SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Introduction

This documentation is a reference for SugarCube, a free (gratis and libre) story format for Twine/Twee.

> **TIP:** This document is a single page, so you may use your browser's find-in-page functionality —CTRL + F or F3—to search for specific terms.

> **NOTE:** If you believe that you've found a bug in SugarCube, or simply wish to make a suggestion, you may do so by creating a new issue at its source code repository.

Contributions to this documentation have been graciously made by:

- **Chapel** (TwineLab, GitHub)

Markup

> **NOTE:** Except where noted, all markup has been available since `v2.0.0`.

Naked Variable

In addition to using one of the print macros (`<<print>>`, `<<=>>`, `<<->>`) to print the values of TwineScript variables, SugarCube's naked variable markup allows printing them simply by including them within your normal passage text—i.e., variables in passage text are interpolated into a string representation of their values.

The following forms are supported by the naked variable markup:

Type	Syntax	Example
Simple variable	`$variable`	`$name`
Property access, dot notation	`$variable.property`	`$thing.name`

Index/property access, square bracket notation	`$variable[numericIndex]`
	`$variable["property"]`
	`$variable['property']`
	`$variable[$indexOrPropertyVariable]`
		`$thing[0]`
		`$thing["name"]`
		`$thing['name']`
		`$thing[$member]`

If you need to print anything more complex—e.g., using a calculation, `$variable[_i + 1]`, or a method call, `$variable.someMethod()`—then you will still need to use one of the print macros.

For example:

```
/* Explicitly printing the value of $name via the <<print>> macro */
Well hello there, <<print $name>>.

/* Implicitly printing the value of $name via the naked variable markup */
Well hello there, $name.

/* Assuming $name is set to "Mr. Freeman", both should yield the following */
Well hello there, Mr. Freeman.
```

Because variables within your passage text are transformed into their values automatically, if you actually want to output a variable as-is—i.e., without interpolation; e.g., for a tutorial, debug output, or whatever—then you'll need to escape it in some fashion. For example:

```
/* Using the nowiki markup: """…""" (triple double-quotes) */
The variable """$name""" is set to: $name

/* Using the nowiki markup: <nowiki>…</nowiki> */
The variable <nowiki>$name</nowiki> is set to: $name

/* Using the double dollar-sign markup (which escapes the $-sigil): $$ */
The variable $$name is set to: $name
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
/* Assuming $name is set to "Mr. Freeman", all of the above should yield the follo
The variable $name is set to: Mr. Freeman
```

Additionally, you could use the inline code markup to escape the variable, though it will also wrap the escaped variable within a `<code>` element, so it's probably best used for examples and tutorials. For example:

```
/* Using the inline code markup: {{{…}}} (triple curly braces) */
The variable {{{$name}}} is set to: $name

/* Assuming $name is set to "Mr. Freeman", it should yield the following */
The variable <code>$name</code> is set to: Mr. Freeman
```

Link

SugarCube's link markup consists of a required `Link` component and optional `Text` and `Setter` components. The `Link` and `Text` components may be either plain text or any valid TwineScript expression, which will be evaluated early—i.e., when the link is initially processed. The `Setter` component, which only works with passage links, must be a valid TwineScript expression, of the `<<set>>` macro variety, which will be evaluated late—i.e., when the link is clicked on.

The `Link` component value may be the title of a passage or any valid URL to a resource (local or remote).

In addition to the standard pipe separator (`|`) used to separate the `Link` and `Text` components (as seen below), SugarCube also supports the arrow separators (`->` & `<-`). Particular to the arrow separators, the arrows' direction determines the order of the components, with the arrow always pointing at the `Link` component—i.e., the right arrow works like the pipe separator, `Text->Link`, while the left arrow is reversed, `Link<-Text`.

> **WARNING (TWINE 2):** Due to how the Twine 2 automatic passage creation feature currently works, using any TwineScript expression for the `Link` component will cause a passage named after the expression to be created that will need to be deleted. To avoid this problem, it's suggested that you use the separate argument form of the `<<link>>` macro in Twine 2 when you need to use an expression.

For the following examples assume: `$go` is `"Grocery"` and `$show` is `"Go buy milk"`

Syntax	Example
`[[Link]]`	`[[Grocery]]` `[[$go]]`
`[[Text|Link]]`	`[[Go buy milk|Grocery]]` `[[$show\|$go]]`
`[[Link][Setter]]`	`[[Grocery][$bought to "milk"]]` `[[$go][$bought to "milk"]]`
`[[Text|Link][Setter]]`	`[[Go buy milk|Grocery][$bought to "milk"]]` `[[$show|$go][$bought to "milk"]]`

Image

SugarCube's image markup consists of a required `Image` component and optional `Title`, `Link`, and `Setter` components. The `Image`, `Title`, and `Link` components may be either plain text or any valid TwineScript expression, which will be evaluated early—i.e., when the link is initially processed. The `Setter` component, which only works with passage links, must be a valid TwineScript expression, of the `<<set>>` macro variety, which will be evaluated late—i.e., when the link is clicked on.

The `Image` component value may be any valid URL to an image resource (local or remote) or the title of an embedded image passage (Twine 1 & Tweego only). The `Link` component value may be the title of a passage or any valid URL to a resource (local or remote).

In addition to the standard pipe separator (`|`) used to separate the `Image` and `Title` components (as seen below), SugarCube also supports the arrow separators (`->` & `<-`). Particular to the arrow separators, the arrows' direction determines the order of the components, with the arrow always pointing at the `Image` component—i.e., the right arrow works like the pipe separator, `Title->Image`, while the left arrow is reversed, `Image<-Title`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL+F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

WARNING (TWINE 2): Due to how the Twine 2 automatic passage creation feature currently works, using any TwineScript expression for the `Link` component will cause a passage named after the expression to be created that will need to be deleted. To avoid this problem, it's suggested that you use the separate argument form of the `<<link>>` macro in Twine 2 when you need to use an expression.

For the following examples assume: `$src` is `home.png`, `$go` is `"Home"`, and `$show` is `"Go home"`

Syntax	Example
`[img[Image]]`	`[img[home.png]]` `[img[$src]]`
`[img[Image][Link]]`	`[img[home.png][Home]]` `[img[$src][$go]]`
`[img[Image][Link][Setter]]`	`[img[home.png][Home][$done to true]]` `[img[$src][$go][$done to true]]`
`[img[Title|Image]]`	`[img[Go home|home.png]]` `[img[$show\|$src]]`
`[img[Title|Image][Link]]`	`[img[Go home|home.png][Home]]` `[img[$show\|$src][$go]]`
`[img[Title|Image][Link][Setter]]`	`[img[Go home|home.png][Home][$done to true]]` `[img[$show|$src][$go][$done to true]]`

Within stylesheets

A restricted subset of the image markup, allowing only the `Image` component, may be used within stylesheets —primarily to allow the easy use of image passages. For example:

```
/* Using the external image "forest.png" as the <body> background. */
body {
	background-image: [img[forest.png]];
}

/* Using the image passage "lagoon" as the <body> background. */
body {
	background-image: [img[lagoon]];
}
```

HTML Attribute

WARNING: None of these features work within the verbatim HTML markup.

Special Attribute

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

Type	Attribute	Example

Passage, Link	`data-passage`	`<a data-passage="PassageName">Do the thing</a>`
		`<area shape="rect" coords="25,25,75,75" data-passage="PassageName`
		`<button data-passage="PassageName">Do the thing</button>`

Passage, Audio	`data-passage`	`<audio data-passage="AudioPassageName">`
Passage, Image	`data-passage`	`<img data-passage="ImagePassageName">`
Passage, Source	`data-passage`	`<source data-passage="AudioOrVideoPassageName">`
Passage, Video	`data-passage`	`<video data-passage="VideoPassageName">`

Setter	`data-setter`	`<a data-passage="PassageName" data-setter="$thing to 'done'">Do t`
		`<area shape="rect" coords="25,25,75,75" data-passage="PassageName`
		`data-setter="$thing to 'done'">`
		`<button data-passage="PassageName" data-setter="$thing to 'done'"`

History:

- `v2.0.0`: Introduced.
- `v2.24.0`: Added `data-passage` attribute support to `<audio>`, `<source>`, and `<video>` tags.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Attribute Directive

HTML attributes may be prefixed with directives, special text, which trigger special processing of such attributes.

Evaluation directive: `sc-eval:`, `@`

The evaluation directive causes the attribute's value to be evaluated as TwineScript. Post-evaluation, the directive will be removed from the attribute's name and the result of the evaluation will be used as the actual value of the attribute.

> **WARNING:** The evaluation directive is not allowed on the `data-setter` attribute—as its function is to evaluate its contents upon activation of its own element—and any such attempt will cause an error.

For the following examples assume: `_id` is `"foo"`

Syntax	Example	Rendered As
`sc-eval:`*`attribute-name`*	`<span sc-eval:id="_id">…</span>`	`<span id="foo">…</span>`
`@attribute-name`	`<span @id="_id">…</span>`	`<span id="foo">…</span>`

History:

- `v2.21.0`: Introduced.
- `v2.23.5`: Fixed an issue with the evaluation directive where using multiple directives on a single HTML tag would result in some being unprocessed.

Line Continuation

> **SEE ALSO:** The various no-break features—`<<nobr>>` macro, `nobr` special tag, and `Config.passages.nobr` setting—all perform a similar, though slightly different, function.

> **WARNING:** Line continuations, or any markup that relies on line positioning, are incompatible with the no-break features because of how the latter function.

A backslash (`\`) that begins or ends a line is the line continuation markup. Upon processing the backslash, associated line break, and all whitespace between them are removed—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, all of the following: (n.b., `·` represents whitespace that will be removed, `¬` represents line breaks)

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

The rain in Spain falls \····¬
mainly on the plain.

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.
```

Heading

An exclamation point (`!`) that begins a line defines the heading markup. It consists of one to six exclamation points, each additional one beyond the first signifying a lesser heading.

Type	Syntax & Example	Rendered As	Displays As (*roughly*)

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Type	Syntax & Example	Rendered As	Displays As (*roughly*)
Level 1	`!Level 1 Heading`	`<h1>Level 1 Heading</h1>`	**Level 1 Heading**
Level 2	`!!Level 2 Heading`	`<h2>Level 2 Heading</h2>`	**Level 2 Heading**
Level 3	`!!!Level 3 Heading`	`<h3>Level 3 Heading</h3>`	**Level 3 Heading**
Level 4	`!!!!Level 4 Heading`	`<h4>Level 4 Heading</h4>`	**Level 4 Heading**
Level 5	`!!!!!Level 5 Heading`	`<h5>Level 5 Heading</h5>`	**Level 5 Heading**
Level 6	`!!!!!!Level 6 Heading`	`<h6>Level 6 Heading</h6>`	**Level 6 Heading**

Style

WARNING: Because the style markups use the same tokens to begin and end each markup, the same style cannot be nested within itself.

Type	Syntax & Example	Rendered As	Displays As (*roughly*)
Emphasis	`//Emphasis//`	`<em>Emphasis</em>`	*Emphasis*
Strong	`''Strong''`	`<strong>Strong</strong>`	**Strong**
Underline	`__Underline__`	`<u>Underline</u>`	Underline
Strikethrough	`==Strikethrough==`	`<s>Strikethrough</s>`	~~Strikethrough~~
Superscript	`Super^^script^^`	`Super<sup>script</sup>`	Super^{script}
Subscript	`Sub~~script~~`	`Sub<sub>script</sub>`	Sub$_{script}$

List

An asterisk (`*`) or number sign (`#`) that begins a line defines a member of the unordered or ordered list markup, respectively.

Type	Syntax & Example	Rendered As	Displays As (*roughly*)

Unordered	`* A list item`
	`* Another list item`
		`<ul>`
		`<li>A list item</li>`
		`<li>Another list item</li>`
		`</ul>`
			• A list item
			• Another list item

Ordered	`# A list item`
	`# Another list item`
		`<ol>`
		`<li>A list item</li>`
		`<li>Another list item</li>`
		`</ol>`
			1. A list item
			2. Another list item

Blockquote

A right angle bracket (`>`) that begins a line defines the blockquote markup. It consists of one or more right angle brackets, each additional one beyond the first signifying a level of nested blockquote.

Syntax & Example	Rendered As	Displays As (*roughly*)

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Syntax & Example	Rendered As	Displays As (*roughly*)
`>Line 1` `>Line 2` `>>Nested 1` `>>Nested 2`	`<blockquote>Line 1 ` `Line 2 ` `<blockquote>Nested 1 ` `Nested 2 ` `</blockquote></blockquote>`	Line 1 Line 2 Nested 1 Nested 2

Code

Type	Syntax & Example	Rendered As	Displays As (*roughly*)
Inline	`{{{Code}}}`	`<code>Code</code>`	`Code`

Block	`{{{`
	`Code`
	`More code`
	`}}}`
		`<pre><code>Code`
		`More code`
		`</code></pre>`
			`Code`
			`More code`

Horizontal Rule

A set of four hyphen/minus characters (`-`) that begins a line defines the horizontal rule markup.

Type	Syntax & Example	Rendered As	Displays As (*roughly*)
Horizontal rule	`----`	`<hr>`	———

Verbatim Text

The verbatim text markup disables processing of *all* markup contained within—both SugarCube and HTML—passing its contents directly into the output as plain text.

Type	Syntax & Example	Rendered As	Displays As (*roughly*)
Triple double quotes	`"""No //format//"""`	`No //format//`	No //format//
<nowiki> tag	`<nowiki>No //format//</nowiki>`	`No //format//`	No //format//

Verbatim HTML

A set of opening and closing <html> tags—i.e., `<html></html>`—defines the verbatim HTML markup. The verbatim HTML markup disables processing of *all* markup contained within—both SugarCube and HTML—passing its contents directly into the output as HTML markup for the browser. Thus, you should only use plain HTML markup within the verbatim markup—meaning using none of SugarCube's special HTML attributes or directives.

NOTE: You should virtually never need to use the verbatim HTML markup.

Custom Style

WARNING: Because the custom style markup uses the same tokens to begin and end the markup, it cannot be nested within itself.

Type	Syntax	Example	Rendered As
Inline	`@@`*style-list*[1]`;Text@@`	`@@#alfa;.bravo;Text@@`	`<span id="alfa" class="bravo">Text<`
		`@@color:red;Text@@`	`<span style="color:red">Text</span>`

Block	`@@`*style-list*[1]`;`
	`Text`
	`@@`
		`@@#alfa;.bravo;`
		`Text`
		`@@`
			`<div id="alfa" class="bravo">Text</`

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Type	Syntax	Example	Rendered As

		`@@color:red;`
		`Text`
		`@@`
			`<div style="color:red">Text</div>`

1. The style-list should be a semi-colon (`;`) separated list consisting of one or more of the following:
   - A single unique hash-prefixed ID—e.g., `#alfa`.
   - Dot-prefixed class names—e.g., `.bravo`.
   - Style properties—e.g., `color:red`.

   As of `v2.31.0`, the ID and class names components may be conjoined without need of extra semi-colons—e.g., `#alfa;.bravo;.charlie;` may also be written as `#alfa.bravo.charlie;`.

Template

A text replacement markup. The template markup begins with a question mark (`?`) followed by the template name—e.g., `?yolo`—and are set up as functions-that-return-strings, strings, or arrays of either—from which a random member is selected whenever the template is processed. They are defined via the `Template` API.

For example, consider the following markup:

```
?He was always willing to lend ?his ear to anyone.
```

Assuming that `?He` resolves to `She` and `?his` to `her`, then that will produce the following output:

```
She was always willing to lend her ear to anyone.
```

History:

- `v2.29.0`: Introduced.

Comment

NOTE: Comments used within passage markup are not rendered into the page output.

Type	Syntax & Example	Supported Within...
C-style, Block	`/* This is a comment. */`	Passage markup, JavaScript, Stylesheets
TiddlyWiki, Block	`/% This is a comment. %/`	Passage markup
HTML, Block	`<!-- This is a comment. -->`	Passage markup

TwineScript

TwineScript in SugarCube is, essentially, JavaScript with an extra spoonful of sugar on top to make it a bit nicer for the uninitiated.

Variables

NOTE: Temporary variables were added in `v2.3.0`.

A variable is a bit of storage where you may stash a value for later use. In SugarCube, they come in two types: story variables and temporary variables. Story variables are a part of the story history and exist for the lifetime of a playthrough session. Temporary variables do not become part of the story history and only exist for the lifetime of the moment/turn that they're created in. You'll likely use story variables most often throughout your project—though, temporary variables are perfect candidates for things like loop variables, if you're using the `<<for>>` macro.

For example, you might use the story variable `$name` to store the main player character's name or the story variable `$cash` to store how much money the player has on hand.

Values may be of most primitive types and some object types, see Supported Types for more information.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Variable Names

The names of both story and temporary variables have a certain format that they must follow—which signifies that they are variables and not some other kind of data.

The very first, and mandatory, character is their sigil, which denotes whether they are a story or temporary variable. The sigil must be a dollar sign (`$`) for story variables or an underscore (`_`) for temporary variables.

The second, and also mandatory, character of the variable name may be one of the following: the letters A though Z (in upper or lower case), the dollar sign, and the underscore (i.e., `A-Za-z$_`)—after their initial use as the sigil, the dollar sign and underscore become regular variable characters.

Subsequent, optional, characters have the same set as the second with the addition of numerals (i.e., `0-9`, so the full set is `A-Za-z0-9$_`). No other characters are allowed.

A few examples of valid names:

```
/* Story variables */
$cash
$hasKeyCard5
$met_alice
$TIMES_POKED_MR_BEAR

/* Temporary variables */
_i
_something2
_some_loop_value
_COUNT
```

Using Variables

NOTE: This is not an exhaustive list. There are many ways to use and interact with variables.

To modify the values contained within variables, see the `<<set>>` macro and setter links.

To print the values contained within variables, see the naked variable markup and the `<<print>>`, `<<=>>`, and `<<->>` macros.

To control aspects of your project based on the values contained within variables, see the `<<if>>` and `<<switch>>` macros.

Supported Types

The following types of values are natively supported by SugarCube and may be safely used within story and temporary variables.

Primitives

- Booleans—e.g., `true` & `false`
- Numbers—e.g., `42`, `3.14`, & `-17.01`
- Strings—e.g., `"I like pie"` & `'You like pie'`
- `null`
- `undefined`

Objects

- `Array`
- `Date`
- `Map`
- `Set`
- Generic objects

Any supported object type may itself contain any supported primitive or object type.

Unsupported object types, either native or custom, can be made compatible by implementing `.clone()` and `.toJSON()` methods for them—see the *Non-generic object types (a.k.a. classes)* guide for more information.

WARNING: Due to how SugarCube stores the state history a few constructs are ***not supported*** within story variables.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- Circular references. If you need them, then you'll need to keep them out of story variables.
- Property attributes, including getters/setters, and symbol properties. If you need them, then you'll need to use a class or similar non-generic object.
- Functions, including static—i.e., non-instance—methods, due to a few issues.

    1. A function's scope ***cannot*** be restored. Thus, if your function depends upon its scope, then it will not work properly when revived from sessions or saves.
    2. Function behavior is immutable. Thus, storing them within story variables is generally wasteful.

    Instance methods of classes are not affected by either issue, as they're never actually stored within story variables, being referenced from their classes' prototypes instead.

Expressions

SEE ALSO: While not specifically about SugarCube, the About Expressions section of the Twine 1 reference documentation may also be helpful.

Expressions are simply units of code that yield values when evaluated. For example:

```
2 + 2       → Yields: 4
"a" + "bc"  → Yields: "abc"
turns()     → Yields: 1 (assuming that it is the first turn)
```

While every valid expression—even those you might not expect—yields a value, there are essentially two types of expressions: those with side effects and those without. A side effect simply means that the evaluation of the expression modifies some state. For example:

```
$a = 5   → Yields: 5; Side effect: assigns 5 to the story variable $a
$x + 10  → Yields: 25 (assuming $x is 15); No side effects
```

In general, you can group expressions into categories based on what kind of value they yield and/or what side effects they cause. For example: *(not an exhaustive list)*

- Arithmetic: The expression yields a number value—e.g., `42` or `3.14`.
- String: The expression yields a string value—e.g., `"Lulamoon"` or `"5678"`.
- Logical: The expression yields a boolean value—e.g., `true` or `false`.
- Assignment: The expression causes an assignment to occur—e.g., `$a = 5`.

Using Expressions

You will, in all likelihood, use expressions most often within macros—e.g., `<<set>>`, `<<print>>`, `<<if>>`, `<<for>>`.

Macros

Macro Arguments

Macros fall into two broad categories based on the kind of arguments they accept: those that want an expression—e.g., `<<set>>` and `<<print>>`—and those that want discrete arguments separated by whitespace—e.g., `<<link>>` and `<<audio>>`. The documentation for each macro will tell you what it expects.

Those that want an expression are fairly straightforward, as you simply supply an expression.

The discrete argument type of macros are also fairly straightforward, most of the time, as you simply supply the requisite arguments separated by whitespace, which may include variables—as SugarCube automatically yields their values to the macro. There are cases, however, where things get a bit more complicated, namely: instances where you need to pass the name of a variable as an argument, rather than its value, and those where you want to pass the result of an expression as argument.

Argument type macros: passing a variable's name as an argument

Passing the name of a variable as an argument is problematic because variable substitution occurs automatically in SugarCube macros. Meaning that when you pass a variable as an argument, its value is

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

passed to the macro rather than its name.

Normally, this is exactly what you want to happen. Occasionally, however, macros will need the name of a variable rather than its value—e.g., data input macros like `<<textbox>>`—so that they may modify the variable. To resolve these instances, you will need to quote the name of the variable—i.e., instead of passing `$pie` as normal, you'd pass `"$pie"`. These, rare, instances are noted in the macros' documentation and shown in their examples.

Argument type macros: passing an expression as an argument

Passing the result of an expression as an argument is problematic for a couple of reasons: because the macro argument parser doesn't treat arguments as expressions by default and because it separates arguments with whitespace.

Normally, those aren't issues as you should not need to use the result of an expression as an argument terribly often. To resolve instances where you do, however, you'll want to use either a temporary variable or a backquote expression.

For example, the following will not work because the macro parser will think that you're passing five discrete arguments, rather than a single expression:

```
<<link "Wake " + $friend + ".">> … <</link>>
```

You could solve the problem by using a temporary variable to hold the result of the expression, then pass that to the macro. For example:

```
<<set _text to "Wake " + $friend + ".">>\
<<link _text>> … <</link>>
```

A better solution, however, would be to use a backquote[1] (`) expression, which is really just a special form of quoting available in macro arguments that causes the contents of the backquotes to be evaluated and then yields the result as a singular argument. For example:

```
<<link `"Wake " + $friend + "."`>> … <</link>>
```

1. A backquote is also known as a grave and is often paired with the tilde (~) on keyboards.

Variables Macros

`<<capture variableList>> … <</capture>>`

Captures story $variables and temporary _variables, creating localized versions of their values within the macro body.

> **NOTE:** Use of this macro is *only* necessary when you need to localize a variable's value for use with an asynchronous macro—i.e., a macro whose contents are executed at some later time, rather than when it's invoked; e.g., interactive macros, `<<repeat>>`, `<<timed>>`. Generally, this means only when the variable's value will change between the time the asynchronous macro is invoked and when it's activated—e.g., a loop variable.

History:

- `v2.14.0`: Introduced.

Arguments:

- `variableList`: A list of story $variables and/or temporary _variables.

Examples:

```
→ Using <<capture>> to localize a temporary loop variable for use within a <<linka
<<set _what to [
        "a crab rangoon",
        "a gaggle of geese",
        "an aardvark",
        "the world's smallest violin"
]>>
<<for _i to 0; _i lt _what.length; _i++>>
        <<capture _i>>
                I spy with my little <<linkappend "eye" t8n>>, _what[_i]<</linkappe
        <</capture>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<for>>

→ Capturing several variables at once
<<capture $aStoryVar, $anotherStoryVar, _aTempVar>> … <</capture>>
```

<<set expression>>

Sets story $variables and temporary _variables based on the given expression.

History:

- v2.0.0: Introduced.

Arguments:

- expression: A valid expression. See Variables and Expressions for more information.

TwineScript assignment operators:

Operator	Description	Example
to	Assigns the value on the right-hand side of the operator to the left-hand side.	<<set $apples to 6>>

JavaScript assignment operators: *(not an exhaustive list)*

Operator	Description	Example
=	Assigns the value on the right-hand side of the operator to the left-hand side.	<<set $apples = 6>>
+=	Adds the value on the right-hand side of the operator to the current value on the left-hand side and assigns the result to the left-hand side.	<<set $apples += 1>>
-=	Subtracts the value on the right-hand side of the operator from the current value on the left-hand side and assigns the result to the left-hand side.	<<set $apples -= 1>>
*=	Multiplies the current value on the left-hand side of the operator by the value on the right-hand side and assigns the result to the left-hand side.	<<set $apples *= 2>>
/=	Divides the current value on the left-hand side of the operator by the value on the right-hand side and assigns the result to the left-hand side.	<<set $apples /= 2>>
%=	Divides the current value on the left-hand side of the operator by the value on the right-hand side and assigns the remainder to the left-hand side.	<<set $apples %= 10>>

Examples:

```
→ Using the TwineScript "to" operator
<<set $cheese to "a nice, sharp cheddar">>  → Assigns "a nice, sharp cheddar" to st
<<set $chestEmpty to true>>                 → Assigns boolean true to story variabl
<<set $gold to $gold + 5>>                  → Adds 5 to the value of story variable
<<set _counter to _counter + 1>>            → Adds 1 to the value of temporary vari

→ Using standard JavaScript operators
<<set $cheese = "a nice, sharp cheddar">>   → Assigns "a nice, sharp cheddar" to st
<<set $chestEmpty = true>>                  → Assigns boolean true to story variabl
<<set $gold += 5>>                          → Adds 5 to the value of story variable
<<set _counter += 1>>                       → Adds 1 to the value of temporary vari
```

<<unset variableList>>

Unsets story $variables and temporary _variables.

History:

- v2.0.0: Introduced.

Arguments:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

- `variableList`: A list of story $variables and/or temporary _variables.

Examples:

```
<<unset $cheese, $chestEmpty, $gold>>
<<unset _someTempVar>>
```

~~<<remember expression>>~~

> **DEPRECATED:** This macro has been deprecated and should no longer be used. See the `memorize()` and `recall()` functions for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.29.0`: Deprecated in favor of `memorize()` and `recall()`.

~~<<forget variableList>>~~

> **DEPRECATED:** This macro has been deprecated and should no longer be used. See the `forget()` function for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.29.0`: Deprecated in favor of `forget()`.

Scripting Macros

<<run expression>>

Functionally identical to `<<set>>`. Intended to be mnemonically better for uses where the expression is arbitrary code, rather than variables to set—i.e., `<<run>>` to run code, `<<set>>` to set variables.

<<script>> … <</script>>

Silently executes its contents as *pure* JavaScript code—i.e., it performs no story or temporary variable substitution or TwineScript operator processing. For instances where you need to run some pure JavaScript and don't want to waste time performing extra processing on code that has no story or temporary variables or TwineScript operators in it and/or worry about the parser possibly clobbering the code.

> **NOTE:** The predefined variable `output`, which is a reference to a local content buffer, is available for use within the macro's code contents. Once the code has been fully executed, the contents of the buffer, if any, will be output.

History:

- `v2.0.0`: Introduced.

Arguments: *none*

Examples:

```
→ Basic
<<script>>
        /* pure JavaScript code */
<</script>>

→ Modifying the content buffer
<<script>>
        /* Parse some markup and append the result to the output buffer. */
        $(output).wiki("Cry 'Havoc!', and let slip the //ponies// of ''friendship''");
<</script>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Display Macros

<<= expression>>

Outputs a string representation of the result of the given expression. This macro is an alias for <<print>>.

> **TIP:** If you only need to print the value of a TwineScript variable, then you may simply include it in your normal passage text and it will be printed automatically via the naked variable markup.

History:

- v2.0.0: Introduced.

Arguments:

- **expression:** A valid expression. See Expressions for more information.

Examples:

```
→ Assuming $gold is 5
You found <<= $gold>> gold.             → Outputs: You found 5 gold.

→ Assuming $weight is 74.6466266
You weigh <<= $weight.toFixed(2)>> kg.  → Outputs: You weigh 74.65 kg.
```

<<- expression>>

Outputs a string representation of the result of the given expression. This macro is functionally identical to <<print>>, save that it also encodes HTML special characters in the output.

> **TIP:** If you only need to print the value of a TwineScript variable, then you may simply include it in your normal passage text and it will be printed automatically via the naked variable markup.

History:

- v2.0.0: Introduced.

Arguments:

- **expression:** A valid expression. See Expressions for more information.

Examples:

```
→ Assuming $gold is 5
You found <<- $gold>> gold.             → Outputs: You found 5 gold.

→ Assuming $weight is 74.6466266
You weigh <<- $weight.toFixed(2)>> kg.  → Outputs: You weigh 74.65 kg.
```

<<include passageName [elementName]>>
<<include linkMarkup [elementName]>>

Outputs the contents of the passage with the given name, optionally wrapping it within an HTML element. May be called either with the passage name or with a link markup.

History:

- v2.15.0: Introduced.

Arguments:

Passage name form

- **passageName:** The name of the passage to include.
- **elementName:** (optional) The HTML element to wrap the included passage in. If used, the element will include the passage's name normalized into a class name. See CSS passage conversions for more information.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).
- `elementName`: *Identical to the passage name form.*

Examples:

```
<<include "Go West">>          → Include the passage "Go West"
<<include [[Go West]]>>        → Include the passage "Go West"
<<include "Go West" "div">>    → Include the passage "Go West", wrapping it within
<<include [[Go West]] "div">>  → Include the passage "Go West", wrapping it within
```

`<<nobr>> … <</nobr>>`

Executes its contents and outputs the result, after removing leading/trailing newlines and replacing all remaining sequences of newlines with single spaces.

> **NOTE:** The `nobr` special tag and `Config.passages.nobr` setting applies the same processing to an entire passage or all passages, respectively. The line continuation markup performs a similar function, though in a slightly different way.

History:

- `v2.0.0`: Introduced.

Arguments: *none*

Examples:

```
→ 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.
```

`<<print expression>>`

Outputs a string representation of the result of the given expression.

> **TIP:** If you only need to print the value of a TwineScript variable, then you may simply include it in your normal passage text and it will be printed automatically via the naked variable markup.

History:

- `v2.0.0`: Introduced.

Arguments:

- `expression`: A valid expression. See Expressions for more information.

Examples:

```
→ Assuming $gold is 5
You found <<print $gold>> gold.             → Outputs: You found 5 gold.

→ Assuming $weight is 74.6466266
You weigh <<print $weight.toFixed(2)>> kg.  → Outputs: You weigh 74.65 kg.
```

`<<silently>> … <</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.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

Arguments: *none*

Examples:

```
→ Basic
<<silently>>

        You'll never see any of this!

<</silently>>

→ Hiding the guts of 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<</repla
                <<else>>
                        <<replace "#countdown">>Too Late<</replace>>
                        /* do something useful here */
                        <<stop>>
                <</if>>
        <</repeat>>
<</silently>>
```

`<<type speed [start delay] [class classes] [element tag] [id ID] [kee`

`...`

`<</type>>`

Outputs its contents a character—technically, a code point—at a time, mimicking a teletype/typewriter. Can type most content: links, markup, macros, etc.

> **WARNING:** Interactions with macros or other code that inject content only after some external action or period—e.g., `<<linkreplace>>`, `<<timed>>`, etc.—may or may not behave as you'd expect. Testing is ***strongly*** advised.

> **SEE ALSO:** `Config.macros.typeSkipKey`, `Config.macros.typeVisitedPassages`, `<<type>>` Events.

History:

- `v2.32.0`: Introduced.
- `v2.33.0`: Added `class`, `element`, and `id` options and `macro-type-done` class.
- `v2.33.1`: Added `skipkey` option.

Arguments:

- **`speed`:** The rate at which characters are typed, as a valid CSS time value—e.g., `1s` and `40ms`. Values in the range `20-60ms` are a good starting point.
- **`start` *`delay`*:** (optional) The amount of time to delay the start of typing, as a valid CSS time value—e.g., `5s` and `500ms`. If omitted, defaults to `400ms`.
- **`class` *`classes`*:** (optional) The space separated list of classes to be added to the typing container.
- **`element` *`tag`*:** (optional) The element to use as the typing container—e.g., `div` and `span`. If omitted, defaults to `div`.
- **`id` *`ID`*:** (optional) The unique ID to be assigned to the typing container.
- **`keep`:** (optional) Keyword, used to signify that the cursor should be kept after typing is complete.
- **`none`:** (optional) Keyword, used to signify that the cursor should not be used at all.
- **`skipkey`:** (optional) The key used to cause typing to finish immediately. If omitted, defaults to the value of `Config.macros.typeSkipKey`.

Examples:

```
<<type 40ms>>
        Type characters from this content every 40 milliseconds.  Including [[links
<</type>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<type 40ms start 2s>>
        Type characters from this content every 40 milliseconds, starting after a 
<</type>>

<<type 40ms class "foo bar">>
        Type characters from this content every 40 milliseconds, adding classes to
<</type>>

<<type 40ms element "span">>
        Type characters from this content every 40 milliseconds, using a <span> as
<</type>>

<<type 40ms id "type01">>
        Type characters from this content every 40 milliseconds, assigning an ID t
<</type>>

<<type 40ms keep>>
        Type characters from this content every 40 milliseconds, keeping the cursor
<</type>>

<<type 40ms skipkey "Control">>
        Type characters from this content every 40 milliseconds, using the Control
<</type>>
```

CSS styles:

The typed text has no default styling. If you want to change the font or color, then you'll need to change the styling of the `macro-type` class. For example:

```
.macro-type {
        color: limegreen;
        font-family: monospace, monospace;
}
```

There's also a `macro-type-done` class that is added to text that has finished typing, which may be used to style it differently from actively typing text.

The default cursor is the block element character **Right Half Block (U+2590)** and it has no default font or color styling. If you want to change the font, color, or character, then you'll need to change the styling of the `:after` pseudo-element of the `macro-type-cursor` class. For example:

```
.macro-type-cursor:after {
        color: limegreen;
        content: "\269C\FE0F"; /* Fleur-de-lis emoji */
        font-family: monospace, monospace;
}
```

~~<<display passageName [elementName]>>~~
~~<<display linkMarkup [elementName]>>~~

DEPRECATED: This macro has been deprecated and should no longer be used. See the `<<include>>` macro for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.15.0`: Deprecated in favor of `<<include>>`.

Control Macros

`<<if conditional>> … [<<elseif conditional>> …] [<<else>> …] <</if>>`

Executes its contents if the given conditional expression evaluates to `true`. If the condition evaluates to `false` and an `<<elseif>>` or `<<else>>` exists, then other contents can be executed.

NOTE: SugarCube does not trim whitespace from the contents of `<<if>>` macros, so that authors don't have to resort to various kludges to get whitespace where they want it. This means, however, that extra care must be taken when writing them to ensure that unwanted whitespace is not created within the final output.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Arguments:

- **`conditional`:** A valid conditional expression, evaluating to either `true` or `false`. See Expressions for more information.

TwineScript conditional operators:

Operator	Description	Example
`is`	Evaluates to `true` if both sides are *strictly* equal.	`<<if $bullets is 6>>`
`isnot`	Evaluates to `true` if both sides are *strictly* not equal.	`<<if $pie isnot "cherry">>`
`eq`	Evaluates to `true` if both sides are equivalent.	`<<if $bullets eq 6>>`
`neq`	Evaluates to `true` if both sides are not equivalent.	`<<if $pie neq "cherry">>`
`gt`	Evaluates to `true` if the left side is greater than the right side.	`<<if $cash gt 5>>`
`gte`	Evaluates to `true` if the left side is greater than or equal to the right side.	`<<if $foundStars gte $neededStars>>`
`lt`	Evaluates to `true` if the left side is less than the right side.	`<<if $shoeCount lt ($peopleCount * 2)>>`
`lte`	Evaluates to `true` if the left side is less than or equal to the right side.	`<<if $level lte 30>>`
`not`	Flips a `true` evaluation to `false`, and vice versa.	`<<if not $hungry>>`
`and`	Evaluates to `true` if all subexpressions evaluate to `true`.	`<<if $age gte 20 and $age lte 30>>`
`or`	Evaluates to `true` if any subexpressions evaluate to `true`.	`<<if $friend is "Sue" or $friend is "Dan">>`
`def`	Evaluates to `true` if the right side is defined.	`<<if def $mushrooms>>`
`ndef`	Evaluates to `true` if the right side is not defined.	`<<if ndef $bottlecaps>>`

WARNING: The `def` and `ndef` operators have very low precedence, so it is ***strongly*** recommended that if you mix them with other operators, that you wrap them in parentheses—e.g., `(def $style) and ($style is "girly")`.

JavaScript conditional operators: *(not an exhaustive list)*

Operator	Description	Example
`===`	Evaluates to `true` if both sides are *strictly* equal.	`<<if $bullets === 6>>`
`!==`	Evaluates to `true` if both sides are *strictly* not equal.	`<<if $pie !== "cherry">>`
`==`	Evaluates to `true` if both sides are equivalent.	`<<if $bullets == 6>>`
`!=`	Evaluates to `true` if both sides are not equivalent.	`<<if $pie != "cherry">>`
`>`	Evaluates to `true` if the left side is greater than the right side.	`<<if $cash > 5>>`
`>=`	Evaluates to `true` if the left side is greater than or equal to the right side.	`<<if $foundStars >= $neededStars>>`

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Operator	Description	Example
`<`	Evaluates to `true` if the left side is less than the right side.	`<<if $shoeCount < ($peopleCount * 2)>>`
`<=`	Evaluates to `true` if the left side is less than or equal to the right side.	`<<if $level <= 30>>`
`!`	Flips a `true` evaluation to `false`, and vice versa.	`<<if !$hungry>>`
`&&`	Evaluates to `true` if all subexpressions evaluate to `true`.	`<<if $age >= 20 && $age <= 30>>`
`||`	Evaluates to `true` if any subexpressions evaluate to `true`.	`<<if $friend === "Sue" || $friend === "Dan">>`

Examples:

```
<<if $daysUntilLoanDue is 0>><<include "Panic">><</if>>

<<if $cash lt 5>>
        I'm sorry, ma'am, but you don't have enough for the pie.
<<else>>
        <<set $cash -= 5, $hasMeatPie = true>>
        One meat pie, fresh out of the oven, coming up!
<</if>>

<<if $affection gte 75>>
        I love you!
<<elseif $affection gte 50>>
        I like you.
<<elseif $affection gte 25>>
        I'm okay with you.
<<else>>
        Get out of my face.
<</if>>

<<if $hullBreached>>
        <<if $wearingHardSuit>>
                <<include "That was close">>
        <<elseif $wearingSoftSuit>>
                <<include "Hole in suit">>
        <<else>>
                <<include "You die">>
        <</if>>
<</if>>
```

`<<for [conditional]>> … <</for>>`
`<<for [init] ; [conditional] ; [post]>> … <</for>>`
`<<for [keyVariable ,] valueVariable range collection>> … <</for>>`

Repeatedly executes its contents. There are three forms: a conditional-only form, a 3-part conditional form, and a range form.

SEE ALSO: `<<break>>` and `<<continue>>`.

History:

- `v2.0.0`: Introduced.
- `v2.20.0`: Added range form.

Notes

- Loop variables are perfect candidates for the use of temporary variables—e.g., `_i`.
- To ensure that line-breaks end up where you want them, or not, extra care may be required.

Conditional forms *(both conditional-only and 3-part)*

Executes its contents while the given conditional expression evaluates to `true`. If no conditional expression is given, it is equivalent to specifying `true`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: The maximum number of loop iterations in the conditional forms is not unlimited by default, however, it is configurable. See `Config.macros.maxLoopIterations` for more information.

Arguments:

- `init`: (optional) A valid expression, evaluated once at loop initialization. Typically used to initialize counter variable(s). See `<<set>>` for more information.
- `conditional`: (optional) A valid conditional expression, evaluated prior to each loop iteration. As long as the expression evaluates to `true`, the loop is executed. See `<<if>>` for more information.
- `post`: (optional) A valid expression, evaluated after each loop iteration. Typically used to update counter variable(s). See `<<set>>` for more information.

Examples: *(only 3-part conditional form shown)*

```
→ Example setup
<<set $dwarves to ["Doc", "Dopey", "Bashful", "Grumpy", "Sneezy", "Sleepy", "Happy'

→ Loop
<<for _i to 0; _i lt $dwarves.length; _i++>>
<<print _i + 1>>. $dwarves[_i]
<</for>>

→ Result
1. Doc
2. Dopey
3. Bashful
4. Grumpy
5. Sneezy
6. Sleepy
7. Happy
```

Range form

Iterates through all enumerable entries of the given collection. For each iteration, it assigns the key/value pair of the associated entry in the collection to the iteration variables and then executes its contents. Valid collection types are: arrays, generic objects, maps, sets, and strings.

Arguments:

- `keyVariable`: (optional) A story or temporary variable that will be set to the iteration key.
- `valueVariable`: A story or temporary variable that will be set to the iteration value.
- `range`: Keyword, used to signify that the loop is using the range form syntax.
- `collection`: An expression that yields a valid collection type, evaluated once at loop initialization.

Iteration Values:

Collection type	Iteration: key, value
Arrays & Sets	Member: index, value
Generic objects	Property: name, value
Maps	Entry: key, value
Strings	Code point: start index, value

NOTE: Strings are iterated by Unicode code point, however, due to historic reasons they are comprised of, and indexed by, individual UTF-16 code units. This means that some code points may span multiple code units—e.g., the character 💩 is one code point, but two code units.

Examples:

```
→ Example setup
<<set $dwarves to ["Doc", "Dopey", "Bashful", "Grumpy", "Sneezy", "Sleepy", "Happy'

→ Loop
<<for _i, _name range $dwarves>>
<<print _i + 1>>. _name
<</for>>

→ Result
1. Doc
2. Dopey
3. Bashful
4. Grumpy
5. Sneezy
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
6. Sleepy
7. Happy
```

<<break>>

Used within <<for>> macros. Terminates the execution of the current <<for>>.

History:

- v2.0.0: Introduced.

