents the content of the Text object, including formatting and icons. This is only available after, and only updated by, the *Update HTML* action completing.

TagAtPosition(x, y)

Look up the tag for a part of the text at a given position and return the tag if any, else return an empty string if no tag is specified. For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then the tag at a position over the word "world" is "mytag", and the tag at a position over the word "Hello" is "" (an empty string).

TagCount(tag)

TagX(tag, index)

TagY(tag, index)

TagWidth(tag, index)

TagHeight(tag, index)

Identify the size and position of all ranges of the text with a given tag. Note the count and the index actually refers to *fragments*, as a single tagged range may be broken up in to multiple pieces - see the section *Tagged range fragmentation* above for more details.

TextWidth

TextHeight

Return the size of the actual text content within the text object's rectangle.