Advanced Twine

HTML Markup

Custom Styles

Type	Syntax	Example	Rendered As
Highlight, Inline	`@@Text@@`	`@@Text@@`	`<span class="marked">Text</span>`
Custom Style, Inline	`@@`*style-list*[1]`;Text@@`	`@@#foo;.bar;Text@@`	`<span id="foo" class="bar">Text</span>`
		`@@color:red;Text@@`	`<span style="color:red">Text</span>`
Highlight, Block	`@@` `Text` `@@`	`@@` `Text` `@@`	`<div class="marked">Text</div>`
Custom Style, Block	`@@`*style-list*[1]`;` `Text` `@@`	`@@#foo;.bar;` `Text` `@@`	`<div id="foo" class="bar">Text</div>`
		`@@color:red;` `Text` `@@`	`<div style="color:red">Text</div>`

1. The style-list should be a semi-colon (`;`) separated list consisting of a single hash-prefixed ID (which should be unique; e.g. `#foo`) and/or any number of dot-prefixed class names (e.g. `.bar`) and/or style properties (e.g. `color:red`).

HTML Attributes

SugarCube provides a few special HTML attributes, which you may add to raw HTML tags to enable special behaviors. There are attributes for passage links, image passages, and setters.

Type	Attribute	Example

Passage, Link	`data-passage`[1]	`<a data-passage="Grocery">Go buy milk</a>`
		`<area shape="rect" coords="25,25,75,75" data-passage="Lake House">`

Passage, Image	`data-passage`[1]	`<img data-passage="PlayerBioPic">`
Setter	`data-setter`	`<a data-passage="Grocery" data-setter="$bought to 'milk'">Go buy milk</a>`

1. The `data-passage` attribute normally acts as a link, however, for `<img>` tags it acts as the source.

Link w/ Setter	`[[Link][Setter]]`	`[[Grocery][$bought to "milk"]]`
		`[[$go][$bought to "milk"]]`
			Text:	`Grocery`

			Link:	`Grocery`
			Setter:	`$bought to "milk"`

Link w/ Text & Setter	`[[Text|Link][Setter]]`	`[[Go buy milk|Grocery][$bought to "milk"]]`
		`[[$show\|$go][$bought to "milk"]]`
			Text:	`Go buy milk`

			Link:	`Grocery`
			Setter:	`$bought to "milk"`

Mastering Whitespace

Line Continuations

A backslash (`\`) which begins or ends a line is the line continuation markup. The line continuation markup causes the backslash and the associated line break to be removed upon processing, thus joining the nearby lines together. This is mostly useful for controlling whitespace when you want to wrap lines for readability, but not generate extra whitespace upon display, and the `<<silently>>` macro isn't an option because you need to generate output.

For example, both of the following:

```
The rain in Spain falls \
mainly on the plain.

The rain in Spain falls
\ mainly on the plain.
```

Yield the single line in the final output:

```
The rain in Spain falls mainly on the plain.
```

SEE: The `<<nobr>>` macro and `nobr` tag perform a similar function.

`<<nobr>>`

Executes its contents and outputs the result, after compressing all line breaks and replacing them with spaces.

Arguments: *none*

Usage:

```
→ Given: $feeling eq "blue", outputs: I'd like a blueberry pie.
I'd like a <<nobr>>
<<if $feeling eq "blue">>
blueberry
<<else>>
cherry
<</if>>
<</nobr>> pie.
```

`<</nobr>>` *or* `<<endnobr>>`

Closes the `<<nobr>>` macro.

`<<silently>>`

Causes any output generated within its body to be discarded, except for errors (which will be displayed). Generally, only really useful for formatting blocks of macros for ease of use/readability, while ensuring that no output is generated, from spacing or whatnot.

Arguments: *none*

Usage:

```
→ A countdown timer
<<set $seconds to 10>>\
Countdown: <span id="countdown">$seconds seconds remaining</span>!\
<<silently>>
    <<repeat 1s>>
        <<set $seconds to $seconds - 1>>
        <<if $seconds gt 0>>
            <<replace "#countdown">>$seconds seconds remaining<</replace>>
        <<else>>
            <<replace "#countdown">>Too Late<</replace>>
            /* do something useful here */
            <<stop>>
        <</if>>
    <</repeat>>
<</silently>>
```

`<</silently>>` *or* `<<endsilently>>`

Closes the `<<silently>>` macro.