Arguments: *none*

<<continue>>

Used within <<for>> macros. Terminates the execution of the current iteration of the current <<for>> and begins execution of the next iteration.

> **NOTE:** May eat line-breaks in certain situations.

History:

- v2.0.0: Introduced.

Arguments: *none*

<<switch expression>> [<<case valueList>> …] [<<default>> …] <</switch>>

Evaluates the given expression and compares it to the value(s) within its <<case>> children. The value(s) within each case are compared to the result of the expression given to the parent <<switch>>. Upon a successful match, the matching case will have its contents executed. If no cases match and an optional <<default>> case exists, which must be the final case, then its contents will be executed. At most one case will execute.

> **NOTE:** SugarCube does not trim whitespace from the contents of <<case>>/<<default>> macros, so that authors don't have to resort to various kludges to get whitespace where they want it. However, this means that extra care must be taken when writing them to ensure that unwanted whitespace is not created within the final output.

History:

- v2.7.2: Introduced.

Arguments:

<<switch>>

- expression: A valid expression. See Expressions for more information.

<<case>>

- valueList: A space separated list of values to compare against the result of the switch expression.

Examples:

```
→ Without a default case
<<switch $hairColor>>
<<case "red" "auburn">>
        You ginger.
<<case "black" "brown">>
        Dark haired, eh?
<<case "blonde">>
        You may have more fun.
<</switch>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
→ With a default case (assume the passage is about a waterfall)
<<switch visited()>>
<<case 1>>
        You gaze in wonder at the magnificent waterfall for the first time, awestru
<<case 2 3>>
        You once again gaze upon the magnificent waterfall.
<<case 4 5>>
        Yet again, you find yourself looking upon the waterfall.
<<default>>
        Oh, look.  It's that waterfall again.  Meh.
<</switch>>
```

Interactive Macros

Warning

Interactive macros are both asynchronous and require interaction from the player. Thus, there are some potential pitfalls to consider:

1. If you plan on using interactive macros within a loop you will likely need to use the `<<capture>>` macro due to their asynchronous nature.
2. Reloading the page or revisiting a passage may not restore the state of some interactive macros, so it is recommended that you only use them in instances where this will not be an issue or where you can work around it.

`<<button linkText [passageName]>> … <</button>>`
`<<button linkMarkup>> … <</button>>`
`<<button imageMarkup>> … <</button>>`

Creates a button that silently executes its contents when clicked, optionally forwarding the player to another passage. May be called with either the link text and passage name as separate arguments, a link markup, or an image markup.

SEE: Interactive macro warning.

NOTE: This macro is functionally identical to `<<link>>`, save that it uses a button element (`<button>`) rather than an anchor element (`<a>`).

History:

- `v2.8.0`: Introduced.

Arguments:

Separate argument form

- `linkText`: The text of the link. May contain markup.
- `passageName`: (optional) The name of the passage to go to.

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).

Image markup form

- `imageMarkup`: The image markup to use (regular syntax only, no setters).

Examples:

```
→ Without forwarding: a very basic statistic setting example
Strength: <<set $pcStr to 10>><span id="stats-str"><<print $pcStr>></span> \
( <<button "[+]">><<set $pcStr++>><<replace "#stats-str">><<print $pcStr>><</repla
| <<button "[-]">><<set $pcStr-->><<replace "#stats-str">><<print $pcStr>><</repla

→ With forwarding: execute a script, then go to the specified passage
<<button "Onward, Reginald!" "On with the story">><<script>>/* (code) */<</script>>
<<button [[Onward, Reginald!|On with the story]]>><<script>>/* (code) */<</script>>
<<button [img[onward.jpg][On with the story]]>><<script>>/* (code) */<</script>><<
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`<<checkbox receiverName uncheckedValue checkedValue [autocheck|checke`

Creates a checkbox, used to modify the value of the variable with the given name.

SEE: Interactive macro warning.

History:

- `v2.0.0`: Introduced.
- `v2.32.0`: Added `autocheck` keyword.

Arguments:

- `receiverName`: The name of the variable to modify, which *must* be quoted—e.g., `"$foo"`. Object and array property references are also supported—e.g., `"$foo.bar"`, `"$foo['bar']"`, & `"$foo[0]"`.
- `uncheckedValue`: The value set by the checkbox when unchecked.
- `checkedValue`: The value set by the checkbox when checked.
- `autocheck`: (optional) Keyword, used to signify that the checkbox should be automatically set to the checked state based on the current value of the receiver variable. **NOTE:** Automatic checking may fail on non-primitive values—i.e., on arrays and objects.
- `checked`: (optional) Keyword, used to signify that the checkbox should be in the checked state.

Examples:

Basic usage

```
What pies do you enjoy?
* <<checkbox "$pieBlueberry" false true autocheck>> Blueberry?
* <<checkbox "$pieCherry" false true autocheck>> Cherry?
* <<checkbox "$pieCoconutCream" false true autocheck>> Coconut cream?
```

```
What pies do you enjoy?
* <<checkbox "$pieBlueberry" false true checked>> Blueberry?
* <<checkbox "$pieCherry" false true>> Cherry?
* <<checkbox "$pieCoconutCream" false true checked>> Coconut cream?
```

With a `<label>` element

TIP: For accessibility reasons, it's recommended that you wrap each `<<checkbox>>` and its accompanying text within a `<label>` element. Doing so allows interactions with the text to also trigger its `<<checkbox>>`.

```
What pies do you enjoy?
* <label><<checkbox "$pieBlueberry" false true autocheck>> Blueberry?</label>
* <label><<checkbox "$pieCherry" false true autocheck>> Cherry?</label>
* <label><<checkbox "$pieCoconutCream" false true autocheck>> Coconut cream?</label
```

```
What pies do you enjoy?
* <label><<checkbox "$pieBlueberry" false true checked>> Blueberry?</label>
* <label><<checkbox "$pieCherry" false true>> Cherry?</label>
* <label><<checkbox "$pieCoconutCream" false true checked>> Coconut cream?</label>
```

`<<cycle receiverName [once] [autoselect]>>`
`[<<option label [value [selected]]>> …]`
`[<<optionsfrom collection>> …]`
`<</cycle>>`

Creates a cycling link, used to modify the value of the variable with the given name. The cycling options are populated via `<<option>>` and/or `<<optionsfrom>>`.

SEE: Interactive macro warning.

History:

- `v2.29.0`: Introduced.
- `v2.36.0`: Fixed the `selected` keyword and added the `once` keyword.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Arguments:

`<<cycle>>`

- `receiverName`: The name of the variable to modify, which *must* be quoted—e.g., `"$foo"`. Object and array property references are also supported—e.g., `"$foo.bar"`, `"$foo['bar']"`, & `"$foo[0]"`.
- `once`: (optional) Keyword, used to signify that the cycle should stop upon reaching the last option and deactivate itself. **NOTE:** Since you likely want to start at the first option when using this keyword, you should either not select an option, so it defaults to the first, or, if you do, select the first option only.
- `autoselect`: (optional) Keyword, used to signify that an option should be automatically selected as the cycle default based on the current value of the receiver variable. **NOTE:** Automatic option selection will fail on non-primitive values—i.e., on arrays and objects.

`<<option>>`

- `label`: The label shown by the cycling link for the option.
- `value`: (optional) The value set by the cycling link when the option is selected. If omitted, the label will be used as the value.
- `selected`: (optional) Keyword, used to signify that the option should be the cycle default. Only one option may be so selected. If no options are selected as the default, the cycling link will default to the first option, unless the cycle `autoselect` keyword is specified. **NOTE:** If specified, the `value` argument is not optional.

`<<optionsfrom>>`

- `collection`: An expression that yields a valid collection type.

Collection type	Option: label, value
Arrays & Sets	Member: value, value
Generic objects	Property: name, value
Maps	Entry: key, value

Examples:

Using `<<option>>`

```
The answer to the //Ultimate Question of Life, the Universe, and Everything// is?
<<cycle "$answer" autoselect>>
        <<option "Towel">>
        <<option "π" 3.14159>>
        <<option 42>>
        <<option 69>>
        <<option "∞" Infinity>>
<</cycle>>
```

Using `<<optionsfrom>>` with an array

```
→ Given: _pieOptions = ["blueberry", "cherry", "coconut cream"]
What's your favorite pie?
<<cycle "$pie" autoselect>>
        <<optionsfrom _pieOptions>>
<</cycle>>
```

Using `<<optionsfrom>>` with an generic object

```
→ Given: _pieOptions = { "Blueberry" : "blueberry", "Cherry" : "cherry", "Coconut
What's your favorite pie?
<<cycle "$pie" autoselect>>
        <<optionsfrom _pieOptions>>
<</cycle>>
```

Using the `once` keyword

```
You see a large red, candy-like button.
<<cycle "$presses" once>>
        <<option "Should you press it?" 0>>
        <<option "Nothing happened.  Press it again?" 1>>
        <<option "Again?" 2>>
        <<option "That time it locked into place with a loud click and began to gl
<</cycle>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`<<link linkText [passageName]>> … <</link>>`
`<<link linkMarkup>> … <</link>>`
`<<link imageMarkup>> … <</link>>`

Creates a link that silently executes its contents when clicked, optionally forwarding the player to another passage. May be called with either the link text and passage name as separate arguments, a link markup, or an image markup.

> **SEE:** Interactive macro warning.

> **NOTE:** If you simply need a passage link that modifies variables, both the link markup and image markup offer setter variants.

History:

- `v2.8.0`: Introduced.

Arguments:

Separate argument form

- `linkText`: The text of the link. May contain markup.
- `passageName`: (optional) The name of the passage to go to.

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).

Image markup form

- `imageMarkup`: The image markup to use (regular syntax only, no setters).

Examples:

```
→ Without forwarding: a very basic statistic setting example
Strength: <<set $pcStr to 10>><span id="stats-str"><<print $pcStr>></span> \
( <<link "[+]">><<set $pcStr++>><<replace "#stats-str">><<print $pcStr>><</replace
| <<link "[-]">><<set $pcStr-->><<replace "#stats-str">><<print $pcStr>><</replace

→ With forwarding: execute a script, then go to the specified passage
<<link "Onward, Reginald!" "On with the story">><<script>>/* (code) */<</script>><
<<link [[Onward, Reginald!|On with the story]]>><<script>>/* (code) */<</script>><
<<link [img[onward.jpg][On with the story]]>><<script>>/* (code) */<</script>><</l
```

`<<linkappend linkText [transition|t8n]>> … <</linkappend>>`

Creates a single-use link that deactivates itself and appends its contents to its link text when clicked. Essentially, a combination of `<<link>>` and `<<append>>`.

> **SEE:** Interactive macro warning.

History:

- `v2.0.0`: Introduced.

Arguments:

- `linkText`: The text of the link. May contain markup.
- `transition`: (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- `t8n`: (optional) Keyword, alias for `transition`.

Examples:

```
→ Without a transition
We—We should <<linkappend "take">> away their METAL BAWKSES<</linkappend>>!

→ With a transition
I spy with my little <<linkappend "eye" t8n>>, a crab rangoon<</linkappend>>.
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

<<linkprepend linkText [transition|t8n]>> … <</linkprepend>>

Creates a single-use link that deactivates itself and prepends its contents to its link text when clicked. Essentially, a combination of <<link>> and <<prepend>>.

SEE: Interactive macro warning.

History:

- v2.0.0: Introduced.

Arguments:

- linkText: The text of the link. May contain markup.
- transition: (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- t8n: (optional) Keyword, alias for transition.

Examples:

```
→ Without a transition
You see a <<linkprepend "robot">>GIANT <</linkprepend>>.

→ With a transition
I <<linkprepend "like" t8n>>do not <</linkprepend>> lemons.
```

<<linkreplace linkText [transition|t8n]>> … <</linkreplace>>

Creates a single-use link that deactivates itself and replaces its link text with its contents when clicked. Essentially, a combination of <<link>> and <<replace>>.

SEE: Interactive macro warning.

History:

- v2.0.0: Introduced.

Arguments:

- linkText: The text of the link. May contain markup.
- transition: (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- t8n: (optional) Keyword, alias for transition.

Examples:

```
→ Without a transition
I'll have a <<linkreplace "cupcake">>slice of key lime pie<</linkreplace>>, please

→ With a transition
<<linkreplace "You'll //never// take me alive!" t8n>>On second thought, don't hurt
```

<<listbox receiverName [autoselect]>> [<<option label [value [selected]]>> …] [<<optionsfrom collection>> …] <</listbox>>

Creates a listbox, used to modify the value of the variable with the given name. The list options are populated via <<option>> and/or <<optionsfrom>>.

SEE: Interactive macro warning.

History:

- v2.26.0: Introduced.
- v2.27.0: Added autoselect keyword.
- v2.28.0: ~~Added <<optionsFrom>> child tag.~~

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.28.1`: Fixed name of `<<optionsfrom>>` child tag, which was erroneously added as `<<optionsFrom>>` in `v2.28.0`.
- `v2.29.0`: Made the `<<option>>` child tag's `value` argument optional.
- `v2.36.0`: Fixed the `selected` keyword.

Arguments:

`<<listbox>>`

- `receiverName`**:** The name of the variable to modify, which *must* be quoted—e.g., `"$foo"`. Object and array property references are also supported—e.g., `"$foo.bar"`, `"$foo['bar']"`, & `"$foo[0]"`.
- `autoselect`**:** (optional) Keyword, used to signify that an option should be automatically selected as the listbox default based on the current value of the receiver variable. **NOTE:** Automatic option selection will fail on non-primitive values—i.e., on arrays and objects.

`<<option>>`

- `label`**:** The label shown by the listbox for the option.
- `value`**:** (optional) The value set by the listbox when the option is selected. If omitted, the label will be used as the value.
- `selected`**:** (optional) Keyword, used to signify that the option should be the listbox default. Only one option may be so selected. If no options are selected as the default, the listbox will default to the first option, unless the listbox `autoselect` keyword is specified. **NOTE:** If specified, the `value` argument is not optional.

`<<optionsfrom>>`

- `collection`**:** An expression that yields a valid collection type.

Collection type	Option: label, value
Arrays & Sets	Member: value, value
Generic objects	Property: name, value
Maps	Entry: key, value

Examples:

Using `<<option>>`

```
The answer to the //Ultimate Question of Life, the Universe, and Everything// is?
<<listbox "$lbanswer" autoselect>>
        <<option "Towel">>
        <<option "π" 3.14159>>
        <<option 42>>
        <<option 69>>
        <<option "∞" Infinity>>
<</listbox>>
```

Using `<<optionsfrom>>` with an array

```
→ Given: _pieOptions = ["blueberry", "cherry", "coconut cream"]
What's your favorite pie?
<<listbox "$pie" autoselect>>
        <<optionsfrom _pieOptions>>
<</listbox>>
```

Using `<<optionsfrom>>` with an generic object

```
→ Given: _pieOptions = { "Blueberry" : "blueberry", "Cherry" : "cherry", "Coconut
What's your favorite pie?
<<listbox "$pie" autoselect>>
        <<optionsfrom _pieOptions>>
<</listbox>>
```

`<<numberbox receiverName defaultValue [passage] [autofocus]>>`

Creates a number input box, used to modify the value of the variable with the given name, optionally forwarding the player to another passage.

SEE: Interactive macro warning.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

- `v2.32.0`: Introduced.

Arguments:

- `receiverName`: The name of the variable to modify, which *must* be quoted—e.g., `"$foo"`. Object and array property references are also supported—e.g., `"$foo.bar"`, `"$foo['bar']"`, & `"$foo[0]"`.
- `defaultValue`: The default value of the number box.
- `passage`: (optional) The name of the passage to go to if the return/enter key is pressed. May be called either with the passage name or with a link markup.
- `autofocus`: (optional) Keyword, used to signify that the number box should automatically receive focus. Only use the keyword *once* per page; attempting to focus more than one element is undefined behavior.

Examples:

```
→ Creates a number box that modifies $wager
Wager how much on Buttstallion in the race? <<numberbox "$wager" 100>>

→ Creates an automatically focused number box that modifies $wager
Wager how much on Buttstallion in the race? <<numberbox "$wager" 100 autofocus>>

→ Creates a number box that modifies $wager and forwards to the "Result" passage
Wager how much on Buttstallion in the race? <<numberbox "$wager" 100 "Result">>

→ Creates an automatically focused number box that modifies $wager and forwards to
Wager how much on Buttstallion in the race? <<numberbox "$wager" 100 "Result" auto
```

`<<radiobutton receiverName checkedValue [autocheck|checked]>>`

Creates a radio button, used to modify the value of the variable with the given name. Multiple `<<radiobutton>>` macros may be set up to modify the same variable, which makes them part of a radio button group.

SEE: Interactive macro warning.

History:

- `v2.0.0`: Introduced.
- `v2.32.0`: Added `autocheck` keyword.

Arguments:

- `receiverName`: The name of the variable to modify, which *must* be quoted—e.g., `"$foo"`. Object and array property references are also supported—e.g., `"$foo.bar"`, `"$foo['bar']"`, & `"$foo[0]"`.
- `checkedValue`: The value set by the radio button when checked.
- `autocheck`: (optional) Keyword, used to signify that the radio button should be automatically set to the checked state based on the current value of the receiver variable. **NOTE:** Automatic checking may fail on non-primitive values—i.e., on arrays and objects.
- `checked`: (optional) Keyword, used to signify that the radio button should be in the checked state. **NOTE:** Only one radio button in a group—i.e., those using the same receiver variable—should be so checked.

Examples:

Basic usage

```
What's your favorite pie?
* <<radiobutton "$pie" "blueberry" autocheck>> Blueberry?
* <<radiobutton "$pie" "cherry" autocheck>> Cherry?
* <<radiobutton "$pie" "coconut cream" autocheck>> Coconut cream?
```

```
What's your favorite pie?
* <<radiobutton "$pie" "blueberry" checked>> Blueberry?
* <<radiobutton "$pie" "cherry">> Cherry?
* <<radiobutton "$pie" "coconut cream">> Coconut cream?
```

With a `<label>` element

TIP: For accessibility reasons, it's recommended that you wrap each `<<radiobutton>>` and its accompanying text within a `<label>` element. Doing so allows interactions with the text to also trigger its `<<radiobutton>>`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
What's your favorite pie?
* <label><<radiobutton "$pie" "blueberry" autocheck>> Blueberry?</label>
* <label><<radiobutton "$pie" "cherry" autocheck>> Cherry?</label>
* <label><<radiobutton "$pie" "coconut cream" autocheck>> Coconut cream?</label>
```

```
What's your favorite pie?
* <label><<radiobutton "$pie" "blueberry" checked>> Blueberry?</label>
* <label><<radiobutton "$pie" "cherry">> Cherry?</label>
* <label><<radiobutton "$pie" "coconut cream">> Coconut cream?</label>
```

<<textarea receiverName defaultValue [autofocus]>>

Creates a multiline text input block, used to modify the value of the variable with the given name.

SEE: Interactive macro warning.

History:

- v2.0.0: Introduced.

Arguments:

- **receiverName:** The name of the variable to modify, which *must* be quoted—e.g., "$foo". Object and array property references are also supported—e.g., "$foo.bar", "$foo['bar']", & "$foo[0]".
- **defaultValue:** The default value of the text block.
- **autofocus:** (optional) Keyword, used to signify that the text block should automatically receive focus. Only use the keyword *once* per page; attempting to focus more than one element is undefined behavior.

Examples:

```
→ Creates a text block that modifies $pieEssay
Write a short essay about pies:
<<textarea "$pieEssay" "">>

→ Creates an automatically focused text block that modifies $pieEssay
Write a short essay about pies:
<<textarea "$pieEssay" "" autofocus>>
```

<<textbox receiverName defaultValue [passage] [autofocus]>>

Creates a text input box, used to modify the value of the variable with the given name, optionally forwarding the player to another passage.

SEE: Interactive macro warning.

History:

- v2.0.0: Introduced.

Arguments:

- **receiverName:** The name of the variable to modify, which *must* be quoted—e.g., "$foo". Object and array property references are also supported—e.g., "$foo.bar", "$foo['bar']", & "$foo[0]".
- **defaultValue:** The default value of the text box.
- **passage:** (optional) The name of the passage to go to if the return/enter key is pressed. May be called either with the passage name or with a link markup.
- **autofocus:** (optional) Keyword, used to signify that the text box should automatically receive focus. Only use the keyword *once* per page; attempting to focus more than one element is undefined behavior.

Examples:

```
→ Creates a text box that modifies $pie
What's your favorite pie? <<textbox "$pie" "Blueberry">>

→ Creates an automatically focused text box that modifies $pie
What's your favorite pie? <<textbox "$pie" "Blueberry" autofocus>>

→ Creates a text box that modifies $pie and forwards to the "Cakes" passage
What's your favorite pie? <<textbox "$pie" "Blueberry" "Cakes">>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
→ Creates an automatically focused text box that modifies $pie and forwards to the
What's your favorite pie? <<textbox "$pie" "Blueberry" "Cakes" autofocus>>
```

~~<<click linkText [passageName]>> … <</click>>~~

~~<<click linkMarkup>>~~

~~<<click imageMarkup>>~~

DEPRECATED: This macro has been deprecated and should no longer be used. See the <<link>> macro for its replacement.

History:

- v2.0.0: Introduced.
- v2.8.0: Deprecated in favor of <<link>>.

Links Macros

<<actions passageList>>

<<actions linkMarkupList>>

<<actions imageMarkupList>>

Creates a list of single-use passage links. Each link removes itself and all other <<actions>> links to the same passage after being activated. May be called either with a list of passages, with a list of link markup, or with a list of image markup. Probably most useful when paired with <<include>>. See the <<actions>> section of the Twine 1 reference documentation for more information.

History:

- v2.0.0: Introduced.

Arguments:

Passage list form

- passageList: A space separated list of passage names.

Link markup list form

- linkMarkupList: A space separated list of link markup to use (full syntax supported, including setters).

Image markup list form

- imageMarkupList: A space separated list of image markup to use (full syntax supported, including setters).

Examples:

```
→ Passage list form
<<actions "Look at the pie" "Smell the pie" "Taste the pie">>

→ Link markup list form
<<actions [[Look at the pie]] [[Smell the pie]] [[Taste the pie]]>>
<<actions [[Look|Look at the pie]] [[Smell|Smell the pie]] [[Taste|Taste the pie]]

→ Image markup list form
<<actions [img[look.png][Look at the pie]] [img[smell.png][Smell the pie]] [img[ta
```

<<back [linkText]>>

<<back linkMarkup>>

<<back imageMarkup>>

Creates a link that undoes past moments within the story history. May be called with, optional, link text or with a link or image markup.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: If you want to return to a previously visited passage, rather than undo a moment within the history, see the `<<return>>` macro or the `previous()` function.

History:

- `v2.0.0`: Introduced.

Arguments:

Link text form

- `linkText`: (optional) The text of the link.

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).

Image markup form

- `imageMarkup`: The image markup to use (regular syntax only, no setters).

Examples:

Basic usage

```
→ Creates a link that undoes the most recent moment, with default text
<<back>>
```

Link text form

```
→ Creates a link that undoes the most recent moment, with text "Home."
<<back "Home.">>
```

Link markup form

```
→ Creates a link that undoes past moments until the most recent "HQ" moment is rea
<<back [[HQ]]>>

→ Creates a link that undoes past moments until the most recent "HQ" moment is rea
<<back [[Home.|HQ]]>>
```

Image markup form

```
→ Creates a link that undoes the most recent moment, with image "home.png"
<<back [img[home.png]]>>

→ Creates a link that undoes past moments until the most recent "HQ" moment is rea
<<back [img[home.png][HQ]]>>
```

`<<choice passageName [linkText]>>`
`<<choice linkMarkup>>`
`<<choice imageMarkup>>`

Creates a single-use passage link that deactivates itself and all other `<<choice>>` links within the originating passage when activated. May be called either with the passage name and link text as separate arguments, with a link markup, or with a image markup.

WARNING: Normally, when both link and text arguments are accepted, the order is text then link. However, due to a historical artifact, the arguments for the separate argument form of `<<choice>>` are in the reverse order (link then text).

History:

- `v2.0.0`: Introduced.

Arguments:

Separate argument form

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

- `passageName`: The name of the passage to go to.
- `linkText`: (optional) The text of the link. If omitted, the `passageName` will be used instead.

Link markup form

- `linkMarkup`: The link markup to use (full syntax supported, including setters).

Image markup form

- `imageMarkup`: The image markup to use (full syntax supported, including setters).

Examples:

```
→ Separate argument form
<<choice "Take the red pill">>
<<choice $someAction>>
<<choice "Entered magic mirror" "Touch the strange mirror.">>
<<choice $go $show>>

→ Link markup form
<<choice [[Take the red pill]]>>
<<choice [[$someAction]]>>
<<choice [[Touch the strange mirror.|Entered magic mirror]]>>
<<choice [[$show|$go]]>>

→ Image markup form
<<choice [img[redpill.png][Take the red pill]]>>
<<choice [img[some-image.jpg][$someAction]]>>
<<choice [img[mirror.jpg][Entered magic mirror]]>>
<<choice [img[$show][$go]]>>
```

`<<return [linkText]>>`
`<<return linkMarkup>>`
`<<return imageMarkup>>`

Creates a link that navigates forward to a previously visited passage. May be called with, optional, link text or with a link or image markup.

> **NOTE:** If you want to undo previous moments within the history, rather than return to a passage, see the `<<back>>` macro.

History:

- `v2.0.0`: Introduced.

Arguments:

Link text form

- `linkText`: (optional) The text of the link.

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).

Image markup form

- `imageMarkup`: The image markup to use (regular syntax only, no setters).

Examples:

> **NOTE:** The versions that forward to a specific passage are largely unnecessary, as you could simply use a normal link, and exist solely for compatibility with the `<<back>>` macro.

Basic usage

```
→ Creates a link that forwards to the previous passage, with default text
<<return>>
```

Link text form

```
→ Creates a link that forwards to the previous passage, with text "Home."
<<return "Home.">>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Link markup form

```
→ Creates a link that forwards to the "HQ" passage, with default text
<<return [[HQ]]>>

→ Creates a link that forwards to the "HQ" passage, with text "Home."
<<return [[Home.|HQ]]>>
```

Image markup form

```
→ Creates a link that forwards to the previous passage, with image "home.png"
<<return [img[home.png]]>>

→ Creates a link that forwards to the "HQ" passage, with image "home.png"
<<return [img[home.png][HQ]]>>
```

DOM Macros

WARNING: All DOM macros require the elements to be manipulated to be on the page. As a consequence, you cannot use them directly within a passage to modify elements within said passage, since the elements they are targeting are still rendering, thus not yet on the page. You must, generally, use them with an interactive macro—e.g., `<<link>>` macro—the `<<done>>` macro, or within the `PassageDone` special passage. Elements that are already part of the page, on the other hand, present no issues.

`<<addclass selector classNames>>`

Adds classes to the selected element(s).

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.

Arguments:

- `selector`: The CSS/jQuery-style selector used to target element(s).
- `classNames`: The names of the classes, separated by spaces.

Examples:

```
<<addclass "body" "day rain">>  → Add the classes "day" and "rain" to the <body> e
<<addclass "#pie" "cherry">>    → Add the class "cherry" to the element with the II
<<addclass ".joe" "angry">>     → Add the class "angry" to all elements containing
```

`<<append selector [transition|t8n]>> … <</append>>`

Executes its contents and appends the output to the contents of the selected element(s).

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.
- `v2.25.0`: Added `transition` and `t8n` keywords.

Arguments:

- `selector`: The CSS/jQuery-style selector used to target element(s).
- `transition`: (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- `t8n`: (optional) Keyword, alias for `transition`.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Without a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Append to the contents of the target element
<<link "Doing">>
        <<append "#dog">> chasing a cat<</append>>
<</link>>

→ Result, after clicking
I saw a <span id="dog">dog chasing a cat</span>.
```

With a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Append to the contents of the target element
<<link "Doing">>
        <<append "#dog" t8n>> chasing a cat<</append>>
<</link>>

→ Result, after clicking
I saw a <span id="dog">dog<span class="macro-append-insert"> chasing a cat</span><
```

`<<copy selector>>`

Outputs a copy of the contents of the selected element(s).

WARNING: Most interactive elements—e.g., passage links, interactive macros, etc.—cannot be properly copied via `<<copy>>`. Attempting to do so will, usually, result in something that's non-functional.

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.

Arguments:

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

Examples:

```
→ Example setup
I'd like a <span id="snack-source">slice of Key lime pie</span>, please.

I'll have a <span id="snack-dest">breadstick</span>, thanks.

→ Replace the contents of the source target element with a copy of the destination
<<link "Have the same">>
        <<replace "#snack-dest">><<copy "#snack-source">> too<</replace>>
<</link>>

→ Result, after the click
I'd like a <span id="snack-source">slice of Key lime pie</span>, please.

I'll have a <span id="snack-dest">slice of Key lime pie too</span>, thanks.
```

`<<prepend selector [transition|t8n]>> … <</prepend>>`

Executes its contents and prepends the output to the contents of the selected element(s).

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.
- `v2.25.0`: Added `transition` and `t8n` keywords.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Arguments:

- `selector`: The CSS/jQuery-style selector used to target element(s).
- `transition`: (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- `t8n`: (optional) Keyword, alias for `transition`.

Examples:

Without a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Prepend to the contents of the target element
<<link "Size">>
        <<prepend "#dog">>big <</prepend>>
<</link>>

→ Result, after clicking
I saw a <span id="dog">big dog</span>.
```

With a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Prepend to the contents of the target element
<<link "Size">>
        <<prepend "#dog" t8n>>big <</prepend>>
<</link>>

→ Result, after clicking
I saw a <span id="dog"><span class="macro-prepend-insert">big </span>dog</span>.
```

`<<remove selector>>`

Removes the selected element(s).

SEE: DOM macro warning.

NOTE: If you simply want to empty the selected element(s), not remove them outright, you should use an empty `<<replace>>` macro instead.

History:

- `v2.0.0`: Introduced.

Arguments:

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

Examples:

```
→ Given the following
I'd like a <span id="huge-cupcake">humongous </span>cupcake, please.

→ Remove the target element
<<link "Go small">>
        <<remove "#huge-cupcake">>
<</link>>

→ Result, after the click
I'd like a cupcake, please.
```

`<<removeclass selector [classNames]>>`

Removes classes from the selected element(s).

SEE: DOM macro warning.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

Arguments:

- **`selector`:** The CSS/jQuery-style selector used to target element(s).
- **`classNames`:** (optional) The names of the classes, separated by spaces. If no class names are given, removes all classes.

Examples:

```
<<removeclass "body" "day rain">>  → Remove the classes "day" and "rain" from the
<<removeclass "#pie" "cherry">>    → Remove the class "cherry" from the element wi
<<removeclass ".joe" "angry">>     → Remove the class "angry" from all elements co
<<removeclass "#begone">>          → Remove all classes from the element with the
```

`<<replace selector [transition|t8n]>> … <</replace>>`

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

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.
- `v2.25.0`: Added `transition` and `t8n` keywords.

Arguments:

- **`selector`:** The CSS/jQuery-style selector used to target element(s).
- **`transition`:** (optional) Keyword, used to signify that a CSS transition should be applied to the incoming insertions.
- **`t8n`:** (optional) Keyword, alias for `transition`.

Usage

Without a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Replace the contents of the target element
<<link "Breed">>
        <<replace "#dog">>Catahoula Cur<</replace>>
<</link>>

→ Result, after clicking
I saw a <span id="dog">Catahoula Cur</span>.
```

With a transition

```
→ Example setup
I saw a <span id="dog">dog</span>.

→ Replace the contents of the target element
<<link "Breed">>
        <<replace "#dog" t8n>>Catahoula Cur<</replace>>
<</link>>

→ Result, after clicking
I saw a <span id="dog"><span class="macro-replace-insert">Catahoula Cur</span></spa
```

`<<toggleclass selector classNames>>`

Toggles classes on the selected element(s)—i.e., adding them if they don't exist, removing them if they do.

SEE: DOM macro warning.

History:

- `v2.0.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Arguments:

- `selector`: The CSS/jQuery-style selector used to target element(s).
- `classNames`: The names of the classes, separated by spaces.

Examples:

```
<<toggleclass "body" "day rain">>  → Toggle the classes "day" and "rain" on the <b
<<toggleclass "#pie" "cherry">>    → Toggle the class "cherry" on the element with
<<toggleclass ".joe" "angry">>     → Toggle the class "angry" on all elements cont
```

Audio Macros

WARNING: The audio subsystem that supports the audio macros comes with some built-in limitations and it is ***strongly*** recommended that you familiarize yourself with them.

`<<audio trackIdList actionList>>`

Controls the playback of audio tracks, which must be set up via `<<cacheaudio>>`.

SEE: Audio macro limitations.

NOTE: The `<<audio>>` macro cannot affect playlist tracks that have been copied into their respective playlist—meaning those set up via `<<createplaylist>>` with its `copy` action or all tracks set up via, the deprecated, `<<setplaylist>>`—as playlist copies are solely under the control of their playlist.

NOTE: The `Config.audio.pauseOnFadeToZero` setting (default: `true`) controls whether tracks that have been faded to `0` volume (silent) are automatically paused.

History:

- `v2.0.0`: Introduced.
- `v2.1.0`: Added `fadeoverto` action.
- `v2.8.0`: Added group ID(s).
- `v2.28.0`: Added `load` and `unload` actions.

Arguments:

- `trackIdList`: The list of track and/or group IDs, separated by spaces. See below for details on group IDs.
- `actionList`: The list of actions to perform. Available actions are:
  - `fadein`: Start playback of the selected tracks and fade them from their current volume level to `1` (loudest) over `5` seconds.
  - `fadeout`: Start playback of the selected tracks and fade them from their current volume level to `0` (silent) over `5` seconds.
  - `fadeoverto` *`seconds`* *`level`*: Start playback of the selected tracks and fade them from their current volume level to the specified level over the specified number of seconds.
  - `fadeto` *`level`*: Start playback of the selected tracks and fade them from their current volume level to the specified level over `5` seconds.
  - `goto` *`passage`*: Forwards the player to the passage with the given name when playback of the first of the selected tracks ends normally. May be called either with the passage name or with a link markup.
  - `load`: Pause playback of the selected tracks and, if they're not already in the process of loading, force them to drop any existing data and begin loading. **NOTE:** This *should not* be done lightly if your audio sources are on the network, as it forces the player to begin downloading them.
  - `loop`: Set the selected tracks to repeat playback upon ending normally.
  - `mute`: Mute the volume of the selected tracks—effectively volume `0`, except without changing the volume level.
  - `pause`: Pause playback of the selected tracks.
  - `play`: Start playback of the selected tracks.
  - `stop`: Stop playback of the selected tracks.
  - `time` *`seconds`*: Set the current playback time of the selected tracks to the specified number of seconds. Valid values are floating-point numbers in the range `0` (start) to the maximum duration—e.g., `60` is `60` is sixty seconds in, `90.5` is ninety-point-five seconds in.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `unload`: Stop playback of the selected tracks and force them to drop any existing data. **NOTE:** Once unloaded, playback cannot occur until a `load` action is issued.
- `unloop`: Set the selected tracks to not repeat playback (this is the default).
- `unmute`: Unmute the volume of the selected tracks (this is the default).
- `volume` *`level`*: Set the volume of the selected tracks to the specified level. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Group IDs:

Group IDs allow several tracks to be selected simultaneously without needing to specify each one individually. There are several predefined group IDs (`:all`, `:looped`, `:muted`, `:paused`, `:playing`) and custom IDs may be defined via `<<createaudiogroup>>`. The `:not()` group modifier syntax (`groupId:not(trackIdList)`) allows a group to have some of its tracks excluded from selection.

Examples:

Basic usage with group IDs

```
→ Start playback of paused tracks
<<audio ":paused" play>>

→ Pause playback of playing tracks
<<audio ":playing" pause>>

→ Stop playback of playing tracks
<<audio ":playing" stop>>

→ Stop playback of all tracks
<<audio ":all" stop>>

→ Stop playback of playing tracks except those in the ":ui" group
<<audio ":playing:not(:ui)" stop>>

→ Change the volume of all tracks except those in the ":ui" group
→ to 40%, without changing the current playback state
<<audio ":all:not(:ui)" volume 0.40>>
```

Basic usage with track IDs

```
→ Given the following (best done in the StoryInit special passage)
<<cacheaudio "bgm_space" "media/audio/space_quest.mp3" "media/audio/space_quest.ogg

→ Start playback
<<audio "bgm_space" play>>

→ Start playback at 50% volume
<<audio "bgm_space" volume 0.5 play>>

→ Start playback at 120 seconds in
<<audio "bgm_space" time 120 play>>

→ Start repeating playback
<<audio "bgm_space" loop play>>

→ Start playback and fade from 0% to 100% volume
<<audio "bgm_space" volume 0 fadein>>

→ Start playback and fade from 75% to 0% volume
<<audio "bgm_space" volume 0.75 fadeout>>

→ Start playback and fade from 25% to 75% volume
<<audio "bgm_space" volume 0.25 fadeto 0.75>>

→ Start playback and fade from 25% to 75% volume over 30 seconds
<<audio "bgm_space" volume 0.25 fadeoverto 30 0.75>>

→ Start playback and goto the "Peace Moon" passage upon ending normally
<<audio "bgm_space" play goto "Peace Moon">>

→ Pause playback
<<audio "bgm_space" pause>>

→ Stop playback
<<audio "bgm_space" stop>>

→ Mute playback, without changing the current playback state
<<audio "bgm_space" mute>>

→ Unmute playback, without changing the current playback state
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<audio "bgm_space" unmute>>

→ Change the volume to 40%, without changing the current playback state
<<audio "bgm_space" volume 0.40>>

→ Seek to 90 seconds in, without changing the current playback state
<<audio "bgm_space" time 90>>
```

Using the `load` and `unload` actions

> **WARNING:** Be *very careful* with these if your audio sources are on the network, as you are forcing players to begin downloading them. Not everyone has blazing fast internet with unlimited data—especially true for mobile users. Pease, ***do not*** take your players' bandwidth and data usage lightly.

```
→ If it's not currently loading, drop existing data buffers and load the track
<<audio "bgm_space" load>>

→ Unload the track, dropping existing data buffers
<<audio "bgm_space" unload>>
```

`<<cacheaudio trackId sourceList>>`

Caches an audio track for use by the other audio macros.

> **NOTE:** The `StoryInit` special passage is normally the best place to set up tracks.

History:

- `v2.0.0`: Introduced.
- `v2.28.0`: Deprecated the old optional format specifier syntax in favor of a new syntax (`formatId|`).

Arguments:

- **`trackId`:** The ID of the track, which will be used to reference it.
- **`sourceList`:** A space separated list of sources for the track. Only one is required, though supplying additional sources in differing formats is recommended, as no single format is supported by all browsers. A source must be either a URL (absolute or relative) to an audio resource, the name of an audio passage, or a data URI. In rare cases where the audio format cannot be automatically detected from the source (URLs are parsed for a file extension, data URIs are parsed for the media type), a format specifier may be prepended to the front of each source to manually specify the format (syntax: `formatId|`, where `formatId` is the audio format—generally, whatever the file extension would normally be; e.g., `mp3`, `mp4`, `ogg`, `weba`, `wav`).

Examples:

```
→ Cache a track with the ID "boom" and one source via relative URL
<<cacheaudio "boom" "media/audio/explosion.mp3">>

→ Cache a track with the ID "boom" and one source via audio passage
<<cacheaudio "boom" "explosion">>

→ Cache a track with the ID "bgm_space" and two sources via relative URLs
<<cacheaudio "bgm_space" "media/audio/space_quest.mp3" "media/audio/space_quest.ogg

→ Cache a track with the ID "what" and one source via URL with a format specifier
<<cacheaudio "what" "mp3|http://an-audio-service.com/a-user/a-track-id">>
```

`<<createaudiogroup groupId>>` … `<</createaudiogroup>>`

Collects tracks, which must be set up via `<<cacheaudio>>`, into a group via its `<<track>>` children. Groups are useful for applying actions to multiple tracks simultaneously and/or excluding the included tracks from a larger set when applying actions.

> **NOTE:** The `StoryInit` special passage is normally the best place to set up groups.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.19.0`: Introduced.

Arguments:

`<<createaudiogroup>>`

- **`groupId`:** The ID of the group that will be used to reference it and *must* begin with a colon. **NOTE:** There are several predefined group IDs (`:all`, `:looped`, `:muted`, `:paused`, `:playing`) and the `:not` group modifier that cannot be reused/overwritten.

`<<track>>`

- **`trackId`:** The ID of the track.

Examples:

```
→ Given the following (best done in the StoryInit special passage)
<<cacheaudio "ui_beep"  "media/audio/ui/beep.mp3">>
<<cacheaudio "ui_boop"  "media/audio/ui/boop.mp3">>
<<cacheaudio "ui_swish" "media/audio/ui/swish.mp3">>

→ Set up a group ":ui" with the tracks: "ui_beep", "ui_boop", and "ui_swish"
<<createaudiogroup ":ui">>
        <<track "ui_beep">>
        <<track "ui_boop">>
        <<track "ui_swish">>
<</createaudiogroup>>
```

`<<createplaylist listId>>` … `<</createplaylist>>`

`[<<track trackId actionList>> …]`

Collects tracks, which must be set up via `<<cacheaudio>>`, into a playlist via its `<<track>>` children.

> **NOTE:** The `StoryInit` special passage is normally the best place to set up playlists.

History:

- `v2.8.0`: Introduced.
- `v2.29.0`: Deprecated `<<track>>` `copy` keyword in favor of `own`.

Arguments:

`<<createplaylist>>`

- **`listId`:** The ID of the playlist, which will be used to reference it.

`<<track>>`

- **`trackId`:** The ID of the track.
- **`actionList`:** The list of actions to perform. Available actions are:
  - **`volume` *`level`*:** (optional) Set the base volume of the track within the playlist to the specified level. If omitted, defaults to the track's current volume. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.
  - **`own`:** (optional) Keyword, used to signify that the playlist should create its own independent copy of the track, rather than simply referencing the existing version. Owned copies are solely under the control of their playlist—meaning `<<audio>>` actions cannot affect them, even when using group IDs.

