Table of Contents

Welcome to the Twine Cookbook!

The *Twine Cookbook* is a collection of examples organized around common topic areas. For each, contributors have tried to provide examples covering each major built-in story format (Chapbook, Harlowe, SugarCube, and Snowman) where possible and appropriate. For some topics, examples are also provided in SugarCane for Twine 1.4.2.

Each example page includes a short description, a live version, and its Twee code. Both the live version and Twee code can be downloaded directly from each example page.

Reading the Cookbook

Created using GitBook, the *Twine Cookbook* is best viewed as a compiled webpage. In this format, the left sidebar provides access to the different topic areas as well as to a search bar for more quickly looking through the titles and text of the entries. New updates are published in the website format every few months or as necessary.

The *Twine CookBook* can also be read without the live versions on GitHub by starting with the summary page. Note, however, that example pages on GitHub are subject change as new changes are made and revisions submitted before and during editing cycles.

Suggesting Examples or Changes

Include as much as possible of the following in an issue:

- A short summary of the example(s) or suggestions(s)
- How (or if) you would like to be recognized for your contribution

Submitting Documentation or Code Updates

- Clone the repository.
- Review the Organization page on the wiki for how folders and files are named and structured.
- Read the recipe formatting page included in the Cookbook and on Github for layout and style guidelines.
- Add your recipe and submit a pull request to request it to be added to the cookbook.

Contributors:

- @videlais (Dan Cox)
- @klembot (Chris Klimas)
- @tmedwards (Thomas Michael Edwards)
- @greyelf (David Tarrant)
- @webbedspace (Leon Arnott)
- @shawngraham (Shawn Graham)
- @Akjosch (Akjosch)
- @ChapelR (Chapel)

License:

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Version 1.5.1 (February 2020)

What can I make with Twine?

At its heart, Twine is a tool for creating *hypertext*.

The difference between hypertext and a linear story, the kind found in books and magazines, is that it allows the reader to have some measure of control. In other words, the reader has some ability over what they interact with next. In a story about a haunted house, for example, the reader might be able to tell the protagonist to "Turn around and run" or "Venture deeper into the mausoleum". In a nonfiction piece, the reader might ask to learn more about my aunt, who went missing.

The convention that has emerged over the past three decades is that readers navigate hypertexts by clicking links. In this sense, you're already a seasoned hypertext reader by reading this very page! You clicked several links to reach this text, after all, and you've probably clicked an uncountable number of links in your life so far.

Because hypertext branches so much, it's easy to get lost in your own work. Much of Twine is dedicated to helping you keep track of your work's structure visually with a Passages View, so you can see what your readers' experience will be like.

Can I build games with Twine?

Of course!

However, things a little more complicated than they initial appear. Twine itself can be thought of as more of an editor that helps package up Stories. What provides the underlining conditional logic, variables, and other trappings of game programming are Story Formats.

Getting Started

Online

Twine 2 can be accessed online.

Offline

Twine 2 can also be downloaded and run on Windows, MacOS X, and Linux systems. The latest releases on GitHub has a listing of the current versions per operating system.

Twine 1

Twine 1 is only a desktop application. While it can still be used, it is not currently maintained or under active development. The current version is 1.4.2.

- Windows
- MacOS X

How do I make links?

A link is a connection between one passage and another, both visually in Twine 2 and as a hyperlink a user can click to navigate to a passage of the same name in the published HTML.

Passage Link

When something is enclosed within the two sets of opening and closing square brackets, this "links" one passage to another of that exact word or phrase. The "link" is the name of that other passage.

For example, the code below links to the passage named "Example":

```
[[Example]]
```

When using Twine 2, it will automatically create the passage if it does not exist when linked from another passage. The Passages View shows the links between them.

Routing Links

It is also possible to "route" links to different passages. Arrows, `->` or `<-`, can be used to point to the destination of the link and "away" from the word or phrase used as the link text.

```
[[Link to another passage->Different Passage Name]]

[[Different Passage Name<-Link to another passage]]
```

The pipe character, `|`, can also be used to route a user to a different passage than shown in the hyperlink.

For example, the code below routes to the passage "Another" rather than "Example" as it would show:

```
[[Example|Another]]
```

How do I style text?

Each Story Format uses their own markdown and special characters to format text. Some common examples are included below.

Bold (Strong Emphasis)

Harlowe

`**Bold** or ''Bold''`

SugarCube

`''Bold''`

Snowman

`**Bold**`

Chapbook

`__Bold__` or `**Bold**`

Italics (Emphasis)

Harlowe

`*Italics*` or `//Italics//`

SugarCube

`//Italics//`

Snowman

`*Italics*`

Chapbook

`*Italics*` or `_Italics_`

Verbatim Text

Harlowe

`Verbatim`

SugarCube

`"""Verbatim"""` or `<nowiki>Verbatim</nowiki>`

Snowman

Any text following a new line and four spaces or a tab until the end of the line.

Chapbook

Any text following a new line and four spaces or a tab until the end of the line.

Where are my stories saved?

Twine 2

Online

Stories are saved in the local storage of a web browser. This is isolated between a browser and its use of incognito or private windows. *Clearing sessions and cookies in a browser may also clear the storage of Twine 2.* Using a different browser also means accessing different local storage.

Desktop

When used as a desktop application, Twine 2 stores its files under the current user's files. The current collection of Stories can be accessed through the View -> Show Story Library menu option.

Twine 1

As a desktop application, it stores its files in either HTML or as Twee source code on the local computer. It can import and export both HTML and Twee source code (`.twee` or `.tw`).

What do I need to know?

Nothing!

You can make a game, interactive project, or experimental essay without knowing programming or anything other than how to navigate a program and publish HTML using Twine 2!

However, getting the most out of Twine 2 can often require some knowledge of HTML, CSS, or JavaScript. All story formats also use markup to style text.

HTML? CSS? JavaScript?

This Cookbook has entries on HTML, CSS, and JavaScript as they are used in connection with Twine 2.

Markup?

All built-in story formats use different ways of marking up text to add some visual *emphasize* or otherwise **style text**. These differ between story formats, and it is highly recommended to consult the individual documentation of the story format for more information and examples.

Using the Twine Cookbook

The *Twine Cookbook* provides live examples, Twee source code, and links to download either on each example page under the different topic areas.

Downloading and Using the Live Examples

1) Download the compiled HTML from an example page.

2) Import the file into Twine

- Twine 2: Use the "Import From File" link on the right-hand side under the "+Story" button.
- Twine 1: Use File->Import and select "Compiled HTML File...".

Downloading and Using Twee Code

Twee source code is provided for all examples to more easily show the passages and what they contain.

Twine 2

Twine 2 does not natively understand Twee source code.

Twine 1

For Twine 1 examples, the Twee source code can be downloaded and imported through going to File->Import and selecting "Twee Source Code...".

Example Formatting

Note: Some examples for specific versions (like those designed for Twine 1.4.2) contain this warning.
Note: Other examples contain additional important information.

Summary

Each example is divided into sections. They start with a summary of the example and a short description of what functionality is used and why. If specific macros or functions are mentioned, they should be *emphasized* and link to their appropriate documentation where possible.

For terms that may not be familiar to a more general audience, they can be added to the Glossary file and their definitions will be available as a tooltip over the term or phrase in the HTML version of the book.

Live Example

The next section demonstrates the code running.

```
<section>
<iframe src="storyformat_recipe_example.html" height=400 wi

Download: <a href="storyformat_recipe_example.html" target=
</section>
```

Twee Code

Finally, in the last section, all code is shown in its Twee notation for reference and download.

```
Twee notation
Download: <a href="storyformat_recipe_twee.txt" target="_bl
```

See Also

Some examples use, reference, or share functionality. In this space, some examples link to others in order to better help bridge connections between different ideas and techniques.

Style Guide

The Twine Cookbook uses certain conventions to indicate text should be considered macros, part of JavaScript, or values to be used with either.

HTML and Macros

All HTML and macro names (such as those used in Chapbook, Harlowe, and SugarCube) are escaped. They will appear like this `<div>` for HTML and this `<<include>>` for macros. This is done for easily allowing users to copy and paste the text from the Cookbook into Twine 2.

Variables and JavaScript Functions

All variables are highlighted with *emphasis*. If a story format provides or an example uses particular JavaScript functionality, it will appear with **strong emphasis**.

Quotation Marks

In some cases, the value of a variable is shown in quotation marks. This is to help uses see which values are used and how they might be transformed or inform certain functionality.

Twine 2 Editor

The Twine 2 editor has two main modes.