Timing!

`<<repeat delay [transition|t8n]>>`

Repeatedly executes its contents after the given delay, inserting any output into the passage in its place.

NOTE: Passage navigation terminates all pending timed executions.

Arguments:

- **delay:** The amount of time to delay, as a valid CSS time value (e.g. `5s` and `500ms`). The minimum delay is `40ms`.
- **transition:** (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- **t8n:** (optional) Keyword, alias for **transition**.

Usage:

```
→ A countdown timer
<<set $seconds to 10>>\
Countdown: <span id="countdown">$seconds seconds remaining</span>!\
<<silently>>
    <<repeat 1s>>
        <<set $seconds to $seconds - 1>>
        <<if $seconds gt 0>>
            <<replace "#countdown">>$seconds seconds remaining<</replace>>
        <<else>>
            <<replace "#countdown">>Too Late<</replace>>
            /* do something useful here */
            <<stop>>
        <</if>>
    <</repeat>>
<</silently>>
```

`<<stop>>`

Terminates the execution of the innermost `<<repeat>>` macro

`<</repeat>>` *or* `<<endrepeat>>`

Closes the `<<repeat>>` macro.

`<<timed delay [transition|t8n]>>`

Executes its contents after the given delay, inserting any output into the passage in its place.

NOTE: Passage navigation terminates all pending timed executions.

Arguments:

- **delay:** The amount of time to delay, as a valid CSS time value (e.g. `5s` and `500ms`). The minimum delay is `40ms`.
- **transition:** (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- **t8n:** (optional) Keyword, alias for **transition**.

Usage:

```
→ Insert some text after 5 seconds with a transition
I want to go to…<<timed 5s t8n>> WONDERLAND!<</timed>>

→ Replace some text after 10 seconds
I like green <span id="eggs">eggs</span> and ham!\
<<timed 10s>><<replace "#eggs">>pancakes<</replace>><</timed>>

→ A execute <<goto>> after 10 seconds
<<timed 10s>><<goto "To the Moon, Alice">><</timed>>
```

`<<goto passage_name>>`
`<<goto wiki_link>>`

Immediately forwards the user to the passage with the given name. May be called either with the passage name or with a wiki link.

NOTE: `<<goto>>` does not terminate the rendering passage in which it was encountered, so care must be taken to ensure that no unwanted state modifications occur after the call to `<<goto>>`.

Arguments (passage name version):

- **passage_name:** The name of the passage to go to.

Arguments (wiki link version):

- **wiki_link:** The wiki link to use (regular syntax only, no setters).

Usage:

```
→ Passage name version
<<goto "Somewhere over yonder">>
<<goto $selectedPassage>>

→ Wiki link version
<<goto [[Somewhere over yonder]]>>
<<goto [[$selectedPassage]]>>
```

`<<replace selector>>`

Executes its contents and replaces the contents of the selected element(s) with the output.

SEE: DOM macro warning.

Arguments:

- **selector:** The CSS/jQuery-style selector used to target element(s).

Usage:

```
→ Given the following
I'd like a <span id="snack">slice of Key lime pie</span>, please.

→ Replace the contents of the target element
<<link "Go cupcake">>
    <<replace "#snack">>cupcake<</replace>>
<</link>>

→ Result, after the click
I'd like a <span id="snack">cupcake</span>, please.
```

`<</replace>>` *or* `<<endreplace>>`

Closes the `<<replace>>` macro.

`<<next [delay]>>`

Chains additional timed executions, in order, to its parent `<<timed>>`, which execute their contents after their given delay and append any output to the passage after the first.

Arguments:

- **delay:** (optional) The amount of time to delay, as a valid CSS time value (e.g. `5s` and `500ms`). The minimum delay is `40ms`. If omitted, the last delay specified, from a `<<next>>` or the parent `<<timed>>`, will be used.