Examples:

```
→ Given the following setup (best done in the StoryInit special passage)
<<cacheaudio "swamped"       "media/audio/Swamped.mp3">>
<<cacheaudio "heavens_a_lie" "media/audio/Heaven's_A_Lie.mp3">>
<<cacheaudio "closer"        "media/audio/Closer.mp3">>
<<cacheaudio "to_the_edge"   "media/audio/To_The_Edge.mp3">>

→ Create a playlist "bgm_lacuna" with the tracks: "swamped", "heavens_a_lie", "clos
<<createplaylist "bgm_lacuna">>
        <<track "swamped"       volume 1>>      → Add "swamped" at 100% volume
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        <<track "heavens_a_lie" volume 0.5>>    → Add "heavens_a_lie" at 50% volume
        <<track "closer"        own>>           → Add an owned copy of "closer" at
        <<track "to_the_edge"   volume 1 own>>  → Add an owned copy of "to_the_edge
<</createplaylist>>
```

<<masteraudio actionList>>

Controls the master audio settings.

> **SEE:** Audio macro limitations.

History:

- `v2.8.0`: Introduced.
- `v2.28.0`: Added `load`, `muteonhide`, `nomuteonhide`, and `unload` actions.

Arguments:

- `actionList`: The list of actions to perform. Available actions are:
  - `load`: Pause playback of *all* tracks and, if they're not already in the process of loading, force them to drop any existing data and begin loading. **NOTE:** This *should not* be done lightly if your audio sources are on the network, as it forces the player to begin downloading them.
  - `mute`: Mute the master volume (effectively volume `0`, except without changing the volume level).
  - `muteonhide`: Enable automatic muting of the master volume when losing visibility—i.e., when switched to another tab or the browser window is minimized.
  - `nomuteonhide`: Disable automatic muting of the master volume when losing visibility (this is the default).
  - `stop`: Stop playback of *all* tracks.
  - `unload`: Stop playback of *all* tracks and force them to drop any existing data. **NOTE:** Once unloaded, playback cannot occur until a `load` action is issued for each track—either a master `load` action, to affect all tracks, or an `<<audio>>`/`<<playlist>>` `load` action, to affect only certain tracks.
  - `unmute`: Unmute the master volume (this is the default).
  - `volume` *`level`*: Set the master volume to the specified level. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Examples:

Basic usage

```
→ Stop playback of all registered tracks, no exceptions
<<masteraudio stop>>

→ Change the master volume to 40%
<<masteraudio volume 0.40>>

→ Mute the master volume
<<masteraudio mute>>

→ Unmute the master volume
<<masteraudio unmute>>

→ Enable automatic muting of the master volume when losing visibility
<<masteraudio muteonhide>>

→ Disable automatic muting of the master volume when losing visibility
<<masteraudio nomuteonhide>>
```

Using the `load` and `unload` actions

> **WARNING:** Be *very careful* with these if your audio sources are on the network, as you are forcing players to begin downloading them. Not everyone has blazing fast internet with unlimited data—especially true for mobile users. Pease, ***do not*** take your players' bandwidth and data usage lightly.

```
→ If they're not currently loading, drop existing data buffers and load all tracks
<<masteraudio load>>

→ Unload all tracks, dropping existing data buffers
<<masteraudio unload>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`<<playlist listId actionList>>`
~~`<<playlist actionList>>`~~

Controls the playback of the playlist, which must be set up via `<<createplaylist>>`—the deprecated `<<setplaylist>>` may be used instead, though it is not recommended.

> **SEE:** Audio macro limitations.

> **NOTE:** The `Config.audio.pauseOnFadeToZero` setting (default: `true`) controls whether tracks that have been faded to `0` volume (silent) are automatically paused.

History:

- `v2.0.0`: Introduced, compatible with `<<setplaylist>>`.
- `v2.1.0`: Added `fadeoverto` action.
- `v2.8.0`: Added `listId` argument, compatible with `<<createplaylist>>`. Deprecated `<<setplaylist>>` compatible form.
- `v2.28.0`: Added `load` and `unload` actions.

Arguments:

`<<createplaylist>>`-compatible form

- `listId`: The ID of the playlist.
- `actionList`: The list of actions to perform. Available actions are:
  - `fadein`: Start playback of the playlist and fade the current track from its current volume level to `1` (loudest) over `5` seconds.
  - `fadeout`: Start playback of the playlist and fade the current track from its current volume level to `0` (silent) over `5` seconds.
  - `fadeoverto` *`seconds`* *`level`*: Start playback of the playlist and fade the current track from its current volume level to the specified level over the specified number of seconds.
  - `fadeto` *`level`*: Start playback of the playlist and fade the current track from its current volume level to the specified level over `5` seconds.
  - `load`: Pause playback of the playlist and, if its tracks are not already in the process of loading, force them to drop any existing data and begin loading. **NOTE:** This *should not* be done lightly if your audio sources are on the network, as it forces the player to begin downloading them.
  - `loop`: Set the playlist to repeat playback upon ending.
  - `mute`: Mute the volume of the playlist (effectively volume `0`, except without changing the volume level).
  - `pause`: Pause playback of the playlist.
  - `play`: Start playback of the playlist.
  - `shuffle`: Set the playlist to randomly shuffle.
  - `skip`: Skip ahead to the next track in the queue. An empty queue will not be refilled unless repeat playback has been set.
  - `stop`: Stop playback of the playlist.
  - `unload`: Stop playback of the playlist and force its tracks to drop any existing data. **NOTE:** Once unloaded, playback cannot occur until a `load` action is issued.
  - `unloop`: Set the playlist to not repeat playback (this is the default).
  - `unmute`: Unmute the volume of the playlist (this is the default).
  - `unshuffle`: Set the playlist to not randomly shuffle (this is the default).
  - `volume` *`level`*: Set the volume of the playlist to the specified level. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

`<<setplaylist>>`-compatible form

- `actionList`: *Identical to the `<<createplaylist>>`-compatible form.*

Examples: *(only `<<createplaylist>>`-compatible form shown)*

Basic usage

```
→ Given the following (best done in the StoryInit special passage)
<<cacheaudio "swamped"       "media/audio/Swamped.mp3">>
<<cacheaudio "heavens_a_lie" "media/audio/Heaven's_A_Lie.mp3">>
<<cacheaudio "closer"        "media/audio/Closer.mp3">>
<<cacheaudio "to_the_edge"   "media/audio/To_The_Edge.mp3">>
<<createplaylist "bgm_lacuna">>
        <<track "swamped"       volume 1>>
        <<track "heavens_a_lie" volume 1>>
        <<track "closer"        volume 1>>
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        <<track "to_the_edge"   volume 1>>
<</createplaylist>>

→ Start playback
<<playlist "bgm_lacuna" play>>

→ Start playback at 50% volume
<<playlist "bgm_lacuna" volume 0.5 play>>

→ Start non-repeating playback
<<playlist "bgm_lacuna" unloop play>>

→ Start playback with a randomly shuffled playlist
<<playlist "bgm_lacuna" shuffle play>>

→ Start playback and fade from 0% to 100% volume
<<playlist "bgm_lacuna" volume 0 fadein>>

→ Start playback and fade from 75% to 0% volume
<<playlist "bgm_lacuna" volume 0.75 fadeout>>

→ Start playback and fade from 25% to 75% volume
<<playlist "bgm_lacuna" volume 0.25 fadeto 0.75>>

→ Start playback and fade from 25% to 75% volume over 30 seconds
<<playlist "bgm_lacuna" volume 0.25 fadeoverto 30 0.75>>

→ Pause playback
<<playlist "bgm_lacuna" pause>>

→ Stop playback
<<playlist "bgm_lacuna" stop>>

→ Mute playback, without changing the current playback state
<<playlist "bgm_lacuna" mute>>

→ Unmute playback, without changing the current playback state
<<playlist "bgm_lacuna" unmute>>

→ Change the volume to 40%, without changing the current playback state
<<playlist "bgm_lacuna" volume 0.40>>

→ Set the playlist to randomly shuffle, without changing the current playback state
<<playlist "bgm_lacuna" shuffle>>
```

Using the `load` and `unload` actions

> **WARNING:** Be *very careful* with these if your audio sources are on the network, as you are forcing players to begin downloading them. Not everyone has blazing fast internet with unlimited data—especially true for mobile users. Pease, ***do not*** take your players' bandwidth and data usage lightly.

```
→ If they're not currently loading, drop existing data buffers and load all of the
<<playlist "bgm_lacuna" load>>

→ Unload all of the playlist's tracks, dropping existing data buffers
<<playlist "bgm_lacuna" unload>>
```

`<<removeaudiogroup groupId>>`

Removes the audio group with the given ID.

> **NOTE:** You may not remove the predefined group IDs (`:all`, `:looped`, `:muted`, `:paused`, `:playing`) or the `:not` group modifier.

History:

- `v2.28.0`: Introduced.

Arguments:

- `groupId`: The ID of the group.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
→ Given a group set up via <<createaudiogroup ":ui">>…<</createplaylist>>
<<removeaudiogroup ":ui">>
```

<<removeplaylist listId>>

Removes the playlist with the given ID.

History:

- v2.8.0: Introduced.

Arguments:

- listId: The ID of the playlist.

Examples:

```
→ Given a playlist set up via <<createplaylist "bgm_lacuna">>…<</createplaylist>>
<<removeplaylist "bgm_lacuna">>
```

<<waitforaudio>>

Displays the loading screen until *all* currently registered audio has either loaded to a playable state or aborted loading due to errors. Requires tracks to be set up via <<cacheaudio>>.

> **NOTE:** This macro should be invoked ***once*** following any invocations of <<cacheaudio>> and <<createplaylist>>, if any <<track>> definitions used the copy keyword, for which you want the loading screen displayed.

History:

- v2.8.0: Introduced.

Arguments: *none*

Examples:

Basic usage

```
<<cacheaudio "a" "a_track.…">>
<<cacheaudio "b" "b_track.…">>
<<cacheaudio "c" "c_track.…">>
<<cacheaudio "d" "d_track.…">>
<<waitforaudio>>
```

Load only selected audio at startup

```
→ First, register the tracks that will be needed soon
<<cacheaudio "a" "a_track.…">>
<<cacheaudio "b" "b_track.…">>

→ Next, load all currently registered tracks (meaning: "a" and "b")
<<waitforaudio>>

→ Finally, register any tracks that won't be needed until later
<<cacheaudio "c" "c_track.…">>
<<cacheaudio "d" "d_track.…">>
```

~~<<setplaylist trackIdList>>~~

> **DEPRECATED:** This macro has been deprecated and should no longer be used. See the <<createplaylist>> macro for its replacement.

History:

- v2.0.0: Introduced.
- v2.8.0: Deprecated in favor of <<createplaylist>>.

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

~~<<stopallaudio>>~~

DEPRECATED: This macro has been deprecated and should no longer be used. See the `<<audio>>` macro for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.8.0`: Deprecated in favor of `<<audio ":all" stop>>`.

Miscellaneous Macros

<<done>> … <</done>>

Silently executes its contents when the incoming passage is done rendering and has been added to the page. Generally, only really useful for running code that needs to manipulate elements from the incoming passage, since you must wait until they've been added to the page.

TIP: If you need to run the same code on multiple passages, consider using the `PassageDone` special passage or, for a JavaScript/TwineScript solution, a `:passagedisplay` event instead. They serve the same basic purpose as the `<<done>>` macro, but are run each time passage navigation occurs.

History:

- `v2.35.0`: Introduced.
- `v2.36.0`: Changed delay mechanism to improve waiting on the DOM.

Arguments: *none*

Examples:

```
@@#spy;@@

<<done>>
        <<replace "#spy">>I spy with my little eye, a crab rangoon.<</replace>>
<</done>>
```

<<goto passageName>>
<<goto linkMarkup>>

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

NOTE: In most cases, you will not need to use `<<goto>>` as there are often better and easier ways to forward the player. For example, a common use of `<<link>>` is to perform various actions before forwarding the player to another passage. In that case, unless you need to dynamically determine the destination passage within the `<<link>>` body, `<<goto>>` is unnecessary as `<<link>>` already includes the ability to forward the player.

WARNING: Using `<<goto>>` to automatically forward players from one passage to another with no input from them will both create junk moments within the story history and make it extremely difficult for players to navigate the history. It is ***strongly*** recommended that you look into other methods to achieve your goals instead—e.g., `Config.navigation.override`.

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

History:

- `v2.0.0`: Introduced.

Arguments:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

Passage name form

- `passageName`: The name of the passage to go to.

Link markup form

- `linkMarkup`: The link markup to use (regular syntax only, no setters).

Examples:

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

→ Link markup form
<<goto [[Somewhere over yonder]]>>
<<goto [[$selectedPassage]]>>
```

<<repeat delay [transition|t8n]>> … <</repeat>>

Repeatedly executes its contents after the given delay, inserting any output into the passage in its place. May be terminated by a `<<stop>>` macro.

> **NOTE:** Passage navigation terminates all pending timed executions.

History:

- `v2.0.0`: Introduced.

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`.

Examples:

```
→ 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<</repla
                <<else>>
                        <<replace "#countdown">>Too Late<</replace>>
                        /* do something useful here */
                        <<stop>>
                <</if>>
        <</repeat>>
<</silently>>
```

<<stop>>

Used within `<<repeat>>` macros. Terminates the execution of the current `<<repeat>>`.

History:

- `v2.0.0`: Introduced.

Arguments: *none*

<<timed delay [transition|t8n]>> …

[<<next [delay]>> …]

<</timed>>

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Executes its contents after the given delay, inserting any output into the passage in its place. Additional timed executions may be chained via `<<next>>`.

> **NOTE:** Passage navigation terminates all pending timed executions.

History:

- `v2.0.0`: Introduced.

Arguments:

`<<timed>>`

- **`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`.

`<<next>>`

- **`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.

Examples:

```
→ 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>>

→ 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>>

→ 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>>

→ Replace some text with a variable interval
→ Given: _delay is "2s" the interval will be 2 seconds
I'll have <span id="drink">some water</span>, please.\
<<timed _delay>><<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>>
```

`<<widget widgetName [container]>> … <</widget>>`

Creates a new widget macro (henceforth, widget) with the given name. Widgets allow you to create macros by using the standard macros and markup that you use normally within your story. All widgets may access arguments passed to them via the `_args` special variable. Block widgets may access the contents they enclose via the `_contents` special variable.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

WARNING: Widgets should *always* be defined within a `widget`-tagged passage—any widgets that are not may be lost on page reload—and you may use as few or as many such passages as you desire. *Do not* add a `widget` tag to any of the specially named passages and attempt to define your widgets there.

WARNING: The array-like object stored in the `_args` variable should be treated as though it were immutable—i.e., unable to be modified—because in the future it will be made thus, so any attempt to modify it will cause an error.

History:

- `v2.0.0`: Introduced.
- `v2.36.0`: Added the `container` keyword, `_args` variable, and `_contents` variable. Deprecated the `$args` variable in favor of `_args`.

Arguments:

- **`widgetName`:** The name of the created widget, which should not contain whitespace or angle brackets (`<`, `>`). If the name of an existing widget is chosen, the new widget will overwrite the older version.
  NOTE: The names of existing macros are invalid widget names and any attempts to use such a name will cause an error.
- **`container`:** (optional) Keyword, used to signify that the widget should be created as a container widget—i.e., non-void, requiring a closing tag; e.g., `<<foo>>…<</foo>>`.

Special variables, `_args` & `_contents`:

The `_args` special variable is used internally to store arguments passed to the widget—as zero-based indices; i.e., `_args[0]` is the first parsed argument, `_args[1]` is the second, etc—and the full argument string in raw and parsed forms—accessed via the `_args.raw` and `_args.full` properties.

The `_contents` special variable is used internally, by container widgets, to store the contents they enclose.

When a widget is called, any existing `_args` variable, and for container widgets `_contents`, is stored for the duration of the call and restored after. This means that non-widget uses of these special variable are completely safe, though this does have the effect that uses external to widgets are inaccessible within them unless passed in as arguments.

WARNING: Unless localized by use of the `<<capture>>` macro, any story or other temporary variables used within widgets are part of a story's normal variable store, so care *must be* taken not to accidentally either overwrite or pick up an existing value.

Examples:

NOTE: No line-break control mechanisms are used in the following examples for readability. In practice, you'll probably want to use either line continuations or one of the no-break methods: `Config.passages.nobr` setting, `nobr` special tag, `<<nobr>>` macro.

Basic usage (non-container)

```
→ Creating a gender pronoun widget
<<widget "he">>
        <<if $pcSex eq "male">>
                he
        <<elseif $pcSex eq "female">>
                she
        <<else>>
                it
        <</if>>
<</widget>>

→ Using it
"Are you sure that <<he>> can be trusted?"
```

```
→ Creating a silly print widget
<<widget "pm">>
        <<if _args[0]>>
                <<print _args[0]>>
        <<else>>
                Mum's the word!
        <</if>>
<</widget>>

→ Using it
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<pm>>        → Outputs: Mum's the word!
<<pm "Hi!">>  → Outputs: Hi!
```

Basic usage (container)

```
→ Creating a simple dialog box widget
<<widget "say" container>>
        <div class="say-box">
                <img class="say-image" @src="'images/' + _args[0].toLowerCase() +
                <p class="say-text">_contents</p>
        </div>
<</widget>>

→ Using it
<<say "Chapel">>Tweego is a pathway to many abilities some consider to be… unnatura
```

Functions

`clone(original)` → *any*

Returns a deep copy of the given value.

> **NOTE:** Only the primitives, generic objects, some JavaScript natives (specifically: `Array`, `Date`, `Map`, `RegExp`, and `Set`), and DOM node objects are supported by default. Unsupported object types, either native or custom, will need to implement `.clone()` method to be properly supported by the `clone()` function—when called on such an object, it will simply defer to the local method; see the *Non-generic object types (a.k.a. classes)* guide for more information.

History:

- `v2.0.0`: Introduced.

Parameters:

- `original`: (*any*) The object to value.

Examples:

```
// Without clone(); given the generic object: $foo = { id : 1 }
<<set $bar to $foo>>
<<set $bar.id to 5>>
$foo.id  → Returns: 5
$bar.id  → Returns: 5

// With clone(); given the generic object: $foo = { id : 1 }
<<set $bar to clone($foo)>>
<<set $bar.id to 5>>
$foo.id  → Returns: 1
$bar.id  → Returns: 5
```

`either(list…)` → *any*

Returns a random value from its given arguments.

History:

- `v2.0.0`: Introduced.

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. **NOTE:** Does not flatten nested arrays—if this is required, the `<Array>.flat()` method may be used to flatten the nested arrays prior to passing them to `either()`.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
// 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: $letters = ["A", "B"]
either($letters, "C", "D")  → Returns a random value from the whole list—i.e., "A"

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

forget(key)

Removes the specified key, and its associated value, from the story metadata store.

SEE ALSO: memorize(), recall().

History:

- v2.29.0: Introduced.

Parameters:

- key: (*string*) The key to remove.

Examples:

```
<<run forget('achievements')>>
```

hasVisited(passages…) → *boolean*

Returns whether the passage with the given title occurred within the story history. If multiple passage titles are given, returns the logical-AND aggregate of the set—i.e., true if all were found, false if any were not found.

History:

- v2.7.0: Introduced.

Parameters:

- passages: (*string* | *Array*<*string*>) The title(s) of the passage(s) to search for. May be a list or an array of passages.

Examples:

```
<<if hasVisited("Bar")>>…has been to the Bar…<</if>>
<<if not hasVisited("Bar")>>…has never been to the Bar…<</if>>
<<if hasVisited("Bar", "Café")>>…has been to both the Bar and Café<</if>>
<<if not hasVisited("Bar", "Café")>>…has never been to either the Bar, Café, or bo
```

lastVisited(passages…) → *integer*

Returns the number of turns that have passed since the last instance of the passage with the given title occurred within the story history or -1 if it does not exist. If multiple passage titles are given, returns the lowest count (which can be -1).

History:

- v2.0.0: Introduced.

Parameters:

- passages: (*string* | *Array*<*string*>) The title(s) of the passage(s) to search for. May be a list or an array of passages.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<if lastVisited("Bar") is -1>>…has never been to the Bar…<</if>>
<<if lastVisited("Bar") is 0>>…is currently in the Bar…<</if>>
<<if lastVisited("Bar") is 1>>…was in the Bar one turn ago…<</if>>
<<if lastVisited("Bar", "Café") is -1>>…has never been to the Bar, Café, or both…<
<<if lastVisited("Bar", "Café") is 2>>…has been to both the Bar and Café, most rec
```

`importScripts(urls…)` → *`Promise`* *object*

Load and integrate external JavaScript scripts.

> **NOTE:** Loading is done asynchronously at run time, so if the script must be available within a tight time frame, then you should use the `Promise` returned by the function to ensure that the script is loaded before it is needed.

> **NOTE:** Your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage) is normally the best place to call `importScripts()`.

History:

- `v2.16.0`: Introduced.

Parameters:

- `urls`: (*string* | *Array*<*string*>) The URLs of the external scripts to import. Loose URLs are imported concurrently, arrays of URLs are imported sequentially.

Examples:

Basic usage

```
// Import all scripts concurrently
importScripts(
	"https://somesite/a/path/a.js",
	"https://somesite/a/path/b.js",
	"https://somesite/a/path/c.js",
	"https://somesite/a/path/d.js"
);


// Import all scripts sequentially
importScripts([
	"https://somesite/a/path/a.js",
	"https://somesite/a/path/b.js",
	"https://somesite/a/path/c.js",
	"https://somesite/a/path/d.js"
]);

// Import scripts a.js, b.js, and the c.js/d.js group concurrently,
// while importing c.js and d.js sequentially relative to each other
importScripts(
	"https://somesite/a/path/a.js",
	"https://somesite/a/path/b.js",
	[
		"https://somesite/a/path/c.js",
		"https://somesite/a/path/d.js"
	]
);
```

Basic usage with the returned `Promise` object

```
// Import a script while using the returned Promise to ensure that
// the script has been fully loaded before executing dependent code
importScripts("https://somesite/a/path/a.js")
	.then(function () {
		// Code that depends on the script goes here.
	})
	.catch(function (err) {
		// There was an error loading the script, log it to the console.
		console.log(err);
	});
```

Saving the returned `Promise` object for later use

```
// Import a script while saving the returned Promise so it may be used later
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
setup.aScriptImport = importScripts( "https://somesite/a/path/aScript.js" );

// Use the returned Promise later on to ensure that the script has been fully
// loaded before executing dependent code
setup.aScriptImport
	.then(function () {
		// Code that depends on the script goes here.
	})
	.catch(function (err) {
		// There was an error loading the script, log it to the console.
		console.log(err);
	});
```

`importStyles(urls…)` → *`Promise`* *object*

Load and integrate external CSS stylesheets.

> **NOTE:** Loading is done asynchronously at run time, so if the stylesheet must be available within a tight time frame, then you should use the `Promise` returned by the function to ensure that the stylesheet is loaded before it is needed.

> **NOTE:** Your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage) is normally the best place to call `importStyles()`.

History:

- `v2.16.0`: Introduced.

Parameters:

- **`urls`:** (*string* | *Array*<*string*>) The URLs of the external stylesheets to import. Loose URLs are imported concurrently, arrays of URLs are imported sequentially.

Examples:

Basic usage

```
// Import all stylesheets concurrently
importStyles(
	"https://somesite/a/path/a.css",
	"https://somesite/a/path/b.css",
	"https://somesite/a/path/c.css",
	"https://somesite/a/path/d.css"
);

// Import all stylesheets sequentially
importStyles([
	"https://somesite/a/path/a.css",
	"https://somesite/a/path/b.css",
	"https://somesite/a/path/c.css",
	"https://somesite/a/path/d.css"
]);

// Import stylesheets a.css, b.css, and the c.css/d.css group concurrently,
// while importing c.css and d.css sequentially relative to each other
importStyles(
	"https://somesite/a/path/a.css",
	"https://somesite/a/path/b.css",
	[
		"https://somesite/a/path/c.css",
		"https://somesite/a/path/d.css"
	]
);
```

Basic usage with the returned `Promise` object

```
// Grab a loading screen lock
var lsLockId = LoadScreen.lock();

// Import a stylesheet while using the returned Promise to ensure that the
// stylesheet has been fully loaded before unlocking the loading screen
importStyles("https://somesite/a/path/a.css")
	.then(function () {
		// The stylesheet has been loaded, release the loading screen lock
		LoadScreen.unlock(lsLockId);
	})
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        .catch(function (err) {
                // There was an error loading the stylesheet, log it to the consol
                console.log(err);
        });
```

`memorize(key, value)`

Sets the specified key and value within the story metadata store, which causes them to persist over story and browser restarts. To update the value associated with a key, simply set it again.

NOTE: The story metadata, like saves, is tied to the specific story it was generated with. It is not a mechanism for moving data between stories.

WARNING: The story metadata store ***is not***, and should not be used as, a replacement for saves. Examples of good uses: achievement tracking, new game+ data, playthrough statistics, etc.

WARNING: This feature is largely incompatible with private browsing modes, which cause all in-browser storage mechanisms to either persist only for the lifetime of the browsing session or fail outright.

SEE ALSO: `forget()`, `recall()`.

History:

- `v2.29.0`: Introduced.

Parameters:

- `key`: (*string*) The key that should be set.
- `value`: (*any*) The value to set.

Examples:

```
// Sets 'achievements', with the given value, in the metadata store.
<<run memorize('achievements', { ateYellowSnow : true })>>

// Sets 'ngplus', with the given value, in the metadata store.
<<run memorize('ngplus', true)>>
```

`passage()` → *string*

Returns the title of the active (present) passage.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
<<if passage() is "Café">>…the active passage is the Café passage…<</if>>
```

`previous()` → *string*

Returns the title of the most recent previous passage whose title does not match that of the active passage or an empty string, if there is no such passage.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<if previous() is "Café">>…the most recent non-active passage is the Café passage.

→ Commonly used as part of a link to return to the most recent non-active passage
[[Return|previous()]]
```

`random([min ,] max)` → *integer*

Returns a pseudo-random whole number (integer) within the range of the given bounds (inclusive)—i.e., [min, max].

> **NOTE:** By default, it uses `Math.random()` as its source of (non-deterministic) randomness, however, when the seedable PRNG has been enabled, via `State.prng.init()`, it uses that (deterministic) seeded PRNG instead.

History:

- `v2.0.0`: Introduced.

Parameters:

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

Examples:

```
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 decimal number (floating-point) within the range of the given bounds (inclusive for the minimum, exclusive for the maximum)—i.e., [min, max).

> **NOTE:** By default, it simply returns non-deterministic results from `Math.random()`, however, when the seedable PRNG has been enabled, via `State.prng.init()`, it returns deterministic results from the seeded PRNG instead.

History:

- `v2.0.0`: Introduced.

Parameters:

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

Examples:

```
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…
```

`recall(key [, defaultValue])` → *any*

Returns the value associated with the specified key from the story metadata store or, if no such key exists, the specified default value, if any.

> **SEE ALSO:** `forget()`, `memorize()`.

History:

- `v2.29.0`: Introduced.

Parameters:

- `key`: (*string*) The key whose value should be returned.
- `defaultValue`: (optional, *any*) The value to return if the key doesn't exist.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
// Set setup.achievements to the 'achievements' metadata or an empty generic objec
<<set setup.achievements to recall('achievements', {})>>

// Set setup.ngplus to the 'ngplus' metadata, with no default.
<<set setup.ngplus to recall('ngplus')>>
```

`setPageElement(idOrElement , passages [, defaultText])` → *`HTMLElement`* ***object | null***

Renders the selected passage into the target element, replacing any existing content, and returns the element. If no passages are found and default text is specified, it will be used instead.

History:

- `v2.0.0`: Introduced.

Parameters:

- `idOrElement`: (*string* | `HTMLElement` *object*) The ID of the element or the element itself.
- `passages`: (*string* | *Array<string>*) The name(s) of the passage(s) to search for. May be a single passage or an array of passages. If an array of passage names is specified, the first passage to be found is used.
- `defaultText`: (optional, *string*) The default text to use if no passages are found.

Examples:

> **NOTE:** As it is highly unlikely that either an array of passage names or default text will be needed in the vast majority of cases, only a few basic examples will be given.

```
// Using an ID; given an existing element on the page: <div id="my-display"></div>
setPageElement("my-display", "MyPassage");

// Using an element; given a reference to an existing element: myElement
setPageElement(myElement, "MyPassage");
```

`tags([passages…])` → ***Array<string>***

Returns a new array consisting of all of the tags of the given passages.

History:

- `v2.0.0`: Introduced.

Parameters:

- `passages`: (optional, *string* | *Array<string>*) The passages from which to collect tags. May be a list or an array of passages. If omitted, will default to the active (present) passage—included passages do not count for this purpose; e.g., passages pulled in via `<<include>>`, `PassageHeader`, etc.

Examples:

```
<<if tags().includes("forest")>>…the active passage is part of the forest…<</if>>
<<if tags("Lonely Glade").includes("forest")>>…the Lonely Glade passage is part of
```

`temporary()` → ***object***

Returns a reference to the current temporary variables store (equivalent to: `State.temporary`). This is only really useful within pure JavaScript code, as within TwineScript you may simply access temporary variables natively.

History:

- `v2.19.0`: Introduced.

Parameters: *none*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
// Given: _selection is 'Zagnut Bar'
if (temporary().selection === 'Zagnut Bar') {
        /* Do something… */
}
```

`time()` → *integer*

Returns the number of milliseconds that have passed since the current passage was rendered to the page.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
→ Links that vary based on the time
In the darkness, something wicked this way comes.  Quickly!  Do you \
<<link "try to run back into the light">>
        <<if time() lt 5000>>
                /% The player clicked the link in under 5s, so they escape %/
                <<goto "Well lit passageway">>
        <<else>>
                /% Else, they're eaten by a grue %/
                <<goto "Eaten by a grue">>
        <</if>>
<</link>> \
or [[stand your ground|Eaten by a grue]]?
```

`turns()` → *integer*

Returns the total number (count) of played turns currently in effect—i.e., the number of played moments up to the present moment; future (rewound/undone) moments are not included within the total.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
<<print "This is turn #" + turns()>>
```

`variables()` → *object*

Returns a reference to the active (present) story variables store (equivalent to: `State.variables`). This is only really useful within pure JavaScript code, as within TwineScript you may simply access story variables natively.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $hasGoldenKey is true
if (variables().hasGoldenKey) {
        /* Do something… */
}
```

`visited([passages…])` → *integer*

Returns the number of times that the passage with the given title occurred within the story history. If multiple passage titles are given, returns the lowest count.

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Parameters:

- **`passages`:** (optional, *string* | *Array<string>*) The title(s) of the passage(s) to search for. May be a list or an array of passages. If omitted, will default to the current passage.

Examples:

```
<<if visited() is 3>>…this is the third visit to the current passage…<</if>>
<<if visited("Bar")>>…has been to the Bar at least once…<</if>>
<<if visited("Café") is 1>>…has been to the Café exactly once…<</if>>
<<if visited("Bar", "Café") is 4>>…has been to both the Bar and Café at least four
```

`visitedTags(tags…)` → *integer*

Returns the number of passages within the story history that are tagged with all of the given tags.

History:

- `v2.0.0`: Introduced.

Parameters:

- **`tags`:** (*string* | *Array<string>*) The tags to search for. May be a list or an array of tags.

Examples:

```
<<if visitedTags("forest")>>…has been to some part of the forest at least once…<</
<<if visitedTags("forest", "haunted") is 1>>…has been to the haunted part of the fo
<<if visitedTags("forest", "burned") is 3>>…has been to the burned part of the fore
```

Methods

Most of the methods listed below are SugarCube extensions, with the rest being either JavaScript natives or bundled library methods that are listed here for their utility—though, this is not an exhaustive list.

For more information see:

- MDN's JavaScript reference for native JavaScript object methods—and more.
- jQuery API reference for native jQuery methods.

Additionally. SugarCube includes polyfills for virtually all JavaScript (ECMAScript) 5 & 6 native object methods—via the es5-shim and es6-shim polyfill libraries (shims only, no shams)—so they may be safely used even if your project will be played in ancient browsers that do not natively support them.

Array Methods

`<Array>.concat(members…)` → *Array<any>*

Concatenates one or more members to the end of the base array and returns the result as a new array. Does not modify the original.

History: *native JavaScript method*

Parameters:

- **`members`:** (*any…*) The members to concatenate. Members that are arrays will be merged—i.e., their members will be concatenated, rather than the array itself.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
// Given: $fruits1 = ["Apples", "Oranges"], $fruits2 = ["Pears", "Plums"]
$fruits1.concat($fruits2)            → Returns ["Apples", "Oranges", "Pears", "Plur
$fruits1.concat($fruits2, $fruits2)  → Returns ["Apples", "Oranges", "Pears", "Plur
$fruits1.concat("Pears")             → Returns ["Apples", "Oranges", "Pears"]
$fruits1.concat("Pears", "Pears")    → Returns ["Apples", "Oranges", "Pears", "Pear
$fruits1.concat($fruits2, "Pears")   → Returns ["Apples", "Oranges", "Pears", "Plur
```

`<Array>.concatUnique(members…)` → *Array<any>*

Concatenates one or more unique members to the end of the base array and returns the result as a new array. Does not modify the original.

History:

- `v2.21.0`: Introduced.

Parameters:

- `members`: (*any…*) The members to concatenate. Members that are arrays will be merged—i.e., their members will be concatenated, rather than the array itself.

Examples:

```
// Given: $fruits1 = ["Apples", "Oranges"], $fruits2 = ["Pears", "Plums"]
$fruits1.concatUnique($fruits2)            → Returns ["Apples", "Oranges", "Pears"
$fruits1.concatUnique($fruits2, $fruits2)  → Returns ["Apples", "Oranges", "Pears"
$fruits1.concatUnique("Pears")             → Returns ["Apples", "Oranges", "Pears"
$fruits1.concatUnique("Pears", "Pears")    → Returns ["Apples", "Oranges", "Pears"
$fruits1.concatUnique($fruits2, "Pears")   → Returns ["Apples", "Oranges", "Pears"
```

`<Array>.count(needle [, position])` → *integer*

Returns the number of times that the given member was found within the array, starting the search at `position`.

History:

- `v2.0.0`: Introduced.

Parameters:

- `needle`: (*any*) The member to count.
- `position`: (optional, *integer*) The zero-based index at which to begin searching for `needle`. If omitted, will default to `0`.

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Plums", "Oranges"]
$fruits.count("Oranges")     → Returns 2
$fruits.count("Oranges", 2)  → Returns 1
```

`<Array>.countWith(predicate [, thisArg])` → *integer*

Returns the number of times that members within the array pass the test implemented by the given predicate function.

History:

- `v2.36.0`: Introduced.

Parameters:

- `predicate`: (*function*) The function used to test each member. It is called with three arguments:
  - `value`: (*any*) The member being processed.
  - `index`: (optional, *integer*) The index of member being processed.
  - `array`: (optional, *array*) The array being processed.
- `thisArg`: (optional, *any*) The value to use as `this` when executing `predicate`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Plums", "Oranges"]
$fruits.countWith(function (fruit) { return fruit === "Oranges"; })  → Returns 2
```

```
// Given: $numbers = [1, 2.3, 4, 76, 3.1]
$numbers.countWith(Number.isInteger)  → Returns 3
```

```
// Given: $items = [
//      { name : 'Healing potion', kind : 'potion' },
//      { name : 'Longsword', kind : 'weapon' },
//      { name : 'Mana potion', kind : 'potion' },
//      { name : 'Dead rat', kind : 'junk' },
//      { name : 'Endurance potion', kind : 'potion' },
//      { name : 'Shortbow', kind : 'weapon' }
// ]
$items.countWith(function (item) { return item.kind === 'junk'; })  → Returns 1
```

<Array>.delete(needles…) → *Array<any>*

Removes all instances of the given members from the array and returns a new array containing the removed members.

History:

- v2.5.0: Introduced.

Parameters:

- needles: (*any*... | *Array<any>*) The members to remove. May be a list of members or an array.

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Plums", "Oranges"]
$fruits.delete("Oranges")          → Returns ["Oranges", "Oranges"]; $fruits ["App
$fruits.delete("Apples", "Plums")  → Returns ["Apples", "Plums"]; $fruits ["Orange
```

<Array>.deleteAt(indices…) → *Array<any>*

Removes all of the members at the given indices from the array and returns a new array containing the removed members.

History:

- v2.5.0: Introduced.

Parameters:

- indices: (*integer*... | *Array<integer>*) The indices of the members to remove. May be a list or array of indices.

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Plums", "Oranges"]
$fruits.deleteAt(2)     → Returns ["Plums"]; $fruits ["Apples", "Oranges", "Orange
$fruits.deleteAt(1, 3)  → Returns ["Oranges", "Oranges"]; $fruits ["Apples", "Plum
$fruits.deleteAt(0, 2)  → Returns ["Apples", "Plums"]; $fruits ["Oranges", "Orange
```

<Array>.deleteWith(predicate [, thisArg]) → *Array<any>*

Removes all of the members from the array that pass the test implemented by the given predicate function and returns a new array containing the removed members.

History:

- v2.25.0: Introduced.

Parameters:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `predicate`: (*function*) The function used to test each member. It is called with three arguments:
  - `value`: (*any*) The member being processed.
  - `index`: (optional, *integer*) The index of member being processed.
  - `array`: (optional, *array*) The array being processed.
- `thisArg`: (optional, *any*) The value to use as `this` when executing `predicate`.

Examples:

```
// Given: $fruits = ["Apples", "Apricots", "Oranges"]

→ Returns ["Apricots"]; $fruits ["Apples", "Oranges"]
$fruits.deleteWith(function (val) {
        return val === "Apricots";
})

→ Returns ["Apples", "Apricots"]; $fruits ["Oranges"]
$fruits.deleteWith(function (val) {
        return val.startsWith("Ap");
})

// Given: $fruits = [{ name : "Apples" }, { name : "Apricots" }, { name : "Oranges'

→ Returns [{ name : "Apricots" }]; $fruits [{ name : "Apples" }, { name : "Oranges'
$fruits.deleteWith(function (val) {
        return val.name === "Apricots";
})

→ Returns [{ name : "Apples" }, { name : "Apricots" }]; $fruits [{ name : "Oranges'
$fruits.deleteWith(function (val) {
        return val.name.startsWith("Ap");
})
```

`<Array>.first()` → *any*

Returns the first member from the array. Does not modify the original.

History:

- `v2.27.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.first()  → Returns "Blueberry"
```

`<Array>.flat(depth)` → *Array<any>*

Returns a new array consisting of the source array with all sub-array elements concatenated into it recursively up to the given depth. Does not modify the original.

History: *native JavaScript method*

Parameters:

- `depth`: (optional, *integer*) The number of nested array levels should be flattened. If omitted, will default to `1`.

Examples:

```
// Given: $npa = [["Alfa", "Bravo"], [["Charlie", "Delta"], ["Echo"]], "Foxtrot"]

$npa.flat()   → Returns ["Alfa", "Bravo", ["Charlie", "Delta"], ["Echo"], "Foxtrot'
$npa.flat(1)  → Returns ["Alfa", "Bravo", ["Charlie", "Delta"], ["Echo"], "Foxtrot'
$npa.flat(2)  → Returns ["Alfa", "Bravo", "Charlie", "Delta", "Echo", "Foxtrot"]
```

`<Array>.flatMap(callback [, thisArg])` → *Array<any>*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Returns a new array consisting of the result of calling the given mapping function on every element in the source array and then concatenating all sub-array elements into it recursively up to a depth of `1`. Does not modify the original.

> **NOTE:** Identical to calling `<Array>.map(…).flat()`.

History: *native JavaScript method*

Parameters:

- `callback`: (*function*) The function used to produce members of the new array. It is called with three arguments:
  - `value`: (*any*) The member being processed.
  - `index`: (optional, *integer*) The index of member being processed.
  - `array`: (optional, *array*) The array being processed.
- `thisArg`: (optional, *any*) The value to use as `this` when executing `callback`.

Examples:

```
// Given: $npa = ["Alfa", "Bravo Charlie", "Delta Echo Foxtrot"]

→ Returns ["Alfa", "Bravo", "Charlie", "Delta", "Echo", "Foxtrot"]
$npa.flatMap(function (val) {
        return val.split(" ");
})
```

`<Array>.includes(needle [, position])` → *boolean*

Returns whether the given member was found within the array, starting the search at `position`.

History: *native JavaScript method*

Parameters:

- `needle`: (*any*) The member to find.
- `position`: (optional, *integer*) The zero-based index at which to begin searching for `needle`. If omitted, will default to `0`.

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
<<if $pies.includes("Cherry")>>…found Cherry pie…<</if>>
<<if $pies.includes("Pecan", 3)>>…found Pecan pie within ["Pecan", "Pumpkin"]…<</i
```

`<Array>.includesAll(needles…)` → *boolean*

Returns whether all of the given members were found within the array.

History:

- `v2.10.0`: Introduced.

Parameters:

- `needles`: (*any*… | *Array*<*any*>) The members to find. May be a list of members or an array.

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
<<if $pies.includesAll("Cherry", "Pecan")>>…found Cherry and Pecan pies…<</if>>

// Given: $search = ["Blueberry", "Pumpkin"]
<<if $pies.includesAll($search)>>…found Blueberry and Pumpkin pies…<</if>>
```

`<Array>.includesAny(needles…)` → *boolean*

Returns whether any of the given members were found within the array.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.10.0`: Introduced.

Parameters:

- `needles`: (*any*... | *Array*<*any*>) The members to find. May be a list of members or an array.

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
<<if $pies.includesAny("Cherry", "Pecan")>>…found Cherry or Pecan pie…<</if>>

// Given: $search = ["Blueberry", "Pumpkin"]
<<if $pies.includesAny($search)>>…found Blueberry or Pumpkin pie…<</if>>
```

`<Array>.last()` → *any*

Returns the last member from the array. Does not modify the original.

History:

- `v2.27.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.last()  → Returns "Pumpkin"
```

`<Array>.pluck()` → *any*

Removes and returns a random member from the base array.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.pluck()  → Removes and returns a random pie from the array
```

`<Array>.pluckMany(want)` → *Array*<*any*>

Randomly removes the given number of members from the base array and returns the removed members as a new array.

History:

- `v2.20.0`: Introduced.

Parameters:

- `want`: (optional, *integer*) The number of members to pluck. Cannot pluck more members than the base array contains.

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.pluckMany(3)  → Removes three random pies from the array and returns them as
```

`<Array>.pop()` → *any*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Removes and returns the last member from the array, or `undefined` if the array is empty.