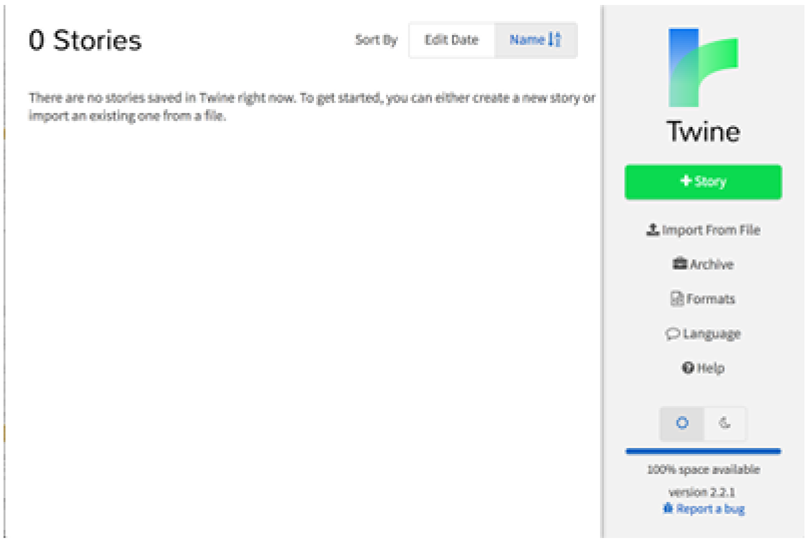
Story Listing

Showing the current stories stories in the browser (online) or on the computer (desktop application), the Story Listing mode is the first encountered in Twine.

Passages View

When opening and changing content in a story, the Passages View shows the passages, any connections between them, and allows for testing and playing the story.

Using the Editor: Story Listing

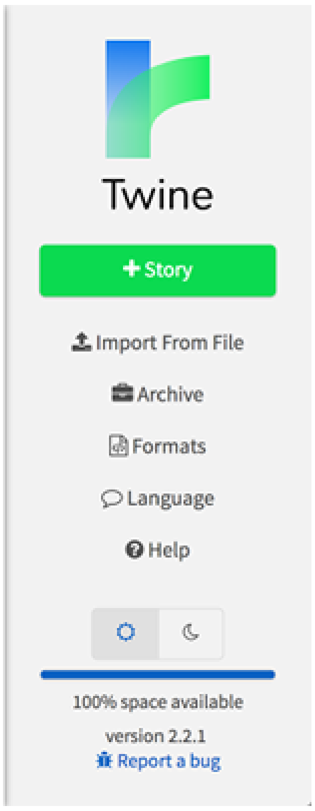


There are no stories saved in Twine right now. To get started, you can either create a new story or import an existing one from a file.

Story Listing

The very first screen upon opening Twine 2 in the desktop or online version is the story view or story listing. This shows all of the stories loaded in this version that can be opened and edited.

Side Menu

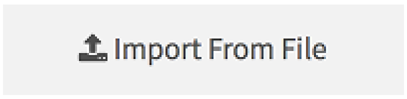


On the right-hand side is a menu for accessing different options in the editor.

+Story

Using the "+Story" allows the user to name and create a new story that will be added to the story listing.

Import From File

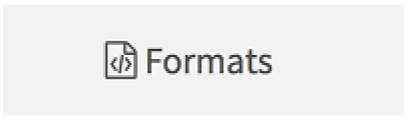


Using the "Import from File" functionality opens a dialog window to select a file and import it into the story listing. All files published with Twine 1 (starting with 1.4.2) are compatible and can be imported. However, major versions of the editor can only publish files matching its number.

Archive

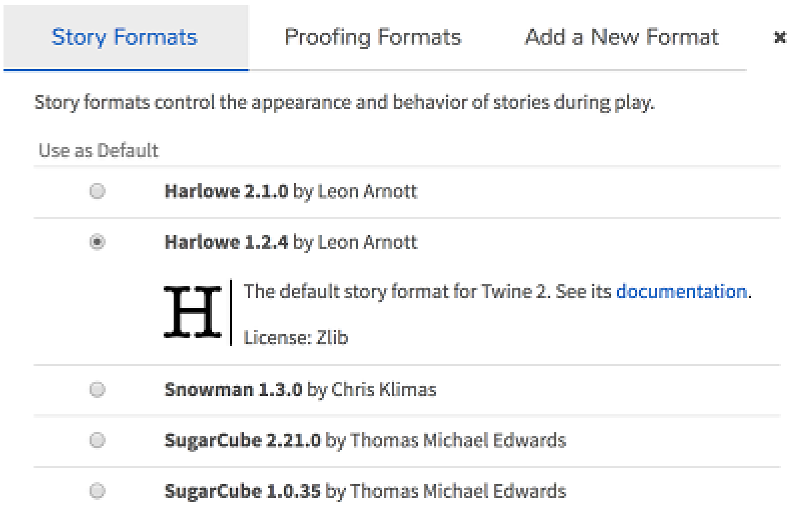
Using the "Archive" button will produce a ZIP'd file containing all of the stories currently listed in the view.

Formats



"Formats" allows for adding to the current listing or changing the current default story format when creating new stories.

Story Formats



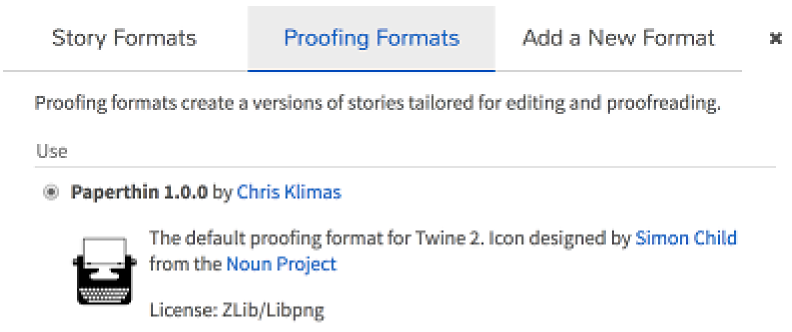
Story formats control the appearance and behavior of stories during play.

Use as Default

- ○ **Harlowe 2.1.0** by Leon Arnott
- ● **Harlowe 1.2.4** by Leon Arnott
  The default story format for Twine 2. See its documentation.
  License: Zlib
- ○ **Snowman 1.3.0** by Chris Klimas
- ○ **SugarCube 2.21.0** by Thomas Michael Edwards
- ○ **SugarCube 1.0.35** by Thomas Michael Edwards

Opened by default, this tab shows all of the current story formats loaded in Twine 2.

Proofing Formats



Proofing formats create a versions of stories tailored for editing and proofreading.

Use

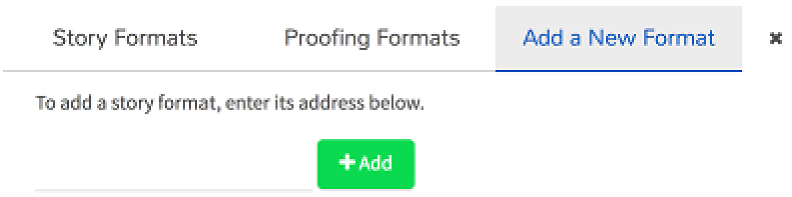
● **Paperthin 1.0.0** by Chris Klimas

The default proofing format for Twine 2. Icon designed by Simon Child from the Noun Project

License: ZLib/Libpng

A listing of all of the current Proofing Formats loaded in Twine 2 as well as which one (if multiple are loaded) is the default for creating a "Proof Copy" of a story.

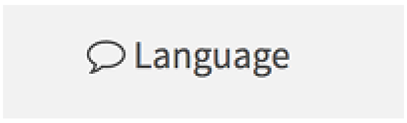
Add a New Format



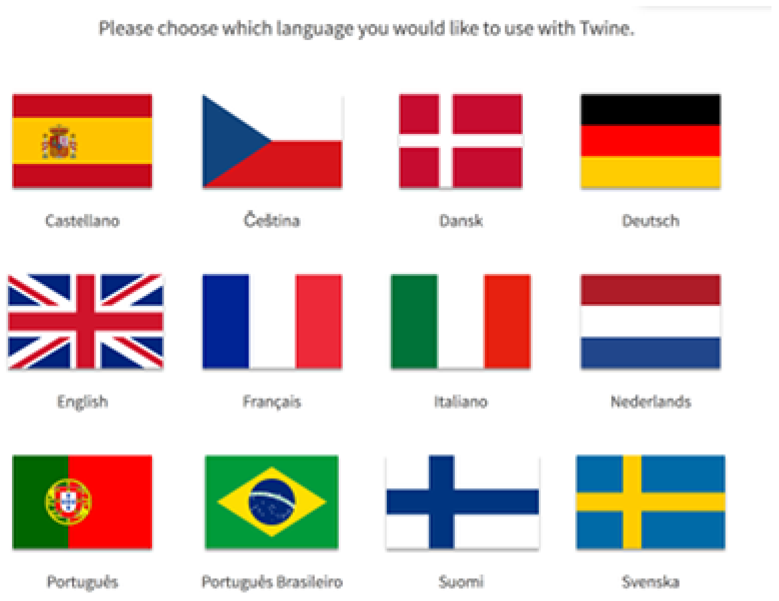
To add a story format, enter its address below.

Through pasting or typing out the URL of a new story format and using the "Add" button, other story formats can be loaded and used. Depending on if the loaded format was story or proof, it will appear as an option on those tabs once loaded and ready for use.

Language



Twine 2 supports over a dozen languages for named entries in its user interface.