Usage:

```
→ Insert some text in 2 second intervals three times (at: 2s, 4s, 6s)
<<timed 2s>>Hi! Ho!
<<next>>Hi! Ho!
<<next>>It's off to work we go!
<</timed>>

→ Replace some text in 1 second intervals
I'll have <span id="drink">some water</span>, please.\
<<timed 1s>><<replace "#drink">>a glass of milk<</replace>>\
<<next>><<replace "#drink">>a can of soda<</replace>>\
<<next>><<replace "#drink">>a cup of coffee<</replace>>\
<<next>><<replace "#drink">>tea, southern style, sweet<</replace>>\
<<next>><<replace "#drink">>a scotch, neat<</replace>>\
<<next>><<replace "#drink">>a bottle of your finest absinthe<</replace>>\
<</timed>>

→ Set a $variable after 4 seconds, 3 seconds, 2 seconds, and 1 second
<<silently>>
<<set $choice to 0>>
<<timed 4s>>
    <<set $choice to 1>>
<<next 3s>>
    <<set $choice to 2>>
<<next 2s>>
    <<set $choice to 3>>
<<next 1s>>
    <<set $choice to 4>>
<</timed>>
<<silently>>\
```

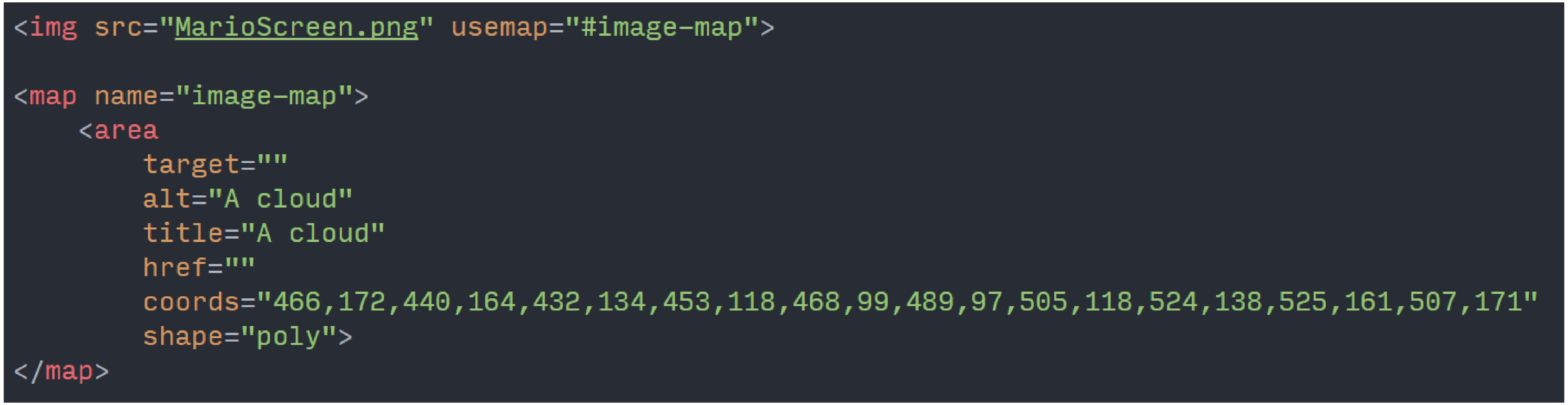
Maps!

HTML Maps

- Can be used to create a navigation map or a more visual way to explore in Twine
- Define clickable regions in an image
- w3schools example and demo
- Map generator: image-map.net/

Map -> Twine

- Code from generator:



```
<img src="MarioScreen.png" usemap="#image-map">

<map name="image-map">
    <area
        target=""
        alt="A cloud"
        title="A cloud"
        href=""
        coords="466,172,440,164,432,134,453,118,468,99,489,97,505,118,524,138,525,161,507,171"
        shape="poly">
</map>
```

1: Update URL

```
<img src="MarioScreen.png" usemap="#image-map">

<map name="image-map">
    <area
        ~~target=""~~          2: Remove target
        alt="A cloud"
        title="A cloud"
        ~~href=""~~            3: Remove href
        coords="466,172,440,164,432,134,453,118,468,99,489,97,505,118,524,138,525,161,507,171"
        shape="poly">
</map>
```

4: Add: data-passage=“Passage Name” to link area to a passage

```html
<img src="http://mikewesthad.github.io/Class-TwineMedia/Images/MarioScreen.png" usemap="#image-map">

<map name="image-map">
    <area
        data-passage="Cloud"
        alt="A cloud"
        title="A cloud"
        coords="466,172,440,164,432,134,453,118,468,99,489,97,505,118,524,138,525,161,507,171"
        shape="poly">
</map>
```

HTML Attributes

SugarCube provides a few special HTML attributes, which you may add to raw HTML tags to enable special behaviors. There are attributes for passage links, image passages, and setters.

Type	Attribute	Example

Passage, Link	`data-passage`[1]	`<a data-passage="Grocery">Go buy milk</a>`
		`<area shape="rect" coords="25,25,75,75" data-passage="Lake House">`

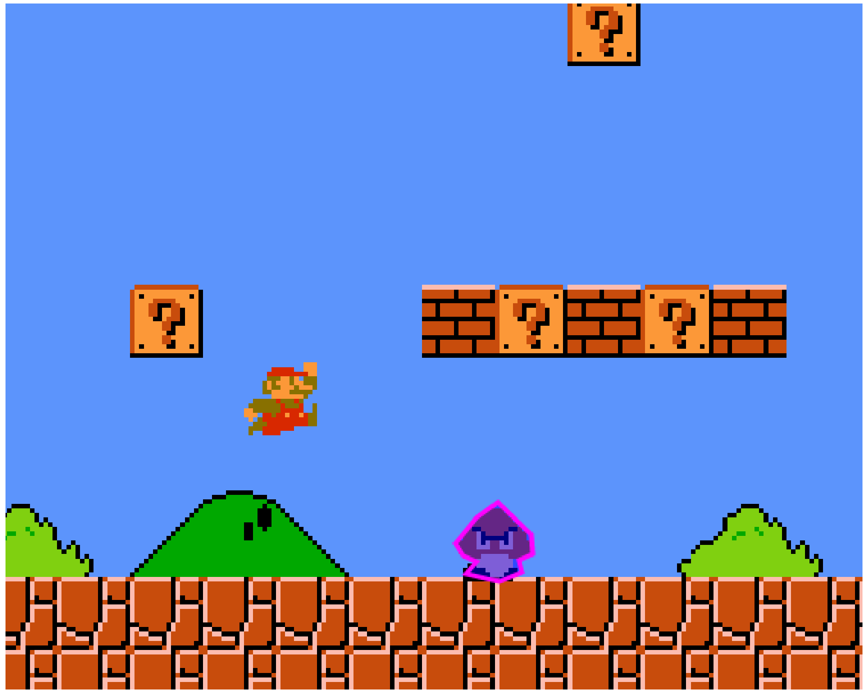
Passage, Image	`data-passage`[1]	`<img data-passage="PlayerBioPic">`
Setter	`data-setter`	`<a data-passage="Grocery" data-setter="$bought to 'milk'">Go buy milk</a>`

1. The `data-passage` attribute normally acts as a link, however, for `<img>` tags it acts as the source.

maphilight.js

maphilight.js

- A JavaScript library for highlighting areas inside of a map



Integrating with Twine

- Copy my modified version of maphilight.js: here
- Paste into your Twine's Story Javascript
- Create special "PassageDone" passage
- Use <<script>> to active maphilight in PassageDone

```
<<script>>

// Apply maphilight to all images that have a usemap attribute
$("img[usemap]").maphilight({
  fill: true,               // Fill the area?
  fillColor: '0000ff',      // HEX format without the starting "#"
  fillOpacity: 0.5,         // Opacity of the filled area
  stroke: true,             // Outline the area?
  strokeColor: 'ff00ff',
  strokeOpacity: 1,
  strokeWidth: 3,           // Outline width
  fade: true,               // Animate when hovered with a fade?
  alwaysOn: false,          // Always show the areas?
  neverOn: false,
  groupBy: false,
  wrapClass: true,
  shadow: false,
  shadowX: 0,
  shadowY: 0,
  shadowRadius: 6,
  shadowColor: '000000',
  shadowOpacity: 0.8,
  shadowPosition: 'outside',
  shadowFrom: false
});

<</script>>
```

```
<<script>>

// Apply maphilight to all images that have a usemap attribute
$("img[usemap]").maphilight({
  // ... styling ommitted ...
});

// If you have multiple maps in your story and you want to apply different
// styling to each:
//  1: Give each map (or maps) that should share a style a class
//  2: Use the below code, but replace "showAllMap" with the class name
$("img.showAllMap").data("maphilight", {
  fillColor: 'ff0000',
  stroke: false,
  alwaysOn: true
});

// If you want to override any of those options for an area:
//  1: Give the area (or areas) a class
//  2: Use the below code, but replace "greenArea" with the class name
$("area.greenArea").data("maphilight", {
  fillColor: '00ff00',
  fillOpacity: 0.5
});

<</script>>
```

Typed Library

Typed.js

- JavaScript text animation library
- Live demo

Setup

- Go to motoslave.net/sugarcube/2/
- Download “typed.js integration module”
- Copy:
  - typedjs-integration-module.min.js --> Story JavaScirpt
  - typedjs-integration-module.css --> Story Stylesheet

`typed` Class Name Examples

Basic usage, speed and delay at defaults:

```
typed
```

To set the typing speed to 80ms:

```
typed-speed80
```

To delay the start of typing for 800ms:

```
typed-delay800
```

To set the typing speed to 80ms and delay the start of typing for 800ms:

```
typed-speed80-delay800
```

Usage Examples

Using custom styles markup, with speed and delay at defaults:

```
@@.typed;
Hello, meatbag.

I am M.E.L., your Master Electronic Leader.

Loyalty and hard work shall be rewarded with continued existence.  Disobedience or poor work ethic shall be rewarded

Have a nice day.  And remember, the computer is your friend.
@@
```

Randomness

SugarCube 2.x Functions

SugarCube is a free (gratis and libre) story format for Twine/Twee, based on TiddlyWiki.

This documentation is a reference for various functions available within SugarCube.

`clone()`, `either()`, `hasVisited()`, `lastVisited()`, `passage()`, `previous()`, `random()`, `randomFloat()`, `setPageElement()`, `tags()`, `time()`, `turns()`, `variables()`, `visited()`, `visitedTags()`

`random([min ,] max)` : *integer*

Returns a pseudo-random whole number (integer) within the range of the given bounds.

Parameters:

- **min:** (optional) The lower bound of the random number (inclusive). If omitted, will default to `0`.
- **max:** The upper bound of the random number (inclusive).

Usage:

```
random(5)     → Returns a number in the range 0-5
random(1, 6)  → Returns a number in the range 1-6
```

`randomFloat([min ,] max)` : *float*

Returns a pseudo-random real number (floating-point) within the range of the given bounds.

NOTE: Unlike with its sibling function `random()`, the `max` parameter is exclusive, not inclusive (i.e. the range goes to, but does not include, the given value).

Parameters:

- **min:** (optional) The lower bound of the random number (inclusive). If omitted, will default to `0.0`.
- **max:** The upper bound of the random number (exclusive).

Usage:

```
randomFloat(5.0)       → Returns a number in the range 0.0-4.9999999…
randomFloat(1.0, 6.0)  → Returns a number in the range 1.0-5.9999999…
```

`either(list…)` : *any*

Returns a random value from its given arguments.

Parameters:

- **list** (*any*) The list of values to operate on. May be any combination of singular values, actual arrays, or array-like objects. All values will be concatenated into a single list for selection.

Usage:

```
// Using singular values
either("Blueberry", "Cherry", "Pecan")  → Returns a random pie from the whole list

// Using arrays; given: $pies = [ "Blueberry", "Cherry", "Pecan" ]
either($pies)  → Returns a random pie from the whole array

// Using singular values and arrays; given: $letters = [ "A", "B", "C" ]
either($letters, "D", "E")  → Returns a random value from the whole list (i.e. "A", "B", "C", "D", "E")

// Using multiple arrays; given: $letters = [ "A", "B", "C" ] & $numerals = [ "1", "2", "3" ]
either($letters, $numerals)  → Returns a random value from the whole list (i.e. "A", "B", "C", "1", "2", "3")
```