History: *native JavaScript method*

Parameters: *none*

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Pears"]
$fruits.pop()  → Returns "Pears"; $fruits ["Apples", "Oranges"]
```

`<Array>.push(members…)` → *number*

Appends one or more members to the end of the base array and returns its new length.

History: *native JavaScript method*

Parameters:

- `members`: (*any*...) The members to append.

Examples:

```
// Given: $fruits = ["Apples", "Oranges"]
$fruits.push("Apples")  → Returns 3; $fruits ["Apples", "Oranges", "Apples"]

// Given: $fruits = ["Apples", "Oranges"]
$fruits.push("Plums", "Plums")  → Returns 4; $fruits ["Apples", "Oranges", "Plums"
```

`<Array>.pushUnique(members…)` → *number*

Appends one or more unique members to the end of the base array and returns its new length.

History:

- `v2.21.0`: Introduced.

Parameters:

- `members`: (*any*...) The members to append.

Examples:

```
// Given: $fruits = ["Apples", "Oranges"]
$fruits.pushUnique("Apples")  → Returns 2; $fruits ["Apples", "Oranges"]

// Given: $fruits = ["Apples", "Oranges"]
$fruits.pushUnique("Plums", "Plums")  → Returns 3; $fruits ["Apples", "Oranges", "
```

`<Array>.random()` → *any*

Returns a random member from the base array. Does not modify the original.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.random()  → Returns a random pie from the array
```

`<Array>.randomMany(want)` → *Array<any>*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Randomly selects the given number of unique members from the base array and returns the selected members as a new array. Does not modify the original.

History:

- `v2.20.0`: Introduced.

Parameters:

- `want`: (optional, *integer*) The number of members to select. Cannot select more members than the base array contains.

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.randomMany(3)  → Returns a new array containing three unique random pies fro
```

`<Array>.shift()` → *any*

Removes and returns the first member from the array, or `undefined` if the array is empty.

History: *native JavaScript method*

Parameters: *none*

Examples:

```
// Given: $fruits = ["Apples", "Oranges", "Pears"]
$fruits.shift()  → Returns "Apples"; $fruits ["Oranges", "Pears"]
```

`<Array>.shuffle()` → *Array<any>*

Randomly shuffles the array.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $pies = ["Blueberry", "Cherry", "Cream", "Pecan", "Pumpkin"]
$pies.shuffle()  → Randomizes the order of the pies in the array
```

`<Array>.unshift(members…)` → *number*

Prepends one or more members to the beginning of the base array and returns its new length.

History: *native JavaScript method*

Parameters:

- `members`: (*any*…) The members to append.

Examples:

```
// Given: $fruits = ["Oranges", "Plums"]
$fruits.unshift("Oranges")  → Returns 3; $fruits ["Oranges", "Oranges", "Plums"]

// Given: $fruits = ["Oranges", "Plums"]
$fruits.unshift("Apples", "Apples")  → Returns 4; $fruits ["Apples", "Apples", "Ora
```

`<Array>.unshiftUnique(members…)` → *number*

Prepends one or more unique members to the beginning of the base array and returns its new length.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.21.0`: Introduced.

Parameters:

- **`members`:** (*any*...) The members to append.

Examples:

```
// Given: $fruits = ["Oranges", "Plums"]
$fruits.unshiftUnique("Oranges")  → Returns 2; $fruits ["Oranges", "Plums"]

// Given: $fruits = ["Oranges", "Plums"]
$fruits.unshiftUnique("Apples", "Apples")  → Returns 3; $fruits ["Apples", "Orange
```

~~`<Array>.contains(needle [, position])`~~ → ~~*boolean*~~

DEPRECATED: This method has been deprecated and should no longer be used. See the `<Array>.includes()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.10.0`: Deprecated in favor of `<Array>.includes()`.

~~`<Array>.containsAll(needles…)`~~ → ~~*boolean*~~

DEPRECATED: This method has been deprecated and should no longer be used. See the `<Array>.includesAll()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.10.0`: Deprecated in favor of `<Array>.includesAll()`.

~~`<Array>.containsAny(needles…)`~~ → ~~*boolean*~~

DEPRECATED: This method has been deprecated and should no longer be used. See the `<Array>.includesAny()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.10.0`: Deprecated in favor of `<Array>.includesAny()`.

~~`<Array>.flatten()`~~ → ~~*Array<any>*~~

DEPRECATED: This method has been deprecated and should no longer be used. See the `<Array>.flat()` method for its replacement. The exactly equivalent call is: `<Array>.flat(Infinity)`.

Returns a new array consisting of the flattened source array. Does not modify the original.

History:

- `v2.0.0`: Introduced.
- `v2.29.0`: Deprecated in favor of `<Array>.flat()`.

~~`Array.random(array)`~~ → ~~*any*~~

DEPRECATED: This method has been deprecated and should no longer be used. In general, look to the `<Array>.random()` method instead. If you need a random member from an array-like object, use the `Array.from()` method to convert it to an array, then use `<Array>.random()`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Returns a random member from the array or array-like object. Does not modify the original.

History:

- `v2.0.0`: Introduced.
- `v2.20.0`: Deprecated.

JSON Methods

`JSON.reviveWrapper(codeString [, reviveData])` → *array*

Returns the given code string, and optional data chunk, wrapped within the JSON deserialization revive wrapper. Intended to allow authors to easily wrap their custom object types (a.k.a. classes) revival code and associated data within the revive wrapper, which should be returned from an object instance's `.toJSON()` method, so that the instance may be properly revived upon deserialization.

SEE: The *Non-generic object types (a.k.a. classes)* guide for more detailed information.

History:

- `v2.0.0`: Introduced.
- `v2.9.0`: Added `reviveData` parameter.

Parameters:

- `codeString`: (*string*) The revival code string to wrap.
- `reviveData`: (optional, *any*) The data that should be made available to the evaluated revival code during deserialization via the special `$ReviveData$` variable. **WARNING:** Attempting to pass the value of an object instance's `this` directly as the `reviveData` parameter will trigger out of control recursion in the serializer, so a clone of the instance's own data must be passed instead.

Examples:

```
JSON.reviveWrapper( /* valid JavaScript code string */ );             → Without da
JSON.reviveWrapper( /* valid JavaScript code string */ , myOwnData);  → With data

// E.g., Assume that you're attempting to revive an instance of a custom class nam
//       `Character`, which is assigned to a story variable named `$pc`.  The call
//       to `JSON.reviveWrapper()` might look something like the following.
var ownData = {};
Object.keys(this).forEach(function (pn) { ownData[pn] = clone(this[pn]); }, this);
return JSON.reviveWrapper('new Character($ReviveData$)', ownData);
```

---

jQuery Methods

`<jQuery>.ariaClick([options ,] handler)` → *jQuery object*

Makes the target element(s) WAI-ARIA-compatible clickables—meaning that various accessibility attributes are set and, in addition to mouse clicks, enter/return and spacebar key presses also activate them. Returns a reference to the current `jQuery` object for chaining.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *object*) The options to be used when creating the clickables.
- `handler`: (*function*) The callback to invoke when the target element(s) are activated.

Options object:

An options object should have some of the following properties:

- `namespace`: (*string*) A period-separated list of event namespaces.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `one`: (*boolean*) Whether the clickables are single-use—i.e., the handler callback runs only once and then removes itself. If omitted, defaults to `false`.
- `selector`: (*string*) A selector applied to the target element(s) to filter the descendants that triggered the event. If omitted or `null`, the event is always handled when it reaches the target element(s).
- `data`: (*any*) Data to be passed to the handler in `event.data` when an event is triggered.
- `controls`: (*string*) Value for the `aria-controls` attribute.
- `pressed`: (*string*) Value for the `aria-pressed` attribute (valid values: `"true"`, `"false"`).
- `label`: (*string*) Value for the `aria-label` and `title` attributes.

Examples:

```
// Given an existing element: <a id="so-clicky">Click me</a>
$('#so-clicky').ariaClick(function (event) {
	/* do stuff */
});

// Creates a basic link and appends it to the `output` element
$('<a>Click me</a>')
	.ariaClick(function (event) {
		/* do stuff */
	})
	.appendTo(output);

// Creates a basic button and appends it to the `output` element
$('<button>Click me</button>')
	.ariaClick(function (event) {
		/* do stuff */
	})
	.appendTo(output);

// Creates a link with options and appends it to the `output` element
$('<a>Click me</a>')
	.ariaClick({
		one   : true,
		label : 'This single-use link does stuff.'
	}, function (event) {
		/* do stuff */
	})
	.appendTo(output);
```

`<jQuery>.ariaDisabled(state)` → *`jQuery`* object

Changes the disabled state of the target WAI-ARIA-compatible clickable element(s). Returns a reference to the current `jQuery` object for chaining.

> **NOTE:** This method is meant to work with clickables created via `<jQuery>.ariaClick()` and may not work with clickables from other sources. SugarCube uses `<jQuery>.ariaClick()` internally to handle all of its various link markup and macros.

History:

- `v2.26.0`: Introduced.

Parameters:

- `state`: (*boolean*) The disabled state to apply. Truthy to disable the element(s), falsy to enable them.

Examples:

```
// Given an existing WAI-ARIA-compatible clickable element with the ID "so-clicky"
$('#so-clicky').ariaDisabled(true)   → Disables the target element
$('#so-clicky').ariaDisabled(false)  → Enables the target element
```

`<jQuery>.ariaIsDisabled()` → *`boolean`*

Returns whether any of the target WAI-ARIA-compatible clickable element(s) are disabled.

> **NOTE:** This method is meant to work with clickables created via `<jQuery>.ariaClick()` and may not work with clickables from other sources. SugarCube uses `<jQuery>.ariaClick()` internally to handle all of its various link markup and macros.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.26.0`: Introduced.

Parameters: *none*

Examples:

```
// Given an existing WAI-ARIA-compatible clickable element with the ID "so-clicky"

// If "#so-clicky" is disabled:
$('#so-clicky').ariaIsDisabled()  → Returns true

// If "#so-clicky" is enabled:
$('#so-clicky').ariaIsDisabled()  → Returns false
```

`jQuery.wiki(sources…)`

Wikifies the given content source(s) and discards the result. If there were errors, an exception is thrown. This is only really useful when you want to invoke a macro for its side-effects and aren't interested in its output.

History:

- `v2.17.0`: Introduced.

Parameters:

- `sources`: (*string*...) The list of content sources.

Examples:

```
$.wiki('<<somemacro>>');  → Invokes the <<somemacro>> macro, discarding any output
```

`<jQuery>.wiki(sources…)` → *`jQuery` object*

Wikifies the given content source(s) and appends the result to the target element(s). Returns a reference to the current `jQuery` object for chaining.

History:

- `v2.0.0`: Introduced.

Parameters:

- `sources`: (*string*...) The list of content sources.

Examples:

```
// Given an element: <div id="the-box"></div>
$('#the-box').wiki('Who //are// you?');  → Appends "Who <em>are</em> you?" to the t
```

Math Methods

`Math.clamp(num , min , max)` → *number*

Returns the given number clamped to the specified bounds. Does not modify the original.

History:

- `v2.0.0`: Introduced.

Parameters:

- `num`: (*number*) The number to clamp. May be an actual number or a numerical string.
- `min`: (*number*) The lower bound of the number.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `max`: (*number*) The upper bound of the number.

Examples:

```
Math.clamp($stat, 0, 200)  → Clamps $stat to the bounds 0–200 and returns the new v
Math.clamp($stat, 1, 6.6)  → Clamps $stat to the bounds 1–6.6 and returns the new v
```

`Math.trunc(num)` → *integer*

Returns the whole (integer) part of the given number by removing its fractional part, if any. Does not modify the original.

History: *native JavaScript method*

Parameters:

- `num`: (*number*) The number to truncate to an integer.

Examples:

```
Math.trunc(12.7)   → Returns 12
Math.trunc(-12.7)  → Returns -12
```

Number Methods

`<Number>.clamp(min , max)` → *number*

Returns the number clamped to the specified bounds. Does not modify the original.

History:

- `v2.0.0`: Introduced.

Parameters:

- `min`: (*number*) The lower bound of the number.
- `max`: (*number*) The upper bound of the number.

Examples:

```
$stat.clamp(0, 200)  → Clamps $stat to the bounds 0–200 and returns the new value
$stat.clamp(1, 6.6)  → Clamps $stat to the bounds 1–6.6 and returns the new value
```

RegExp Methods

`RegExp.escape(text)` → *string*

Returns the given string with all regular expression metacharacters escaped. Does not modify the original.

History:

- `v2.0.0`: Introduced.

Parameters:

- `text`: (*string*) The string to escape.

Examples:

```
RegExp.escape('That will be $5 (cash only)')   → Returns 'That will be \$5 \(cash
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

String Methods

NOTE: Strings in TwineScript/JavaScript are Unicode, however, due to historic reasons they are comprised of, and indexed by, individual UTF-16 code units rather than code points. This means that some code points may span multiple code units—e.g., the emoji 💩 is one code point, but two code units.

`<String>.count(needle [, position])` → *integer*

Returns the number of times that the given substring was found within the string, starting the search at `position`.

History:

- `v2.0.0`: Introduced.

Parameters:

- `needle`: (*any*) The substring to count.
- `position`: (optional, *integer*) The zero-based index at which to begin searching for `needle`. If omitted, will default to `0`.

Examples:

```
// Given: $text = "How now, brown cow."
$text.count("ow")     → Returns 4
$text.count("ow", 8)  → Returns 2
```

`<String>.first()` → *string*

Returns the first Unicode code point within the string. Does not modify the original.

SEE: String methods note.

History:

- `v2.27.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $text = "abc"
$text.first()  → Returns "a"

// Given: $text = "👩‍👩‍👧‍👧"
$text.first()  → Returns "👩"
```

`String.format(format , arguments…)` → *string*

Returns a formatted string, after replacing each format item in the given format string with the text equivalent of the corresponding argument's value.

History:

- `v2.0.0`: Introduced.

Parameters:

- `format`: (*string*) The format string, which consists of normal text and format items.
- `arguments`: (*any*... | *Array<any>*) Either a list of arguments, which correspond by-index to the format items within the format string, or an array, whose members correspond by-index.

Format items:

A format item has the syntax `{index[,alignment]}`, square-brackets denoting optional elements.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `index`: (*integer*) The (zero-based) index of the argument whose string representation will replace the format item.
- `alignment`: (optional, *integer*) The total length of the field into which the argument is inserted, and whether it's right- or left-aligned (positive aligns right, negative aligns left).

Examples:

```
String.format("{0}, {1}!", "Hello", "World")      → List of arguments; Returns "He
String.format("{0}, {1}!", [ "Hello", "World" ])  → Array argument; Returns "Hello
String.format("{0,6}", "foo")                     → Returns "   foo"
String.format("{0,-6}", "foo")                    → Returns "foo   "
```

`<String>.includes(needle [, position])` → *boolean*

Returns whether the given substring was found within the string, starting the search at `position`.

History: *native JavaScript method*

Parameters:

- `needle`: (*any*) The substring to find.
- `position`: (optional, *integer*) The zero-based index at which to begin searching for `needle`. If omitted, will default to `0`.

Examples:

```
// Given: $text = "How now, brown cow."
$text.includes("row")      → Returns true
$text.includes("row", 14)  → Returns false
$text.includes("cow", 14)  → Returns true
$text.includes("pow")      → Returns false
```

`<String>.last()` → *string*

Returns the last Unicode code point within the string. Does not modify the original.

SEE: String methods note.

History:

- `v2.27.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $text = "abc"
$text.last()  → Returns "c"

// Given: $text = "👨‍👩‍👧"
$text.last()  → Returns "👧"
```

`<String>.toLocaleUpperFirst()` → *string*

Returns the string with its first Unicode code point converted to upper case, according to any locale-specific rules. Does not modify the original.

SEE: String methods note.

History:

- `v2.9.0`: Introduced.

Parameters: *none*

Examples:

```
// Using the Turkish (Türkçe) locale and given: $text = "ışık"
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
$text.toLocaleUpperFirst()  → Returns "Işık"

// Using the Turkish (Türkçe) locale and given: $text = "iki"
$text.toLocaleUpperFirst()  → Returns "İki"
```

`<String>.toUpperFirst()` → *string*

Returns the string with its first Unicode code point converted to upper case. Does not modify the original.

SEE: String methods note.

History:

- `v2.9.0`: Introduced.

Parameters: *none*

Examples:

```
// Given: $text = "hello."
$text.toUpperFirst()  → Returns "Hello."

// Given: $text = "χαίρετε."
$text.toUpperFirst()  → Returns "Χαίρετε."
```

Special Names

Passage, tag, and variable names that have special meaning to SugarCube.

Warning

1. All special names listed herein are case sensitive, so their spelling and capitalization must be ***exactly*** as shown.
2. ***Never*** combine special passages with special tags. By doing so, you will probably break things in subtle and hard to detect ways.

Passage Names

`PassageDone`

Used for post-passage-display tasks, like redoing dynamic changes (happens after the rendering and display of each passage). Generates no output.

Roughly equivalent to the `:passagedisplay` event.

History:

- `v2.0.0`: Introduced.

`PassageFooter`

Appended to each rendered passage.

Roughly equivalent to the `:passagerender` event.

History:

- `v2.0.0`: Introduced.

`PassageHeader`

Prepended to each rendered passage.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Roughly equivalent to the `:passagestart` event.

History:

- `v2.0.0`: Introduced.

`PassageReady`

Used for pre-passage-display tasks, like redoing dynamic changes (happens before the rendering of each passage). Generates no output.

Roughly equivalent to the `:passagestart` event.

History:

- `v2.0.0`: Introduced.

`Start`

Twine 2: *Not special.* Any passage may be chosen as the starting passage by selecting it via the *Start Story Here* passage context-menu item—n.b. older versions of Twine 2 used a 🚀 icon for the same purpose.

Twine 1/Twee: *Required.* The starting passage, the first passage displayed. Configurable, see `Config.passages.start` for more information.

History:

- `v2.0.0`: Introduced.

`StoryAuthor`

Used to populate the authorial byline area in the UI bar (element ID: `story-author`).

History:

- `v2.0.0`: Introduced.

`StoryBanner`

Used to populate the story's banner area in the UI bar (element ID: `story-banner`).

History:

- `v2.0.0`: Introduced.

`StoryCaption`

Used to populate the story's caption area in the UI bar (element ID: `story-caption`). May also be, and often is, used to add additional story UI elements and content to the UI bar.

History:

- `v2.0.0`: Introduced.

`StoryDisplayTitle`

Sets the story's display title in the browser's titlebar and the UI bar (element ID: `story-title`). If omitted, the story title will be used instead.

History:

- `v2.31.0`: Introduced.

`StoryInit`

Used for pre-story-start initialization tasks, like variable initialization (happens at the beginning of story initialization). Generates no output.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

StoryInterface

Used to replace SugarCube's default UI. Its contents are treated as raw HTML markup—i.e., *none* of SugarCube's special HTML processing is performed. It must contain, at least, an element with the ID `passages` that will be the main passage display area.

Additional elements, aside from the `#passages` element, may include either the `data-init-passage` or `data-passage` content attribute, whose value is the name of the passage used to populate the element—the passage will be processed as normal, meaning that markup and macros will work as expected. The `data-init-passage` attribute causes the element to be updated once at initialization, while the `data-passage` attribute causes the element to be updated upon each passage navigation.

> **WARNING:** Elements that include either a `data-init-passage` or `data-passage` content attribute *should not* themselves contain additional elements—since such elements' contents are replaced each turn via their associated passage, any child elements would be lost.

History:

- `v2.18.0`: Introduced.
- `v2.28.0`: Added processing of the `data-passage` content attribute.
- `v2.36.0`: Added processing of the `data-init-passage` content attribute.

Examples:

Minimal working example

```
<div id="passages"></div>
```

With `data-init-passage` and `data-passage` content attributes

```
<div id="interface">
        <div id="menu" data-init-passage="Menu"></div>
        <div id="notifications" data-passage="Notifications"></div>
        <div id="passages"></div>
</div>
```

StoryMenu

Used to populate the story's menu items in the UI bar (element ID: `menu-story`).

> **NOTE:** The story menu only displays links—specifically, anything that creates an anchor element (`<a>`). While it renders content just as any other passage does, instead of displaying the rendered output as-is, it sifts through the output and builds its menu from the generated links contained therein.

History:

- `v2.0.0`: Introduced.

Examples:

```
[[Inventory]]
<<link "Schedule">>…<</link>>
```

StorySettings

> **WARNING:** Unused by SugarCube. The `Config` API serves the same basic purpose.

StoryShare

Used to populate the contents of the Share dialog. Intended for social media links.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: The Share dialog only displays links—specifically, anything that creates an anchor element (`<a>`). While it renders content just as any other passage does, instead of displaying the rendered output as-is, it sifts through the output and builds its contents from the generated links contained therein.

History:

- `v2.0.0`: Introduced.

`StorySubtitle`

Sets the story's subtitle in the UI bar (element ID: `story-subtitle`).

History:

- `v2.0.0`: Introduced.

`StoryTitle`

WARNING: The story title is used to create the storage ID that is used to store all player data, both temporary and persistent. It should be plain text, containing no code, markup, or macros of any kind.

TIP: If you want to set a title for display that contains code, markup, or macros, see the `StoryDisplayTitle` special passage.

Twine 2: *Unused.* The story's title is part of the story project.

Twine 1/Twee: *Required.* Sets the story's title.

History:

- `v2.0.0`: Introduced.

Tag Names

`bookmark`

Registers the passage into the *Jump To* menu.

History:

- `v2.0.0`: Introduced.

`nobr`

Causes leading/trailing newlines to be removed and all remaining sequences of newlines to be replaced with single spaces before the passage is rendered. Equivalent to wrapping the entire passage in a `<<nobr>>` macro. See the `Config.passages.nobr` setting for a way to apply the same processing to all passages at once.

NOTE: Does not affect `script` or `stylesheet` tagged passages, for Twine 1/Twee.

History:

- `v2.0.0`: Introduced.

`init`

Registers the passage as an initialization passage. Used for pre-story-start initialization tasks, like variable initialization (happens at the beginning of story initialization). Generates no output.

NOTE: This is chiefly intended for use by add-ons/libraries. For normal projects, authors are encouraged to continue to use the `StoryInit` special named passage.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.36.0`: Introduced.

`script`

Twine 2: *Not special.* Use the *Edit Story JavaScript* story editor menu item for scripts.

Twine 1/Twee: Registers the passage as JavaScript code, which is executed during startup.

History:

- `v2.0.0`: Introduced.

`stylesheet`

Twine 2: *Not special.* Use the *Edit Story Stylesheet* story editor menu item for styles.

Twine 1/Twee: Registers the passage as a CSS stylesheet, which is loaded during startup. It is ***strongly*** recommended that you use only one stylesheet passage. Additionally, see the tagged stylesheet warning.

History:

- `v2.0.0`: Introduced.

`Twine.audio`

Registers the passage as an audio passage. See Guide: Media Passages for more information.

History:

- `v2.24.0`: Introduced.

`Twine.image`

Registers the passage as an image passage. See Guide: Media Passages for more information.

History:

- `v2.0.0`: Introduced.

`Twine.video`

Registers the passage as a video passage. See Guide: Media Passages for more information.

History:

- `v2.24.0`: Introduced.

`Twine.vtt`

Registers the passage as a VTT passage. See Guide: Media Passages for more information.

History:

- `v2.24.0`: Introduced.

`widget`

Registers the passage as `<<widget>>` macro definitions, which are loaded during startup.

History:

- `v2.0.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Variable Names

$

Alias for `jQuery`, by default. **NOTE:** This should not be confused with story variables, which start with a `$`—e.g., `$foo`.

History:

- `v2.0.0`: Introduced.

_args

Widget arguments array (only inside widgets). See `<<widget>>` for more information.

History:

- `v2.36.0`: Introduced.

_contents

Widget contents string (only inside block widgets). See `<<widget>>` for more information.

History:

- `v2.36.0`: Introduced.

Config

Configuration API. See `Config` API for more information.

History:

- `v2.0.0`: Introduced.

Dialog

Dialog API. See `Dialog` API for more information.

History:

- `v2.0.0`: Introduced.

Engine

Engine API. See `Engine` API for more information.

History:

- `v2.0.0`: Introduced.

Fullscreen

Fullscreen API. See `Fullscreen` API for more information.

History:

- `v2.31.0`: Introduced.

---

jQuery

jQuery library function.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

`l10nStrings`

Strings localization object. See Localization for more information.

History:

- `v2.10.0`: Introduced.

`LoadScreen`

LoadScreen API. See `LoadScreen` API for more information.

History:

- `v2.15.0`: Introduced.

`Macro`

Macro API. See `Macro` API for more information.

History:

- `v2.0.0`: Introduced.

`Passage`

Passage API. See `Passage` API for more information.

History:

- `v2.0.0`: Introduced.

`Save`

Save API. See `Save` API for more information.

History:

- `v2.0.0`: Introduced.

`Setting`

Setting API. See `Setting` API for more information.

History:

- `v2.0.0`: Introduced.

`settings`

Player settings object, set up by the author/developer. See `Setting` API for more information.

History:

- `v2.0.0`: Introduced.

`setup`

Object that authors/developers may use to set up various bits of static data. Generally, you would use this for data that does not change and should not be stored within story variables, which would make it part of the history.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

`SimpleAudio`

SimpleAudio API. See `SimpleAudio` API for more information.

History:

- `v2.28.0`: Introduced.

`State`

State API. See `State` API for more information.

History:

- `v2.0.0`: Introduced.

`Story`

Story API. See `Story` API for more information.

History:

- `v2.0.0`: Introduced.

`Template`

Template API. See `Template` API for more information.

History:

- `v2.29.0`: Introduced.

`UI`

UI API. See `UI` API for more information.

History:

- `v2.0.0`: Introduced.

`UIBar`

UIBar API. See `UIBar` API for more information.

History:

- `v2.17.0`: Introduced.

`$args`

> **DEPRECATED:** The `$args` special variable has been deprecated and should no longer be used. See the `_args` special variable for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.36.0`: Deprecated.

~~`postdisplay`~~

> **DEPRECATED:** `postdisplay` tasks have been deprecated and should no longer be used. See the `:passagedisplay` event for its replacement.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~postrender~~

> **DEPRECATED:** `postrender` tasks have been deprecated and should no longer be used. See the `:passagerender` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~predisplay~~

> **DEPRECATED:** `predisplay` tasks have been deprecated and should no longer be used. See the `:passagestart` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~prehistory~~

> **DEPRECATED:** `prehistory` tasks have been deprecated and should no longer be used. See the `:passageinit` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~prerender~~

> **DEPRECATED:** `prerender` tasks have been deprecated and should no longer be used. See the `:passagestart` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

CSS

Passage Conversions

IDs and classes automatically generated from passage names and tags are normalized to kebab case with all lowercase letters—which entails: removing characters that are not alphanumerics, underscores, hyphens, en-/em-dashes, or whitespace, then replacing any remaining non-alphanumeric characters with hyphens, one per group, and finally converting the result to lowercase.

Passage Names

Passage names have `passage-` prepended to their converted forms and are converted both into IDs and classes depending on how the passage is used—an ID for the active passage, classes for included (via `<<include>>`) passages.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

For example, if the passage name was `Gone fishin'`, then:

- When the active passage, it would become the ID `passage-gone-fishin` (selector: `#passage-gone-fishin`).
- When included, it would become the class `passage-gone-fishin` (selector: `.passage-gone-fishin`).

Passage Tags

When displaying a passage, its tags are:

1. Added to the active passage's container element, `<html>` element, and `<body>` element as a space separated list within the `data-tags` attribute.
2. Added to the active passage's container element and `<body>` element as classes. The following special tags are excluded from this mapping:

Twine 2:	`debug`, `nobr`, `passage`, `widget`, and any tag starting with `twine.`
Twine 1/Twee:	`debug`, `nobr`, `passage`, `script`, `stylesheet`, `widget`, and any tag starting with `twine.`

For example, if the tag name was `Sector_42`, then it would become both the `data-tags` attribute member `Sector_42` (selector: `[data-tags~="Sector_42"]`) and the class `sector-42` (selector: `.sector-42`).

Example Selectors

Selector	Description
`html`	The document element. The default font stack is set here. The active passage's tags will be added to its `data-tags` attribute (see: Passage Conversions).
`body`	The body of the page. The default foreground and background colors are set here. The active passage's tags will be added to its `data-tags` attribute and classes (see: Passage Conversions).
`#story`	Selects the story element.
`#passages`	Selects the element that contains passage elements. All created passage elements will be children of this element.
`.passage`	Selects the passage element. Normally, there will be only one such passage per turn, however, during passage navigation there may briefly be two—the incoming (a.k.a. active) and outgoing passages. The active passage's name will be added as its ID (see: Passage Conversions). The active passage's tags will be added to its `data-tags` attribute and classes (see: Passage Conversions).
`.passage a`	Selects all `<a>` elements within the passage element.
`.passage a:hover`	Selects `<a>` elements within the passage element that are being hovered over.
`.passage a:active`	Selects `<a>` elements within the passage element that are being clicked on.
`.passage .link-broken`	Selects all internal link elements within the passage element whose passages do not exist within the story.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Selector	Description
`.passage .link-disabled`	Selects all internal link elements within the passage element who have been disabled—e.g., already chosen `<<choice>>` macro links.
`.passage .link-external`	Selects all external link elements within the passage element—e.g., links to other pages and websites.
`.passage .link-internal`	Selects all internal link elements within the passage element—e.g., passage and macro links.
`.passage .link-visited`[1]	Selects all internal link elements within the passage element whose passages are within the in-play story history—i.e., passages the player has been to before.
`.passage .link-internal:not(.link-visited)`[1]	Selects all internal link elements within the passage element whose passages are not within the in-play story history—i.e., passages the player has never been to before.

1. The `.link-visited` class is not enabled by default, see the `Config` API's `Config.addVisitedLinkClass` property for more information.

Warnings

Multiple Stylesheets *(for Twine 1/Twee only)*

When using Twine 1/Twee, it is ***strongly*** recommended that you use only a single `stylesheet` tagged passage. CSS styles cascade in order of load, so if you use multiple `stylesheet` tagged passages, then it is all too easy for your styles to be loaded in the wrong order, since Twine 1/Twee gives you no control over the order that multiple `stylesheet` tagged passages load.

Tagged Stylesheets

SugarCube does not support the Twine 1.4+ vanilla story formats' tagged stylesheets. In SugarCube, you would instead simply prefix the selectors of your styles with the appropriate tag-based selectors—e.g., either `[data-tags~="…"]` attribute selectors or class selectors.

For example, if some story passages were tagged with `forest`, then styles for those forest passages might look like this:

```
/* Using [data-tags~="…"] attribute selectors on <html> */
html[data-tags~="forest"] { background-image: url(forest-bg.jpg); }
html[data-tags~="forest"] .passage { color: darkgreen; }
html[data-tags~="forest"] a { color: green; }
html[data-tags~="forest"] a:hover { color: lime; }

/* Using [data-tags~="…"] attribute selectors on <body> */
body[data-tags~="forest"] { background-image: url(forest-bg.jpg); }
body[data-tags~="forest"] .passage { color: darkgreen; }
body[data-tags~="forest"] a { color: green; }
body[data-tags~="forest"] a:hover { color: lime; }

/* Using class selectors on <body> */
body.forest { background-image: url(forest-bg.jpg); }
body.forest .passage { color: darkgreen; }
body.forest a { color: green; }
body.forest a:hover { color: lime; }
```

Built-in Stylesheets

These are SugarCube's built-in stylesheets, in order of load/cascade. The most interesting of which, from an end-user's standpoint, are 4–10. The links go to the most recent release versions of each in SugarCube's source code repository.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

1. normalize.css
2. init-screen.css
3. font.css
4. core.css
5. core-display.css
6. core-passage.css
7. core-macro.css
8. ui-dialog.css
9. ui.css
10. ui-bar.css
11. ui-debug.css

HTML

The hierarchy of the document body, including associated HTML IDs and class names is as follows.

Notes:

- Periods of ellipsis (…) signify data that is dynamically generated at run time.
- The `#story-title-separator` element is normally unused.
- The story menu, `#menu-story`, will only exist if the `StoryMenu` special passage is used.
- The core menu item for the Settings dialog, `#menu-item-settings`, will only exist if the `Setting` API is used.
- The core menu item for the Share dialog, `#menu-item-share`, will only exist if the `StoryShare` special passage is used.

```
<body class="…">
	<div id="init-screen"></div>
	<div id="ui-overlay" class="ui-close"></div>
	<div id="ui-dialog" tabindex="0" role="dialog" aria-labelledby="ui-dialog-
		<div id="ui-dialog-titlebar">
			<h1 id="ui-dialog-title"></h1>
			<button id="ui-dialog-close" class="ui-close" tabindex="0"
		</div>
		<div id="ui-dialog-body"></div>
	</div>
	<div id="ui-bar">
		<div id="ui-bar-tray">
			<button id="ui-bar-toggle" tabindex="0" title="…" aria-lab
			<div id="ui-bar-history">
				<button id="history-backward" tabindex="0" title="
				<button id="history-jumpto" tabindex="0" title="…"
				<button id="history-forward" tabindex="0" title="…
			</div>
		</div>
		<div id="ui-bar-body">
			<header id="title" role="banner">
				<div id="story-banner"></div>
				<h1 id="story-title"></h1>
				<div id="story-subtitle"></div>
				<div id="story-title-separator"></div>
				<p id="story-author"></p>
			</header>
			<div id="story-caption"></div>
			<nav id="menu" role="navigation">
				<ul id="menu-story">…<ul>
				<ul id="menu-core">
					<li id="menu-item-saves"><a tabindex="0">…
					<li id="menu-item-settings"><a tabindex="0
					<li id="menu-item-restart"><a tabindex="0"
					<li id="menu-item-share"><a tabindex="0">…
				</ul>
			</nav>
		</div>
	</div>
	<div id="story" role="main">
		<div id="passages">
			<div class="passage …" id="…" data-passage="…">
				<!-- The active (present) passage content -->
			</div>
		</div>
	</div>
	<!-- The story data chunk, which depends on the compiler release (see belo
```

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

```
        <script id="script-sugarcube" type="text/javascript"><!-- The main SugarCul
</body>
```

Story data chunks:

Periods of ellipsis (…) signify data that is generated at compile time.

Twine 2 style data chunk

```
<tw-storydata name="…" startnode="…" creator="…" creator-version="…"
        ifid="…" zoom="…" format="…" format-version="…" options="…" hidden>
        <!-- Passage data nodes… -->
</tw-storydata>
```

Twine 1 style data chunk

```
<div id="store-area" data-size="…" hidden>
        <!-- Passage data nodes… -->
</div>
```

Events

Events are messages that are sent (a.k.a.: fired, triggered) to notify code that something has taken place, from player interactions to automated happenings. Each event is represented by an object that has properties that may be used to get additional information about what happened.

This section offers a list of SugarCube-specific events, triggered at various points during story operation.

SEE ALSO: For standard browser/DOM events, see the *Event reference* @MDN.

Dialog Events

`Dialog` events allow the execution of JavaScript code at specific points during the opening and closing of dialogs.

SEE: `Dialog` API.

`:dialogclosed` event

Global event triggered as the last step in closing the dialog when `Dialog.close()` is called.

WARNING: You cannot obtain data about the closing dialog from the dialog itself—e.g., title or classes—when using the `:dialogclosed` event, as the dialog has already closed and been reset by the time the event is fired. If you need that kind of information from the dialog itself, then you may use the `:dialogclosing` event instead.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

NOTE: While there are no custom properties, the event is fired from the dialog's body, thus the `target` property will refer to its body element—i.e., `#ui-dialog-body`.

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':dialogclosed', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':dialogclosed', function (ev) {
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        /* JavaScript code */
});
```

`:dialogclosing` event

Global event triggered as the first step in closing the dialog when `Dialog.close()` is called.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

> **NOTE:** While there are no custom properties, the event is fired from the dialog's body, thus the `target` property will refer to its body element—i.e., `#ui-dialog-body`.

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':dialogclosing', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':dialogclosing', function (ev) {
        /* JavaScript code */
});
```

`:dialogopened` event

Global event triggered as the last step in opening the dialog when `Dialog.open()` is called.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

> **NOTE:** While there are no custom properties, the event is fired from the dialog's body, thus the `target` property will refer to its body element—i.e., `#ui-dialog-body`.

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':dialogopened', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':dialogopened', function (ev) {
        /* JavaScript code */
});
```

`:dialogopening` event

Global event triggered as the first step in opening the dialog when `Dialog.open()` is called.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

> **NOTE:** While there are no custom properties, the event is fired from the dialog's body, thus the `target` property will refer to its body element—i.e., `#ui-dialog-body`.

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':dialogopening', function (ev) {
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
$(document).on(':dialogopening', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':dialogopening', function (ev) {
        /* JavaScript code */
});
```

Navigation Events

Navigation events allow the execution of JavaScript code at specific points during passage navigation.

In order of processing: *(for reference, this also shows tasks and various special passages)*

1. Passage init. Happens before the modification of the state history.
    1. `:passageinit` event.
    2. ~~`prehistory` tasks.~~ *(deprecated)*
2. Passage start. Happens before the rendering of the incoming passage.
    1. ~~`predisplay` tasks.~~ *(deprecated)*
    2. `PassageReady` special passage.
    3. `:passagestart` event.
    4. ~~`prerender` tasks.~~ *(deprecated)*
    5. `PassageHeader` special passage.
3. Passage render. Happens after the rendering of the incoming passage.
    1. `PassageFooter` special passage.
    2. `:passagerender` event.
    3. ~~`postrender` tasks.~~ *(deprecated)*
4. Passage display. Happens after the display—i.e., output—of the incoming passage.
    1. `PassageDone` special passage.
    2. `:passagedisplay` event.
    3. ~~`postdisplay` tasks.~~ *(deprecated)*
5. UI bar special passages update. Happens before the end of passage navigation.
    1. `StoryBanner` special passage.
    2. `StoryDisplayTitle` special passage.
    3. `StorySubtitle` special passage.
    4. `StoryAuthor` special passage.
    5. `StoryCaption` special passage.
    6. `StoryMenu` special passage.
6. Passage end. Happens at the end of passage navigation.
    1. `:passageend` event.

`:passageinit` event

Triggered before the modification of the state history.

History:

- `v2.20.0`: Introduced.

Event object properties:

- **`passage`:** (*`Passage` object*) The incoming passage object. See the `Passage` API for more information.

Examples:

```
/* Execute the handler function each time the event triggers. */
$(document).on(':passageinit', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':passageinit', function (ev) {
        /* JavaScript code */
});
```

`:passagestart` event

Triggered before the rendering of the incoming passage.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.20.0`: Introduced.

Event object properties:

- **`content`:** (*`HTMLElement` object*) The, currently, empty element that will eventually hold the rendered content of the incoming passage.
- **`passage`:** (*`Passage` object*) The incoming passage object. See the `Passage` API for more information.

Examples:

Basic usage

```
/* Execute the handler function each time the event triggers. */
$(document).on(':passagestart', function (ev) {
	/* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':passagestart', function (ev) {
	/* JavaScript code */
});
```

Modifying the content buffer

```
/*
	Process the markup "In the //beginning//." and append the result
	to the incoming passage's element.
*/
$(document).on(':passagestart', function (ev) {
	$(ev.content).wiki("In the //beginning//.");
});
```

`:passagerender` event

Triggered after the rendering of the incoming passage.

History:

- `v2.20.0`: Introduced.

Event object properties:

- **`content`:** (*`HTMLElement` object*) The element holding the fully rendered content of the incoming passage.
- **`passage`:** (*`Passage` object*) The incoming passage object. See the `Passage` API for more information.

Examples:

Basic usage

```
/* Execute the handler function each time the event triggers. */
$(document).on(':passagerender', function (ev) {
	/* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':passagerender', function (ev) {
	/* JavaScript code */
});
```

Modifying the content buffer

```
/*
	Process the markup "At the //end// of some renderings." and append the resu
	to the incoming passage's element.
*/
$(document).on(':passagerender', function (ev) {
	$(ev.content).wiki("At the //end// of some renderings.");
});
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`:passagedisplay` event

Triggered after the display—i.e., output—of the incoming passage.

History:

- `v2.20.0`: Introduced.
- `v2.31.0`: Added `content` property to event object.

Event object properties:

- **`content`:** (*`HTMLElement` object*) The element holding the fully rendered content of the incoming passage.
- **`passage`:** (*`Passage` object*) The incoming passage object. See the `Passage` API for more information.

Examples:

Basic usage

```
/* Execute the handler function each time the event triggers. */
$(document).on(':passagedisplay', function (ev) {
	/* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':passagedisplay', function (ev) {
	/* JavaScript code */
});
```

Modifying the content buffer

```
/*
	Process the markup "It's //showtime//!" and append the result
	to the incoming passage's element.
*/
$(document).on(':passagedisplay', function (ev) {
	$(ev.content).wiki("It's //showtime//!");
});
```

`:passageend` event

Triggered at the end of passage navigation.

History:

- `v2.20.0`: Introduced.
- `v2.31.0`: Added `content` property to event object.

Event object properties:

- **`content`:** (*`HTMLElement` object*) The element holding the fully rendered content of the incoming passage.
- **`passage`:** (*`Passage` object*) The incoming passage object. See the `Passage` API for more information.

Examples:

Basic usage

```
/* Execute the handler function each time the event triggers. */
$(document).on(':passageend', function (ev) {
	/* JavaScript code */
});

/* Execute the handler function exactly once. */
$(document).one(':passageend', function (ev) {
	/* JavaScript code */
});
```

Modifying the content buffer

```
/*
	Process the markup "So long and //thanks for all the fish//!" and append tl
	to the incoming passage's element.
*/
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
$(document).on(':passageend', function (ev) {
        $(ev.content).wiki("So long and //thanks for all the fish//!");
});
```

~~`prehistory` tasks~~

DEPRECATED: `prehistory` tasks have been deprecated and should no longer be used. See the `:passageinit` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~`predisplay` tasks~~

DEPRECATED: `predisplay` tasks have been deprecated and should no longer be used. See the `:passagestart` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~`prerender` tasks~~

DEPRECATED: `prerender` tasks have been deprecated and should no longer be used. See the `:passagestart` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~`postrender` tasks~~

DEPRECATED: `postrender` tasks have been deprecated and should no longer be used. See the `:passagerender` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

~~`postdisplay` tasks~~

DEPRECATED: `postdisplay` tasks have been deprecated and should no longer be used. See the `:passagedisplay` event for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.31.0`: Deprecated.

`SimpleAudio` Events

`SimpleAudio` events allow the execution of JavaScript code at specific points during audio playback.

SEE: To add or remove event listeners to audio tracks managed by the `SimpleAudio` API see:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `AudioTrack` API methods: `<AudioTrack>.off()`, `<AudioTrack>.on()`, `<AudioTrack>.one()`.
- `AudioRunner` API methods: `<AudioRunner>.off()`, `<AudioRunner>.on()`, `<AudioRunner>.one()`.

`:faded` event

Track event triggered when a fade completes normally.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers for one track via <AudioTra
aTrack.on(':faded', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function when the event triggers for multiple tracks via <Au
someTracks.on(':faded', function (ev) {
        /* do something */
});
```

`:fading` event

Track event triggered when a fade starts.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers for one track via <AudioTra
aTrack.on(':fading', function (ev) {
        /* JavaScript code */
});

/* Execute the handler function when the event triggers for multiple tracks via <Au
someTracks.on(':fading', function (ev) {
        /* do something */
});
```

`:stopped` event

Track event triggered when playback is stopped after `<AudioTrack>.stop()` or `<AudioRunner>.stop()` is called—either manually or as part of another process.

SEE ALSO: `ended` and `pause` for information on somewhat similar native events.

History:

- `v2.29.0`: Introduced.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers for one track via <AudioTra
aTrack.on(':stopped', function (ev) {
        /* JavaScript code */
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
});

/* Execute the handler function when the event triggers for multiple tracks via <Au
someTracks.on(':stopped', function (ev) {
        /* do something */
});
```

System Events

System events allow the execution of JavaScript code at specific points during story startup and teardown.

`:storyready` event

Global event triggered once just before the dismissal of the loading screen at startup.

History:

- `v2.31.0`: Introduced.

Event object properties: *none*

Examples:

```
/* Execute the handler function exactly once, since it's only fired once. */
$(document).one(':storyready', function (ev) {
        /* JavaScript code */
});
```

`:enginerestart` event

Global event triggered once just before the page is reloaded when `Engine.restart()` is called.

History:

- `v2.23.0`: Introduced.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).one(':enginerestart', function (ev) {
        /* JavaScript code */
});
```

<<type>> Events

`<<type>>` macro events allow the execution of JavaScript code at specific points during typing.

`:typingcomplete` event

Global event triggered when all `<<type>>` macros within a passage have completed.

> **NOTE:** Injecting additional `<<type>>` macro invocations *after* a `:typingcomplete` event has been fired will cause another event to eventually be generated, since you're creating a new sequence of typing.

History:

- `v2.32.0`: Introduced.

Event object properties: *none*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':typingcomplete', function (ev) {
	/* JavaScript code */
});
```

`:typingstart` event

Local event triggered on the typing wrapper when the typing of a section starts.

History:

- `v2.32.0`: Introduced
- `v2.33.0`: Changed to a local event that bubbles up the DOM tree.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':typingstart', function (ev) {
	/* JavaScript code */
});
```

`:typingstop` event

Local event triggered on the typing wrapper when the typing of a section stops.

History:

- `v2.32.0`: Introduced
- `v2.33.0`: Changed to a local event that bubbles up the DOM tree.

Event object properties: *none*

Examples:

```
/* Execute the handler function when the event triggers. */
$(document).on(':typingstop', function (ev) {
	/* JavaScript code */
});
```

Config API

The `Config` object controls various aspects of SugarCube's behavior.

> **NOTE:** `Config` object settings should be placed within your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage).

Audio Settings

`Config.audio.pauseOnFadeToZero` ↔ *boolean* (default: `true`)

Determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
Config.audio.pauseOnFadeToZero = false;
```

`Config.audio.preloadMetadata` ↔ *boolean* (default: `true`)

Determines whether the audio subsystem attempts to preload track metadata—meaning information about the track (e.g., duration), not its audio frames.

> **NOTE:** It is unlikely that you will ever want to disable this setting.

History:

- `v2.28.0`: Introduced.

Examples:

```
Config.audio.preloadMetadata = false;
```

History Settings

`Config.history.controls` ↔ *boolean* (default: `true`)

Determines whether the story's history controls (Backward, Jump To, & Forward buttons) are enabled within the UI bar.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.history.controls = false;
```

`Config.history.maxStates` ↔ *integer* (default: `40`)

Sets the maximum number of states (moments) to which the history is allowed to grow. Should the history exceed the limit, states will be dropped from the past (oldest first).

> **TIP:** For game-oriented projects, as opposed to more story-oriented interactive fiction, a setting of `1` is ***strongly recommended***.

History:

- `v2.0.0`: Introduced.
- `v2.36.0`: Reduced the default to `40`.

Examples:

```
// Limit the history to a single state (recommended for games)
Config.history.maxStates = 1;

// Limit the history to 80 states
Config.history.maxStates = 80;
```

Macros Settings

`Config.macros.ifAssignmentError` ↔ *boolean* (default: `true`)

Determines whether the `<<if>>` macro returns an error when the `=` assignment operator is used within its conditional—e.g., `<<if $suspect = "Bob">>`. Does not flag other assignment operators.

> **NOTE:** This setting exists because it's unlikely that you'll ever want to actually perform an assignment within a conditional expression and typing `=` when you meant `===` (or `==`) is a fairly

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

> easy to mistake make—either from a finger slip or because you just don't know the difference between the operators.

History:

- `v2.0.0`: Introduced.

Examples:

```
// No error is returned when = is used within an <<if>> or <<elseif>> conditional
Config.macros.ifAssignmentError = false;
```

`Config.macros.maxLoopIterations` ↔ *integer* (default: `1000`)

Sets the maximum number of iterations allowed before the `<<for>>` macro conditional forms are terminated with an error.

> **NOTE:** This setting exists to prevent a misconfigured loop from making the browser unresponsive.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Allow only 5000 iterations
Config.macros.maxLoopIterations = 5000;
```

`Config.macros.typeSkipKey` ↔ *string* (default: `" "`, space)

Sets the default `KeyboardEvent.key` value that causes the currently running `<<type>>` macro instance to finish typing its content immediately.

History:

- `v2.33.1`: Introduced.

Examples:

```
// Change the default skip key to Control (CTRL)
Config.macros.typeSkipKey = "Control";
```

`Config.macros.typeVisitedPassages` ↔ *boolean* (default: `true`)

Determines whether the `<<type>>` macro types out content on previously visited passages or simply outputs it immediately.

History:

- `v2.32.0`: Introduced.

Examples:

```
// Do not type on previously visited passages
Config.macros.typeVisitedPassages = false;
```

Navigation Settings

`Config.navigation.override` ↔ *function* (default: *none*)

Allows the destination of passage navigation to be overridden. The callback is passed one parameter, the original destination passage title. If its return value is falsy, the override is cancelled and navigation to the original destination continues unperturbed. If its return value is truthy, the override succeeds and that value is used as the new destination of the navigation.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.13.0`: Introduced.

Examples:

```
Config.navigation.override = function (destinationPassage) {
        /* code */
};
```

Based upon a story variable

```
// Force the player to the "You Died" passage if they let $health get too low.
Config.navigation.override = function (dest) {
        var sv = State.variables;

        // If $health is less-than-or-equal to 0, go to the "You Died" passage inst
        if (sv.health <= 0) {
                return "You Died";
        }
};
```

Passages Settings

`Config.passages.descriptions` ↔ *boolean* | *object* | *function* (default: *none*)

Determines whether alternate passage descriptions are used by the *Saves* and *Jump To* menus—by default an excerpt from the passage is used. Valid values are boolean `true`, which simply causes the passages' titles to be used, an object, which maps passages' titles to their descriptions, or a function, which should return the passages' description.

> **NOTE:**
> - **As boolean:** You should ensure that all encounterable passage titles are meaningful.
> - **As object:** If the mapping object does not contain an entry for the passage in question, then the passage's excerpt is used instead.
> - **As function:** The function is called with the passage in question as its `this` value. If the function returns falsy, then the passage's excerpt is used instead.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Uses passages' titles
Config.passages.descriptions = true;

// Uses the description mapped by the title
Config.passages.descriptions = {
        "title" : "description"
};

// Returns the description to be used
Config.passages.descriptions = function () {
        if (this.title === "title") {
                return "description";
        }
};
```

`Config.passages.displayTitles` ↔ *boolean* (default: `false`)

Determines whether passage titles are combined with the story title, within the browser's/tab's titlebar, when passages are displayed.

History:

- `v2.0.0`: Introduced.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
Config.passages.displayTitles = true;
```

`Config.passages.nobr` ↔ *boolean* (default: `false`)

Determines whether rendering passages have their leading/trailing newlines removed and all remaining sequences of newlines replaced with single spaces before they're rendered. Equivalent to including the `nobr` special tag on every passage.

> **NOTE:** Does not affect `script` or `stylesheet` tagged passages, for Twine 1/Twee, or the Story JavaScript or Story Stylesheet sections, for Twine 2.

History:

- `v2.19.0`: Introduced.

Examples:

```
Config.passages.nobr = true;
```

`Config.passages.onProcess` ↔ *function* (default: *none*)

Allows custom processing of passage text. The function is invoked each time the `<Passage>.processText()` method is called. It is passed an abbreviated version of the associated passage's `Passage` instance—containing only the `tags`, `text`, and `title` properties. Its return value should be the post-processed text.

> **NOTE:** Does not affect `script` or `stylesheet` tagged passages, for Twine 1/Twee, or the Story JavaScript or Story Stylesheet sections, for Twine 2.

> **NOTE:** The function will be called just before the built-in no-break passage processing if you're also using that—see the `Config.passages.nobr` setting and `nobr` special tag.

History:

- `v2.30.0`: Introduced.

Examples:

```
/* Change instancess of "cat" to "dog". */
Config.passages.onProcess = function (p) {
	return p.text.replace(/\bcat(s?)\b/g, 'dog$1');
};
```

`Config.passages.start` ↔ *string* (Twine 2 default: *user-selected*; Twine 1/Twee default: `"Start"`)

Sets the starting passage, the very first passage that will be displayed.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.passages.start = "That Other Starting Passage";
```

`Config.passages.transitionOut` ↔ *string* | *integer* (default: *none*)

Determines whether outgoing passage transitions are enabled. Valid values are the name of the property being animated, which causes the outgoing passage element to be removed once that transition animation is complete, or an integer delay (in milliseconds), which causes the outgoing passage element to be removed once the delay has expired. You will also need some CSS styles to make this work—examples given below.

> **NOTE:** If using an integer delay, ideally, it should probably be slightly longer than the outgoing transition delay that you intend to use—e.g., an additional 10ms or so should be sufficient.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Examples:

```
// Remove outgoing elements when their opacity animation ends
Config.passages.transitionOut = "opacity";

// Remove outgoing elements after 1010ms (1.01s)
Config.passages.transitionOut = 1010;
```

CSS styles:

At the very least you will need to specify a `.passage-out` style that defines the transition's end state. For example:

```
.passage-out {
	opacity: 0;
}
```

That probably won't be very pleasing to the eye, however, so you will likely need several styles to make something that looks half-decent. For example, the following will give you a basic crossfade:

```
#passages {
	position: relative;
}
.passage {
	left: 0;
	position: absolute;
	top: 0;
	transition: opacity 1s ease;
}
.passage-out {
	opacity: 0;
}
```

Saves Settings

`Config.saves.autoload` ↔ *boolean* | *string* | *function* (default: *none*)

Determines whether the autosave, if it exists, is automatically loaded upon story startup. Valid values are boolean `true`, which simply causes the autosave to be loaded, the string `"prompt"`, which prompts the player via a dialog to load the autosave, or a function, which causes the autosave to be loaded if its return value is truthy.

> **NOTE:** If the autosave cannot be loaded, for any reason, then the start passage is loaded instead.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Automatically loads the autosave
Config.saves.autoload = true;

// Prompts the player about loading the autosave
Config.saves.autoload = "prompt";

// Loads the autosave if it returns a truthy value
Config.saves.autoload = function () {
	/* code */
};
```

`Config.saves.autosave` ↔ *boolean* | *Array<string>* | *function* (default: *none*)

Determines whether the autosave is created/updated when passages are displayed.

Valid values are:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- Boolean `true`, which causes the autosave to be updated with every passage.
- Boolean `false`, which causes the autosave to never update automatically—i.e., you must do it manually via the `Save.autosave.save()` static method.
- An array of strings, which causes the autosave to be updated for each passage with at least one matching tag.
- A function, which causes the autosave to be updated for each passage where its return value is truthy.

> **WARNING:** When setting the value to boolean `true`, you will likely also need to use the `Config.saves.isAllowed` property to disallow saving on the start passage. Or, if you use the start passage as real part of your story and allow the player to reenter it, rather than just as the initial landing/cover page, then you may wish to only disallow saving on the start passage the very first time it's displayed—i.e., at story startup.

History:

- `v2.0.0`: Introduced.
- `v2.30.0`: Added function values and deprecated string values.

Examples:

```
// Autosaves every passage
Config.saves.autosave = true;

// Allows manual autosaving
Config.saves.autosave = false;

// Autosaves on passages tagged with any of "bookmark" or "autosave"
Config.saves.autosave = ["bookmark", "autosave"];

// Autosaves on passages if it returns a truthy value
Config.saves.autosave = function () {
        /* code */
};
```

`Config.saves.id` ↔ *string* (default: *slugified story title*)

Sets the story ID associated with saves.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.saves.id = "a-big-huge-story-part-1";
```

`Config.saves.isAllowed` ↔ *function* (default: *none*)

Determines whether saving is allowed within the current context. The callback is invoked each time a save is requested. If its return value is falsy, the save is disallowed. If its return value is truthy, the save is allowed to continue unperturbed.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.saves.isAllowed = function () {
        /* code */
};
```

`Config.saves.slots` *integer* (default: `8`)

Sets the maximum number of available save slots.

History:

- `v2.0.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
Config.saves.slots = 4;
```

`Config.saves.tryDiskOnMobile` ↔ *boolean* (default: `true`)

Determines whether saving to disk is enabled on mobile devices—i.e., smartphones, tablets, etc.

> **WARNING:** Mobile browsers can be fickle, so saving to disk may not work as expected in all browsers.

History:

- `v2.34.0`: Introduced.

Examples:

```
/* To disable saving to disk on mobile devices. */
Config.saves.tryDiskOnMobile = false;
```

`Config.saves.version` ↔ *any* (default: *none*)

Sets the `version` property of saves.

> **NOTE:** This setting is only used to set the `version` property of saves. Thus, it is only truly useful if you plan to upgrade out-of-date saves via a `Config.saves.onLoad` callback.

History:

- `v2.0.0`: Introduced.

Examples:

```
// As an integer (recommended)
Config.saves.version = 3;

// As a string (not recommended)
Config.saves.version = "v3";
```

~~`Config.saves.onLoad`~~ ↔ ~~*function* (default: *none*)~~

> **DEPRECATED:** This setting has been deprecated and should no longer be used. See the `Save.onLoad.add()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.36.0`: Deprecated in favor of the `Save` Events API.

~~`Config.saves.onSave`~~ ↔ ~~*function* (default: *none*)~~

> **DEPRECATED:** This setting has been deprecated and should no longer be used. See the `Save.onSave.add()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.33.0`: Added save operation details object parameter to the callback function.
- `v2.36.0`: Deprecated in favor of the `Save` Events API.

UI Settings

`Config.ui.stowBarInitially` ↔ *boolean* | *integer* (default: `800`)

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

Determines whether the UI bar (sidebar) starts in the stowed (shut) state initially. Valid values are boolean `true`/`false`, which causes the UI bar to always/never start in the stowed state, or an integer, which causes the UI bar to start in the stowed state if the viewport width is less-than-or-equal-to the specified number of pixels.

History:

- `v2.11.0`: Introduced.

Examples:

```
// As a boolean; always start stowed
Config.ui.stowBarInitially = true;

// As a boolean; never start stowed
Config.ui.stowBarInitially = false;

// As an integer; start stowed if the viewport is 800px or less
Config.ui.stowBarInitially = 800;
```

`Config.ui.updateStoryElements` ↔ *boolean* (default: `true`)

Determines whether certain elements within the UI bar are updated when passages are displayed. The affected elements are the story: banner, subtitle, author, caption, and menu.

> **NOTE:** The story title is not included in updates because SugarCube uses it as the basis for the key used to store and load data used when playing the story and for saves.

History:

- `v2.0.0`: Introduced.

Examples:

```
// If you don't need those elements to update
Config.ui.updateStoryElements = false;
```

Miscellaneous Settings

`Config.addVisitedLinkClass` ↔ *boolean* (default: `false`)

Determines whether the `link-visited` class is added to internal passage links that go to previously visited passages—i.e., the passage already exists within the story history.

> **NOTE:** You *must* provide your own styling for the `link-visited` class as none is provided by default.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.addVisitedLinkClass = true;
```

CSS styles:

You will also need to specify a `.link-visited` style that defines the properties visited links should have. For example:

```
.link-visited {
	color: purple;
}
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`Config.cleanupWikifierOutput` ↔ *boolean* (default: `false`)

Determines whether the output of the Wikifier is post-processed into more sane markup—i.e., where appropriate, it tries to transition the plethora of `<br>` elements into `<p>` elements.

History:

- `v2.0.0`: Introduced.

Examples:

```
Config.cleanupWikifierOutput = true;
```

`Config.debug` ↔ *boolean* (default: `false`)

Indicates whether SugarCube is running in test mode, which enables debug views. See the *Test Mode* guide for more information.

> **NOTE:** This property is automatically set based on whether you're using a testing mode in a Twine compiler—i.e., *Test* mode in Twine 2, *Test Play From Here* in Twine 1, or the test mode option (`-t`, `--test`) in Tweego. You may, however, forcibly enable it if you need to for some reason—e.g., if you're using another compiler, which doesn't offer a way to enable test mode.

History:

- `v2.2.0`: Introduced.

Examples:

```
// Forcibly enable test mode
Config.debug = true;

// Check if test mode is enabled in JavaScript
if (Config.debug) {
	/* do something debug related */
}

// Check if test mode is enabled via the <<if>> macro
<<if Config.debug>>
	/* do something debug related */
<</if>>
```

`Config.loadDelay` ↔ *integer* (default: `0`)

Sets the integer delay (in milliseconds) before the loading screen is dismissed, once the document has signaled its readiness. Not generally necessary, however, some browsers render slower than others and may need a little extra time to get a media-heavy page done. This allows you to fine tune for those cases.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Delay the dismissal of the loading screen by 2000ms (2s)
Config.loadDelay = 2000;
```

`Dialog` API

`Dialog.append(content)` → *`Dialog` object*

Appends the given content to the dialog's content area. Returns a reference to the `Dialog` object for chaining.

> **NOTE:** If your content contains any SugarCube markup, you'll need to use the `Dialog.wiki()` method instead.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.9.0`: Introduced.

Parameters:

- `content`: The content to append. As this method is essentially a shortcut for `jQuery(Dialog.body()).append(…)`, see jQuery's `append()` method for the range of valid content types.

Examples:

```
Dialog.append("Cry 'Havoc!', and let slip the <em>ponies</em> of <strong>friendshi

Dialog.append( /* some DOM nodes */ );
```

`Dialog.body()` → *`HTMLElement`* *object*

Returns a reference to the dialog's content area.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
jQuery(Dialog.body()).append("Cry 'Havoc!', and let slip the <em>ponies</em> of <s

jQuery(Dialog.body()).wiki("Cry 'Havoc!', and let slip the //ponies// of ''friends
```

`Dialog.close()` → *`Dialog`* *object*

Closes the dialog. Returns a reference to the `Dialog` object for chaining.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Dialog.close();
```

`Dialog.isOpen([classNames])` → *boolean*

Returns whether the dialog is currently open.

History:

- `v2.0.0`: Introduced.

Parameters:

- `classNames`: (optional, *string*) The space-separated-list of classes to check for when determining the state of the dialog. Each of the built-in dialogs contains a name-themed class that can be tested for in this manner—e.g., the Saves dialog contains the class `saves`.

Examples:

```
if (Dialog.isOpen()) {
        /* code to execute if the dialog is open… */
}

if (Dialog.isOpen("saves")) {
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        /* code to execute if the Saves dialog is open… */
}
```

`Dialog.open([options [, closeFn]])` → *`Dialog`* object

Opens the dialog. Returns a reference to the `Dialog` object for chaining.

> **NOTE:** Call this only after populating the dialog with content.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *null* | *object*) The options to be used when opening the dialog.
- `closeFn`: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

Options object:

An options object should have some of the following properties:

- `top`: Top y-coordinate of the dialog (default: `50`; in pixels, but without the unit).

Examples:

```
Dialog.open();
```

`Dialog.setup([title [, classNames]])` → *`HTMLElement`* object

Prepares the dialog for use and returns a reference to its content area.

History:

- `v2.0.0`: Introduced.

Parameters:

- `title`: (optional, *string*) The title of the dialog.
- `classNames`: (optional, *string*) The space-separated-list of classes to add to the dialog.

Examples:

```
// Basic example.
Dialog.setup();
Dialog.wiki("Blah //blah// ''blah''.");
Dialog.open();

// Adding a title to the dialog.
Dialog.setup("Character Sheet");
Dialog.wiki(Story.get("PC Sheet").processText());
Dialog.open();

// Adding a title and class to the dialog.
Dialog.setup("Character Sheet", "charsheet");
Dialog.wiki(Story.get("PC Sheet").processText());
Dialog.open();
```

`Dialog.wiki(wikiMarkup)` → *`Dialog`* object

Renders the given markup and appends it to the dialog's content area. Returns a reference to the `Dialog` object for chaining.

> **NOTE:** If your content consists of DOM nodes, you'll need to use the `Dialog.append()` method instead.

History:

- `v2.9.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters:

- `wikiMarkup`: The markup to render.

Examples:

```
Dialog.wiki("Cry 'Havoc!', and let slip the //ponies// of ''friendship''.");
```

~~Dialog.addClickHandler(targets [, options [, startFn [, doneFn [, cle~~

DEPRECATED: This method has been deprecated and should no longer be used. The core of what it does is simply to wrap a call to `Dialog.open()` within a call to `<jQuery>.ariaClick()`, which can be done directly and with greater flexibility.

History:

- `v2.0.0`: Introduced.
- `v2.29.0`: Deprecated.

Engine API

`Engine.lastPlay` → *number*

Returns a timestamp representing the last time `Engine.play()` was called.

History:

- `v2.0.0`: Introduced.

Examples:

```
Engine.lastPlay  → The timestamp at which Engine.play() was last called
```

`Engine.state` → *string*

Returns the current state of the engine (`"idle"`, `"playing"`, `"rendering"`).

History:

- `v2.7.0`: Introduced.

States:

- `"idle"`: The engine is idle, awaiting the triggering of passage navigation—the default state.
- `"playing"`: Passage navigation has been triggered and a turn is being processed.
- `"rendering"`: The incoming passage is being rendered and added to the page—takes place during turn processing, so implies `"playing"`.

Examples:

```
Engine.state  → Returns the current state of the engine
```

`Engine.backward()` → *boolean*

Moves backward one moment within the full history (past + future), if possible, activating and showing the moment moved to. Returns whether the history navigation was successful (should only fail if already at the beginning of the full history).

History:

- `v2.0.0`: Introduced.

Parameters: *none*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
Engine.backward()  → Rewinds the full history by one moment—i.e., undoes the moment
```

Engine.forward() → *boolean*

Moves forward one moment within the full history (past + future), if possible, activating and showing the moment moved to. Returns whether the history navigation was successful (should only fail if already at the end of the full history).

History:

- v2.0.0: Introduced.

Parameters: *none*

Examples:

```
Engine.forward()  → Fast forwards the full history by one moment—i.e., redoes the r
```

Engine.go(offset) → *boolean*

Activates the moment at the given offset from the active (present) moment within the full state history and show it. Returns whether the history navigation was successful (should only fail if the offset from the active (present) moment is not within the bounds of the full history).

History:

- v2.0.0: Introduced.

Parameters:

- **offset:** (*integer*) The offset from the active (present) moment of the moment to go to.

Examples:

```
Engine.go(2)   → Fast forwards the full history by two moments—i.e., redoes the mor
Engine.go(-4)  → Rewinds the full history by four moments—i.e., undoes the moments
```

Engine.goTo(index) → *boolean*

Activates the moment at the given index within the full state history and show it. Returns whether the history navigation was successful (should only fail if the index is not within the bounds of the full history).

History:

- v2.0.0: Introduced.

Parameters:

- **index:** (*integer*) The index of the moment to go to.

Examples:

```
Engine.goTo(0)  → Goes to the first moment
Engine.goTo(9)  → Goes to the tenth moment
```

Engine.isIdle() → *boolean*

Returns whether the engine is idle.

History:

- v2.16.0: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters: *none*

Examples:

```
Engine.isIdle()  → Returns whether the engine is idle
```

`Engine.isPlaying()` → *boolean*

Returns whether the engine is processing a turn—i.e., passage navigation has been triggered.

History:

- `v2.16.0`: Introduced.

Parameters: *none*

Examples:

```
Engine.isPlaying()  → Returns whether the engine is playing
```

`Engine.isRendering()` → *boolean*

Returns whether the engine is rendering the incoming passage.

History:

- `v2.16.0`: Introduced.

Parameters: *none*

Examples:

```
Engine.isRendering()  → Returns whether the engine is rendering
```

`Engine.play(passageTitle [, noHistory])` → *`HTMLElement` object*

Renders and displays the passage referenced by the given title, optionally without adding a new moment to the history.

History:

- `v2.0.0`: Introduced.

Parameters:

- `passageTitle`: (*string*) The title of the passage to play.
- `noHistory`: (optional, *boolean*) Disables the update of the history—i.e., no moment is added to the history.

Examples:

```
Engine.play("Foo")        → Renders, displays, and adds a moment for the passage "
Engine.play("Foo", true)  → Renders and displays the passage "Foo", but does not a
```

`Engine.restart()`

Restarts the story.

> **WARNING:** The player will *not* be prompted and all unsaved state will be lost.

> **NOTE:** In general, you should not call this method directly. Instead, call the `UI.restart()` static method, which prompts the player with an OK/Cancel dialog before itself calling `Engine.restart()`, if they accept.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Engine.restart()  → Restarts the story
```

`Engine.show()` → *`HTMLElement`* object

Renders and displays the active (present) moment's associated passage without adding a new moment to the history.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Engine.show()  → Renders and displays the present passage without adding new histo
```

`Fullscreen` API

Provides access to browsers' fullscreen functionality.

Backgrounds in fullscreen

If you wish to use custom backgrounds, either simply colors or with images, then you should place them on the `body` element. For example:

```
body {
	background: #111 fixed url("images/background.png") center / contain no-re
}
```

> **WARNING:** It is ***strongly recommended*** that you do not place background properties on the `html` element in addition to the `body` element as this can cause background jitter in Internet Explorer when scrolling outside of fullscreen mode.

> **WARNING:** If setting a background image via the `background` shorthand property, then you should also specify a `background-color` value with it or include a separate `background-color` property after the `background` property. The reason being is that the `background` property resets the background color, so if you do not set one either as one of its values or via a following `background-color` property, then the browser's default background color could show through if the background image does not cover the entire viewport or includes transparency.

Fullscreen limitations

The `Fullscreen` API comes with some built-in limitations:

1. Fullscreen requests must be initiated by the player, generally via click/touch—i.e., the request must be made as a result of player interaction; e.g., activating a button/link/etc whose code makes the request.

`Fullscreen.element` → *`HTMLElement`* object | *null*

Returns the current fullscreen element or, if fullscreen mode is not active, `null`.

History:

- `v2.31.0`: Introduced.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
Fullscreen.element  → The current fullscreen element
```

`Fullscreen.isEnabled()` → *boolean*

Returns whether fullscreen is both supported and enabled.

History:

- `v2.31.0`: Introduced.

Parameters: *none*

Examples:

```
Fullscreen.isEnabled()  → Whether fullscreen mode is available
```

`Fullscreen.isFullscreen()` → *boolean*

Returns whether fullscreen mode is currently active.

History:

- `v2.31.0`: Introduced.

Parameters: *none*

Examples:

```
Fullscreen.isFullscreen()  → Whether fullscreen mode is active
```

`Fullscreen.request([options [, requestedEl]])` → *`Promise` object*

Request that the browser enter fullscreen mode.

SEE: Backgrounds and limitations.

History:

- `v2.31.0`: Introduced.

Parameters:

- `options`: (optional, *object*) The fullscreen options object.
- `requestedEl`: (optional, *`HTMLElement` object*) The element to enter fullscreen mode with. If omitted, defaults to the entire page.

Options object:

A fullscreen options object should have some of the following properties:

- `navigationUI`: (*string*) Determines the fullscreen navigation UI preference. Valid values are (default: `"auto"`):
  - `"auto"`: Indicates no preference.
  - `"hide"`: Request that the browser's navigation UI be hidden. The full dimensions of the screen will be used to display the element.
  - `"show"`: Request that the browser's navigation UI be shown. The screen dimensions allocated to the element will be clamped to leave room for the UI.

NOTE: Browsers are not currently required to honor the `navigationUI` setting.

Examples:

Basic usage (recommended)

```
/* Request to enter fullscreen mode. */
Fullscreen.request();
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

With options and a specified element

```
/* Request to enter fullscreen mode while showing its navigation UI and with the g
Fullscreen.request({ navigationUI : "show" }, myElement);
```

`Fullscreen.exit()` → *`Promise`* *object*

Request that the browser exit fullscreen mode.

History:

- `v2.31.0`: Introduced.

Parameters: *none*

Examples:

```
/* Request to exit fullscreen mode. */
Fullscreen.exit();
```

`Fullscreen.toggle([options [, requestedEl]])` → *`Promise`* *object*

Request that the browser toggle fullscreen mode—i.e., enter or exit as appropriate.

History:

- `v2.31.0`: Introduced.

Parameters:

- `options`: (optional, *object*) The fullscreen options object. See `Fullscreen.request()` for more information.
- `requestedEl`: (optional, *`HTMLElement`* *object*) The element to toggle fullscreen mode with. If omitted, defaults to the entire page.

Examples:

Basic usage (recommended)

```
/* Request to toggle fullscreen mode. */
Fullscreen.toggle();
```

With options and a specified element

```
/* Request to toggle fullscreen mode while showing its navigation UI and with the g
Fullscreen.toggle({ navigationUI : "show" }, myElement);
```

`Fullscreen.onChange(handlerFn [, requestedEl])`

Attaches fullscreen change event handlers.

History:

- `v2.31.0`: Introduced.

Parameters:

- `handlerFn`: (*function*) The function to invoke when fullscreen mode is changed.
- `requestedEl`: (optional, *`HTMLElement`* *object*) The element to attach the handler to.

Examples:

Basic usage (recommended)

```
/* Attach a hander to listen for fullscreen change events. */
Fullscreen.onChange(function (ev) {
        /* Fullscreen mode changed, so do something. */
});
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

With a specified element

```
/* Attach a hander to the given element to listen for fullscreen change events. */
Fullscreen.onChange(function (ev) {
	/* Fullscreen mode changed on myElement, so do something. */
}, myElement);
```

Fullscreen.offChange([handlerFn [, requestedEl]])

Removes fullscreen change event handlers.

History:

- `v2.31.0`: Introduced.

Parameters:

- `handlerFn`: (optional, *function*) The function to remove. If omitted, will remove all handler functions.
- `requestedEl`: (optional, `HTMLElement` *object*) The element to remove the handler(s) from.

Examples:

Basic usage (recommended)

```
/* Remove all fullscreen change event handers. */
Fullscreen.offChange();
```

```
/* Remove the given fullscreen change event hander. */
/* NOTE: Requires that the original handler function was saved. */
Fullscreen.offChange(originalHandlerFn);
```

With a specified element

```
/* Remove all fullscreen change event handers from myElement. */
Fullscreen.offChange(null, myElement);
```

```
/* Remove the given fullscreen change event hander from myElement. */
/* NOTE: Requires that the original handler function was saved. */
Fullscreen.offChange(originalHandlerFn, myElement);
```

Fullscreen.onError(handlerFn [, requestedEl])

Attaches fullscreen error event handlers.

History:

- `v2.31.0`: Introduced.

Parameters:

- `handlerFn`: (*function*) The function to invoke when fullscreen mode encounters an error.
- `requestedEl`: (optional, `HTMLElement` *object*) The element to attach the handler to.

Examples:

Basic usage (recommended)

```
/* Attach a hander to listen for fullscreen error events. */
Fullscreen.onError(function (ev) {
	/* Fullscreen mode changed, so do something. */
});
```

With a specified element

```
/* Attach a hander to the given element to listen for fullscreen error events. */
Fullscreen.onError(function (ev) {
	/* Fullscreen mode changed on myElement, so do something. */
}, myElement);
```

Fullscreen.offError([handlerFn [, requestedEl]])

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

Removes fullscreen error event handlers.

History:

- `v2.31.0`: Introduced.

Parameters:

- **`handlerFn`:** (optional, *function*) The function to remove. If omitted, will remove all handler functions.
- **`requestedEl`:** (optional, *`HTMLElement` object*) The element to remove the handler(s) from.

Examples:

Basic usage (recommended)

```
/* Remove all fullscreen error event handers. */
Fullscreen.offError();
```

```
/* Remove the given fullscreen error event hander. */
/* NOTE: Requires that the original handler function was saved. */
Fullscreen.offError(originalHandlerFn);
```

With a specified element

```
/* Remove all fullscreen error event handers from myElement. */
Fullscreen.offError(null, myElement);
```

```
/* Remove the given fullscreen error event hander from myElement. */
/* NOTE: Requires that the original handler function was saved. */
Fullscreen.offError(originalHandlerFn, myElement);
```

LoadScreen API

NOTE: To simply add a delay to the dismissal of the loading screen to hide initial flashes of unstyled content (FOUC)—e.g., style changes and page reflows—you do not need to use this API. See the `Config.loadDelay` configuration setting.

`LoadScreen.lock()` → *number*

Acquires a loading screen lock and returns its ID. Displays the loading screen, if necessary.

History:

- `v2.15.0`: Introduced.

Parameters: *none*

Examples:

```
LoadScreen.lock()  → Locks the loading screen and returns the lock ID
```

`LoadScreen.unlock(lockId)`

Releases the loading screen lock with the given ID. Hides the loading screen, if no other locks exist.

History:

- `v2.15.0`: Introduced.

Parameters:

- **`lockId`:** (*integer*) The loading screen lock ID.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
var lockId = LoadScreen.lock();

// Do something whose timing is unpredictable that should be hidden by the loading

LoadScreen.unlock(lockId);
```

Macro API

SEE ALSO: `MacroContext` API.

`Macro.add(name , definition)`

Add new macro(s).

History:

- `v2.0.0`: Introduced
- `v2.33.0`: Obsoleted the `deep` parameter.

Parameters:

- `name`: (*string* | *Array<string>*) Name, or array of names, of the macro(s) to add. **NOTE:** Names must consist of characters from the basic Latin alphabet and start with a letter, which may be optionally followed by any number of letters, numbers, the underscore, or the hyphen.
- `definition`: (*object* | *string*) Definition of the macro(s) or the name of an existing macro whose definition to copy.

Definition object:

A macro definition object should have some of the following properties (only `handler` is absolutely required):

- `skipArgs`: (optional, *boolean* | *Array<string>*) Disables parsing argument strings into discrete arguments. Used by macros that only use the raw/full argument strings. Boolean `true` to affect all tags or an array of tag names to affect.
- `tags`: (optional, *null* | *Array<string>*) Signifies that the macro is a container macro—i.e., not self-closing. An array child tag names or `null`, if there are no child tags.
- `handler`: (*function*) The macro's main function. It will be called without arguments, but with its `this` set to a macro context object.

Additional properties may be added for internal use.

Examples:

```
/*
	Example of a very simple/naive <<if>>/<<elseif>>/<<else>> macro implementat
*/
Macro.add('if', {
	skipArgs : true,
	tags     : ['elseif', 'else'],
	handler  : function () {
		try {
			for (var i = 0, len = this.payload.length; i < len; ++i) {
				if (
					this.payload[i].name === 'else' ||
					!!Scripting.evalJavaScript(this.payload[i]
				) {
					jQuery(this.output).wiki(this.payload[i].c
					break;
				}
			}
		}
		catch (ex) {
			return this.error('bad conditional expression: ' + ex.messa
		}
	}
});
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`Macro.delete(name)`

Remove existing macro(s).

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (*string* | *Array<string>*) Name, or array of names, of the macro(s) to remove.

Examples:

```
Macro.delete("amacro")
Macro.delete(["amacro", "bmacro"])
```

`Macro.get(name)` → *object*

Return the named macro definition, or `null` on failure.

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the macro whose definition should be returned.

Examples:

```
Macro.get("print")
```

`Macro.has(name)` → *boolean*

Returns whether the named macro exists.

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the macro to search for.

Examples:

```
Macro.has("print")
```

`Macro.tags.get(name)` → *Array<string>*

Return the named macro tag's parents array (includes the names of all macros who have registered the tag as a child), or `null` on failure.

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the macro tag whose parents array should be returned.

Examples:

```
Macro.tags.get("else")  → For the standard library, returns: ["if"]
```

`Macro.tags.has(name)` → *boolean*

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

Returns whether the named macro tag exists.

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the macro tag search for.

Examples:

```
Macro.tags.has("else")
```

MacroContext API

> **SEE ALSO:** Macro API.

Macro handlers are called with no arguments, but with their `this` set to a macro (execution) context object. Macro context objects contain the following data and method properties.

<MacroContext>.args → *Array<any>*

An array of discrete arguments parsed from the argument string.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Given: <<someMacro "a" "b" "c">>
this.args.length  → Returns 3
this.args[0]      → Returns 'a'
this.args[1]      → Returns 'b'
this.args[2]      → Returns 'c'
```

<MacroContext>.args.full → *string*

The argument string after converting all TwineScript syntax elements into their native JavaScript counterparts.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Given: <<if "a" is "b">>
this.args.full  → Returns '"a" === "b"'
```

<MacroContext>.args.raw → *string*

The unprocessed argument string.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Given: <<if "a" is "b">>
this.args.raw  → Returns '"a" is "b"'
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`<MacroContext>.name` → *string*

The name of the macro.

History:

- `v2.0.0`: Introduced.

Examples:

```
// Given: <<someMacro …>>
this.name  → Returns 'someMacro'
```

`<MacroContext>.output` → *`HTMLElement`* *object*

The current output element.

History:

- `v2.0.0`: Introduced.

`<MacroContext>.parent` → *null* | *object*

The (execution) context object of the macro's parent, or `null` if the macro has no parent.

History:

- `v2.0.0`: Introduced.

`<MacroContext>.parser` → *object*

The parser instance that generated the macro call.

History:

- `v2.0.0`: Introduced.

`<MacroContext>.payload` → *null* | *Array<object>*

The text of a container macro parsed into discrete payload objects by tag. Payload objects have the following properties:

- `name`: (*string*) Name of the current tag.
- `args`: (*Array<any>*) The current tag's argument string parsed into an array of discrete arguments. Equivalent in function to `<MacroContext>.args`.
  - `args.full`: (*string*) The current tag's argument string after converting all TwineScript syntax elements into their native JavaScript counterparts. Equivalent in function to `<MacroContext>.args.full`.
  - `args.raw`: (*string*) The current tag's unprocessed argument string. Equivalent in function to `<MacroContext>.args.raw`.
- `contents`: (*string*) The current tag's contents—i.e., the text between the current tag and the next.

History:

- `v2.0.0`: Introduced.

`<MacroContext>.self` → *object*

The macro's definition—created via `Macro.add()`.

History:

- `v2.0.0`: Introduced.

`<MacroContext>.contextHas(filter)` → *boolean*

Returns whether any of the macro's ancestors passed the test implemented by the given filter function.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Parameters:

- `filter`: (*function*) The function used to test each ancestor execution context object, which is passed in as its sole parameter.

Examples:

```
var includeAncestor = function (ctx) { return ctx.name === "include"; };
this.contextHas(includeAncestor);  → Returns true if any ancestor was an <<include
```

`<MacroContext>.contextSelect(filter)` → *null | object*

Returns the first of the macro's ancestors that passed the test implemented by the given filter function or `null`, if no members pass.

History:

- `v2.0.0`: Introduced.

Parameters:

- `filter`: (*function*) The function used to test each ancestor execution context object, which is passed in as its sole parameter.

Examples:

```
var includeAncestor = function (ctx) { return ctx.name === "include"; };
this.contextSelect(includeAncestor);  → Returns the first <<include>> macro ancesto
```

`<MacroContext>.contextSelectAll(filter)` → *Array<object>*

Returns a new array containing all of the macro's ancestors that passed the test implemented by the given filter function or an empty array, if no members pass.

History:

- `v2.0.0`: Introduced.

Parameters:

- `filter`: (*function*) The function used to test each ancestor execution context object, which is passed in as its sole parameter.

Examples:

```
var includeAncestor = function (ctx) { return ctx.name === "include"; };
this.contextSelectAll(includeAncestor);  → Returns an array of all <<include>> macı
```

`<MacroContext>.createShadowWrapper(callback [, doneCallback [, startC` → *function*

Returns a callback function that wraps the specified callback functions to provide access to the variable shadowing system used by the `<<capture>>` macro.

NOTE: All of the specified callbacks are invoked as the wrapper is invoked—meaning, with their `this` set to the `this` of the wrapper and with whatever parameters were passed to the wrapper.

WARNING: Only useful when you have an asynchronous callback that invokes code/content that needs to access story and/or temporary variables shadowed by `<<capture>>`. If you don't know what that means, then this API is likely not for you.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

History:

- `v2.14.0`: Introduced.
- `v2.23.3`: Fixed an issue where shadows would fail for multiple layers of nested asynchronous code due to loss of context.

Parameters:

- `callback`: (*function*) The main callback function, executed when the wrapper is invoked. Receives access to variable shadows.
- `doneCallback`: (optional, *function*) The finalization callback function, executed after the main `callback` returns. Does not receive access to variable shadows.
- `startCallback`: (optional, *function*) The initialization callback function, executed before the main `callback` is invoked. Does not receive access to variable shadows.

Examples:

Basic usage

```
$someElement.on('some_event', this.createShadowWrapper(function (ev) {
	/* main asynchronous code */
}));
```

With an optional `doneCallback`

```
$someElement.on('some_event', this.createShadowWrapper(
	function (ev) {
		/* main asynchronous code */
	},
	function (ev) {
		/* asynchronous code invoked after the main code */
	}
));
```

With an optional `doneCallback` and `startCallback`

```
$someElement.on('some_event', this.createShadowWrapper(
	function (ev) {

		/* main asynchronous code */
	},
	function (ev) {
		/* asynchronous code invoked after the main code */
	},
	function (ev) {
		/* asynchronous code invoked before the main code */
	}
));
```

`<MacroContext>.error(message)` → *boolean:false*

Renders the message prefixed with the name of the macro and returns `false`.

History:

- `v2.0.0`: Introduced.

Parameters:

- `message`: (*string*) The error message to output.

Examples:

```
// Given: <<someMacro …>>
return this.error("oops");  → Outputs '<<someMacro>>: oops'
```

Passage API

Instances of the `Passage` object are returned by the `Story.get()` static method.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

All properties of `Passage` objects should be treated as if they were ***read-only***, as modifying them could result in unexpected behavior.

`<Passage>.domId` → *string*

The DOM ID of the passage, created from the slugified passage title.

History:

- `v2.0.0`: Introduced.

`<Passage>.tags` → *Array<string>*

The tags of the passage.

History:

- `v2.0.0`: Introduced.

`<Passage>.text` → *string*

The raw text of the passage.

History:

- `v2.0.0`: Introduced.

`<Passage>.title` → *string*

The title of the passage.

History:

- `v2.0.0`: Introduced.

`<Passage>.description()` → *string*

Returns the description of the passage, created from either an excerpt of the passage or the `Config.passages.descriptions` setting.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
var passage = Story.get("The Ducky");

passage.description()  → Returns the description of "The Ducky" passage
```

`<Passage>.processText()` → *string*

Returns the processed text of the passage, created from applying `nobr` tag and image passage processing to its raw text.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
var passage = Story.get("The Ducky");

passage.processText()  → Returns the fully processed text of "The Ducky" passage
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Save API

NOTE: There are several configuration settings for saves that it would be wise for you to familiarize yourself with.