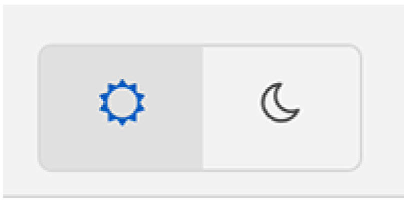

Please choose which language you would like to use with Twine.

Selecting one of them will change the current language of the editor and its menus.

Help

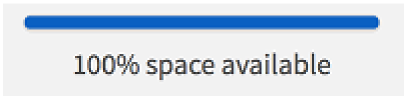
Clicking on "Help" opens a new tab or window in the default browser on the "Twine 2 Guide" of the wiki.

Theme Options



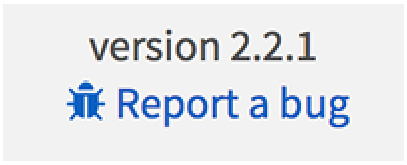
Twine 2 comes with two themes: dark and light. Clicking on either icon switches the interface to that theme.

Current Storage (Online-version only)



When used online, the editor will keep track of how much local storage usage remains for creating and saving stories in Twine 2. Depending on the browser, system, and other settings, this amount can be different.

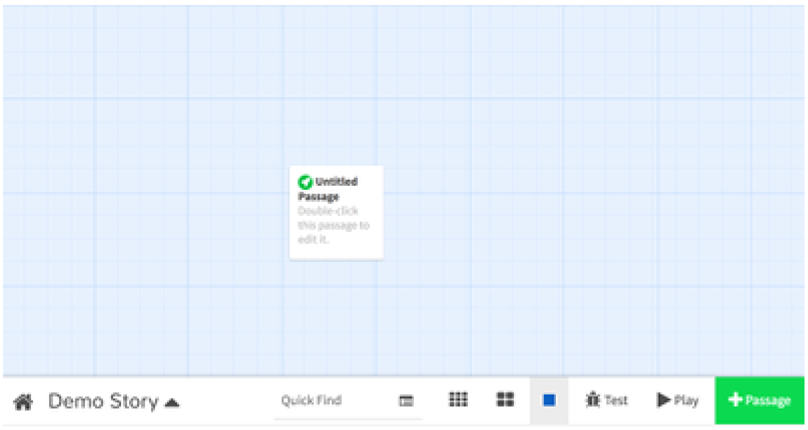
Version Information and Bug Reporting



At the bottom of the menu is the current version of Twine 2 and a link to the Issues page of the GitHub repository.

Using the Editor: Passages View

Passages View



At the bottom of the Passages View is a menu with access to different functionality.

Return to Story Listing

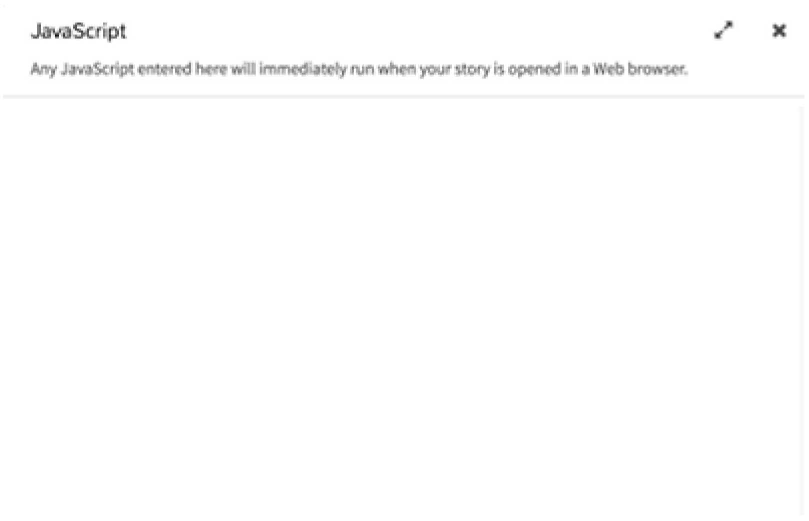
Return to the Story List.

Story Menu

Demo Story ▲

The Story Menu gives access to different functionality about the story itself.

Edit Story JavaScript

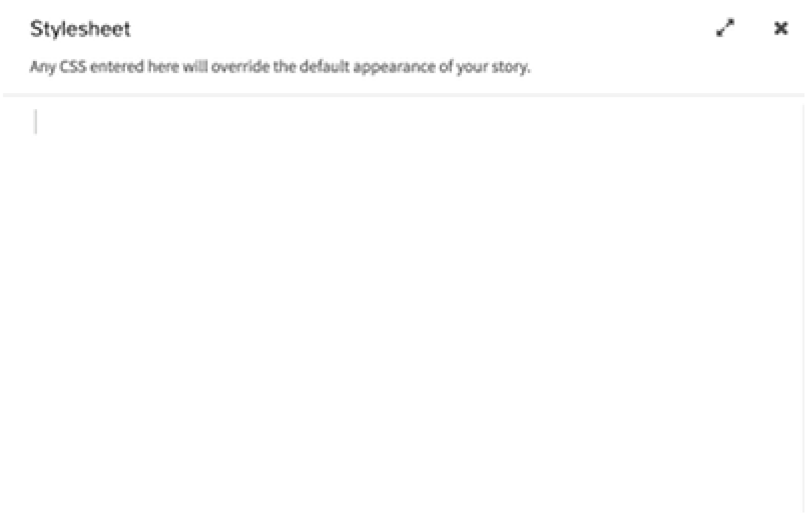


JavaScript

Any JavaScript entered here will immediately run when your story is opened in a Web browser.

The Edit Story JavaScript screen allows for adding or changing JavaScript code that will be included in the story when run or published in a HTML file.

Edit Story Stylesheet

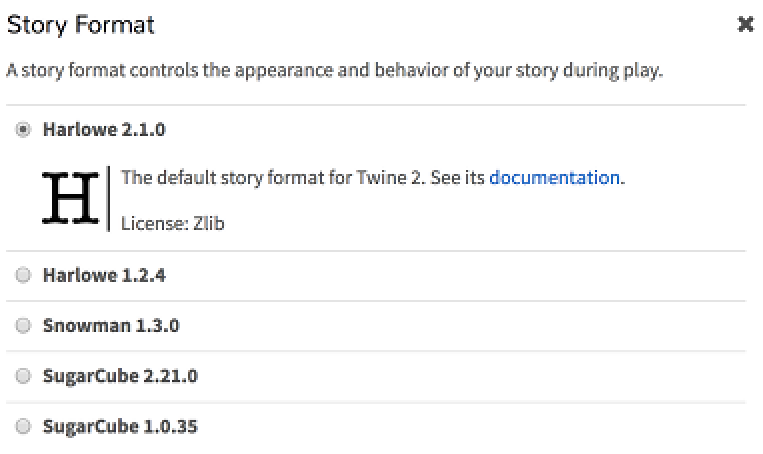


Stylesheet

Any CSS entered here will override the default appearance of your story.

The Edit Story Stylesheet screen allows for adding or changing CSS rules that will be included in the story when run or published in a HTML file.

Change Story Format



Story Format

A story format controls the appearance and behavior of your story during play.

[✓ Harlowe 2.1.0]

The default story format for Twine 2. See its documentation.

License: Zlib

Harlowe 1.2.4

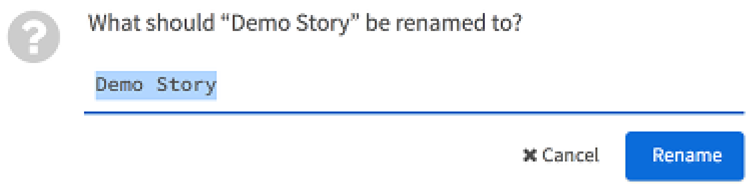
Snowman 1.3.0

SugarCube 2.21.0

SugarCube 1.0.35

The Story Format screens allows for changing the story format in use when running and publishing the story in HTML.

Rename Story



What should "Demo Story" be renamed to?

Demo Story

The Rename Story option allows for changing the name of the story.

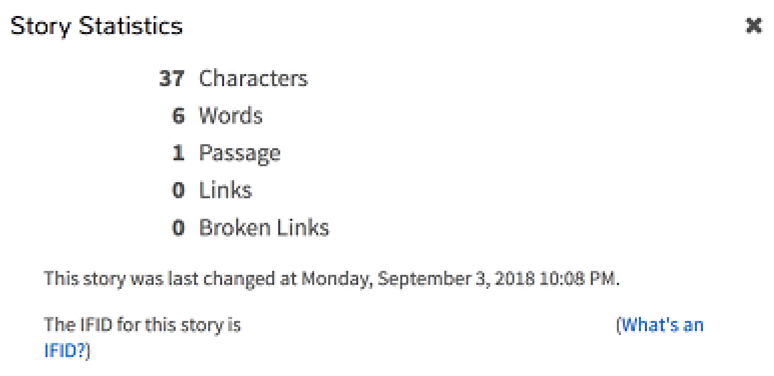
Select All Passages

The Select All Passages places all passages into a single selection for moving or rearranging them as a group

Snap to Grid

The Snap to Grid option turns off or on if passages should move to the closest grid position when moved.

Story Statistics


Story Statistics

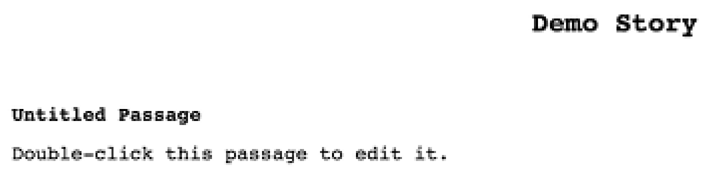
37 Characters
6 Words
1 Passage
0 Links
0 Broken Links

This story was last changed at Monday, September 3, 2018 10:08 PM.

The IFID for this story is (What's an IFID?)

The Story Statistics screen shows data on the story including the number of characters and words.

View Proofing Copy


Demo Story

Untitled Passage

Double-click this passage to edit it.

The View Proofing Copy of a story is the text of all passages except for the Story JavaScript and Story Stylesheet.

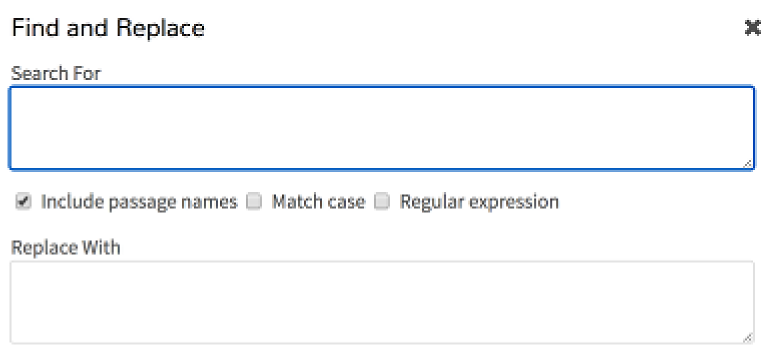
Publish to File

The Publish to File option compiles the current story and any Story JavaScript and Story Stylesheet code into a single HTML file.

Quick Find

The Quick Find functionality is used for searching for words or longer phrases across all passages.

Find and Replace


Find and Replace

Search For

☑ Include passage names ☐ Match case ☐ Regular expression

Replace With

The Find and Replace functionality search for words, phrases, or certain expresses for the purpose of replacing them, if found, with other words or longer phrases.

Story Stucture

The Story Structure view shows the connection between passages only.

Passage Titles

The Passage Tiles view shows simply that: passage titles.

Titles and Excerpts

The Titles and Excerpts view shows passages, their contents, and excerpts of their contents.

Play Story in Test Mode

The "Play Story in Text Mode" button starts turns on debugging functionality and starts the story.

Play Story

The "Play Story" button starts the story in a new tab or window.

Create New Passage

The "Create New Passage" button creates a new, unconnected passage in the Passage View.

Passages

Working in Twine is working with passages. They are foundational to how Twine works, and also what is shown to users when they view a Twine story.

A passage is a way of thinking about different parts of a story. They can be rooms in a house, different time periods, or compartments for storage. The connections between them are made by the author or as part of the playing experience by the reader.

Links

Generally, the action of clicking a link in Twine is a movement between passages. In interactive fiction terminology, this can be thought of as a turn. The act of clicking the link signaled that a choice was made and it is now time to progress to the next content or set of choices.

Content AND Code

While passages can be thought of as content sections, they are also where code goes to change how text appears and how the passage should respond to the user. Writing prose and code both happen within passages. They are not separated in Twine: writing in a passage has the potential to be either or both at the same time.

Story Formats

While using passages are a part of all Twine authoring, there are different collections of rules and styling options called story formats.

In Twine 2, they are accessed as part of the Story Menu. Clicking on Change Story Format opens up the options for picking different story formats and, in most cases, also have links to their externally-hosted documentation.

What are story formats?

In Twine 1, story formats were different visual layouts. Instead of content appearing one way, it could be changed using a different story format. That changed with Twine 2.

Now, story formats are more like dialects of the overall language of Twine along with their own visual changes. Some, like Harlowe, are much more designed for beginners while another, Snowman, is only recommended for more advanced users who want to write they own functionality.

What are the differences between story formats?

There are many. Harlowe and Chapbook were created to be more user-friendly in many ways. However, this also means that more advanced functionality is much harder or nearly impossible. SugarCube follows many of the patterns started with Twine 1 with a large, expanded set of functionality, but also expects some knowledge on the part of the author. Snowman comes with very little built-in functionality and expects the author to write or otherwise supply their own.

Macros

The main differences between story formats come in how they handle macros. SugarCube, for example, supplies a large number of macros for doing many different things. Harlowe has less macros for authors to use, but is also aimed at a different, more general audience.

Chapbook also uses macros but calls them inserts and modifiers.

Style Markup

The rules for styling text are often very different between story formats. The "Style Markup" examples show a great breakdown of the different options per story format (Chapbook, Harlowe, SugarCube, and Snowman).

What is Twine?

Twine is an open-source tool for telling interactive, nonlinear stories. Navigation works by clicking (or, on mobile devices, tapping) links to change old, refresh current, or even load new content.

What can you make with Twine?

Anything! Because Twine produces HTML that web browsers can read, the limitations on Twine are not in what can be developed with the tool, but in the web browsers that run it. Anything that can be done in a web browser can be done in Twine.

Why a cookbook?

The idea for a *Twine Cookbook* was heavily inspired by the *Inform Recipe Book*, a collection of examples to learn Inform. Compiling some of the most requested code solutions across multiple versions, histories, and even years of development, the original editorial team sought to create the same project for Twine. After publishing the first version in August 2017, the *Twine Cookbook* was born.

What this is

The *Twine Cookbook* is a living document. It has rolling deadlines and is often updated multiple times a year as new requests and solutions to old problems are found and submitted. It is driven by the Twine community and finds inspiration in what others create and share online.

Stories

Anything made using Twine can be called by any name. They are no rules on naming conventions and everything from experimental games to more traditional novels can be created in Twine. Everything is welcome. In general, the Twine editor calls individual projects Stories.

Stories can be published to HTML and are readable in a web browser without Twine. In their published form, they can also be imported into Twine for further editing.

IFID

When created, each Story is each given a series of letters and numbers called an Interactive Fiction IDentifier (IFID).

The IFID is always retained when importing or publishing Stories. This helps authors track their projects on different platforms, or if other authors have copied or tried to claim it as their own without their knowledge or consent.

Passages

Passages can be thought of as divisions of time, space, or combinations of the two. They can also be thought of as blocks of dialogue, sections of code, or simply ways to break up a complicated project into more easily understood parts. In Twine, passages are at the core of any story.

Connecting Passages

The simplest way to connect passages is through adding two opening and closing brackets around any collection of letters, numbers, or punctuation. If a passage exists with those exact characters in the same ordering and combination, the passages will be linked together. When viewing the compiled HTML version, there will now be a link to navigate between the two.

By default, passage links are one-way. Navigating the link means moving away from one to another.

Link to Another Passage named "Link to another passage":

```
[[Link to another passage]]
```

Link to Another Passage Named "Different Passage":

The pipe, "`|`", can be used to rename a link from its original to some other name for the same purpose.

```
[[Link to another passage|Different Passage]]
```

Link to Another Passage Named "Different Passage":

Starting in Twine 2, to route links, arrows, "->" or "<-", can be used to *point* to the destination of the link.

```
[[Link to another passage->Different Passage]]
```

```
[[Different Passage<-Link to another passage]]
```

Variables

In programming terminology, a variable is a container for a value that can change. In Twine, a variable is a way of storing and acting on data of some sort. Anything from a number to a series of characters can be stored in a variable. Unlike other code or text in a Passage, variables most commonly start with either the dollar sign ($) or the underscore (`_`) in the Harlowe and SugarCube story formats. (In Chapbook, variables are part of a 'vars section'.)

Story Variables (Harlowe and SugarCube)

Once created, story variables in Twine can be accessed from any passage at any time. They are *globally* accessible to all functionality everywhere.

Example:

```
$numberVariable
```

Variables are translated into their values when used by themselves in a Passage. To display their value, they can simply be included as part of any other text.

Example:

```
The value of the variable is $numberVariable.
```

Temporary Variables (Harlowe and SugarCube)

It can often be useful to work with values in a more controlled manner. For this purpose, temporary variables can be used. They are *locally* accessible. They only exist while the current passage is shown. They start with an underscore (`_`).

Example:

```
_tempVariable
```

Temporary variables can also be used to display their values with other text like Story Variables.

Example:

```
The value of the variable is _numberVariable.
```

Differences in Chapbook

Chapbook handles variables differently. Instead of variables needing to start with the dollar sign `$` or an underscore, `_`, Chapbook also allows variable names to start with upper or lowercase letters as well.

Example:

```
strength: 18
$dexterity: 7
_constitution: 14
```

Differences in Snowman

Snowman uses JavaScript variables. It provides three global variables: *window.story* (for working with the story), *window.passage* (for working with the current passage), and *s* (as a way to access values across passages).

Example:

```
s.strength = 14;
```

Markdown / Markup

All built-in story formats use a form of markdown or markup.

Markdown?

John Gruber and Aaron Swartz created Markdown in 2004 with the goal to create a way of adding some extra symbols to text to easily convert it into HTML. Since then, it has become very popular with sites like GitHub supporting it for text input.

Snowman and Chapbook use a modified version of Markdown for styling text.

Markup?

Harlowe and SugarCube use what is known as markup. To change the presentation of text, extra symbols are added to create visual effects like *emphasis* and a **stronger emphasis**. Most wiki software and sites like Wikipedia use markup to style text.

What's the difference?

Markdown *is* markup. It is a way of changing the presentation of text through adding extra symbols that have special meaning when other programs read it and convert it into a different format like HTML. The only real difference is that Markdown has a name whereas "markup" is a category of all languages that perform the same actions.

Macros

Macros allow programming code to be intermixed with text shown onscreen. They allow a wide variety of functionality to be added to a story, from changing the appearance of text to reacting to mouse and touch events. Story Formats are often chosen based on the macros they provide and how they can be used together.

Twine 1 and SugarCube

In Twine 1, macros were written with two less-than (`<<`) and two-greater-than signs (`>>`) around code. (SugarCube, as a successor of this form, follows the same syntax.)

Example:

```
<<display "Another Passage">>
```

Harlowe

The Harlowe story format uses a different syntax for macros. They are wrapped in a single open `(` and close parenthesis `)` , and use brackets, `[]` , to indicate which text or sections are associated or acted upon by the macro.

Example:

```
(font: “Arial”)[This text will be in Arial.]
```

Snowman

Snowman does not provide macros in the same sense that SugarCube and Harlowe do, but allow mixing JavaScript code in text with `<%` and `%>` , with `<%=` and `%>` displaying the result on the page.

Examples

```
The chalkboard reads 2 + 2 = <%= 2 + 2 %>.
```

Chapbook

Chapbook provides inserts and modifiers to work with variables and other values. However, any variable testing must be done within the vars section itself.

Examples

```
largeFamily: cousins > 10
--
```

Story Formats

Each story format provides a different visual layout, set of macros, and internal JavaScript functionality.

Harlowe

Harlowe is the default story format in Twine 2. It is designed for ease-of-use and for those using Twine 2 for the first time.

Harlowe Example:

```
(if: $hasKey)[It looks like the key will open the door.]
(else:)[No way forward!]
```

SugarCube

SugarCube continues the traditions of Twine 1 while also expanding the available macros. It has more functionality than Harlowe, but can sometimes require greater knowledge of programming techniques and development patterns for more advanced usage.

SugarCube Example:

```
<<if $hasKey>>
It looks like the key will open the door.
<<else>>
No way forward!
<</if>>
```

Snowman

Snowman is designed to be written with custom JavaScript and CSS. It has no built-in macros, but includes the Underscore.js, Marked, and jQuery JavaScript libraries.

Snowman Example:

```
<% if (s.hasKey) { %>
It looks like the key will open the door.
<% } else { %>
No way forward!
<% } %>
```

Chapbook

Chapbook is a "second-generation" Twine 2 story format that seperates its functionality into "inserts", which cause text to appear, and "modifiers", functionality that affect text in some way.

Chapbook Example

```
[if hasKey]
It looks like the key will open the door.
[else]
No way forward!
```

CSS

Cascading Style Sheets (CSS) is programming language for describing the presentation of HTML elements (*i.e.* the colors, fonts, spacing, and general layout of a web page). "Cascading" means rules move from a parent element to any children. Any specific rules also overrule any general ones.

CSS styles are associated with HTML elements using the element's (tag)name, id, classes, and/or other, possibly custom, attributes. Each built-in story format in Twine 2 uses different, sometimes custom, HTML elements to organize the story and then applies its own CSS rules.

Story Stylesheet

When using Twine, additional CSS rules can be added through the Story Stylesheet screen. This CSS is inserted into the final story and provides an opportunity to override the color and formatting choices expressed in the story format's own stylesheet. (When using Twee, styles can be added using one or more passages tagged "stylesheet".)

Considering the complex nature of CSS cascading, it is always highly recommended to use a story format's own macros where possible to change the presentation or layout of a story.

Common areas involved in story format styling include the full page or window, a sidebar (if present), and the current passage as a sub-area of the page. Often, there is also a mechanism to style passages according to their tags (as assigned in Twine 2). See the CSS Selectors recipes for more details.

JavaScript

The programming language JavaScript is embedded in all modern web browsers and is a foundational part of how Twine works.

Knowledge of JavaScript is not required to create stories using Twine. However, understanding how JavaScript works and the expectations of how things are structured in the language can be helpful when using advanced functionality in SugarCube and when using Snowman.

Story JavaScript

When using Twine, extra functionality can be added through the Story JavaScript screen. This is run before the Story is run and provides an opportunity to write specialized code or include external libraries and files. Some story formats, like SugarCube, provide the ability to translate between JavaScript and macro usage in Twine. Others, like Snowman, expect this to be used when creating more complex projects.

window.setup

Based on the object provided by SugarCube of the same name, this cookbook suggests using or creating a *window.setup* global object when working with Story JavaScript in Twine for greater portability between story formats.

window.setup **Example**

```
window.setup = window.setup || {};
```

What is Twee?

Twee is the source code of a Twine story. In Twine 1, stories could be exported into its source, changed, and imported again. Twine 2 has moved away from this functionality, but has been heavily influenced through having sections (passages in Twine 1) where the user can add CSS (Story Stylesheet) and JavaScript (Story JavaScript).

Notation

Starting with Twee 3, there is a standard for reading and writing Twee when working with Twine 2 passages.

Twee 3 notation is written as a series of four parts for the header of each passage:

- Sigil: (Required) Two colons ("::") followed by a space
- Passage Name: (Required) The name of the passage
- Tags: (Optional) Optional tags
- Metadata: (Optional) Information about the passage

The content of a passage continues until the next header of a passage is found or the input ends with at least a single empty line between passage headers.

Example Twee Notation:

```
:: Snoopy [dog peanuts]
Snoopy is a dog in the comic Peanuts.

:: Charlie Brown [person peanuts] {"position":"600,400","si
Charlie Brown is a person in the comic Peanuts
```

Special Passage Names

Some compilers understand and process certain keywords differently. The following is a common set of case-sensitive reserved passage names across Twine 1 and Twee 3.

Start (Twine 1 and Twee 3)

The first passage.

```
:: Start
A beginning!
```

StoryTitle (Twine 1 and Twee 3)

The title of the story.

```
:: StoryTitle
A Named Story
```

StorySubtitle (Twine 1)

The subtitle of the story.

```
:: StorySubtitle
The subtitle of this story
```

StoryAuthor (Twine 1)

The author of the story.

```
:: StoryAuthor
John Smith
```

StoryMenu (Twine 1)

Corresponds to the menu that hovers in the upper-right corner of the page in the Jonah story format, or on the left side of the page in the Sugarcane story format.

```
:: StoryMenu
Content of the story menu!
```

StorySettings (Twine 1)

Used to specify certain options and settings.

- Undo: enables the player to "undo moves." In Sugarcane, this means being able to use the Back button in the browser. In Jonah, this means being able to use the “Rewind to here” link, and being able to click links in previous passages.
- Bookmark: enables the player to use the “Bookmark” link in Sugarcane and Jonah. On by default.

- Hash updates: this causes the current passage's bookmark URL to be automatically placed in the player's browser address bar whenever they change passages. This is off by default because the URLs can become very long and ugly quickly.
- Prompt before closing: If the player tries to reload or close the page, the browser will prompt for confirmation. This is useful for long games - it would be unfortunate if the player lost a lot of progress due to an idle key-press.
- Don't use default CSS: This removes most of the CSS used by the story format, allowing CSS programmers to write their own stylesheet redesigns more easily. Off by default - but including the text “blank stylesheet” in a stylesheet will set it on automatically.
- ROT13: obfuscates the story's HTML source to dissuade people from spoiling themselves by reading it. Off by default.
- jQuery: include the library or not
- Modernizr: include the the library or not

```
:: StorySettings
undo:on
bookmark:on
hash:on
exitprompt:on
blankcss:on
obfuscate:rot13
jquery:on
modernizr:on
```

StoryIncludes (Twine 1)

Includes, "imports", other local or remote files during the HTML compilation process. In Twine 1.4.2, both Twine Story (.tws) and Twine Source (.twee) files can be used.

```
:: StoryIncludes
localfile.tws
```

StoryData (Twee 3)

A JSON chunk encapsulating various Twine 2-compatible details about the story.

- ifid: (Required) IFID of the story
- format: (Optional) Story format
- format-version: (Optional) Story format version
- start: (Optional) PID of starting passage

- tag-colors: (Optional) Pairs of tags and colors
- zoom: (Optional) Decimal zoom level

```
:: StoryData
{
    "ifid": "D674C58C-DEFA-4F70-B7A2-27742230C0FC",
    "format": "SugarCube",
    "format-version": "2.28.2",
    "start": "My Starting Passage",
    "tag-colors": {
        "bar": "green",
        "foo": "red",
        "qaz": "blue"
    },
    "zoom": 0.25
}
```

Special Tag Names

Twee 3 defines two special case-sensitive lowercase passage tags: `stylesheet` and `script`.

(Passages are also loaded according to alphabetical order if others exist with the same special passage tags.)

Stylesheet

Any additional or overriding CSS rules for the story.

```
:: UserStylesheet[stylesheet]
```

Script:

Any additional or overriding JavaScript code for the story.

```
:: UserScript[script]
```

HTML

The HyperText Markup Language (HTML) is the standard for all documents designed for a web browser. It consists of a series of elements defining its structure and the layout of its content.

Story Format Layout

Each story format handles its own layout and HTML structure. While CSS can be used to style its elements, it is often recommended to use any exisitng macros for this purpose in a story format, if available.

- Harlowe: Named Hooks
- SugarCube: CSS Selectors
- Snowman: HTML Elements
- Chapbook: Customization

Twine 2 HTML

All data of a Twine 2 story is stored as a series of HTML elements within a page according to the HTML Output Specification.

Example:

```
<tw-storydata>
  <style
    role="stylesheet"
    id="twine-user-stylesheet"
    type="text/twine-css">
  </style>
  <script
    role="script"
    id="twine-user-script"
    type="text/twine-javascript">
  </script>
  <tw-tag
    name="tagName"
    color="orange">
  </tw-tag>
  <tw-passagedata
        pid="1"
        name="Start"
        tags="tag1 tag2"
        position="102,99"
        size="100,100">
    Some content
    </tw-passagedata>
</tw-storydata>
```

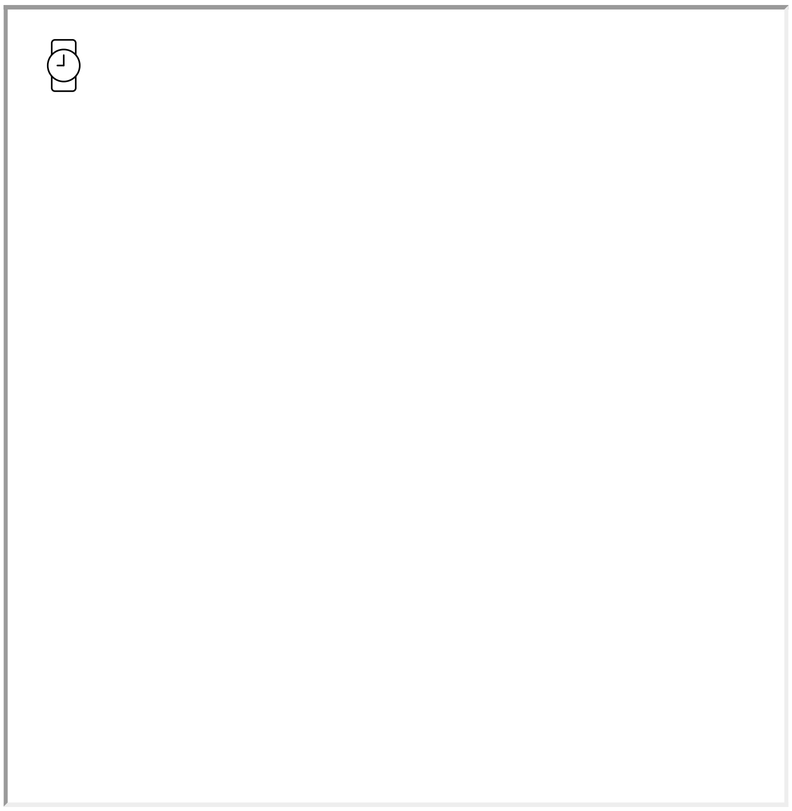
"Adding Functionality": Chapbook (v1.0.0)

Summary

Chapbook allows for creation of custom inserts and modifiers.

The example below adds an insert that displays a ⌚ emoji.

Live Example


⌚

Download: Live Example

Twee Code

```
:: StoryTitle
Adding Functionality

:: UserScript[script]
engine.extend('1.0.0', () => {
    config.template.inserts = [{
        match: /^smiley face$/i,
        render: () => '⍰'
    }, ...config.template.inserts];
});

:: Start
Hello there {smiley face}
```

Download: Twee Code

"Adding Functionality": SugarCube (v2.18)

Summary

In SugarCube, additional functionality can be added through the *Macro.add()* function.

In this example, the *Date()* JavaScript function is used to get the current time. This is saved to *payload.contents*, and the *jQuery.wiki()* function is used to convert and append it to the current passage.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Adding Functionality in SugarCube

:: UserScript[script]
Macro.add("currenttime", {
    tags: null,
    handler: function() {
        // Try the following code and catch any errors
        try {

            // Get the current time and save it to the payl
            this.payload.contents = new Date();

            // Wikify (and append) the current payload cont
            jQuery(this.output).wiki(this.payload.contents)

        }
        catch (ex) {
            // Return any errors
            return this.error("Error: " + ex.message);
        }
    }
});

:: Start
<<currenttime>><</currenttime>>
```

Download: Twee Code

"Adding Functionality": Snowman (v1.3.0)

Summary

Snowman does not provide macros. However, additional functionality can be added through the use of the Underscore.js JavaScript library provided with Snowman.

In this example, a global function, *showCurrentTime()*, is added to the *window.setup* object. It is called in a passage through using the interpolation functionality of Underscore's template system to show a value.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Adding Functionality in Snowman

:: UserScript[script]
// Use or create window.setup
window.setup = window.setup || {};

// Create global function
window.setup.showCurrentTime = function() {
    return new Date();
}

:: Start
The current time is <%= setup.showCurrentTime() %>
```

Download: Twee Code

"Arrays": Chapbook (1.0.0)

Summary

Using the Vars Section, variables can be set using any JavaScript values, such as arrays. However, Chapbook expressions will only show certain types of values.

While Chapbook cannot show an array or its value by position, a new variable can be set and then shown. This following example shows how to do this approach.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Arrays for Chapbook

:: Start
arrayExample: [13, 15]
exampleValue: arrayExample[0]
--
Chapbook can't display indexed array values currently. Howe

Here is an {exampleValue}.
```

Download: Twee Code

"Arrays": Harlowe (v2.1)

Summary

Arrays are a collection of values. Each entry in an array is assigned an *index*, which is a number that corresponds to its position in the array. In Harlowe, and unlike in JavaScript, arrays are one-based, meaning the first element in the array is given the index "1". Arrays can be created using the `(a:)` or `(array:)` macro and assigning a variable to it: `(set: $myArray to (a:))` .

Specific elements in an array can be accessed by following its variable name with a possessive `'s` and an ordinal number referencing the index to check, (`$myArray's 2nd`); the final entry, `$myArray's last` , points to the final element. Its contents can be tested using the `contains` operator (e.g. `(if: $myArray contains 'something') [...]`), add new items using the `+` operator (e.g. `(set: $myArray to + (a: 'something'))`), and remove items using the `-` operator. All elements in an array can be passed to macros as separate arguments with the spread operator (`...`).

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Arrays in Harlowe

:: init [startup]
<!-- it is always a good idea to initialize your variables,
(set: $inventory to (a:))
(set: $chest to (a: 'a shield', 'a suit of armor'))
(set: $chestOpen to false)

:: inventory [header]
You are currently carrying:
<!-- if the inventory contains nothing, show "nothing" -->\
(if: $inventory's length is 0)[\
    nothing.
](else:)[\
    <!-- we iterate over the array and print each item -->\
    (for: each _item, ...$inventory)[\
        _item (unless: $inventory's last is _item)[, ]\
    ].
]
-----



:: Start
<!-- we use the + operator and wrap the target elements in
You find yourself inside a small room. In the corner, you s

(set: $inventory to it + (a: 'a sword'))\
[[Continue|hallway]]

:: hallway
You see a chest here in the hallway.  \
(unless: $chestOpen)[\
    Do you want to open it?

    {
    (link: 'Open the chest.')[
        <!-- adding to arrays together can also be done wit
        (set: $inventory to it + $chest)
        (set: $chestOpen to true)
        (goto: 'chest')
    ]
    }
](else:)[\
    It's open, and there's nothing inside.
]\

[[Move on.|dart trap]]
```

```
:: chest
You open the chest and find (for: each _item, ...$chest)[\
    _item (unless: $chest's last is _item)[ and ]\
].

(link: 'Okay')[ (goto: (history:)'s last) ]

:: dart trap
Several darts shoot out of a wall at you!
<!-- we can check to see if the player has a given item wit
(if: $inventory contains 'a shield')[\
    Luckily, your shield will protect you.
](else:)[\
    With no way to defend yourself, you die.
]
```

Download: Twee Code

"Arrays": SugarCube (v2.18)

Summary

Arrays are a collection of values. Each value in an array is assigned an *index*, which is a number that corresponds to the position of that item or element. Arrays have many built-in methods and other features, and SugarCube adds many more. Arrays can be created by assigning a variable to the array literal, which is a pair of brackets (`[]`): `<<set $myArray to []>>` .

Specific elements can be accessed in an array by following its variable name with a pair of brackets containing the index to check. Testing whether an array contains an element can be done using the **Array#includes()** function; adding new items can be done using the **Array#push()** function.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Arrays in SugarCube

:: StoryInit
/% it is always a good idea to initialize your variables, b
<<set $inventory to []>>
<<set $chest to ['a shield', 'a suit of armor']>>
<<set $chestOpen to false>>

:: PassageHeader
You are currently carrying:
/% if the inventory contains nothing, show "nothing" %/\
<<if $inventory.length is 0>>\
    nothing.
<<else>>\
    /% the Array#join() method combines all array elements 
    <<= $inventory.join(', ')>>.
<</if>>
-----



:: Start
/% we use the Array#push() method to add new items to our i
You find yourself inside a small room. In the corner, you s

<<run $inventory.push('a sword')>>\
[[Continue|hallway]]

:: hallway
You see a chest here in the hallway.  \
<<if not $chestOpen>>\
    Do you want to open it?

    <<link [[Open the chest.|chest]]>>
        /% concatenating the arrays and setting the result 
        <<set $inventory to $inventory.concat($chest)>>
        <<set $chestOpen to true>>
    <</link>>
<<else>>\
    It's open, and there's nothing inside.
<</if>>

[[Move on.|dart trap]]

:: chest
You open the chest and find <<= $chest.join(' and ')>>.

[[Okay.|previous()]]
```

```
:: dart trap
Several darts shoot out of a wall at you!
/% we can check to see if the player has a given item with
<<if $inventory.includes('a shield')>>\
    Luckily, your shield will protect you.
<<else>>\
    With no way to defend yourself, you die.
<</if>>
```

Download: Twee Code

"Arrays": Snowman (v1.3)

Summary

Arrays are a collection of values. Each value in an array is assigned an *index*, which is a number that corresponds to the position of that item or element in the array. In JavaScript, arrays are *zero-based*, meaning the first element in the array is given the index "0". Arrays have many built-in methods and other features for your use. You can create an array by assigning a variable to the array literal, which is a pair of brackets (`[]`): `<% s.myArray = [] %>` .

Specific elements can be accessed in an array by following its variable name with a pair of brackets containing the index to check. Testing whether an array contains an element can be done using the **Array#includes()** function; adding new items can be doone using the **Array#push()** function.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Arrays in Snowman

:: UserScript [script]
(function () {
    var s = window.story.state;
    s.inventory = [];
    s.chest = ['a shield', 'a suit of armor'];
    s.chestOpen = false;
}());

:: Header
You are currently carrying:
<% if (s.inventory.length === 0) { %>
nothing.
<% } else { %>
<%= s.inventory.join(', ') + '.' %>
<% } %>

:: Start
<%= window.story.render("Header") %><hr />

You find yourself inside a small room. In the corner, you s

<% s.inventory.push('a sword') %>
[[Continue|hallway]]

:: hallway
<%= window.story.render("Header") %><hr />
You see a chest here in the hallway.
<% if (!s.chestOpen) { %>
Do you want to open it?

[[Open the chest.|chest]]
<% } else { %>
It's open, and there's nothing inside.
<% } %>

[[Move on.|dart trap]]

:: chest
<% s.inventory = s.inventory.concat(s.chest) %>
<% s.chestOpen = true %>
<%= window.story.render("Header") %><hr />

You open the chest and find <%= s.chest.join(' and ') %>.

[[Okay.|hallway]]
```

```
:: dart trap
<%= window.story.render("Header") %><hr />
Several darts shoot out of a wall at you!
<% if (s.inventory.includes('a shield')) { %>

Luckily, your shield will protect you.
<% } else { %>

With no way to defend yourself, you die.
<% } %>
```

Download: Twee Code

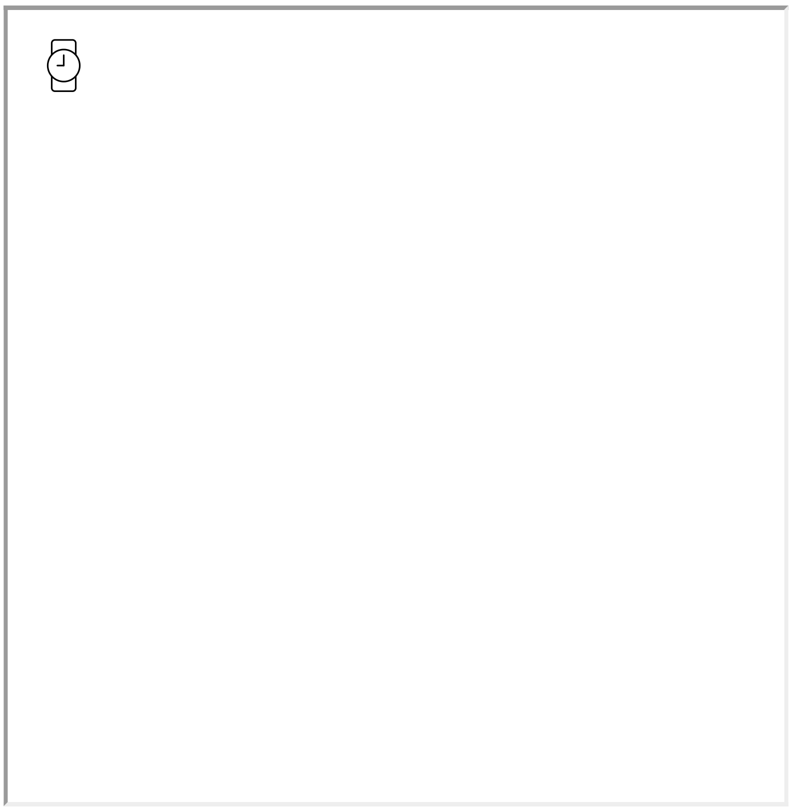
"Audio": Chapbook (v1.0.0)

Summary

Chapbook supports both looping sounds (which it calls ambient) and one-off sound (which it calls effects). It only allows playing one sound at a time.

Note: This examples uses two additional files, testpattern.ogg and testpattern.wav. Both files need to be downloaded and placed in the same folder as the HTML file in order to work as designed.

Live Example



Download: Live Example

Twee Code

```
:: Start
sound.ambient.test.url: 'testpattern.ogg'
sound.ambient.test.description: 'An audio test pattern'
--

> [[Play sound]]
> [[Stop the sound]]

:: Play sound
{ambient sound: 'test'}

[[Return->Start]]

:: Stop the sound
sound.ambient.test.playing: false
--

[[Return->Start]]
```

Download: Twee Code

"Audio": Harlowe (v2.0)

Summary

Harlowe does not have direct macro support for audio resources. However, additional JavaScript can be added to work with audio elements within a story.

Audio elements rely on sources either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the resource in relation to the webpage. Because audio files are external resources, they must also be accessed from a remote service, file hosting location, or stored separately with the webpage.

Due to browser differences in licensing, some audio formats are not universally supported. For best results in using audio in Twine, it is recommended to use multiple formats, allowing the browser to choose which one is best supported when first loaded.

Note: This examples uses two additional files, testpattern.ogg and testpattern.wav. Both files need to be downloaded and placed in the same folder as the HTML file in order to work as designed.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Audio in Harlowe

:: Start
<audio controls>
  <source src="testpattern.ogg" type="audio/ogg">
  <source src="testpattern.wav" type="audio/wav">
Your browser does not support the audio element.
</audio>
```

Download: Twee Code

"Audio": SugarCube (v2.18)

Summary

SugarCube supports audio through multiple macros. For basic playing of audio, resources must be first cached through the `<<cacheaudio>>` macro and then can be referenced in others like the `<<audio>>` macro for playing and stopping.

Audio elements rely on sources either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the resource in relation to the webpage. Because audio files are external resources, they must also be accessed from a remote service, file hosting location, or stored separately with the webpage.

Due to browser differences in licensing, some audio formats are not universally supported. For best results in using audio in Twine, it is recommended to use multiple formats, allowing the browser to choose which format is best supported when first loaded.

Note: This examples uses two additional files, testpattern.ogg and testpattern.wav. Both files need to be downloaded and placed in the same folder as the HTML file in order to work as designed.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Audio in SugarCube

:: Start
<<link "Start audio!">>
    <<audio "testpattern" play>>
<</link>>

<<link "Stop audio!">>
    <<audio "testpattern" stop>>
<</link>>

:: StoryInit
<<cacheaudio "testpattern" "testpattern.ogg" "testpattern.w
```

Download: Twee Code

"Audio": Snowman (v1.3.0)

Summary

Snowman does not have direct macro support for audio resources. However, additional JavaScript can be added to work with audio elements within a story.

Audio elements rely on sources either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the resource in relation to the webpage. Because audio files are external resources, they must also be accessed from a remote service, file hosting location, or stored separately with the webpage.

Due to browser differences in licensing, some audio formats are not universally supported. For best results in using audio in Twine, it is recommended to use multiple formats, allowing the browser to choose which one is best supported when first loaded.

Note: This examples uses two additional files, testpattern.ogg and testpattern.wav. Both files need to be downloaded and placed in the same folder as the HTML file in order to work as designed.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Audio in Snowman

:: Start
<audio controls>
  <source src="testpattern.ogg" type="audio/ogg">
  <source src="testpattern.wav" type="audio/wav">
Your browser does not support the audio element.
</audio>
```

Download: Twee Code

"Conditional Statements": Chapbook (v1.0.0)

Summary

The `[if condition]` and `[else]` modifiers found in the Conditional Display section of the Chapbook guide can be used to conditionally display content based on the current value of a variable.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Conditional Statements in Chapbook

:: Start
animal: "horse"
--
[if animal === "dog"]
It's a dog!
[else]
It's a horse!
[continue]
```

Download: Twee Code

See Also

Setting and Showing Variables

"Conditional Statements": Harlowe (v2.0)

Summary

The `(if:)` and `(else:)` macros conditionally produce commands that can be attached to hooks in Harlowe. If the statement is true, the `(if:)` section will be run. Otherwise, the `(else:)` section will be.

The `(unless:)` macro can also be used in place of `(if:)` for the opposite effect. Furthermore, variables can be attached to hooks to control whether they are displayed based on if they are "true" (will be displayed) or "false" (will not be displayed).

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Conditional Statements in Harlowe

:: Start
(set: $animal to "horse")

(if: $animal is "dog")[It's a dog!]
(else:)[It's a horse!]

(unless: $animal is "dog")[It's a horse!]
(else:)[It's a dog!]

(set: $isDog to $animal is "horse")
$isDog[It's a dog!]
```

Download: Twee Code

See Also

Setting and Showing Variables

"Conditional Statements": SugarCube (v2.18)

Summary

In SugarCube, the `<<if>>` and `<<else>>` macros conditionally run sections. If the statement is true, the `<<if>>` section will be run. Otherwise, the `<<else>>` section will be.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Conditional Statements in SugarCube

:: Start
<<set $animal to "horse">>

<<if $animal is "dog">>
It's a dog!
<<else>>
It's a horse!
<</if>>
```

Download: Twee Code

See Also

Setting and Showing Variables

"Conditional Statements": Snowman (v1.3.0)

Summary

Through using the *s* global variable and the built-in Underscore template functionality, JavaScript conditional statements can be run to show content in Snowman.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Conditional Statements in Snowman

:: Start
<%
    s.animal = "horse";
%>

<% if(s.animal == "dog"){ %>
It's a dog!
<% } else { %>
It's a horse!
<% } %>
```

Download: Twee Code

See Also

Setting and Showing Variables

"CSS Selectors": Chapbook (v1.0)

Summary

This example shows how to use CSS selectors to style different areas of the page. In Chapbook, the *backdrop* covers the whole page, the *page* is like a "story" area in other story formats, and the first `<div>` inside the `<article>` is the main passage area.

```
<div id="backdrop">
    <div class="page">
        <header>
            <div class="left"></div>
            <div class="center"></div>
            <div class="right"></div>
        </header>
        <article>
            <div></div>
        </article>
        <footer>
            <div class="left"></div>
            <div class="center"></div>
            <div class="right"></div>
        </footer>
    </div>
</div>
```

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS Selectors in Chapbook

:: UserStylesheet[stylesheet]
#backdrop {
    border: 5px solid green;
}

#page {
    border: 2px solid red;
}

#page article>:first-child {
    border: 1px solid blue;
}

:: Start
The backdrop has a green border; it contains this page (rec

[[Second]]

:: Second
This passage also has a blue border.
```

Download: Twee Code

"CSS Selectors": Harlowe (v2.0)

Note: The following example is designed for use in Harlowe 2.x and later

Summary

This example shows how to use CSS selectors to style different areas of the page. In Harlowe, custom HTML elements are used for layout: the `<tw-story>` element contains the page as well as a an element containing the currently shown passage, `<tw-passage>`, and an element containing the sidebar, `<tw-sidebar>`.

```
<tw-story>
    <tw-passage>
        <tw-sidebar>...</tw-sidebar>
        ...
    </tw-passage>
</tw-story>
```

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS Selectors in Harlowe

:: UserStylesheet [stylesheet]
tw-story {
    border: 5px solid lightgreen;
}

tw-sidebar {
    border: 2px solid blue;
}

tw-passage {
    border: 1px solid red;
}

:: Start
The page has a green border; it contains this passage (red
[[Second]]

:: Second
This passage also has a red border.
```

Download: Twee Code

See Also

See the left sidebar recipe for another example of stylying custom elements like `<tw-sidebar>`. Harlowe also supports styling passages by tag using its custom 'tags' attribute.

"CSS Selectors": SugarCube (v2.18)

Summary

This example shows how to use CSS selectors to style different areas of the page. SugarCube uses nested `<div>` s with ids to structure its significant elements. Most notably the sidebar is a `<div>` with id "ui-bar". However, these elements are often more easily styled by other means, such as selecting the `<body>` element to style the entire page, and the "passage" class to style the current passage.

```
<body>
    <div id="ui-bar">...</div>
    <div id="story">
        <div id="passages">
            <div id="a-passage-name" class="passage">...</c
        </div>
    </div>
</body>
```

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
CSS Selectors in SugarCube

:: UserStylesheet [stylesheet]
body {
    background-color: darkgreen;
}

#ui-bar {
    border: 2px solid blue;
}

.passage {
    border: 1px solid red;
}

:: Start
The page has a green background; it contains this passage (
[[Second]]

:: Second
This passage also has a red border.
```

Download: Twee Code

See Also

See the SugarCube images recipe for an example of using a single *class* CSS selector to style a different element. See styling passages by tag for an example of using two classes to style a single element.

Bleached

Bleached is an alternate, light-colored stylesheet for SugarCube, and a good example of how to change SugarCube's default colors using CSS.

"CSS Selectors": Snowman (v1.3.0)

Summary

This example shows how to use CSS selectors to style different areas of the page. Snowman uses a simple HTML structure of a `<div>` with id "passage" directly below (inside) the `<body>` tag.

```
<body>
    <div id="passage">...</div>
</body>
```

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS Selectors in Snowman

:: UserStylesheet [stylesheet]
body {
    background-color: lightgreen;
}

#passage {
    border: 1px solid red;
}

:: Start
The page has a green background; it contains this passage,

[[Second]]

:: Second
This passage also has a red border.
```

Download: Twee Code

See Also

See the Snowman left sidebar recipe for an example of styling passages using the *id* CSS selector.

"CSS Selectors": Snowman (v2.0.2)

Summary

This example shows how to use CSS selectors to style different areas of the page. Snowman 2.0 uses the elements `<tw-story>` and `<tw-passage>` to show a passage. It also changes from an *id* to a *class* named "passage" from previous versions of Snowman.

```
<body>
    <tw-story>
        <tw-passage class="passage"></tw-passage>
    </tw-story>
</body>
```

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS Selectors in Snowman

:: UserStylesheet [stylesheet]
body {
    background-color: lightgreen;
}

tw-passage {
    border: 1px solid red;
}

:: Start
The page has a green background; it contains this passage,

[[Second]]

:: Second
This passage also has a red border.
```

Download: Twee Code

See Also

See the Snowman left sidebar recipe for an example of styling passages using the *id* CSS selector.

"CSS and Passage Tags": Chapbook (v1.0.0)

Summary

Using the built-in global variable *config*, text colors and other CSS properties can be programmatically adjusted using the `[JavaScript]` modifer in Chapbook.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: CSS and Passage Tags

:: Start
config.style.page.color: "yellow-4"
--
This is one color!
[JavaScript]
// Overwrite the default text color
config.style.page.color = "violet-6";
[continued]

And this is the same color!
```

Download: Twee Code

"CSS and Passage Tags": Harlowe (v2.0)

Note: The following example is designed for use in Harlowe 2.x and later

Summary

This example shows how to use CSS selectors to selectively style different passages based on how they are tagged. In Harlowe, the "tags" attribute can be used to create different sets of styles and apply them when shown.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS and Passage Tags in Harlowe

:: UserStylesheet[stylesheet]
[tags="grey"] {
    background: grey;
}

[tags="yellow"] {
    background: yellow;
      color: black;
}

:: Start[grey]
This passage has a grey background and (default) white text
[[Second]]

:: Second[yellow]
This passage has a yellow background and black text.
```

Download: Twee Code

"CSS and Passage Tags": SugarCube (v2.18)

Summary

This example shows how to use CSS selectors to style different passages based on how they are tagged. In SugarCube, the tag name is applied to both the *body* and the passage shown. To style different parts, use either the "body" selector or a combination of "passage" and the tag name.

SugarCube 2.x and later also applies a "data-tags" attribute to the `<html>` , `<body>` , and elements with the class ".passage". These can also be used to style the page at different levels.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
CSS and Passage Tags in SugarCube

:: UserStylesheet[stylesheet]
/* Style the entire body when showing passage tagged with g
body.grey {
  color: green;
}

/* Style only the inner part of the passage tagged with "gr
.passage.grey {
  background: grey;
}

/* Style only the inner part of the passage tagged with "ye
.passage.yellow {
  background: yellow;
  color: black;
}

:: Start[grey]
This passage has a grey background and green text.
[[Second]]

:: