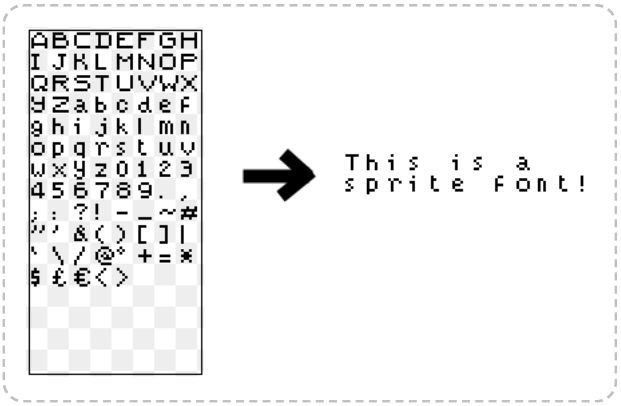
SPRITE FONT

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

The **Sprite Font** object uses an image to display text. The "sprite font" is the object image, which contains a grid of every character that can be drawn. By drawing parts of this image in sequence, strings of text can be rendered. This allows complete artistic control over the appearance of text, rather than having to choose from existing fonts.



Scripting

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

Comparison with Text object

Both the Sprite Font and Text objects can display text in the project. Typically the Text object shows monochrome characters from an existing font or web font, which can use a range of sizes and possibly also bold and italic options. On the other hand Sprite Font uses images for each character. While this means any kind of artwork can be used for text, notably allowing for multi-colored text, it has the trade-off that it only really supports one font size and one bold/italic setting (those that it is drawn with).

Another important difference is that traditional fonts as used by the Text object often have good support for unicode characters. This allows them to display a wide range of characters, including many alphabets and character sets from many different languages, as well as emoji. Sprite Fonts however can only use the characters for which an image has been drawn. It is very difficult to make a sprite font that covers much of the tens of thousands of possible unicode characters. If a Sprite Font is set to show some text which contains a character that has not been drawn, it will simply show an empty space for that character. If the entire string is in a different language which the sprite font does not cover, nothing will render at all. Bear in mind that if you allow user-inputted text, such as the player's name, or you wish to translate the project in future, then Text objects are probably more suitable.

Re-coloring SpriteFonts

SpriteFont objects have a *Color* property that can be used to conveniently re-color the text (and the `[color]` BBCode tag works similarly; see below). This works by applying a color filter, which works best if the SpriteFont is drawn with white text, since that can be filtered to any other color. For this reason the default SpriteFont is drawn white. The *Color* property is set to blue by default in order to help identify that the color of the SpriteFont is set by this property. So long as the image is drawn white, the color of the text can be set to any color with this property.

Using BBCode

By default the SpriteFont object allows the use of BBCode, a simple way of marking up text for formatting. 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 hide a word, wrap it in `[hide]` and `[/hide]`, e.g. `[hide]Hello[/hide]`. Some tags take a parameter, such as the scale, which is specified after an equals sign in the opening tag, e.g. `[scale=2]Hello[/scale]`.

The following tags are supported. Note that due to the fact SpriteFonts render images for text, the supported BBCode tags differ from those used by the Text object.

- `[scalex=2]stretch wider[/scalex]`
- `[scaley=2]stretch taller[/scaley]`
- `[scale=2]stretch both axes[/scale]`
- `[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. Note that for SpriteFont, the color is applied as a tint. To ensure you can use any color text, use a SpriteFont with the characters drawn in a white color.
- `[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
- `[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* in the Text object manual entry for more details (as fragmentation works the same for both SpriteFont and Text objects).

Sprite font properties

Text

The initial text to display.

Sprite font

Click the Edit link to edit the source image that text characters are rendered from. The image can be any size, but it should fit the characters it contains exactly. Characters start in the top-left and the sequence moves to the right, wrapping down to the next line when it reaches the right edge of the image. If the character is narrower than the cell, and you change its width using *Spacing data* or the *Set character width* action, the image should be drawn left-aligned in the cell.

Character width

Character height

The size of each character's cell in the sprite font image. Individual characters can be displayed with a different width using *Spacing data* or the *Set character width* action. In this case, the character should be drawn left-aligned within its cell.

Character set

A string of characters that describes the sequence of letters in the sprite font image. This is used to map text to images. While the default starts with the English alphabet, it could be changed to another language or sequence and the image updated accordingly. Note however the Sprite Font can only display characters that are in the character set; any characters not in the character set with a corresponding image will appear as an empty space.

Spacing data

Some data in JSON format that lists the widths of individual characters. This allows improved text layout by using narrower spaces for narrower characters. The spacing data also affects the display of the text in the Layout View. The data is an array of pairs. Each pair is a width, and then a string of all the characters that width applies to. For example the pair `[10, "aeou"]` will set the width of the characters *a*, *e*, *o* and *u* to 10 pixels. The characters are case-sensitive, allowing you to choose different widths for uppercase characters. You can also set the width of the space character. Each pair must be listed in an array, e.g. `[[10, "aeou"], [12, "mvw"]]`.

Scale

A multiplier to scale the rendered text with, such as 0.5 for half as big or 2 for twice as big. This can be used to "fake" different font sizes, but remember it's only stretching images; you may want to draw the font again at a different size instead of using a scale.

Character spacing

Extra space in pixels to add horizontally between characters.

Line height

Extra space in pixels to add vertically between lines. 0 is the default size, negative values make lines closer together, and positive values space lines out further apart.

Horizontal alignment

The horizontal alignment of the text within the object bounding rectangle.

Vertical alignment

The vertical alignment of the text within the object bounding box.

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.

Initially visible

Whether the object is initially visible or invisible when the layout starts.

Origin

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

Sprite font conditions

Compare text

Compare the current text the object is showing.

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.

Sprite font actions

Append text

Add some text to the end of the existing text.

Set character spacing

Set line height

Set scale

Set horizontal alignment

Set vertical alignment

Set wrapping

Set the corresponding object properties. For more information, see *Sprite font properties*.

Set character width

Set the width of certain characters. Normally it is preferable to use the *Spacing data* property, since it displays with correct spacing in the Layout View. In this action you can specify multiple characters at the same time to set their widths simultaneously, including the space character.

Set text

Replace the current text with a new string.

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.

Sprite font expressions

CharacterHeight

Return the sprite font cell height.

CharacterScale

CharacterSpacing

LineHeight

Return the corresponding object properties. For more information, see *Sprite font properties*.

CharacterWidth(char)

Return the width of a character. A character must be passed (as a string) so the *Spacing data* or *Set character width* action can be taken in to account. Since the expression can only return one value, if there are multiple characters in the string, only the first is used.

Text

Return 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`.

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* in the Text object manual entry for more details (as fragmentation works the same for both SpriteFont and Text objects).

TextWidth

TextHeight

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