WARNING: In-browser saves—i.e., autosave and slot saves—are largely incompatible with private browsing modes, which cause all in-browser storage mechanisms to either persist only for the lifetime of the browsing session or fail outright.

Save Objects

NOTE: Adding additional properties directly to save objects is not recommended. Instead, use the `metadata` property.

Save objects have some of the following properties:

- `id`: (*string*) The story's save ID.
- `state`: (*object*) The marshaled story history (see below for details).
- `title`: (*string*) The title of the save.
- `date`: (*integer*) The date when the save was created (in milliseconds elapsed since epoch).
- `metadata`: (optional, *any*) Save metadata (end-user specified; must be JSON-serializable).
- `version`: (optional, *any*) Save version (end-user specified via `Config.saves.version`).

The `state` object has the following properties:

- `history`: (*Array<object>*) The array of moment objects (see below for details).
- `index`: (*integer*) The index of the active moment.
- `expired`: (optional, *Array<string>*) The array of expired moment passage titles, exists only if any moments have expired.
- `seed`: (optional, *string*) The seed of SugarCube's seedable PRNG, exists only if enabled.

Each `moment` object has the following properties:

- `title`: (*string*) The title of the associated passage.
- `variables`: (*object*) The current variable store object, which contains sigil-less name ⇒ value pairs—e.g., `$foo` exists as `foo`.
- `pull`: (optional, *integer*) The current pull count of SugarCube's seedable PRNG, exists only if enabled.

General

`Save.clear()`

Deletes all slot saves and the autosave, if it's enabled.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.clear()
```

`Save.get()`

Returns the saves object.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.get()
```

`Save.ok()` → *boolean*

Returns whether both the slot saves and autosave are available and ready.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
if (Save.ok()) {
	/* Code to manipulate saves. */
}
```

Slots

`Save.slots.length` → *integer*

Returns the total number of available slots.

History:

- `v2.0.0`: Introduced.

Examples:

```
Save.slots.length
```

`Save.slots.count()` → *integer*

Returns the total number of filled slots.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.slots.count()
```

`Save.slots.delete(slot)`

Deletes a save from the given slot.

History:

- `v2.0.0`: Introduced.

Parameters:

- `slot`: (*integer*) Save slot index (0-based).

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
Save.slots.delete(5)  → Deletes the sixth slot save
```

`Save.slots.get(slot)` → *object*

Returns a save object from the given slot or `null`, if there was no save in the given slot.

History:

- `v2.0.0`: Introduced.

Parameters:

- `slot`: (*integer*) Save slot index (0-based).

Examples:

```
Save.slots.get(5)  → Returns the sixth slot save
```

`Save.slots.has(slot)` → *boolean*

Returns whether the given slot is filled.

History:

- `v2.0.0`: Introduced.

Parameters:

- `slot`: (*integer*) Save slot index (0-based).

Examples:

```
if (Save.slots.has(5)) {
        /* Code to manipulate the sixth slot save. */
}
```

`Save.slots.isEmpty()` → *boolean*

Returns whether there are any filled slots.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.slots.isEmpty()  → Effectively returns: Save.slots.count() === 0
```

`Save.slots.load(slot)`

Loads a save from the given slot.

History:

- `v2.0.0`: Introduced.

Parameters:

- `slot`: (*integer*) Save slot index (0-based).

Examples:

```
Save.slots.load(5)  → Load the sixth slot save
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`Save.slots.ok()` → *boolean*

Returns whether the slot saves are available and ready.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
if (Save.slots.ok()) {
        /* Code to manipulate slot saves. */
}
```

`Save.slots.save(slot [, title [, metadata]])`

Saves to the given slot.

History:

- `v2.0.0`: Introduced.

Parameters:

- `slot`: (*integer*) Save slot index (0-based).
- `title`: (optional, *string*) The title of the save. If omitted or `null`, defaults to the passage's description.
- `metadata`: (optional, *any*) The data to be stored in the save object's `metadata` property. Must be JSON-serializable.

Examples:

```
→ Save to the sixth slot save with the default title and no metadata
Save.slots.save(5)


→ Save to the sixth slot save with the title "Midgar" and no metadata
Save.slots.save(5, "Midgar")

→ Save to the sixth slot save with the default title and metadata someMetadata
Save.slots.save(5, null, someMetadata)

→ Save to the sixth slot save with the title "Midgar" and metadata someMetadata
Save.slots.save(5, "Midgar", someMetadata)
```

Autosave

`Save.autosave.delete()`

Deletes the autosave.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.autosave.delete()  → Deletes the autosave
```

`Save.autosave.get()` → *object*

Returns the save object from the autosave or `null`, if there was no autosave.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.autosave.get()  → Returns the autosave
```

`Save.autosave.has()` → *boolean*

Returns whether the autosave is filled.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
if (Save.autosave.has()) {
	/* Code to manipulate the autosave. */
}
```

`Save.autosave.load()`

Loads the autosave.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Save.autosave.load()  → Load the autosave
```

`Save.autosave.ok()` → *boolean*

Returns whether the autosave is available and ready.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
if (Save.autosave.ok()) {
	/* Code to manipulate the autosave. */
}
```

`Save.autosave.save([title [, metadata]])`

Saves to the autosave.

History:

- `v2.0.0`: Introduced.

Parameters:

- `title`: (optional, *string*) The title of the save. If omitted or `null`, defaults to the passage's description.
- `metadata`: (optional, *any*) The data to be stored in the save object's `metadata` property. Must be JSON-serializable.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
→ Save to the autosave with the default title and no metadata
Save.autosave.save()

→ Save to the autosave with the title "Midgar" and no metadata
Save.autosave.save("Midgar")

→ Save to the autosave with the default title and metadata someMetadata
Save.autosave.save(null, someMetadata)

→ Save to the autosave with the title "Midgar" and metadata someMetadata
Save.autosave.save("Midgar", someMetadata)
```

Disk

`Save.export([filename [, metadata]])`

Saves to disk.

History:

- `v2.0.0`: Introduced.
- `v2.8.0`: Added `metadata` parameter.

Parameters:

- `filename`: (optional, *string*) The base filename of the save, which gets slugified to remove most symbols. Appended to this is a datestamp (format: `YYYYMMDD-hhmmss`) and the file extension `.save`—e.g., `"The Scooby Chronicles"` would result in the full filename: `the-scooby-chronicles-{datestamp}.save`. If omitted or `null`, defaults to the story's title.
- `metadata`: (optional, *any*) The data to be stored in the save object's `metadata` property. Must be JSON-serializable.

Examples:

```
→ Export a save with the default filename and no metadata
Save.export()

→ Export a save with the filename "the-7th-fantasy-{datestamp}.save" and no metadat
Save.export("The 7th Fantasy")

→ Export a save with the default filename and metadata someMetadata
Save.export(null, someMetadata)

→ Export a save with the filename "the-7th-fantasy-{datestamp}.save" and metadata s
Save.export("The 7th Fantasy", someMetadata)
```

`Save.import(event)`

Loads a save from disk.

NOTE: You do not call this manually, it *must* be called by the `change` event handler of an `<input type="file">` element.

History:

- `v2.0.0`: Introduced.

Parameters:

- `event`: (*event object*) The event object that was passed to the `change` event handler of the associated `<input type="file">` element.

Examples:

Basic usage

```
// Assuming you're creating a file input something like the following
var input  = document.createElement('input');
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
input.type = 'file';

input.id   = 'saves-import-file';
input.name = 'saves-import-file';

// Set up Save.import() as the event handler for when a file has been chosen
jQuery(input).on('change', Save.import);
```

In case you needed to do more than simply load the save, you may do something like the following:

```
// Assuming you're creating a file input something like the following
var input  = document.createElement('input');
input.type = 'file';
input.id   = 'saves-import-file';
input.name = 'saves-import-file';

// Set up a custom event handler for when a file has been chosen, which will call S
jQuery(input).on('change', function (ev) {
        // You must pass in the event when calling Save.import() manually
        Save.import(ev);

        // Put anything else you needed to do here
});
```

As a self-contained link/button using macros

```
<<link "Load From Disk">>
        <<script>>
        jQuery(document.createElement('input'))
                .prop('type', 'file')
                .on('change', Save.import)
                .trigger('click');
        <</script>>
<</link>>
```

Serialization

`Save.serialize([metadata])` → *string* | *null*

Returns a save as a serialized string, or `null` if saving is not allowed within the current context.

History:

- `v2.21.0`: Introduced.

Parameters:

- `metadata`: (optional, *any*) The data to be stored as metadata. Must be JSON-serializable.

Examples:

```
→ Serialize a save with no metadata
const myGameState = Save.serialize();
if (myGameState === null) {
        /* Failure.  You've disallowed saving. */
}

→ Serialize a save with metadata someMetadata
const myGameState = Save.serialize(someMetadata);
if (myGameState === null) {
        /* Failure.  You've disallowed saving. */
}
```

`Save.deserialize(saveStr)` → *any* | *null*

Deserializes the given save string, created via `Save.serialize()`, and loads the save. Returns the bundled metadata, if any, or `null` if the given save could not be deserialized and loaded.

History:

- `v2.21.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters:

- `saveStr`: (*string*) The serialized save string.

Examples:

```
→ Deserialize a save with no metadata
const loadResult = Save.deserialize(myGameState);
if (loadResult === null) {
	/* Failure.  An error was displayed to the player. */
}

→ Deserialize a save with metadata
const loadResult = Save.deserialize(myGameState);
if (loadResult !== null) {
	/* Success.  Do something with loadResult, which contains the metadata. */
}
else {
	/* Failure.  An error was displayed to the player. */
}
```

Events

`Save.onLoad.add(handler)`

Performs any required processing before the save data is loaded—e.g., upgrading out-of-date save data. The handler is passed one parameter, the save object to be processed. If it encounters an unrecoverable problem during its processing, it may throw an exception containing an error message; the message will be displayed to the player and loading of the save will be terminated.

History:

- `v2.36.0`: Introduced.

Parameters:

- `handler`: (*function*) The handler function to be executed upon the loading of a save.

Handler parameters:

- `save`: (*object*) The save object to be processed.

Examples:

```
Save.onLoad.add(function (save) {
	/* code to process the save object before it's loaded */
});
```

`Save.onLoad.clear()`

Deletes all currently registered on-load handlers.

History:

- `v2.36.0`: Introduced.

Parameters: *none*

Examples:

```
Save.onLoad.clear();
```

`Save.onLoad.delete(handler)` → *boolean*

Deletes the specified on-load handler, returning `true` if the handler existed or `false` if not.

History:

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

- `v2.36.0`: Introduced.

Parameters:

- `handler`: (*function*) The handler function to be deleted.

Examples:

```
// Given:
//      let myOnLoadHandler = function (save) {
//              /* code to process the save object before it's loaded */
//      };
//      Save.onLoad.add(myOnLoadHandler);

Save.onLoad.delete(myOnLoadHandler);
```

`Save.onLoad.size` → *integer*

Returns the number of currently registered on-load handlers.

History:

- `v2.36.0`: Introduced.

Parameters: *none*

Examples:

```
console.log('There are %d onLoad handlers registered.', Save.onLoad.size);
```

`Save.onSave.add(handler)`

Performs any required processing before the save data is saved. The handlers is passed two parameters, the save object to be processed and save operation details object.

History:

- `v2.36.0`: Introduced.

Parameters:

- `handler`: (*function*) The handler function to be executed upon the saving of a save.

Handler parameters:

- `save`: (*object*) The save object to be processed.
- `details`: (*object*) The save operation details object.

Save operation details object:

A save operation details object will have the following properties:

- `type`: (*string*) A string representing what caused the save operation. Possible values are: `'autosave'`, `'disk'`, `'serialize'`, `'slot'`.

Examples:

Using only the save object parameter

```
Save.onSave.add(function (save) {
        /* code to process the save object before it's saved */
});
```

Using both the save object and operation details parameters

```
Save.onSave.add(function (save, details) {
        switch (details.type) {
                case 'autosave': {
                        /* code to process the save object from autosaves before i
                        break;
                }
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
                case 'disk': {
                        /* code to process the save object from disk saves before
                        break;
                }

                case 'serialize': {
                        /* code to process the save object from serialize saves be
                        break;
                }

                default: { /* slot */
                        /* code to process the save object from slot saves before
                        break;
                }
        }
});
```

Save.onSave.clear()

Deletes all currently registered on-save handlers.

History:

- v2.36.0: Introduced.

Parameters: *none*

Examples:

```
Save.onSave.clear();
```

Save.onSave.delete(handler) → *boolean*

Deletes the specified on-save handler, returning true if the handler existed or false if not.

History:

- v2.36.0: Introduced.

Parameters:

- handler: (*function*) The handler function to be deleted.

Examples:

```
// Given:
//      let myOnSaveHandler = function (save, details) {
//              /* code to process the save object before it's saved */
//      };
//      Save.onSave.add(myOnSaveHandler);

Save.onSave.delete(myOnSaveHandler);
```

Save.onSave.size → *integer*

Returns the number of currently registered on-save handlers.

History:

- v2.36.0: Introduced.

Parameters: *none*

Examples:

```
console.log('There are %d onSave handlers registered.', Save.onSave.size);
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Setting API

Manages the Settings dialog and `settings` object.

> **WARNING:** `Setting` API method calls ***must*** be placed within your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage) or settings will not function correctly.

`Setting.addHeader(name [, desc])`

Adds a header to the Settings dialog.

History:

- `v2.7.1`: Introduced.

Parameters:

- `name`: (*string*) Name of the header.
- `desc`: (optional, *string*) Description explaining the header in greater detail.

Examples:

```
// Setting up a basic header
Setting.addHeader("Content Settings");

// Setting up a header w/ a description
Setting.addHeader("Content Settings", "Settings controlling what content is made av
```

`Setting.addToggle(name, definition)`

Adds the named property to the `settings` object and a toggle control for it to the Settings dialog.

History:

- `v2.0.0`: Introduced.
- `v2.26.0`: Added `desc` property to definition object.

Parameters:

- `name`: (*string*) Name of the `settings` property to add, which the control will manage.
- `definition`: (*object*) Definition of the control.

Definition object:

A toggle definition object should have some of the following properties:

- `label`: (*string*) Label to use for the control.
- `desc`: (optional, *string*) Description explaining the control in greater detail.
- `default`: (optional, *boolean*) The default value for the setting and default state of the control. Leaving it undefined means to use `false` as the default.
- `onInit`: (optional, *function*) The function to call during initialization.
- `onChange`: (optional, *function*) The function to call when the control's state is changed.

Examples:

Basic toggle setting

```
// Setting up a basic toggle control for the settings property 'mature'
Setting.addToggle("mature", {
        label : "Content for mature audiences?"
}); // default value not defined, so false is used
```

Toggle that adds/removes a CSS class

```
// Setting up a toggle control for the settings property 'widescreen' w/ callbacks
var settingWidescreenHandler = function () {
        if (settings.widescreen) { // is true
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
                $("html").addClass("widescreen");
        }
        else { // is false
                $("html").removeClass("widescreen");
        }
};
Setting.addToggle("widescreen", {
        label    : "Allow the story to use the full width of your browser window?"
        default  : false,
        onInit   : settingWidescreenHandler,
        onChange : settingWidescreenHandler
});
```

And the requisite CSS style rule:

```
html.widescreen #passages {
        max-width: none;
}
```

`Setting.addList(name, definition)`

Adds the named property to the `settings` object and a list control for it to the Settings dialog.

History:

- `v2.0.0`: Introduced.
- `v2.26.0`: Added `desc` property to definition object.

Parameters:

- **`name`:** (*string*) Name of the `settings` property to add, which the control will manage.
- **`definition`:** (*object*) Definition of the control.

Definition object:

A list definition object should have some of the following properties:

- **`label`:** (*string*) Label to use for the control.
- **`list`:** (*Array<string>*) The array of items.
- **`desc`:** (optional, *string*) Description explaining the control in greater detail.
- **`default`:** (optional, *[as `list` array]*) The default value for the setting and default state of the control. It should have the same value as one of the members of the `list` array. Leaving it undefined means to use the first array member as the default.
- **`onInit`:** (optional, *function*) The function to call during initialization.
- **`onChange`:** (optional, *function*) The function to call when the control's state is changed.

Examples:

```
// Setting up a basic list control for the settings property 'difficulty'
Setting.addList("difficulty", {
        label   : "Choose a difficulty level.",
        list    : ["Easy", "Normal", "Hard", "INSANE"],
        default : "Normal"
});

// Setting up a list control for the settings property 'theme' w/ callbacks
var settingThemeNames = ["(none)", "Bright Lights", "Charcoal", "Midnight", "Tinse
var settingThemeHandler = function () {
        // cache the jQuery-wrapped <html> element
        var $html = $("html");

        // remove any existing theme class
        $html.removeClass("theme-bright-lights theme-charcoal theme-midnight theme

        // switch on the theme name to add the requested theme class
        switch (settings.theme) {
        case "Bright Lights":
                $html.addClass("theme-bright-lights");
                break;
        case "Charcoal":
                $html.addClass("theme-charcoal");
                break;
        case "Midnight":
                $html.addClass("theme-midnight");
                break;
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        case "Tinsel City":
                $html.addClass("theme-tinsel-city");
                break;
        }
};
Setting.addList("theme", {
        label    : "Choose a theme.",
        list     : settingThemeNames,
        onInit   : settingThemeHandler,
        onChange : settingThemeHandler
}); // default value not defined, so the first array member "(none)" is used
```

Setting.addRange(name, definition)

Adds the named property to the `settings` object and a range control for it to the Settings dialog.

History:

- `v2.26.0`: Introduced.

Parameters:

- **`name`:** (*string*) Name of the `settings` property to add, which the control will manage.
- **`definition`:** (*object*) Definition of the control.

Definition object:

A range definition object should have some of the following properties:

- **`label`:** (*string*) Label to use for the control.
- **`min`:** (*number*) The minimum value.
- **`max`:** (*number*) The maximum value.
- **`step`:** (*number*) Limits the increments to which the value may be set. It must be evenly divisible into the full range—i.e., `max - min`.
- **`desc`:** (optional, *string*) Description explaining the control in greater detail.
- **`default`:** (optional, *number*) The default value for the setting and default state of the control. Leaving it undefined means to use the value of `max` as the default.
- **`onInit`:** (optional, *function*) The function to call during initialization.
- **`onChange`:** (optional, *function*) The function to call when the control's state is changed.

Examples:

```
// Setting up a volume control for the settings property 'masterVolume' w/ callbacl
Setting.addRange("masterVolume", {
        label    : "Master volume.",
        min      : 0,
        max      : 10,
        step     : 1,
        onChange : function () {
                SimpleAudio.volume(settings.masterVolume / 10);
        }
}); // default value not defined, so max value (10) is used
```

Setting.load()

Loads the settings from storage.

> **NOTE:** The API automatically calls this method at startup, so you should never need to call this method manually.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Setting.load();
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`Setting.reset([name])`

Resets the setting with the given name to its default value. If no name is given, resets all settings.

History:

- `v2.0.0`: Introduced.

Parameters:

- `name`: (optional, *string*) Name of the `settings` object property to reset.

Examples:

```
// Reset the setting 'difficulty'
Setting.reset("difficulty");

// Reset all settings
Setting.reset();
```

`Setting.save()`

Saves the settings to storage.

> **NOTE:** The controls of the Settings dialog automatically call this method when settings are changed, so you should normally never need to call this method manually. Only when manually modifying the values of `settings` object properties, outside of the controls, would you need to call this method.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
Setting.save();
```

`settings` object

A prototype-less generic object whose properties and values are defined by the `Setting.addToggle()`, `Setting.addList()`, and `Setting.addRange()` methods.

Normally, the values of its properties are automatically managed by their associated Settings dialog control. If necessary, however, you may manually change their values—n.b. you'll need to call the `Setting.save()` after having done so.

History:

- `v2.0.0`: Introduced.

`SimpleAudio` API

The core audio subsystem and backend for the audio macros.

> **SEE ALSO:** `AudioTrack` API, `AudioRunner` API, and `AudioList` API.

Audio limitations

The audio subsystem is based upon the HTML Media Elements APIs and comes with some built-in limitations:

1. True gapless transitions between tracks is not supported.
2. In mobile browsers, playback volume is controlled by the device hardware. Thus, all volume adjustments are ignored by the device, though muting should work normally.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

3. In mobile browsers and, more recently, most desktop browsers, playback must be initiated by the player—generally via click/touch. In these cases, audio will not automatically play on the starting passage, nor is it likely to play if initiated from within asynchronous code—e.g., via `<<timed>>`—though this ultimately depends on various factors. A simple solution for the former is to use some kind of click/touch-through screen—e.g., a splash screen, which the player goes through to the real starting passage. The latter is harder to resolve, so is best avoided.
4. The load and playback states of tracks are not currently recorded within the active play session or saves. Thus, if you need either to be recoverable, then you'll have to handle that yourself.

General

`SimpleAudio.load()`

Pauses playback of *all* currently registered tracks and, if they're not already in the process of loading, force them to drop any existing data and begin loading.

> **WARNING:** This *should not* be done lightly if your audio sources are on the network, as it forces players to begin downloading them.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.load();
```

`SimpleAudio.loadWithScreen()`

Displays the loading screen until *all* currently registered audio tracks have either loaded to a playable state or aborted loading due to errors. The loading process is as described in `SimpleAudio.load()`.

> **WARNING:** This *should not* be done lightly if your audio sources are on the network, as it forces players to begin downloading them.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.loadWithScreen();
```

`SimpleAudio.mute([state])` → get: *boolean* | set: *undefined*

Gets or sets the mute state for the master volume (default: `false`).

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The mute state.

Examples:

```
// Get the current master volume mute state.
var isMuted = SimpleAudio.mute();

// Mute the master volume.
SimpleAudio.mute(true);
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
// Unmute the master volume.
SimpleAudio.mute(false);
```

SimpleAudio.muteOnHidden([state]) → get: *boolean* | set: *undefined*

Gets or sets the mute-on-hidden state for the master volume (default: `false`). The mute-on-hidden state controls whether the master volume is automatically muted/unmuted when the story's browser tab loses/gains visibility. Loss of visibility is defined as when the browser window is either switched to another tab or minimized.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The mute-on-hidden state.

Examples:

```
// Get the current master volume mute-on-hidden state.
var isMuteOnHidden = SimpleAudio.muteOnHidden();

// Enable automatic muting of the master volume when visibility is lost.
SimpleAudio.muteOnHidden(true);

// Disable automatic muting of the master volume when visibility is lost.
SimpleAudio.muteOnHidden(false);
```

SimpleAudio.select(selector) → *AudioRunner object*

Returns an `AudioRunner` instance for the tracks matching the given selector.

History:

- `v2.28.0`: Introduced.

Parameters:

- `selector`: (*string*) The list of audio track(s) and/or group ID(s), separated by spaces. There are several predefined group IDs (`:all`, `:looped`, `:muted`, `:paused`, `:playing`). The `:not()` group modifier syntax (`groupId:not(selector)`) allows a group to have some of its tracks excluded from selection.

Examples:

Basic usage

```
SimpleAudio.select(":ui")  → Returns the AudioRunner instance for the tracks match
```

Typical usage

```
// Return the AudioTrack instance matching "swamped" and do something with it
SimpleAudio.select("swamped").volume(1).play();

// Start playback of paused audio tracks
SimpleAudio.select(":paused").play();

// Pause playback of playing audio tracks
SimpleAudio.select(":playing").pause();

// Stop playback of playing audio tracks
SimpleAudio.select(":playing").stop();

// Stop playback of all audio tracks (not uniquely part of a playlist)
SimpleAudio.select(":all").stop();

// Stop playback of playing audio tracks except those in the ":ui" group
SimpleAudio.select(":playing:not(:ui)").stop();

// Change the volume of all audio tracks except those in the ":ui" group
// to 40%, without changing the current playback state
SimpleAudio.select(":all:not(:ui)").volume(0.40);
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`SimpleAudio.stop()`

Stops playback of *all* currently registered tracks.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.stop();
```

`SimpleAudio.unload()`

Stops playback of *all* currently registered tracks and force them to drop any existing data.

> **NOTE:** Once a track has been unloaded, playback cannot occur until it is reloaded.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.unload();
```

`SimpleAudio.volume([level])` → get: *number* | set: *undefined*

Gets or sets the master volume level (default: `1`).

History:

- `v2.28.0`: Introduced.

Parameters:

- `level`: (optional, *number*) The volume level to set. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Examples:

```
// Get the current master volume level.
var currentMasterVolume = SimpleAudio.volume();

// Set the master volume level to 75%.
SimpleAudio.volume(0.75);
```

Tracks

`SimpleAudio.tracks.add(trackId, sources…)`

Adds an audio track with the given track ID.

History:

- `v2.28.0`: Introduced.

Parameters:

- `trackId`: (*string*) The ID of the track, which will be used to reference it.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `sources`: (*string*... | *Array*<*string*>) The audio sources for the track, which may be a list of sources or an array. Only one is required, though supplying additional sources in differing formats is recommended, as no single format is supported by all browsers. A source must be either a URL (absolute or relative) to an audio resource, the name of an audio passage, or a data URI. In rare cases where the audio format cannot be automatically detected from the source (URLs are parsed for a file extension, data URIs are parsed for the media type), a format specifier may be prepended to the front of each source to manually specify the format (syntax: `formatId|`, where `formatId` is the audio format—generally, whatever the file extension would normally be; e.g., `mp3`, `mp4`, `ogg`, `weba`, `wav`).

Examples:

```
// Cache a track with the ID "boom" and one source via relative URL
SimpleAudio.tracks.add("boom", "media/audio/explosion.mp3");

// Cache a track with the ID "boom" and one source via audio passage
SimpleAudio.tracks.add("boom", "explosion");

// Cache a track with the ID "bgm_space" and two sources via relative URLs
SimpleAudio.tracks.add("bgm_space", "media/audio/space_quest.mp3", "media/audio/spa

// Cache a track with the ID "what" and one source via URL with a format specifier
SimpleAudio.tracks.add("what", "mp3|http://an-audio-service.com/a-user/a-track-id")
```

`SimpleAudio.tracks.clear()`

Deletes all audio tracks.

> **NOTE:** Cannot delete tracks solely under the control of a playlist.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.tracks.clear();
```

`SimpleAudio.tracks.delete(trackId)`

Deletes the audio track with the given track ID.

> **NOTE:** Cannot delete tracks solely under the control of a playlist.

> **WARNING:** Does not currently remove the track from either groups or playlists. Thus, any groups or playlists containing the deleted track should be rebuilt.

History:

- `v2.28.0`: Introduced.

Parameters:

- `trackId`: (*string*) The ID of the track.

Examples:

```
SimpleAudio.tracks.delete("bgm_space");
```

`SimpleAudio.tracks.get(trackId)` → *`AudioTrack`* ***object | null***

Returns the `AudioTrack` instance with the given track ID, or `null` on failure.

> **NOTE:** To affect multiple tracks and/or groups at once, see the `SimpleAudio.select()` method.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.28.0`: Introduced.

Parameters:

- **`trackId`:** (*string*) The ID of the track.

Examples:

Basic usage

```
SimpleAudio.tracks.get("swamped")  → Returns the AudioTrack instance matching "swar
```

Typical usage

```
// Return the AudioTrack instance matching "swamped" and do something with it
SimpleAudio.tracks.get("swamped").volume(1).play();
```

`SimpleAudio.tracks.has(trackId)` → *boolean*

Returns whether an audio track with the given track ID exists.

History:

- `v2.28.0`: Introduced.

Parameters:

- **`trackId`:** (*string*) The ID of the track.

Examples:

```
if (SimpleAudio.tracks.has("bgm_space")) {
        // Track "bgm_space" exists.
}
```

Groups

`SimpleAudio.groups.add(groupId, trackIds…)`

Adds an audio group with the given group ID. Groups are useful for applying actions to multiple tracks simultaneously and/or excluding the included tracks from a larger set when applying actions.

> **NOTE:** If you want to play tracks in a sequence, then you want a playlist instead.

History:

- `v2.28.0`: Introduced.

Parameters:

- **`groupId`:** (*string*) The ID of the group, which will be used to reference it and *must* begin with a colon.
  NOTE: There are several predefined group IDs (`:all`, `:looped`, `:muted`, `:paused`, `:playing`) and the `:not` group modifier that cannot be reused/overwritten.
- **`trackIds`:** (*string*... | *Array*<*string*>) The IDs of the tracks to make part of the group, which may be a list of track IDs or an array.

Examples:

```
// Set up a group ":ui" with the tracks: "ui_beep", "ui_boop", and "ui_swish"
SimpleAudio.groups.add(":ui", "ui_beep", "ui_boop", "ui_swish");
```

`SimpleAudio.groups.clear()`

Deletes all audio groups.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: Only deletes the groups themselves, does not affect their component tracks.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.groups.clear();
```

`SimpleAudio.groups.delete(groupId)`

Deletes the audio group with the given group ID.

NOTE: Only deletes the group itself, does not affect its component tracks.

History:

- `v2.28.0`: Introduced.

Parameters:

- `groupId`: (*string*) The ID of the group.

Examples:

```
SimpleAudio.groups.delete(":ui");
```

`SimpleAudio.groups.get(groupId)` → *Array<string> | null*

Returns the array of track IDs with the given group ID, or `null` on failure.

NOTE: To actually affect multiple tracks and/or groups, see the `SimpleAudio.select()` method.

History:

- `v2.28.0`: Introduced.

Parameters:

- `groupId`: (*string*) The ID of the group.

Examples:

```
SimpleAudio.groups.get(":ui")  → Returns the array of track IDs matching ":ui"
```

`SimpleAudio.groups.has(groupId)` → *boolean*

Returns whether an audio group with the given group ID exists.

History:

- `v2.28.0`: Introduced.

Parameters:

- `groupId`: (*string*) The ID of the group.

Examples:

```
if (SimpleAudio.groups.has(":ui")) {
        // Group ":ui" exists.
}
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Lists

`SimpleAudio.lists.add(listId, sources…)`

Adds a playlist with the given list ID. Playlists are useful for playing tracks in a sequence—i.e., one after another.

> **NOTE:** If you simply want to apply actions to multiple tracks simultaneously, then you want a group instead.

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Changed descriptor object `copy` property to `own`.

Parameters:

- **`listId`:** (*string*) The ID of the list, which will be used to reference it.
- **`sources`:** (*string* | *object* | *Array<string | object>*) The track IDs or descriptors of the tracks to make part of the list, which may be specified as a list or an array.

Descriptor objects:

Track descriptor objects come in two forms and should have some of the noted properties:

- **Existing track form:** `{ id, [own], [volume] }`
  - **`id`:** (*string*) The ID of an existing track.
  - **`own`:** (optional, *boolean*) When `true`, signifies that the playlist should create its own independent copy of the track, rather than simply referencing the existing instance. Owned copies are solely under the control of their playlist and cannot be selected with either the `SimpleAudio.tracks.get()` method or the `SimpleAudio.select()` method.
  - **`volume`:** (optional, *number*) The base volume level of the track within the playlist. If omitted, defaults to the track's current volume. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.
- **New track form:** `{ sources, [volume] }`
  - **`sources`:** (*Array<string>*) The audio sources for the track. Only one is required, though supplying additional sources in differing formats is recommended, as no single format is supported by all browsers. A source must be either a URL (absolute or relative) to an audio resource, the name of an audio passage, or a data URI. In rare cases where the audio format cannot be automatically detected from the source (URLs are parsed for a file extension, data URIs are parsed for the media type), a format specifier may be prepended to the front of each source to manually specify the format (syntax: `formatId|`, where `formatId` is the audio format—generally, whatever the file extension would normally be; e.g., `mp3`, `mp4`, `ogg`, `weba`, `wav`).
  - **`volume`:** (optional, *number*) The base volume level of the track within the playlist. If omitted, defaults to `1` (loudest). Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Examples:

Basic usage with track IDs

```
// Add existing tracks at their current volumes
SimpleAudio.lists.add("bgm_lacuna", "swamped", "heavens_a_lie", "closer", "to_the_
```

Using a mix of track IDs and descriptors

```
SimpleAudio.lists.add("bgm_lacuna",
        // Add existing track "swamped" at its current volume
        "swamped",

        // Add existing track "heavens_a_lie" at 50% volume
        {
                id     : "heavens_a_lie",
                volume : 0.5
        },

        // Add an owned copy of existing track "closer" at its current volume
        {
                id  : "closer",
                own : true
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        },

        // Add an owned copy of existing track "to_the_edge" at 100% volume
        {
                id     : "to_the_edge",
                own    : true,
                volume : 1
        }
);
```

Using descriptors with sources

```
SimpleAudio.lists.add("bgm_lacuna",
        // Add a track from the given sources at the default volume (100%)
        {
                sources : ["media/audio/Swamped.mp3"]
        }

        // Add a track from the given sources at 50% volume
        {
                sources : ["media/audio/Heaven's_A_Lie.mp3"],
                volume  : 0.5
        },

        // Add a track from the given sources at the default volume (100%)
        {
                sources : ["media/audio/Closer.mp3"]
        },

        // Add a track from the given sources at 100% volume
        {
                sources : ["media/audio/To_The_Edge.mp3"],
                volume  : 1
        }
);
```

SimpleAudio.lists.clear()

Deletes all playlists.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
SimpleAudio.lists.clear();
```

SimpleAudio.lists.delete(listId)

Deletes the playlist with the given list ID.

History:

- v2.28.0: Introduced.

Parameters:

- listId: (*string*) The ID of the playlist.

Examples:

```
SimpleAudio.lists.delete("bgm_lacuna");
```

SimpleAudio.lists.get(listId) → *AudioList* ***object | null***

Returns the AudioList instance with the given list ID, or null on failure.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.28.0`: Introduced.

Parameters:

- `listId`: (*string*) The ID of the playlist.

Examples:

Basic usage

```
SimpleAudio.lists.get("bgm_lacuna")  → Returns the AudioList instance matching "bgr
```

Typical usage

```
// Return the AudioList instance matching "bgm_lacuna" and do something with it
SimpleAudio.lists.get("bgm_lacuna").volume(1).loop(true).play();
```

`SimpleAudio.lists.has(listId)` → *boolean*

Returns whether a playlist with the given list ID exists.

History:

- `v2.28.0`: Introduced.

Parameters:

- `listId`: (*string*) The ID of the playlist.

Examples:

```
if (SimpleAudio.lists.has("bgm_lacuna")) {
        // Playlist "bgm_lacuna" exists.
}
```

AudioTrack API

Audio tracks encapsulate and provide a consistent interface to an audio resource.

SEE ALSO: `SimpleAudio` API, `AudioRunner` API, and `AudioList` API.

`<AudioTrack>.clone()` → *`AudioTrack` object*

Returns a new independent copy of the track.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
var trackCopy = aTrack.clone();
```

`<AudioTrack>.duration()` → *number*

Returns the track's total playtime in seconds, `Infinity` for a stream, or `NaN` if no metadata exists.

History:

- `v2.28.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters: *none*

Examples:

```
var trackLength = aTrack.duration();
```

`<AudioTrack>.fade(duration , toVol [, fromVol])` → *`Promise`* object

Starts playback of the track and fades it between the specified starting and destination volume levels over the specified number of seconds.

> **NOTE:** The `Config.audio.pauseOnFadeToZero` setting (default: `true`) determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.

Parameters:

- `duration`: (*number*) The number of seconds over which the track should be faded.
- `toVol`: (*number*) The destination volume level.
- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the track's current volume level.

Examples:

```
// Fade the track from volume 0 to 1 over 6 seconds.
aTrack.fade(6, 1, 0);
```

`<AudioTrack>.fadeIn(duration [, fromVol])` → *`Promise`* object

Starts playback of the track and fades it from the specified volume level to `1` (loudest) over the specified number of seconds.

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Updated to return a `Promise`.

Parameters:

- `duration`: (*number*) The number of seconds over which the track should be faded.
- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the track's current volume level.

Examples:

```
// Fade the track in from volume 0 over 5 seconds.
aTrack.fadeIn(5, 0);
```

`<AudioTrack>.fadeOut(duration [, fromVol])` → *`Promise`* object

Starts playback of the track and fades it from the specified volume level to `0` (silent) over the specified number of seconds.

> **NOTE:** The `Config.audio.pauseOnFadeToZero` setting (default: `true`) determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Updated to return a `Promise`.

Parameters:

- `duration`: (*number*) The number of seconds over which the track should be faded.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the track's current volume level.

Examples:

```
// Fade the track out from volume 1 over 8 seconds.
aTrack.fadeOut(8, 1);
```

`<AudioTrack>.fadeStop()`

Interrupts an in-progress fade of the track, or does nothing if no fade is progressing.

NOTE: This does not alter the volume level.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aTrack.fadeStop();
```

`<AudioTrack>.hasData()` → *boolean*

Returns whether enough data has been loaded to play the track through to the end without interruption.

NOTE: This is an estimate calculated by the browser based upon the currently downloaded data and the download rate.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.hasData()) {
	/* do something */
}
```

`<AudioTrack>.hasMetadata()` → *boolean*

Returns whether, at least, the track's metadata has been loaded.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.hasMetadata()) {
	/* do something */
}
```

`<AudioTrack>.hasNoData()` → *boolean*

Returns whether none of the track's data has been loaded.

History:

- `v2.28.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters: *none*

Examples:

```
if (aTrack.hasNoData()) {
	/* do something */
}
```

<AudioTrack>.hasSomeData() → *boolean*

Returns whether, at least, some of the track's data has been loaded.

> **TIP:** The <AudioTrack>.hasData() method is generally more useful.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.hasSomeData()) {
	/* do something */
}
```

<AudioTrack>.hasSource() → *boolean*

Returns whether any valid sources were registered.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.hasSource()) {
	/* do something */
}
```

<AudioTrack>.isEnded() → *boolean*

Returns whether playback of the track has ended.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isEnded()) {
	/* do something */
}
```

<AudioTrack>.isFading() → *boolean*

Returns whether a fade is in-progress on the track.

History:

- v2.28.0: Introduced.

Parameters: *none*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

Examples:

```
if (aTrack.isFading()) {
	/* do something */
}
```

<AudioTrack>.isFailed() → *boolean*

Returns whether an error has occurred.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isFailed()) {
	/* do something */
}
```

<AudioTrack>.isLoading() → *boolean*

Returns whether the track is loading data.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isLoading()) {
	/* do something */
}
```

<AudioTrack>.isPaused() → *boolean*

Returns whether playback of the track has been paused.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isPaused()) {
	/* do something */
}
```

<AudioTrack>.isPlaying() → *boolean*

Returns whether the track is playing.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isPlaying()) {
	/* do something */
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
}
```

`<AudioTrack>.isSeeking()` → *boolean*

Returns whether the track is seeking.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isSeeking()) {
	/* do something */
}
```

`<AudioTrack>.isStopped()` → *boolean*

Returns whether playback of the track has been stopped.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isStopped()) {
	/* do something */
}
```

`<AudioTrack>.isUnavailable()` → *boolean*

Returns whether the track is currently unavailable for playback. Possible reasons include: no valid sources are registered, no sources are currently loaded, an error has occurred.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isUnavailable()) {
	/* do something */
}
```

`<AudioTrack>.isUnloaded()` → *boolean*

Returns whether the track's sources are currently unloaded.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
if (aTrack.isUnloaded()) {
	/* do something */
}
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`<AudioTrack>.load()`

Pauses playback of the track and, if it's not already in the process of loading, forces it to drop any existing data and begin loading.

> **WARNING:** This *should not* be done lightly if your audio sources are on the network, as it forces players to begin downloading them.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aTrack.load();
```

`<AudioTrack>.loop([state])` → get: *boolean* | set: *`AudioTrack`* *object*

Gets or sets the track's repeating playback state (default: `false`). When used to set the loop state, returns a reference to the current `AudioTrack` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The loop state.

Examples:

```
// Get the track's current loop state.
var isLooped = aTrack.loop();

// Loop the track.
aTrack.loop(true);

// Unloop the track.
aTrack.loop(false);
```

`<AudioTrack>.mute([state])` → get: *boolean* | set: *`AudioTrack`* *object*

Gets or sets the track's volume mute state (default: `false`). When used to set the mute state, returns a reference to the current `AudioTrack` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The mute state.

Examples:

```
// Get the track's current volume mute state.
var isMuted = aTrack.mute();

// Mute the track's volume.
aTrack.mute(true);

// Unmute the track's volume.
aTrack.mute(false);
```

`<AudioTrack>.off(...args)` → *`AudioTrack`* *object*

Removes event handlers from the track. Returns a reference to the current `AudioTrack` instance for chaining.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: Shorthand for jQuery's `.off()` method applied to the audio element.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g., `.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.off()` in the jQuery API docs for more information.

Examples:

```
// Remove any handlers for the ended event.
aTrack.off('ended.myEvents');
```

`<AudioTrack>.on(...args)` → *`AudioTrack`* object

Attaches event handlers to the track. Returns a reference to the current `AudioTrack` instance for chaining.

NOTE: Shorthand for jQuery's `.on()` method applied to the audio element.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g., `.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.on()` in the jQuery API docs for more information.

Examples:

```
// Add a handler for the ended event.
aTrack.on('ended.myEvents', function () {
        /* do something */
});
```

`<AudioTrack>.one(...args)` → *`AudioTrack`* object

Attaches single-use event handlers to the track. Returns a reference to the current `AudioTrack` instance for chaining.

NOTE: Shorthand for jQuery's `.one()` method applied to the audio element.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g., `.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.one()` in the jQuery API docs for more information.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color − +

Introduction

Markup +

TwineScript +

Macros +

Functions +

Methods +

Special Names +

CSS +

HTML

Events +

Config API +

Dialog API +

Engine API +

Fullscreen API +

LoadScreen API +

Macro API +

▸ MacroContext API +

Passage API +

Save API +

Setting API +

SimpleAudio API +

▸ AudioTrack API +

▸ AudioRunner API +

▸ AudioList API +

State API +

Story API +

Template API +

UI API +

UIBar API +

Guide: State, Sessions, and Saving +

Guide: Tips +

Guide: Media Passages +

Guide: Harlowe to SugarCube +

Guide: Test Mode +

Guide: TypeScript

Examples:

```
// Add a single-use handler for the ended event.
aTrack.one('ended.myEvents', function () {
	/* do something */
});
```

<AudioTrack>.pause()

Pauses playback of the track.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
aTrack.pause();
```

<AudioTrack>.play() → *Promise* object

Begins playback of the track.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

Basic usage

```
aTrack.play();
```

Using the returned Promise

```
aTrack.play()
	.then(function () {
		console.log('The track is playing.');
	})
	.catch(function (problem) {
		console.warn('There was a problem with playback: ' + problem);
	});
```

<AudioTrack>.playWhenAllowed()

Begins playback of the track or, failing that, sets the track to begin playback as soon as the player has interacted with the document.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
aTrack.playWhenAllowed();
```

<AudioTrack>.remaining() → *number*

Returns how much remains of the track's total playtime in seconds, Infinity for a stream, or NaN if no metadata exists.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
var trackRemains = aTrack.remaining();
```

`<AudioTrack>.stop()`

Stops playback of the track.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTrack.stop();
```

`<AudioTrack>.time([seconds])` → get: *number* | set: *`AudioTrack`* *object*

Gets or sets the track's current time in seconds. When used to set a value, returns a reference to the current `AudioTrack` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `seconds`: (optional, *number*) The time to set. Valid values are floating-point numbers in the range `0` (start) to the maximum duration—e.g., `60` is `60` is sixty seconds in, `90.5` is ninety-point-five seconds in.

Examples:

```
// Get the track's current time.
var trackTime = aTrack.time();

// Set the track's current time to 30 seconds from its beginning.
aTrack.time(30);

// Set the track's current time to 30 seconds from its end.
aTrack.time(aTrack.duration() - 30);
```

`<AudioTrack>.unload()`

Stops playback of the track and forces it to drop any existing data.

> **NOTE:** Once unloaded, playback cannot occur until the track's data is loaded again.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aTrack.unload();
```

`<AudioTrack>.volume([level])` → get: *number* | set: *`AudioTrack`* *object*

Gets or sets the track's volume level (default: `1`). When used to set the volume, returns a reference to the current `AudioTrack` instance for chaining.

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

History:

- `v2.28.0`: Introduced.

Parameters:

- **`level`:** (optional, *number*) The volume level to set. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Examples:

```
// Get the track's current volume level.
var trackVolume = aTrack.volume();

// Set the track's volume level to 75%.
aTrack.volume(0.75);
```

AudioRunner API

Audio runners are useful for performing actions on multiple tracks at once.

SEE ALSO: `SimpleAudio` API, `AudioTrack` API, and `AudioList` API.

<AudioRunner>.fade(duration , toVol [, fromVol])

Starts playback of the selected tracks and fades them between the specified starting and destination volume levels over the specified number of seconds.

NOTE: The `Config.audio.pauseOnFadeToZero` setting (default: `true`) determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.

Parameters:

- **`duration`:** (*number*) The number of seconds over which the tracks should be faded.
- **`toVol`:** (*number*) The destination volume level.
- **`fromVol`:** (optional, *number*) The starting volume level. If omitted, defaults to the tracks' current volume level.

Examples:

```
// Fade the selected tracks from volume 0 to 1 over 6 seconds.
someTracks.fade(6, 1, 0);
```

<AudioRunner>.fadeIn(duration [, fromVol])

Starts playback of the selected tracks and fades them from the specified volume level to `1` (loudest) over the specified number of seconds.

History:

- `v2.28.0`: Introduced.

Parameters:

- **`duration`:** (*number*) The number of seconds over which the tracks should be faded.
- **`fromVol`:** (optional, *number*) The starting volume level. If omitted, defaults to the tracks' current volume level.

Examples:

```
// Fade the selected tracks in from volume 0 over 5 seconds.
someTracks.fadeIn(5, 0);
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
someTracks.fadeIn(3, 0);
```

<AudioRunner>.fadeOut(duration [, fromVol])

Starts playback of the selected tracks and fades them from the specified volume level to `0` (silent) over the specified number of seconds.

> **NOTE:** The `Config.audio.pauseOnFadeToZero` setting (default: `true`) determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.

Parameters:

- `duration`: (*number*) The number of seconds over which the tracks should be faded.
- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the tracks' current volume level.

Examples:

```
// Fade the selected tracks out from volume 1 over 8 seconds.
someTracks.fadeOut(8, 1);
```

<AudioRunner>.fadeStop()

Interrupts an in-progress fade of the selected tracks, or does nothing if no fade is progressing.

> **NOTE:** This does not alter the volume level.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTracks.fadeStop();
```

<AudioRunner>.load()

Pauses playback of the selected tracks and, if they're not already in the process of loading, forces them to drop any existing data and begin loading.

> **WARNING:** This *should not* be done lightly if your audio sources are on the network, as it forces players to begin downloading them.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTracks.load();
```

<AudioRunner>.loop(state) → *AudioRunner object*

Sets the selected tracks' repeating playback state (default: `false`). Returns a reference to the current `AudioRunner` instance for chaining.

History:

- `v2.28.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters:

- `state`: (*boolean*) The loop state.

Examples:

```
// Loop the selected tracks.
someTracks.loop(true);

// Unloop the selected tracks.
someTracks.loop(false);
```

`<AudioRunner>.mute(state)` → *`AudioRunner`* object

Sets the selected tracks' volume mute state (default: `false`). Returns a reference to the current `AudioRunner` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (*boolean*) The mute state.

Examples:

```
// Mute the selected tracks' volume.
someTracks.mute(true);

// Unmute the selected tracks' volume.
someTracks.mute(false);
```

`<AudioRunner>.off(...args)` → *`AudioRunner`* object

Removes event handlers from the selected tracks. Returns a reference to the current `AudioRunner` instance for chaining.

NOTE: Shorthand for jQuery's `.off()` method applied to each of the audio elements.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g., `.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.off()` in the jQuery API docs for more information.

Examples:

```
// Remove any handlers for the ended event.
someTracks.off('ended.myEvents');
```

`<AudioRunner>.on(...args)` → *`AudioRunner`* object

Attaches event handlers to the selected tracks. Returns a reference to the current `AudioRunner` instance for chaining.

NOTE: Shorthand for jQuery's `.on()` method applied to each of the audio elements.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g.,

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.on()` in the jQuery API docs for more information.

Examples:

```
// Add a handler for the ended event.
someTracks.on('ended.myEvents', function () {
	/* do something */
});
```

`<AudioRunner>.one(...args)` → *`AudioRunner`* object

Attaches single-use event handlers to the selected tracks. Returns a reference to the current `AudioRunner` instance for chaining.

NOTE: Shorthand for jQuery's `.one()` method applied to each of the audio elements.

WARNING: The `SimpleAudio` APIs use events internally for various pieces of functionality. To prevent conflicts, it is ***strongly*** suggested that you specify a custom user namespace—e.g., `.myEvents`—when attaching your own handlers. It is further ***strongly*** suggested that you provide that same custom user namespace when removing them.

History:

- `v2.28.0`: Introduced.

Parameters:

SEE: `<jQuery>.one()` in the jQuery API docs for more information.

Examples:

```
// Add a single-use handler for the ended event.
someTracks.one('ended.myEvents', function () {
	/* do something */
});
```

`<AudioRunner>.pause()`

Pauses playback of the selected tracks.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTracks.pause();
```

`<AudioRunner>.play()`

Begins playback of the selected tracks.

History:

- `v2.28.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters: *none*

Examples:

```
someTracks.play();
```

<AudioRunner>.playWhenAllowed()

Begins playback of the selected tracks or, failing that, sets the tracks to begin playback as soon as the player has interacted with the document.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
someTracks.playWhenAllowed();
```

<AudioRunner>.stop()

Stops playback of the selected tracks.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
someTracks.stop();
```

<AudioRunner>.time(seconds) → *AudioRunner* object

Sets the selected tracks' current time in seconds. Returns a reference to the current AudioRunner instance for chaining.

History:

- v2.28.0: Introduced.

Parameters:

- seconds: (*number*) The time to set. Valid values are floating-point numbers in the range 0 (start) to the maximum duration—e.g., 60 is 60 is sixty seconds in, 90.5 is ninety-point-five seconds in.

Examples:

```
// Set the selected tracks' current time to 30 seconds from their beginning.
someTracks.time(30);
```

<AudioRunner>.unload()

Stops playback of the selected tracks and forces them to drop any existing data.

NOTE: Once unloaded, playback cannot occur until the selected tracks' data is loaded again.

History:

- v2.28.0: Introduced.

Parameters: *none*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
someTracks.unload();
```

<AudioRunner>.volume(level) → *AudioRunner object*

Sets the selected tracks' volume level (default: 1). Returns a reference to the current AudioRunner instance for chaining.

History:

- v2.28.0: Introduced.

Parameters:

- **level:** (*number*) The volume level to set. Valid values are floating-point numbers in the range 0 (silent) to 1 (loudest)—e.g., 0 is 0%, 0.5 is 50%, 1 is 100%.

Examples:

```
// Set the selected tracks' volume level to 75%.
someTracks.volume(0.75);
```

AudioList API

Audio lists (playlists) are useful for playing tracks in a sequence—i.e., one after another.

SEE ALSO: SimpleAudio API, AudioTrack API, and AudioRunner API.

<AudioList>.duration() → *number*

Returns the playlist's total playtime in seconds, Infinity if it contains any streams, or NaN if no metadata exists.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
var listLength = aList.duration();
```

<AudioList>.fade(duration , toVol [, fromVol]) → *Promise object*

Starts playback of the playlist and fades the currently playing track between the specified starting and destination volume levels over the specified number of seconds.

NOTE: The Config.audio.pauseOnFadeToZero setting (default: true) determines whether the audio subsystem automatically pauses tracks that have been faded to 0 volume (silent).

History:

- v2.28.0: Introduced.
- v2.29.0: Updated to return a Promise.

Parameters:

- **duration:** (*number*) The number of seconds over which the currently playing track should be faded.
- **toVol:** (*number*) The destination volume level.
- **fromVol:** (optional, *number*) The starting volume level. If omitted, defaults to the currently playing track's current volume level.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
// Fade the playlist from volume 0 to 1 over 6 seconds.
aList.fade(6, 1, 0);
```

<AudioList>.fadeIn(duration [, fromVol]) → *Promise* object

Starts playback of the playlist and fades the currently playing track from the specified volume level to `1` (loudest) over the specified number of seconds.

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Updated to return a `Promise`.

Parameters:

- `duration`: (*number*) The number of seconds over which the currently playing track should be faded.
- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the currently playing track's current volume level.

Examples:

```
// Fade the playlist in from volume 0 over 5 seconds.
aList.fadeIn(5, 0);
```

<AudioList>.fadeOut(duration [, fromVol]) → *Promise* object

Starts playback of the playlist and fades the currently playing track from the specified volume level to `0` (silent) over the specified number of seconds.

NOTE: The `Config.audio.pauseOnFadeToZero` setting (default: `true`) determines whether the audio subsystem automatically pauses tracks that have been faded to `0` volume (silent).

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Updated to return a `Promise`.

Parameters:

- `duration`: (*number*) The number of seconds over which the currently playing track should be faded.
- `fromVol`: (optional, *number*) The starting volume level. If omitted, defaults to the currently playing track's current volume level.

Examples:

```
// Fade the playlist out from volume 1 over 8 seconds.
aList.fadeOut(8, 1);
```

<AudioList>.fadeStop()

Interrupts an in-progress fade of the currently playing track, or does nothing if no fade is progressing.

NOTE: This does not alter the volume level.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
aList.fadeStop();
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

<AudioList>.isEnded() → *boolean*

Returns whether playback of the playlist has ended.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aList.isEnded()) {
        /* do something */
}
```

<AudioList>.isFading() → *boolean*

Returns whether a fade is in-progress on the currently playing track.

History:

- v2.29.0: Introduced.

Parameters: *none*

Examples:

```
if (aList.isFading()) {
        /* do something */
}
```

<AudioList>.isPaused() → *boolean*

Returns whether playback of the playlist has been paused.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aList.isPaused()) {
        /* do something */
}
```

<AudioList>.isPlaying() → *boolean*

Returns whether the playlist is playing.

History:

- v2.28.0: Introduced.

Parameters: *none*

Examples:

```
if (aList.isPlaying()) {
        /* do something */
}
```

<AudioList>.isStopped() → *boolean*

Returns whether playback of the playlist has been stopped.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
if (aList.isStopped()) {
        /* do something */
}
```

`<AudioList>.load()`

Pauses playback of the playlist and, if they're not already in the process of loading, forces its tracks to drop any existing data and begin loading.

> **WARNING:** This *should not* be done lightly if your audio sources are on the network, as it forces players to begin downloading them.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aList.load();
```

`<AudioList>.loop([state])` → get: *boolean* | set: *`AudioList`* *object*

Gets or sets the playlist's repeating playback state (default: `false`). When used to set the loop state, returns a reference to the current `AudioList` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The loop state.

Examples:

```
// Get the playlist's current loop state.
var isLooped = aList.loop();

// Loop the playlist.
aList.loop(true);

// Unloop the playlist.
aList.loop(false);
```

`<AudioList>.mute([state])` → get: *boolean* | set: *`AudioList`* *object*

Gets or sets the playlist's volume mute state (default: `false`). When used to set the mute state, returns a reference to the current `AudioList` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- `state`: (optional, *boolean*) The mute state.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
// Get the playlist's current volume mute state.
var isMuted = aList.mute();

// Mute the playlist's volume.
aList.mute(true);

// Unmute the playlist's volume.
aList.mute(false);
```

`<AudioList>.pause()`

Pauses playback of the playlist.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aList.pause();
```

`<AudioList>.play()` → *`Promise` object*

Begins playback of the playlist.

History:

- `v2.28.0`: Introduced.
- `v2.29.0`: Updated to return a `Promise`.

Parameters: *none*

Examples:

Basic usage

```
aList.play();
```

Using the returned `Promise`

```
aList.play()
	.then(function () {
		console.log('The playlist is playing.');
	})
	.catch(function (problem) {
		console.warn('There was a problem with playback: ' + problem);
	});
```

`<AudioList>.playWhenAllowed()`

Begins playback of the playlist or, failing that, sets the playlist to begin playback as soon as the player has interacted with the document.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
aList.playWhenAllowed();
```

`<AudioList>.remaining()` → *number*

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Returns how much remains of the playlist's total playtime in seconds, `Infinity` if it contains any streams, or `NaN` if no metadata exists.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
var listRemains = aList.remaining();
```

`<AudioList>.shuffle([state])` → get: *boolean* | set: *`AudioList` object*

Gets or sets the playlist's randomly shuffled playback state (default: `false`). When used to set the shuffle state, returns a reference to the current `AudioList` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- **`state`:** (optional, *boolean*) The shuffle state.

Examples:

```
// Get the playlist's current shuffle state.
var isShuffled = aList.shuffle();

// Enable shuffling of the playlist.
aList.shuffle(true);

// Disable shuffling of the playlist.
aList.shuffle(false);
```

`<AudioList>.skip()`

Skips ahead to the next track in the playlist, if any.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTrack.skip();
```

`<AudioList>.stop()`

Stops playback of the playlist.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
someTrack.stop();
```

`<AudioList>.time()` → *number*

Returns the playlist's current time in seconds, or `NaN` if no metadata exists.

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
var listTime = aList.time();
```

`<AudioList>.unload()`

Stops playback of the playlist and forces its tracks to drop any existing data.

> **NOTE:** Once unloaded, playback cannot occur until the track's data is loaded again.

History:

- `v2.28.0`: Introduced.

Parameters: *none*

Examples:

```
aList.unload();
```

`<AudioList>.volume([level])` → get: *number* | set: *`AudioList`* *object*

Gets or sets the playlist's volume level (default: `1`). When used to set the volume, returns a reference to the current `AudioList` instance for chaining.

History:

- `v2.28.0`: Introduced.

Parameters:

- **`level`:** (optional, *number*) The volume level to set. Valid values are floating-point numbers in the range `0` (silent) to `1` (loudest)—e.g., `0` is 0%, `0.5` is 50%, `1` is 100%.

Examples:

```
// Get the playlist's current volume level.
var trackVolume = aList.volume();

// Set the playlist's volume level to 75%.
aList.volume(0.75);
```

State API

The story history contains moments (states) created during play. Since it is possible to navigate the history—i.e., move backward and forward though the moments within the history—it may contain both past moments—i.e., moments that have been played—and future moments—i.e., moments that had been played, but have been rewound/undone, yet are still available to be restored.

In addition to the history, there is also the active moment—i.e., present—and expired moments—i.e., moments that had been played, but have expired from the history, thus cannot be navigated to.

API members dealing with the history work upon either the active moment—i.e., present—or one of the history subsets: the full in-play history—i.e., past + future—the past in-play subset—i.e., past only—or the extended past subset—i.e., expired + past. These instances will be noted.

`State.active` → *object*

Returns the active (present) moment.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

> **NOTE:** Using `State.active` directly is generally unnecessary as there exist a number of shortcut properties, `State.passage` and `State.variables`, and story functions, `passage()` and `variables()`, which grant access to its normal properties.

History:

- `v2.0.0`: Introduced.

Examples:

```
State.active.title      → The title of the present moment
State.active.variables  → The variables of the present moment
```

`State.bottom` → *object*

Returns the bottommost (least recent) moment from the full in-play history (past + future).

History:

- `v2.0.0`: Introduced.

Examples:

```
State.bottom.title      → The title of the least recent moment within the full in-
State.bottom.variables  → The variables of the least recent moment within the full
```

`State.current` → *object*

Returns the current moment from the full in-play history (past + future), which is the pre-play version of the active moment.

> **WARNING:** `State.current` *is not* a synonym for `State.active`. You will, very likely, never need to use `State.current` directly within your code.

History:

- `v2.8.0`: Introduced.

Examples:

```
State.current.title      → The title of the current moment within the full in-play
State.current.variables  → The variables of the current moment within the full in-
```

`State.length` → *integer*

Returns the number of moments within the past in-play history (past only).

History:

- `v2.0.0`: Introduced.

Examples:

```
if (State.length === 0) {
        /* No moments within the past in-play history. Egad! */
}
```

`State.passage` → *string*

Returns the title of the passage associated with the active (present) moment.

History:

- `v2.0.0`: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
State.passage  → The passage title of the present moment
```

State.size → *integer*

Returns the number of moments within the full in-play history (past + future).

History:

- v2.0.0: Introduced.

Parameters: *none*

Examples:

```
if (State.size === 0) {
        /* No moments within the full in-play history. Egad! */
}
```

State.temporary → *object*

Returns the current temporary variables.

History:

- v2.13.0: Introduced.

Examples:

```
State.temporary  → The current temporary variables
```

State.top → *object*

Returns the topmost (most recent) moment from the full in-play history (past + future).

WARNING: State.top *is not* a synonym for State.active. You will, very likely, never need to use State.top directly within your code.

History:

- v2.0.0: Introduced.

Examples:

```
State.top.title      → The title of the most recent moment within the full in-play
State.top.variables  → The variables of the most recent moment within the full in-
```

State.turns → *integer*

Returns the total number (count) of played moments within the extended past history (expired + past).

History:

- v2.0.0: Introduced.

Examples:

```
if (State.turns === 1) {
        /* Initial turn.  The starting passage is displayed. */
}
```

State.variables → *object*

Returns the variables from the active (present) moment.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Examples:

```
State.variables  → The variables of the present moment
```

`State.getVar(varName)` → *any*

Returns the value of the story or temporary variable by the given name.

History:

- `v2.22.0`: Introduced.

Parameters:

- `varName`: (*string*) The name of the story or temporary variable, including its sigil—e.g., `$charName`.

Examples:

```
State.getVar("$charName")  → Returns the value of $charName
```

`State.has(passageTitle)` → *boolean*

Returns whether any moments with the given title exist within the past in-play history (past only).

> **NOTE:** `State.has()` *does not* check expired moments. If you need to know if the player has ever been to a particular passage, then you *must* use the `State.hasPlayed()` method or the `hasVisited()` story function.

History:

- `v2.0.0`: Introduced.

Parameters:

- `passageTitle`: (*string*) The title of the moment whose existence will be verified.

Examples:

```
State.has("The Ducky")  → Returns whether a moment matching "The Ducky" exists
```

`State.hasPlayed(passageTitle)` → *boolean*

Returns whether any moments with the given title exist within the extended past history (expired + past).

> **NOTE:** If you need to check for multiple passages, the `hasVisited()` story function will likely be more convenient to use.

History:

- `v2.0.0`: Introduced.

Parameters:

- `passageTitle`: (*string*) The title of the moment whose existence will be verified.

Examples:

```
State.hasPlayed("The Ducky")  → Returns whether a moment matching "The Ducky" ever
```

`State.index(index)` → *object*

Returns the moment, relative to the bottom of the past in-play history (past only), at the given index.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

History:

- `v2.0.0`: Introduced.

Parameters:

- **`index`:** (*integer*) The index of the moment to return.

Examples:

```
State.index(0)                → Returns the least recent moment within the past i
State.index(1)                → Returns the second to least recent moment within
State.index(State.length - 1)  → Returns the most recent moment within the past in
```

`State.isEmpty()` → *boolean*

Returns whether the full in-play history (past + future) is empty.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
if (State.isEmpty()) {
	/* No moments within the full in-play history. Egad! */
}
```

`State.peek([offset])` → *object*

Returns the moment, relative to the top of the past in-play history (past only), at the, optional, offset.

History:

- `v2.0.0`: Introduced.

Parameters:

- **`offset`:** (optional, *integer*) The offset, from the top of the past in-play history, of the moment to return. If not given, an offset of `0` is used.

Examples:

```
State.peek()                → Returns the most recent moment within the past in-
State.peek(0)               → Returns the most recent moment within the past in-
State.peek(1)               → Returns the second most recent moment within the pa
State.peek(State.length - 1)  → Returns the least recent moment within the past in
```

`State.metadata.size` → *integer*

Returns the size of the story metadata store—i.e., the number of stored pairs.

History:

- `v2.30.0`: Introduced.

Examples:

```
// Determines whether the metadata store has any members.
if (State.metadata.size > 0) {
	/* store is not empty */
}
```

`State.metadata.clear()`

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Empties the story metadata store.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
// Removes all values from the metadata store.
State.metadata.clear();
```

`State.metadata.delete(key)`

Removes the specified key, and its associated value, from the story metadata store.

History:

- `v2.29.0`: Introduced.

Parameters:

- `key`: (*string*) The key to delete.

Examples:

```
// Removes 'achievements' from the metadata store.
State.metadata.delete('achievements');
```

`State.metadata.entries()` → *Array<Array<string, any>>*

Returns an array of the story metadata store's key/value pairs as `[key, value]` arrays.

History:

- `v2.36.0`: Introduced.

Parameters: *none*

Examples:

```
// Iterate over the pairs with a `for` loop.
var metadata = State.metadata.entries();
for (var i = 0; i < metadata.length; ++i) {
        var key   = metadata[i][0];
        var value = metadata[i][1];

        /* do something */
}
```

```
// Iterate over the pairs with `<Array>.forEach()`.
State.metadata.entries().forEach(function (pair) {
        var key   = pair[0];
        var value = pair[1];

        /* do something */
});
```

`State.metadata.get(key)` → *any*

Returns the value associated with the specified key from the story metadata store.

History:

- `v2.29.0`: Introduced.

Parameters:

- `key`: (*string*) The key whose value should be returned.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
// Returns the value of 'achievements' from the metadata store.
var playerAchievements = State.metadata.get('achievements');
```

State.metadata.has(key) → *boolean*

Returns whether the specified key exists within the story metadata store.

History:

- v2.29.0: Introduced.

Parameters:

- key: (*string*) The key whose existence should be tested.

Examples:

```
// Returns whether 'achievements' exists within the metadata store.
if (State.metadata.has('achievements')) {
	/* do something */
}
```

State.metadata.keys() → *Array<string>*

Returns an array of the story metadata store's keys.

History:

- v2.36.0: Introduced.

Parameters: *none*

Examples:

```
// Iterate over the keys with a `for` loop.
var metadataKeys = State.metadata.keys();
for (var i = 0; i < metadataKeys.length; ++i) {
	var key = metadataKeys[i];

	/* do something */
}
```

```
// Iterate over the keys with `<Array>.forEach()`.
State.metadata.keys().forEach(function (key) {
	/* do something */
});
```

State.metadata.set(key, value)

Sets the specified key and value within the story metadata store, which causes them to persist over story and browser restarts—n.b. private browsing modes do interfere with this. To update the value associated with a key, simply set it again.

NOTE: The story metadata, like saves, is tied to the specific story it was generated with. It is not a mechanism for moving data between stories.

WARNING: The story metadata store ***is not***, and should not be used as, a replacement for saves. Examples of good uses: achievement tracking, new game+ data, playthrough statistics, etc.

WARNING: This feature is largely incompatible with private browsing modes, which cause all in-browser storage mechanisms to either persist only for the lifetime of the browsing session or fail outright.

History:

- v2.29.0: Introduced.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters:

- `key`: (*string*) The key that should be set.
- `value`: (*any*) The value to set.

Examples:

```
// Sets 'achievements', with the given value, in the metadata store.
State.metadata.set('achievements', { ateYellowSnow : true });

// Sets 'ngplus', with the given value, in the metadata store.
State.metadata.set('ngplus', true);
```

`State.prng.init([seed [, useEntropy]])`

Initializes the seedable pseudo-random number generator (PRNG) and integrates it into the story state and saves. Once initialized, the `State.random()` method and story functions, `random()` and `randomFloat()`, return deterministic results from the seeded PRNG—by default, they return non-deterministic results from `Math.random()`.

> **NOTE:** `State.prng.init()` *must be* called during story initialization, within either your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage) or the `StoryInit` special passage. Additionally, it is ***strongly*** recommended that you do not specify any arguments to `State.prng.init()` and allow it to automatically seed itself. If you should chose to use an explicit seed, however, it is ***strongly*** recommended that you also enable additional entropy, otherwise all playthroughs for all players will be exactly the same.

History:

- `v2.29.0`: Introduced.

Parameters:

- `seed`: (optional, *string*) The explicit seed used to initialize the pseudo-random number generator.
- `useEntropy`: (optional, *boolean*) Enables the use of additional entropy to pad the specified explicit seed.

Examples:

```
State.prng.init()                       → Automatically seed the PRNG (recommended
State.prng.init("aVeryLongSeed")        → Seed the PRNG with "aVeryLongSeed" (not r
State.prng.init("aVeryLongSeed", true)  → Seed the PRNG with "aVeryLongSeed" and pa
```

`State.prng.isEnabled()` → *boolean*

Returns whether the seedable PRNG has been enabled.

History:

- `v2.29.0`: Introduced.

Examples:

```
State.prng.isEnabled()  → Returns whether the seedable PRNG is enabled
```

`State.prng.pull` → *integer* | *NaN*

Returns the current pull count—i.e., how many requests have been made—from the seedable PRNG or, if the PRNG is not enabled, `NaN`.

> **NOTE:** The pull count is automatically included within saves and sessions, so this is not especially useful outside of debugging purposes.

History:

- `v2.29.0`: Introduced.

Examples:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
State.prng.pull  → Returns the current PRNG pull count
```

`State.prng.seed` → *string* | *null*

Returns the seed from the seedable PRNG or, if the PRNG is not enabled, `null`.

> **NOTE:** The seed is automatically included within saves and sessions, so this is not especially useful outside of debugging purposes.

History:

- `v2.29.0`: Introduced.

Examples:

```
State.prng.seed  → Returns the PRNG seed
```

`State.random()` → *number*

Returns a pseudo-random decimal number (floating-point) in the range `0` (inclusive) up to, but not including, `1` (exclusive).

> **NOTE:** By default, it simply returns non-deterministic results from `Math.random()`, however, when the seedable PRNG has been enabled, via `State.prng.init()`, it returns deterministic results from the seeded PRNG instead.

History:

- `v2.0.0`: Introduced.

Parameters: *none*

Examples:

```
State.random()  → Returns a pseudo-random floating-point number in the range [0, 1)
```

`State.setVar(varName, value)` → *boolean*

Sets the value of the story or temporary variable by the given name. Returns whether the operation was successful.

History:

- `v2.22.0`: Introduced.

Parameters:

- `varName`: (*string*) The name of the story or temporary variable, including its sigil—e.g., `$charName`.
- `value`: (*any*) The value to assign.

Examples:

```
State.setVar("$charName", "Jane Doe")  → Assigns the string "Jane Doe" to $charName
```

~~`State.initPRNG([seed [, useEntropy]])`~~

> **DEPRECATED:** This method has been deprecated and should no longer be used. See the `State.prng.init()` method for its replacement.

History:

- `v2.0.0`: Introduced.
- `v2.29.0`: Deprecated in favor of `State.prng.init()`.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Story API

`Story.domId` → *string*

The DOM ID of the story, created from the slugified story title.

History:

- `v2.0.0`: Introduced.

`Story.ifId` → *string*

The IFID (Interactive Fiction IDentifier) of the story, if any.

History:

- `v2.5.0`: Introduced.

`Story.title` → *string*

The title of the story.

History:

- `v2.0.0`: Introduced.

`Story.get(title)` → *`Passage`* *object*

Returns the `Passage` object referenced by the given title, or an empty `Passage` object on failure.

NOTE: This method will not return "code" passages—i.e., script, stylesheet, and widget passages.

History:

- `v2.0.0`: Introduced.

Parameters:

- `title`: (*string*) The title of the `Passage` object to return.

Examples:

```
Story.get("The Ducky")  → Returns the Passage object matching "The Ducky"
```

`Story.has(title)` → *boolean*

Returns whether a `Passage` object referenced by the given title exists.

NOTE: This method will not detect "code" passages—i.e., script, stylesheet, and widget passages.

History:

- `v2.0.0`: Introduced.

Parameters:

- `title`: (*string*) The title of the `Passage` object whose existence will be verified.

Examples:

```
Story.has("The Ducky")  → Returns whether a Passage object matching "The Ducky" ex
```

`Story.lookup(propertyName , searchValue [, sortProperty])` →

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`Passage` *object array*

Returns a new array filled with all `Passage` objects that contain the given property, whose value matches the given search value, or an empty array, if no matches are made.

> **NOTE:** This method will not return "code" passages—i.e., script, stylesheet, and widget passages.

History:

- `v2.0.0`: Introduced.

Parameters:

- `propertyName`: (*string*) The name of property whose value will be compared to the search value.
- `searchValue`: (*string* | *number*) The value to search for within the matched property. The type of the property determines how the search occurs—non-arrays are directly compared, while arrays are searched. If the property's value, for non-arrays, or any of its members, for arrays, match, then the `Passage` object is added to the results.
- `sortProperty`: (optional, *string*) The property whose value will be used to lexicographically sort the returned array. If not given, the `Passage` object's `title` property is used.

Examples:

```
→ Returns all 'forest'-tagged Passage objects, sorted by their titles
Story.lookup("tags", "forest");
```

`Story.lookupWith(predicate [, sortProperty])` → `Passage` *object array*

Returns a new array filled with all `Passage` objects that pass the test implemented by the given predicate function or an empty array, if no objects pass.

> **NOTE:** This method will not return "code" passages—i.e., script, stylesheet, and widget passages.

History:

- `v2.11.0`: Introduced.

Parameters:

- `predicate`: (*function*) The function used to test each `Passage` object, which is passed into the function as its sole parameter. If the function returns `true`, then the `Passage` object is added to the results.
- `sortProperty`: (optional, *string*) The property whose value will be used to lexicographically sort the returned array. If not given, the `Passage` object's `title` property is used.

Examples:

```
→ Returns all 'forest'-tagged Passage objects, sorted by their titles
Story.lookupWith(function (p) {
        return p.tags.includes("forest");
});

→ Returns all Passage objects whose titles contain whitespace, sorted by their titl
var hasWhitespaceRegExp = /\s/;
Story.lookupWith(function (p) {
        return hasWhitespaceRegExp.test(p.title);
});
```

`Template` API

`Template.size` → *number*

Returns the number of existing templates.

History:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

- `v2.29.0`: Introduced.

Examples:

```
if (Template.size === 0) {
	/* No templates exist. */
}
```

`Template.add(name , definition)`

Add new template(s).

History:

- `v2.29.0`: Introduced.

Parameters:

- `name`: (*string* | *Array<string>*) Name, or array of names, of the template(s) to add. **NOTE:** Names must consist of characters from the basic Latin alphabet and start with a letter, which may be optionally followed by any number of letters, numbers, the underscore, or the hyphen.
- `definition`: (*function* | *string* | *Array<function | string>*) Definition of the template(s), which may be a: function, string, or an array of either. **NOTE:** Each time array definitions are referenced, one of their member templates is randomly selected to be the output source.

Function templates:

Function templates should return a string, which may itself contain markup. They are called with no arguments, but with their `this` set to a template (execution) context object that contains the following data properties:

- `name`: (*string*) The template's name.

String templates:

String templates consist solely of a string, which may itself contain markup.

Examples:

Basic usage

```
/* Define a function template named ?yolo. */
Template.add('yolo', function () {
	return either('YOLO', 'You Only Live Once');
});

/* Define a string template named ?nolf. */
Template.add('nolf', 'No One Lives Forever');

/* Define an array of string templates named ?alsoYolo. */
Template.add('alsoYolo', ['YOLO', 'You Only Live Once']);

/* Define an array of mixed string and function templates named ?cmyk. */
Template.add('cmyk', [
	'Cyan',
	function () {
		return either('Magenta', 'Yellow');
	},
	'Black'
]);
```

Using the context object (`this`)

```
/* Define a function template with two names, ?color and ?Color, whose output chang
Template.add(['color', 'Color'], function () {
	var color = either('red', 'green', 'blue');
	return this.name === 'Color' ? color.toUpperFirst() : color;
});
```

`Template.delete(name)`

Remove existing template(s).

SugarCube v2 Documentation
2. 36. 1 (2021-12-22)
Find in page: CTRL + F or F3
Code Color
Introduction
Markup
TwineScript
Macros
Functions
Methods
Special Names
CSS
HTML
Events
Config API
Dialog API
Engine API
Fullscreen API
LoadScreen API
Macro API
MacroContext API
Passage API
Save API
Setting API
SimpleAudio API
AudioTrack API
AudioRunner API
AudioList API
State API
Story API
Template API
UI API
UIBar API
Guide: State, Sessions, and Saving
Guide: Tips
Guide: Media Passages
Guide: Harlowe to SugarCube
Guide: Test Mode
Guide: TypeScript

History:

- `v2.29.0`: Introduced.

Parameters:

- `name`: (*string* | *Array<string>*) Name, or array of names, of the template(s) to remove.

Examples:

```
/* Deletes the template ?yolo. */
Template.delete('yolo');

/* Deletes the templates ?yolo and ?nolf. */
Template.delete(['yolo', 'nolf']);
```

`Template.get(name)` → *function | string | Array<function | string>*

Return the named template definition, or `null` on failure.

History:

- `v2.29.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the template whose definition should be returned.

Examples:

```
/* Returns the template ?yolo, or null if it doesn't exist. */
var yolo = Template.get('yolo');
```

`Template.has(name)` → *boolean*

Returns whether the named template exists.

History:

- `v2.29.0`: Introduced.

Parameters:

- `name`: (*string*) Name of the template to search for.

Examples:

```
if (Template.has('yolo')) {
	/* A ?yolo template exists. */
}
```

UI API

`UI.alert(message [, options [, closeFn]])`

Opens the built-in alert dialog, displaying the given message to the player.

History:

- `v2.0.0`: Introduced.

Parameters:

- `message`: (*string*) The message to display to the player.
- `options`: (optional, *null* | *object*) The options object. See `Dialog.open()` for more information.
- `closeFn`: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
UI.alert("You smell of elderberries!");
```

UI.jumpto([options [, closeFn]])

Opens the built-in jump to dialog, which is populated via the `bookmark` tag.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *null* | *object*) The options object. See `Dialog.open()` for more information.
- `closeFn`: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

Examples:

```
UI.jumpto();
```

UI.restart([options])

Opens the built-in restart dialog, prompting the player to restart the story.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *null* | *object*) The options object. See `Dialog.open()` for more information.

Examples:

```
UI.restart();
```

UI.saves([options [, closeFn]])

Opens the built-in saves dialog.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *null* | *object*) The options object. See `Dialog.open()` for more information.
- `closeFn`: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

Examples:

```
UI.saves();
```

UI.settings([options [, closeFn]])

Opens the built-in settings dialog, which is populated from the `Setting` API.

History:

- `v2.0.0`: Introduced.

Parameters:

- `options`: (optional, *null* | *object*) The options object. See `Dialog.open()` for more information.
- `closeFn`: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Examples:

```
UI.settings();
```

UI.share([options [, closeFn]])

Opens the built-in share dialog, which is populated from the StoryShare passage.

History:

- v2.0.0: Introduced.

Parameters:

- options: (optional, *null* | *object*) The options object. See Dialog.open() for more information.
- closeFn: (optional, *null* | *function*) The function to execute whenever the dialog is closed.

Examples:

```
UI.share();
```

UIBar API

UIBar.destroy()

Completely removes the UI bar and all of its associated styles and event handlers.

History:

- v2.17.0: Introduced.

Parameters: *none*

Examples:

```
UIBar.destroy();
```

UIBar.hide() → *UIBar object*

Hides the UI bar. Returns a reference to the UIBar object for chaining.

> **NOTE:** This does not reclaim the space reserved for the UI bar. Thus, a call to UIBar.stow() may also be necessary. Alternatively, if you simply want the UI bar gone completely and permanently, either using UIBar.destroy() or the StoryInterface special passage may be a better choice.

History:

- v2.29.0: Introduced.

Parameters: *none*

Examples:

Basic usage

```
UIBar.hide();
```

With stow

```
UIBar.hide().stow();
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

`UIBar.isHidden()` → *boolean*

Returns whether the UI bar is currently hidden.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
if (UIBar.isHidden()) {
	/* code to execute if the UI bar is hidden… */
}

if (!UIBar.isHidden()) {
	/* code to execute if the UI bar is not hidden… */
}
```

`UIBar.isStowed()` → *boolean*

Returns whether the UI bar is currently stowed.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
if (UIBar.isStowed()) {
	/* code to execute if the UI bar is stowed… */
}

if (!UIBar.isStowed()) {
	/* code to execute if the UI bar is not stowed… */
}
```

`UIBar.show()` → *`UIBar` object*

Shows the UI bar. Returns a reference to the `UIBar` object for chaining.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

Basic usage

```
UIBar.show();
```

With unstow

```
UIBar.unstow().show();
```

`UIBar.stow([noAnimation])` → *`UIBar` object*

Stows the UI bar, so that it takes up less space. Returns a reference to the `UIBar` object for chaining.

History:

- `v2.17.0`: Introduced.
- `v2.29.0`: Added returned `UIBar` chaining reference.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Parameters:

- `noAnimation`: (optional, *boolean*) Whether to skip the default animation.

Examples:

Basic usage

```
UIBar.stow();
```

With no animation

```
UIBar.stow(true);
```

`UIBar.unstow([noAnimation])` → *`UIBar` object*

Unstows the UI bar, so that it is fully accessible again. Returns a reference to the `UIBar` object for chaining.

History:

- `v2.17.0`: Introduced.
- `v2.29.0`: Added returned `UIBar` chaining reference.

Parameters:

- `noAnimation`: (optional, *boolean*) Whether to skip the default animation.

Examples:

Basic usage

```
UIBar.unstow();
```

With no animation

```
UIBar.unstow(true);
```

`UIBar.update()`

Updates all sections of the UI bar that are populated by special passages—e.g., `StoryBanner`, `StoryCaption`, `StoryMenu`, etc.

> **WARNING:** As *all* special passage populated sections are updated it is recommended that `UIBar.update()` be used sparingly. Ideally, if you need to update UI bar content outside of the normal passage navigation update, then you should update only the specific areas you need to rather than the entire UI bar.

History:

- `v2.29.0`: Introduced.

Parameters: *none*

Examples:

```
UIBar.update();
```

Guide: State, Sessions, and Saving

SugarCube preserves the state of the story as it's being played in a number of ways to both prevent the loss of progress and allow players to save stories. This guide will detail how these features work.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Story History

The story history is a collection of **moments**. A new moment is created whenever passage navigation occurs, and **only when passage navigation occurs**. Each moment contains data regarding the active passage and the state of all story variables—that is, the ones you use the `$` sigil to interact with—as they exist when the moment is created. The history allows players to navigate through these moments.

The number of moments contained within the story history is, generally, limited, via the `Config.history.maxStates` setting. As new moments are added, older moments that exceed the maximum number are expired in order of age, oldest first. Expired moments are recorded in a separate expired collection and can no longer be navigated to. If you limit the moments within the history to `1`, via setting `Config.history.maxStates` to `1`, then there will only ever be one moment in the history, but passage navigation is still required for new moments to be created.

> **NOTE:** All user functions and macros that check for the existence of moments within the history check both the story history and expired moments, so will work as expected even if the history is limited to a single moment as described above.

Saving the story records the story's state up until the last moment that was created. This is not necessarily the same as the current state of the story: because moment creation is tied to passage navigation, changes that occur *between* one passage navigation and the next are *not* part of the current moment and will not be preserved by a moment until the *next* navigation, when the next moment is created.

Consider the following:

```
:: one passage
<<set $var to 1>>

[[another passage]]

:: another passage
<<link "Click me!">>
        <<set $var to 2>>
<</link>>
```

In the above example, if you save the story after reaching the passage called *`another passage`*, the `$var` variable will be saved in the state as `1`, as you would expect. If you click the link that sets the variable to `2`, and then save the story, the `$var` variable will *still* be saved as `1`, because a new moment has not yet been created.

Playthrough Session

> **NOTE:** The autosave feature is occasionally confused with the playthrough session feature, but they are in fact distinct systems.

SugarCube automatically stores the current playthrough state to the browser's session storage whenever a new moment is created. This can be thought of as a special, temporary saved story, which is automatically deleted after the player's current browsing session ends. This temporary playthrough session is intended to prevent players from losing data. Some browsers, particularly mobile ones, will free up memory by unloading web pages that are running in the background. This functionally refreshes the webpage, and can cause users to lose their progress. When SugarCube is reloaded by the browser, it checks if a playthrough session exists and loads it to prevent any inadvertent loss of progress.

This feature also prevents players from losing progress if they try to use the browser back and forward buttons to navigate, or if they refresh their browser for any reason. The built-in *`Restart`* button, along with the methods `UI.restart()` and `Engine.restart()` are provided so that the story can be restarted without restoring a session.

To recap:

- When a new moment is created, SugarCube stores the playthrough state to session storage.
- If SugarCube is reloaded by the browser for whatever reason—e.g., due to a refresh, back/forward navigation, being unloaded in the background, etc.—then the session is restored.
- If SugarCube is reloaded by one of its own built-in restart methods, then the session is *not* restored.

Autosave

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: The playthrough session feature is occasionally confused with the autosave feature, but they are in fact distinct systems.

SugarCube features a configurable autosave system. The autosave is, for the most part, a normal save slot, but with a few special features built in. You can set the autosave to save either on every passage or only on certain passages. It can be loaded manually by the player or automatically by the autoload feature, which can be configured to, upon start up, either load the autosave automatically or prompt the player about loading it.

SEE: `Config.saves.autosave` setting, `Config.saves.autoload` setting, and `Save` API: Autosave.

What Happens When a Save is Loaded?

When a saved story is loaded, the state loaded from the save *replaces* the current state. This process is the same regardless of where the loaded state is coming from: it could be a normal save, the autosave, or the playthrough session. The previous state is completely lost—the new state is not added to or combined with the current state, instead it **replaces it in its entirety**. The easiest way to understand this is to look at what happens when you make some changes to `StoryInit` and then load a saved story from before those changes were made. For example:

```
:: StoryInit
<<set $x to 0>>

:: Start
$$x is <<if def $x>> $x <<else>> undefined <</if>>
```

If you run the above, you'll see `$x is 0`. Create a save, then edit the code as follows:

```
:: StoryInit
<<set $x to 0>>
<<set $y to 1>>

:: Start
$$x is <<if def $x>> $x <<else>> undefined <</if>>
$$y is <<if def $y>> $y <<else>> undefined <</if>>
```

Running that, you'll see `$x is 0` and `$y is 1`. Now, load the saved story from before the changes were made, and you'll see `$y is undefined`, since it doesn't exist at all in the loaded state.

Refreshing and Restarting

Whenever your story is first started or, *for any reason*, restarted—e.g., the browser window/tab was refreshed/reloaded—it undergoes its startup sequence. Several things occur **each and every time startup happens**, regardless of whether or not a playthrough session will be restored, an autosave loaded, or the starting passage run. First, the CSS, JavaScript, and Widget sections are processed. Next, the `StoryInit` special passage is processed. Finally, one of three things happen (in order): the existing playthrough session is restored, if it exists, else the autosave is loaded, if it exists and is configured to do so, else the starting passage is run.

Some users have the false impression that `StoryInit` is not run when the story is restarted when the playthrough session is restored or autosave is loaded. Code like `<<set $y to 1>>` *seems* to have no effect because the startup state is replaced by the of the incoming state, **but they are still executed by the engine**. You can see this effect by changing data *outside* the state. For example, let's return to the example above and change it again:

```
:: StoryInit
<<set $x to 0>>
<<set setup.y to 1>>

:: Start
$$x is <<if def $x>> $x <<else>> undefined <</if>>
setup.y is <<if def setup.y>> <<= setup.y>> <<else>> undefined <</if>>
```

You'll see that `setup.y` is being set to `1` and displayed properly regardless of whether you load a saved story or not, because it is not part of the state.

When the story is restarted by SugarCube rather than refreshed via the browser, the playthrough session, if any, is not loaded. `StoryInit` is run, as always. If the autosave exists and the story is configured to automatically load it, then the autosave is loaded and the state is replaced by the autosave's state and the

active passage is rendered, just as if the user had loaded any other save. If no autosave exists, then the starting passage is rendered.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Guide: Tips

This is a collection of tips, from how-tos to best practices.

Suggestions for new entries may be submitted by creating a new issue at SugarCube's source code repository.

Arbitrarily long return

WARNING: Navigating back to a previous passage, for whatever reason, can be problematic. There's no way for the system to know ahead of time whether it's safe to re-execute a passage's contents. Even if it did know that, there's no way for it to know which operations may or may not have side-effects—e.g., changing variables. Thus, if you allow players to return to passages, then you should either: ensure the passages contain no code that has side-effects or wrap that code in something to prevent re-execution—e.g., `<<if visited() is 1>>side-effects<</if>>`.

NOTE: An alternative to navigating to passages to create menus, inventories, and the like would be to use the `Dialog` API.

When you have a situation where you're using a set of passages as some kind of menu/inventory/etc and it's possible for the player to interact with several of those passages, or even simply the same one multiple times, then returning them to the passage they were at before entering the menu can be problematic as they're possibly several passages removed from that originating passage—thus, the `<<return>>` macro and link constructs like `[[Return|previous()]]` will not work.

The most common way to resolve this arbitrarily long return issue is to use a bit of JavaScript to record the last non-menu passage the player visited into a story variable and then to create a link with that.

For example, you may use the following JavaScript code to record the last non-menu passage into the `$return` story variable: (Twine 2: the Story JavaScript, Twine 1/Twee: a `script`-tagged passage)

```
$(document).on(':passagestart', function (ev) {
	if (!ev.passage.tags.includes('noreturn')) {
		State.variables.return = ev.passage.title;
	}
});
```

You'll need to tag each and every one of your menu passages with `noreturn`—you may use any tag you wish (e.g., `menu`, `inventory`), just ensure you change the name in the code if you decide upon another. If necessary, you may also use multiple tags by switching from `<Array>.includes()` to `<Array>.includesAny()` in the above example.

In your menu passages, your long return links will simply reference the `$return` story variable, like so:

```
→ Using link markup
[[Return|$return]]

→ Using <<link>> macro (separate argument form)
<<link "Return" $return>><</link>>
```

WARNING (TWINE 2): Due to how the Twine 2 automatic passage creation feature currently works, using the link markup form will cause a passage named `$return` to be created that will need to be deleted. To avoid this problem, it's suggested that you use the separate argument form of the `<<link>>` macro in Twine 2—as shown above.

Non-generic object types (a.k.a. classes)

As a basic working definition, non-generic object types—a.k.a. classes—are instantiable objects whose own prototype is not `Object`—e.g., `Array` is a native non-generic object type.

Many of the commonly used native non-generic object types are already fully compatible with and supported for use within story variables—e.g., `Array`, `Date`, `Map`, and `Set`. All other non-generic object types, on the other hand, must be made compatible to be successfully stored within story variables.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Making custom non-generic object types fully compatible requires that two methods be added to their prototype, `.clone()` and `.toJSON()`, to support cloning—i.e., deep copying—instances of the type.

- The `.clone()` method needs to return a clone of the instance.
- The `.toJSON()` method needs to return a code string that when evaluated will return a clone of the instance.

In both cases, since the end goal is roughly the same, this means creating a new instance of the base object type and populating it with clones of the original instance's data. There is no one size fits all example for either of these methods because an instance's properties, and the data contained therein, are what determine what you need to do.

> **SEE ALSO:** The `JSON.reviveWrapper()` method for additional information on implementing the `.toJSON()` method.

Examples: *(not an exhaustive list)*

Config/option object parameter constructor (automatic copying of own data)

Here's a simple example whose constructor takes a single config/option object parameter:

```
window.ContactInfo = function (config) {
	// Set up our own data properties with some defaults.
	this.name    = '';
	this.address = '';
	this.phone   = '';
	this.email   = '';

	// Clone the given config object's own properties into our own properties.
	//
	// NOTE: We use the SugarCube built-in `clone()` function to make deep
	// copies of each of the properties' values.
	Object.keys(config).forEach(function (pn) {
		this[pn] = clone(config[pn]);
	}, this);
};

ContactInfo.prototype.clone = function () {
	// Return a new instance containing our own data.
	return new ContactInfo(this);
};

ContactInfo.prototype.toJSON = function () {
	// Return a code string that will create a new instance containing our
	// own data.
	//
	// NOTE: Supplying `this` directly as the `reviveData` parameter to the
	// `JSON.reviveWrapper()` call will trigger out of control recursion in
	// the serializer, so we must pass it a clone of our own data instead.
	var ownData = {};
	Object.keys(this).forEach(function (pn) {
		ownData[pn] = clone(this[pn]);
	}, this);
	return JSON.reviveWrapper('new ContactInfo($ReviveData$)', ownData);
};
```

Creating a new instance of this `ContactInfo` example would be something like:

```
<<set $Joe to new ContactInfo({
	name  : 'Joe Blow',
	phone : '1 555 555 1212',
	email : 'joe@blow.com'
})>>
```

Discrete parameters constructor (manual copying of own data)

Here's a simple example whose constructor takes multiple discrete parameters:

```
window.ContactInfo = function (name, addr, phone, email) {
	// Set up our own data properties with the given values or defaults.
	this.name    = name || '';
	this.address = addr || '';
	this.phone   = phone || '';
	this.email   = email || '';
};

ContactInfo.prototype.clone = function () {
	// Return a new instance containing our own data.
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        return new ContactInfo(
                this.name,
                this.address,
                this.phone,
                this.email
        );
};

ContactInfo.prototype.toJSON = function () {

        // Return a code string that will create a new instance containing our
        // own data.
        return JSON.reviveWrapper(String.format(
                'new ContactInfo({0},{1},{2},{3})',
                JSON.stringify(this.name),
                JSON.stringify(this.address),
                JSON.stringify(this.phone),
                JSON.stringify(this.email)
        ));
};
```

Creating a new instance of this `ContactInfo` example would be something like:

```
<<set $Joe to new ContactInfo(
        'Joe Blow',
        '',
        '1 555 555 1212',
        'joe@blow.com'
)>>
```

Discrete parameters constructor (automatic copying of own data)

Here's a simple example whose constructor takes multiple discrete parameters, but also includes an `._init()` helper method to allow the `.clone()` and `.toJSON()` methods to require less manual tinkering than the previous discrete parameters example by automatically copying an instance's own data:

```
window.ContactInfo = function (name, addr, phone, email) {
        // Set up our own data properties with the given values or defaults.
        this.name    = name || '';
        this.address = addr || '';
        this.phone   = phone || '';
        this.email   = email || '';
};

ContactInfo.prototype._init = function (obj) {
        // Clone the given object's own properties into our own properties.
        Object.keys(obj).forEach(function (pn) {
                this[pn] = clone(obj[pn]);
        }, this);

        // Return `this` to make usage easier.
        return this;
};

ContactInfo.prototype.clone = function () {
        // Return a new instance containing our own data.
        return (new ContactInfo())._init(this);
};

ContactInfo.prototype.toJSON = function () {
        // Return a code string that will create a new instance containing our
        // own data.
        //
        // NOTE: Supplying `this` directly as the `reviveData` parameter to the
        // `JSON.reviveWrapper()` call will trigger out of control recursion in
        // the serializer, so we must pass it a clone of our own data instead.
        var ownData = {};
        Object.keys(this).forEach(function (pn) {
                ownData[pn] = clone(this[pn]);
        }, this);
        return JSON.reviveWrapper('(new ContactInfo())._init($ReviveData$)', ownDat
};
```

Creating a new instance of this `ContactInfo` example would be something like:

```
<<set $Joe to new ContactInfo(
        'Joe Blow',
        '',
        '1 555 555 1212',
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
        'joe@blow.com'
)>>
```

Guide: Media Passages

Media passages are simply a way to embed media into your project—specially tagged passages that contain the data URI of a Base64-encoded media source. Audio, image, video, and VTT passages are supported.

For example, the following is the data URI of a Base64-encoded PNG image of a red dot (●):

```

lEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg==
```

History:

- `v2.0.0`: Image passages.
- `v2.24.0`: Added audio, video, and VTT passages.

Creation

Generally, it's expected that you will use a compiler that supports the automatic creation of media passages, however, they may be created manually.

Automatically

Compilers supporting automatic creation of media passages:

- Tweego supports all media passages.
- Twine 1 supports image passages.

Manually

> **WARNING (TWINE 2):** Due to various limitations in its design, if you're using Twine 2 as your IDE/compiler, then it is ***strongly*** recommended that you do not create more than a few media passages and definitely do not use large sources.

To manually create a media passage:

1. Create a new passage, which will only be used as a media passage—one per media source.
2. Tag it with the appropriate media passage special tag, and only that tag—see below.
3. Paste in the Base64-encoded media source as the passage's content.

See the MDN article *Media formats for HTML audio and video* for more information on formats commonly supported in browsers—pay special attention to the *Browser compatibility* section.

Media passage special tags

> **NOTE:** As with all special tags, media passage tags are case sensitive, so their spelling and capitalization must be *exactly* as shown.

Passage type	Tag
Audio passage	`Twine.audio`
Image passage	`Twine.image`
Video passage	`Twine.video`
VTT passage	`Twine.vtt`

Guide: Harlowe to SugarCube

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

There are many differences between Harlowe and SugarCube, this guide will document some of the most critical you will need to account for if you're coming to SugarCube from a background in Harlowe.

Macro Overview

Aside from general syntax, SugarCube macros do not use hooks, separate arguments differently, and don't allow other macros to be passed as arguments.

Macro Arguments

Like in Harlowe, some SugarCube macros accept expressions and others accept discreet arguments. In SugarCube, discreet arguments passed to a macro are separated by spaces instead of commas. To pass expressions or the results of functions to macros as an argument, you must wrap the expression in backquotes (`).

Additionally, macros in SugarCube do not return values, so macros cannot be used as arguments to other macros. SugarCube provides a variety of functions and methods that may be used instead, and standard JavaScript functions and methods may also be used.

Consider the following Harlowe code:

```
(link-goto: "Go somewhere else", (either: "this passage", "that passage", "the othe
```

A version of the above code in SugarCube might look like this:

```
<<link "Go somewhere else" `either("this passage", "that passage", "the other passa
```

SEE: Macro Arguments.

Container Macros

Where Harlowe uses its hook syntax (square brackets) to associate a macro with its contents, SugarCube instead uses "container" macros—macros that can have content associated with them have opening and closing tags.

Consider the following Harlowe code:

```
(if: $var is 1)[
        The variable is 1.
]
```

In SugarCube, you instead open and close the `<<if>>` macro itself:

```
<<if $var is 1>>
        The variable is 1.
<</if>>
```

Specific Macros

Some macros in Harlowe and SugarCube share a name but work a bit differently. We'll cover some of these differences below.

Link and Click Macros

SugarCube does not have any equivalents to Harlowe's `(click:)` family of macros. Additionally, SugarCube's normal `<<link>>` macro does not have an output element associated with it and is not, by default, a single-use link like its Harlowe equivalent. Both of these features can be constructed in SugarCube, however, using macros like `<<linkreplace>>` or by combining `<<link>>` macros with DOM macros. Additionally, SugarCube's link macro accepts a passage argument, that, if included, turns any `<<link>>` into something similar to Harlowe's `(link-goto:)` macro.

Consider the following Harlowe link macros:

```
(link: "Hey there.")[Hay is for horses.]
(link-repeat: "Get some money")[(set: $cash to it + 1)]
```

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
(link-goto: "Move on", "next passage")
```

The equivalent SugarCube code for each link might look something like this:

```
<<linkreplace "Hey there.">>Hay is for horses.<</linkreplace>>
<<link "Get some money">><<set $cash += 1>><</link>>
<<link "Move on" "next passage">><</link>>
```

SugarCube's `<<link>>` and `<<button>>` macros can also accept the link markup as an argument:

```
<<link [[Move on|next passage]]>><</link>>
```

DOM Macros

NOTE: Harlowe refers to these as "revision macros".

SugarCube's DOM macros can target any HTML element on the page, not just hooks, and unlike their Harlowe equivalents, they cannot target arbitrary strings. You can use custom style markup or HTML to create the elements, and then target them with a query selector.

Consider the following Harlowe code:

```
(set: _greetings to (a: "hi", "hello", "good day", "greetings"))\
The man says, "|target>[(either: ..._greetings)]."

{
(link-repeat: "Change")[
        (replace: ?target)[(either: ..._greetings)]
]
}
```

The equivalent SugarCube code to achieve a similar result would be:

```
<<set _greetings to ["hi", "hello", "good day", "greetings"]>>\
The man says, "@@#target;<<= _greetings.random()>>@@."

<<link "Change">>
        <<replace "#target">><<= _greetings.random()>><</replace>>
<</link>>
```

NOTE: The DOM macros do have a limitation that you should familiarize yourself with.

The Goto Macro

Harlowe's implementation of the `(goto:)` macro terminates the rendering passage. In SugarCube, the passage is not terminated, and anything in the code below the `<<goto>>` macro will have side effects.

Consider this Harlowe code:

```
:: some passage
(set: $count to 0)
(goto: "next")
(set: $count to it + 1)

:: next
$count <!-- 0 -->
```

In the above, the second `(set:)` macro is never run, and the `$count` variable remains at 0.

The equivalent SugarCube code works a bit differently:

```
:: some passage
<<set $count to 0>>
<<goto "next">>
<<set $count += 1>>

:: next
$count /* 1 */
```

SugarCube does not terminate the parsing of the calling passage, so some care is required when calling `<<goto>>`.

As with `<<link>>` and `<<button>>`, `<<goto>>` can accept link markup as its argument:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
<<goto [[next]]>>
```

User Input

SugarCube's user input macros, like `<<textbox>>`, cannot be nested inside a `<<set>>` macro, as you might do with a `(prompt:)` and a `(set:)` in Harlowe. Instead, the macro is passed a *receiver variable* which is set to the value input by the user.

For example, if you wanted to ask the user to enter a name, your code may look like this in Harlowe:

```
(set: $name to (prompt: "What is your name?", "Frank"))
```

In SugarCube, you would likely want to use the `<<textbox>>` macro instead, and pass `$name` in as the receiving variable:

```
<label>What is your name? <<textbox "$name" "Frank">></label>
```

Harlowe's newer input macros, like `(dropdown:)` and `(cycling-link:)` use "bound" variables, which are similar in concept to SugarCube's receiver variables.

Data Types

Harlowe's implementation of data types differs significantly from SugarCube's. A data type refers to the "type" of data a variable is holding, such as a number, a string, an array, or anything else. Harlowe has stricter typing than SugarCube, requiring authors to call macros like `(str:)` or `(num:)` on variables to change their type. SugarCube, like JavaScript, uses *dynamic* typing.

Dynamic Typing

SugarCube, like JavaScript, will try to make sense of expressions passed to it by *coercing* their values if necessary:

```
<<set $number to 1>>
<<set $string to "2">>
<<= $string + $number>> /* "21" */
```

In the above case, since the string value `"2"` cannot be added to a number value, the number value is *coerced* into a string, and the two strings are then *concatenated*. In Harlowe, the same operation will yield an error:

```
(set: $number to 1)
(set: $string to "2")
(print: $string + $number) <!-- error! -->
```

You *must* convert the values to the same type in Harlowe. In SugarCube you *can* convert them if you need to.

In Harlowe:

```
(set: $number to 1)
(set: $string to "2")
(print: $string + $number) <!-- error! -->
(print: $string + (str: $number)) <!-- "21" -->
(print: (num: $string) + $number) <!-- 3 -->
```

In SugarCube:

```
<<set $number to 1>>
<<set $string to "2">>
<<= $string + $number>> /* "21" */
<<= $string + String($number)>> /* "21" */
<<= Number($string) + $number>> /* 3 */
```

Arrays, Datamaps, and Datasets

Harlowe's arrays, datamaps, and datasets are functionally similar to JavaScript `Array`s, `Map`s, and `Set`s, but with a few key differences. SugarCube requires authors to define and work with these data types using the standard JavaScript methods rather than providing macros for them.

Using an array in Harlowe:

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

```
(set: $array to (a:))
(set: $array to it + (a: "something"))
(if: $array contains "something")[…]
```

In SugarCube:

```
<<set $array to []>>
<<run $array.push("something")>>
<<if $array.includes("something")>>…<</if>>
```

Using a datamap in Harlowe:

```
(set: $map to (dm: "key", "value"))
(set: $map's key to "another value")
(if: $map contains key)[…]
```

In SugarCube:

```
<<set $map to new Map([["key", "value"]])>>
<<run $map.set("key", "another value")>>
<<if $map.has("key")>>…<</if>>
```

SugarCube also allows the use of JavaScript generic objects, which may be better in some situations than a map:

```
<<set $object to { key : "value" }>>
<<set $object.key to "another value">>
<<if $object.hasOwnProperty("key")>>…<</if>>
```

Another important difference in the way Harlowe handles its non-primitive data types like arrays, datamaps, and datasets is that they are *passed by value* rather than *passed by reference*.

Consider the following Harlowe code:

```
(set: $player to (dm: "hp", 100, "mp", 50))
(set: $partyMember to $player)
(set: $partyMember's hp to it - 50)
(print: $player's hp) <!-- 100 -->
(print: $partyMember's hp) <!-- 50 -->
```

As you can see, Harlowe creates a deep copy/clone of its non-primitive data types each time they're modified.

In SugarCube, both variables would still point to the same underlying object—at least initially (see below):

```
<<set $player to { hp : 100, mp : 50 }>>
<<set $partyMember to $player>>
<<set $partyMember.hp -= 50>>
$player.hp /* 50 */
$partyMember.hp /* 50 */
```

SugarCube does eventually clone its non-primitive data types as well, but does at the start of passage navigation, rather than each time they're modified.

Guide: Test Mode

History:

- `v2.2.0`: Introduced.

Introduction

In test mode, SugarCube will wrap all macros, and some non-macro markup—e.g., link & image markup—within additional HTML elements, called "debug views" ("views" for short). Views make their associated code visible, thus providing onscreen feedback—they may also be hovered over which, generally, exposes additional information about the underlying code.

> **WARNING:** Because of the additional HTML elements added by the debug views, some nested markup and selectors may be broken. This only affects test mode.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

TIP: In versions of SugarCube ≥v2.23.0, the debugging interface offers additional tools, namely variable watches and arbitrary history navigation.

Enabling Test Mode

Automatically

In Tweego

To enable test mode, use the test option (`-t`, `--test`).

In Twine 2 (≥v2.2)

To enable test mode from the *Stories* screen, click on the story's gear menu and select the *Test Story* menu item.

To enable test mode from the story editor/map screen, click on the *Test* menu item (right side of the bottom bar).

To enable test mode from the story editor/map screen while starting at a specific passage, hover over a passage and select the ▶ menu item.

In Twine 2 (<v2.2)

To enable test mode from the *Stories* screen, click on the story's gear menu and select the *Test Play* menu item.

To enable test mode from the story editor/map screen, click on the *Test* menu item (right side of the bottom bar).

To enable test mode from the story editor/map screen while starting at a specific passage, hover over a passage and select the menu item.

In Twine 1

To enable test mode while starting at a specific passage, right-click on a passage and select the *Test Play From Here* context menu item.

NOTE: Unfortunately, due to limitations in the current release of Twine 1, the *Build* menu's *Test Play* menu item is not able to trigger test mode. You may, however, simply use the *Test Play From Here* context menu item on the `Start` passage to achieve the same result.

Manually

You may forcibly enable test mode manually by setting the `Config` object's `debug` property to `true`. For example:

```
Config.debug = true; // forcibly enable test mode
```

SEE: The `Config.debug` setting for more information.

Debug Bar (≥v2.23.0)

The debug bar (bottom right corner of the page) allows you to: watch the values of story and temporary variables, toggle the debug views, and jump to any moment/turn within the history.

The variable watch panel may be toggled via the *Watch* button. To add a watch for a variable, type its name into the *Add* field and then either press enter/return or click the ✚ button—n.b. depending on the age of your browser, you may also see a list of all current variables when interacting with the *Add* field. To delete a watch, click the ✖ button next to its name in the watch panel. To add watches for all current variables, click the button. To delete all current watches, click the button.

The debug views may be toggled via the *Views* button.

To jump to any moment/turn within the available history, select the moment/turn from the *Turn* select field.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Debug Views (≤v2.22.0)

The debug views themselves may be toggled on and off (default: on) via the *Debug View* button (top of the UI bar).

If you've removed/hidden the UI bar, a construct like the following will allow you to toggle the views on and off:

```
<<button "Toggle Debug Views">><<script>>DebugView.toggle()<</script>><</button>>
```

NOTE: That will only toggles the views, test mode must still be enabled first.

Guide: TypeScript

TypeScript bindings for SugarCube APIs can found as the Definitely Typed package: `@types/twine-sugarcube`.

To install the package via NPM, use the following command:

```
npm install --save-dev @types/twine-sugarcube
```

Guide: Installation

This is a reference on how to install SugarCube in Tweego, Twine 2, and Twine 1/Twee.

NOTE (TWINE 2): Newer versions of Twine 2 come bundled with a version of SugarCube v2, so you only need to read these instructions if you want to install a newer version of SugarCube v2 than is bundled or a non-standard release.

Local Install For Tweego

See Tweego's documentation for more information.

Local Install For Twine 2

There are two primary branches of Twine 2 as far as SugarCube is concerned:

- Those that bundle SugarCube v2: Any series of Twine 2 with a version ≥2.1.
- Those that do not bundle SugarCube v2: Only the older Twine 2.0 series.

Regardless of the version of Twine 2 you're using, follow these instructions to install a local copy of SugarCube v2:

1. Download the current version of SugarCube v2 for Twine 2—comes as a ZIP archive. **NOTE:** There are separate downloads for Twine ≥2.1 and Twine 2.0, so you ***must*** ensure that you download the one that matches your version of Twine 2 or you will likely run into issues.
2. Extract the archive to a safe location on your computer and make note of the path to it. Make sure to keep the files together if you move them out of the included directory.
3. Launch Twine 2.
4. Click on the `Formats` link in the Twine 2 sidebar.
5. In the dialog that opens, click on the `Add a New Format` tab.
6. Finally, paste a file URL to the `format.js` file, based upon the path from step #2, into the textbox and click the `+Add` button (see below for examples).

UNIX (and similar) file URL examples

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

NOTE: If constructing the file URL from a shell path, ensure that either it does not contain escapes or you properly convert them into the correct URL percent-encoded form.

If the full path to the contents of the archive is something like:

```
/home/soandso/Twine/StoryFormats/SugarCube-2/format.js
```

Then the file URL to it would be:

```
file:///home/soandso/Twine/StoryFormats/SugarCube-2/format.js
```

Windows file URL examples

If the full path to the contents of the archive is something like:

```
C:\Users\soandso\Documents\Twine\StoryFormats\SugarCube-2\format.js
```

Then the file URL to it would be (note the changed slashes):

```
file:///C:/Users/soandso/Documents/Twine/StoryFormats/SugarCube-2/format.js
```

Online Install For Twine 2

The online SugarCube install, delivered by the jsDelivr CDN, supports only versions of Twine 2 ≥2.1.

Copy the following URL and paste it into the *Add a New Format* tab of the *Formats* menu, from Twine 2's sidebar.

URL: https://cdn.jsdelivr.net/gh/tmedwards/sugarcube-2/dist/format.js

Local Install For Twine 1/Twee

Follow these instructions to install a local copy of SugarCube v2:

1. Download the current version of SugarCube v2 for Twine 1/Twee—comes as a ZIP archive.
2. Go to your Twine 1/Twee installation directory and open the `targets` directory within.
3. Move the downloaded archive into the `targets` directory and extract it, included directory and all.

If you followed the steps correctly, within Twine 1/Twee's `targets` directory you should now have a `sugarcube-2` directory, which contains several files—e.g., `header.html`, `sugarcube-2.py`, etc.

WARNING (TWINE 1): Due to a flaw in the current release of Twine 1/Twee (`v1.4.2`), if you rename the directory included in the archive (or simply copy its contents to your current SugarCube v2 install), then you ***must*** ensure that the file with the extension `.py` (the story format's custom Twine 1 Header class file) within is named the same as the directory—i.e., the name of the directory and `.py` file must match.

For example, if the name of SugarCube's directory is `sugarcube`, then the name of the `.py` file within must be `sugarcube.py`. Similarly, if the directory is `sugarcube-2`, then the name of the `.py` file within must be `sugarcube-2.py`. Etc.

The directory and `.py` file names within the archive available for download are already properly matched—as `sugarcube-2` and `sugarcube-2.py`—and to avoid issues it recommended that you simply do not rename them.

Guide: Code Updates

This is a reference on how to update existing SugarCube code to work with newer versions of SugarCube.

NOTE: The majority of newer SugarCube versions do not have any changes that would require an update. For those versions that do, the updates are normally completely elective and may be addressed at your leisure, or not at all. Sometimes there are breaking changes, however, and these must be addressed immediately.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Updating to any version ≥2.30.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

`Config` API

Property	Changes
`Config.saves.autosave`	This setting property has been updated to accept function values and its acceptance of string values has been deprecated. String values will still be accepted for further releases of v2, however, switching to an array is recommended—e.g., the string value `"autosave"` would become the array `["autosave"]`. See the `Config.saves.autosave` property for more information.

Updating to any version ≥2.29.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

`Dialog` API

Method	Changes
`Dialog.addClickHandler()`	This method has been deprecated and should no longer be used. The core of what it does is simply to wrap a call to `Dialog.open()` within a call to `<jQuery>.ariaClick()`, which can be done directly and with greater flexibility.

Macro library

Macro	Changes
`<<track>>` (of: `<<createplaylist>>`)	The `<<createplaylist>>` macro's <<track>> child macro has had its `copy` keyword deprecated in favor of `own`. See the `<<createplaylist>>` macro for more information.
`<<forget>>`	The `<<forget>>` macro has been deprecated in favor of the `forget()` function.
`<<remember>>`	The `<<remember>>` macro has been deprecated in favor of the `memorize()` and `recall()` functions.

Method library

Method	Changes
`<Array>.flatten()`	This method has been deprecated in favor of the `<Array>.flat()` method.

`State` API

Method	Changes
`State.initPRNG()`	This method has been deprecated in favor of the `State.prng.init()` method.

Updating to any version ≥2.28.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

Macro library

Macro	Changes

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Macro	Changes
`<<cacheaudio>>`	The `<<cacheaudio>>` macro's original optional format specifier syntax, `format:formatId;…`, has been deprecated in favor of the new syntax, `formatId|…`.

Updating to any version ≥2.20.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

Method library

Method	Changes
`Array.random()`	This method has been deprecated and should no longer be used. In general, look to the `<Array>.random()` method instead. If you need a random member from an array-like object or iterable, use the `Array.from()` method to convert it to an array, then use `<Array>.random()`—e.g., `Array.from(something).random()`.

Updating to any version ≥2.15.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

Macro library

Macro	Changes
`<<display>>`	The `<<display>>` macro has been deprecated in favor of the `<<include>>` macro.

Updating to any version ≥2.10.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

Method library

Method	Changes
`<Array>.contains()`	The `<Array>.contains()` method has been deprecated in favor of the `<Array>.includes()` method.
`<Array>.containsAll()`	The `<Array>.containsAll()` method has been deprecated in favor of the `<Array>.includesAll()` method.
`<Array>.containsAny()`	The `<Array>.containsAny()` method has been deprecated in favor of the `<Array>.includesAny()` method.

`strings` object

The `strings` API object has been replaced by the `l10nStrings` object. See the Localization guide for more information.

Updating to any version ≥2.8.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

Macro library

Macro	Changes
`<<click>>`	The `<<click>>` macro has been deprecated in favor of the `<<link>>` macro.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Macro	Changes
`<<playlist>>`	The `<<playlist>>` macro has had its argument list changed, for compatibility with `<<createplaylist>>`. See the `<<playlist>>` macro for more information.
`<<setplaylist>>`	The `<<setplaylist>>` macro has been deprecated in favor of the `<<createplaylist>>` macro.
`<<stopallaudio>>`	The `<<stopallaudio>>` macro has been deprecated in favor of `<<audio ":all" stop>>`. See the `<<audio>>` macro for more information.

Updating to any version ≥2.5.0 from a lesser version

All changes within this version are elective changes that you may address at your leisure.

`config` API

The `config` API has been renamed `Config` for better consistency with the other APIs.

`State` API

Several `State` API methods have moved to the new `Engine` API. See the `Engine` API docs for more information.

Old `State` method	New `Engine` method
`State.backward()`	`Engine.backward()`
`State.display()`	`Engine.display()`[1]
`State.forward()`	`Engine.forward()`
`State.play()`	`Engine.play()`
`State.restart()`	`Engine.restart()`
`State.show()`	`Engine.show()`

1. While the `Engine.display()` static methods exists, it, like `State.display()` before it, is deprecated. See the `Engine.play()` static method for the replacement. **NOTE:** Their parameters differ, so read the description of `Engine.play()` carefully.

`UI` API

Several `UI` API methods have moved to the new `Dialog` API. See the `Dialog` API docs for more information.

Old `UI` method	New `Dialog` method
`UI.addClickHandler()`	`Dialog.addClickHandler()`
`UI.body()`	`Dialog.body()`
`UI.close()`	`Dialog.close()`
`UI.isOpen()`	`Dialog.isOpen()`
`UI.open()`	`Dialog.open()`
`UI.setup()`	`Dialog.setup()`

Updating to any version ≥2.0.0 from a lesser version

WARNING: All changes within this version are **breaking changes** that you ***must*** address immediately.

HTML & CSS

The HTML & CSS have undergone ***significant*** changes. See the `HTML` and `CSS` docs for more information.

Special passages

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Passage	Changes
`MenuOptions`	The `MenuOptions` special passage has been removed. See the Options system section for more information.
`MenuShare`	The `MenuShare` special passage has been removed. Instead, use the `StoryShare` special passage.
`MenuStory`	The `MenuStory` special passage has been removed. Instead, use the `StoryMenu` special passage.

`config` object

The `config` object has been renamed to `Config` and some of its properties have also changed. See the `Config` API docs for more information.

Property	Changes
`config.altPassageDescription`	Changed the `config.altPassageDescription` property to `Config.passages.descriptions`.
`config.disableHistoryControls`	Changed the `config.disableHistoryControls` property to `Config.history.controls`. This change also inverted the meaning of the property, so take note of that.
`config.disableHistoryTracking`	Replaced the `config.disableHistoryTracking` property with `Config.history.maxStates`. The new property works differently, so take note of that.
`config.displayPassageTitles`	Changed the `config.displayPassageTitles` property to `Config.passages.displayTitles`.
`config.historyMode`	Removed the `config.historyMode` property. It's unnecessary since there's now only one history mode in the engine.
`config.macros.disableIfAssignmentError`	Changed the `config.macros.disableIfAssignmentError` property to `Config.macros.ifAssignmentError`. This change also inverted the meaning of the property, so take note of that.
`config.passageTransitionOut`	Changed the `config.passageTransitionOut` property to `Config.passages.transitionOut`. Additionally, it no longer accepts a boolean value, which has been replaced by the name of the animating property (necessitated by changes to browser handling of transition animations).
`config.startPassage`	Changed the `config.startPassage` property to `Config.passages.start`.
`config.updatePageElements`	Changed the `config.updatePageElements` property to `Config.ui.updateStoryElements`.

History object & prototype (instance: `state`)

The `History` API object has been renamed to `State` and some of its methods have also changed. Furthermore, it is no longer instantiated into the legacy `state` object—which still exists, so legacy code will continue to work. See the `State` API docs for more information.

The `State.display()` method—formerly `state.display()`—is no longer overridable, meaning it cannot be wrapped—e.g., the "StoryRegions" 3rd-party add-ons do this. Instead, use Navigation Events or Tasks.

Calling the `State.prng.init()` method—formerly `History.initPRNG()`—outside of story initialization will now throw an error. It has always been required that the call happen during story initialization, the only change is the throwing of the error.

Seedable pseudo-random number generator (PRNG)

`Math.random()` is no longer replaced by the integrated seedable PRNG when `State.prng.init()` is called. Instead, use either the built-in functions `random()` & `randomFloat()` or the `State.random()` method, if you need direct access to the PRNG—since it returns a call to either `Math.random()` or the seedable PRNG, as appropriate.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Macro system

The `Macros` API object has been renamed to `Macro` and several of its methods have also changed, for better consistency with the other APIs. Furthermore, it is no longer instantiated into the legacy `macros` object—which still exists, so SugarCube-compatible legacy macros will continue to work. See the `Macro` API docs for more information.

Macro library

Macro	Changes
`<<back>>` & `<<return>>`	Replaced the ungainly link text syntax —`<<back::linktext "…">>`/`<<return::linktext "…">>`—with `l10nStrings` object properties—`l10nStrings.macroBackText` and `l10nStrings.macroReturnText`.
`<<if>>` & `<<elseif>>`	The `<<if>>` macro will now, optionally, return an error if the JavaScript `=` assignment operator is used (default: enabled). Configured via the `Config.macros.ifAssignmentError` config property.
Options macros	The various Options macros have been removed. See the Options system section for more information.

Options system

The entire Options system—`MenuOptions` special passage, `options` special variable, and associated macros—has been scrapped for numerous reasons—it was always a hack, required copious amounts of boilerplate code to be useful, etc. It is replaced by the `Setting` API and `settings` special variable. See the `Setting` API docs for more information.

Save system

The `SaveSystem` API object has been renamed to `Save` and several of its methods have also changed, for better consistency with the other APIs. See the `Save` API docs for more information.

UI system

The `UISystem` API object has been split into two APIs `Dialog` and `UI`, and some of its methods have also changed. In particular, the parameter list for the `Dialog.setup()` method has changed. See the `Dialog` API and `UI` API docs for more information.

Guide: Localization

This is a reference for localizing SugarCube's default UI text, in general, and its `l10nStrings` object specifically.

> **NOTE:** If you're simply looking to download ready-to-use localizations, see SugarCube's website (under *Downloads* > *Localizations*).

History:

- `v2.0.0`: Introduced.
- `v2.10.0`: Added `l10nStrings` object. Deprecated `strings` object.

A note about `strings` vs. `l10nStrings`

Prior to SugarCube `v2.10.0`, the strings localization object was named `strings`. The new `l10nStrings` object has a simpler, flatter, set of properties and better support for replacement strings. Unfortunately, this means that the two objects are incompatible.

To ensure backwards compatibility of existing `strings` objects, if one exists within a project's scripts, the older object is mapped to the new `l10nStrings` object.

SugarCube v2 Documentation

2. 36. 1 (2021-12-22)

Find in page: CTRL + F or F3

Code Color

Introduction

Markup

TwineScript

Macros

Functions

Methods

Special Names

CSS

HTML

Events

Config API

Dialog API

Engine API

Fullscreen API

LoadScreen API

Macro API

MacroContext API

Passage API

Save API

Setting API

SimpleAudio API

AudioTrack API

AudioRunner API

AudioList API

State API

Story API

Template API

UI API

UIBar API

Guide: State, Sessions, and Saving

Guide: Tips

Guide: Media Passages

Guide: Harlowe to SugarCube

Guide: Test Mode

Guide: TypeScript

Translation Notes

The capitalization and punctuation used within the default replacement strings is deliberate, especially within the error and warning strings. You would do well to keep your translations similar when possible.

Replacement patterns have the format `{NAME}`—e.g., `{identity}`—where NAME is the name of a property within either the `l10nStrings` object or, in a few cases, an object supplied locally where the string is used—these instances will be commented.

By convention, properties starting with an underscore—e.g., `_warningIntroLacking`—are used as templates, only being included within other localized strings. Feel free to add your own if that makes localization easier—e.g., for gender, plurals, and whatnot. As an example, the default replacement strings make use of this to handle various warning intros and outros.

In use, replacement patterns are replaced recursively, so replacement strings may contain patterns whose replacements contain other patterns. Because replacement is recursive, care must be taken to ensure infinite loops are not created—the system will detect an infinite loop and throw an error.

Usage

Properties on the strings localization object (`l10nStrings`) should be set within your project's JavaScript section (Twine 2: the Story JavaScript; Twine 1/Twee: a `script`-tagged passage) to override the defaults.

For the template that should be used as the basis of localizations, see the `locale/l10n-template.js` file @github.com.

Examples

```
// Changing the project's reported identity to "story"
l10nStrings.identity = "story";

// Changing the text of all dialog OK buttons to "Eeyup"
l10nStrings.ok = "Eeyup";

// Localizing the title of the Restart dialog (n.b. machine translations, for exam
l10nStrings.restartTitle = "Neustart";  // German (de)
l10nStrings.restartTitle = "Reiniciar"; // Spanish (es)
```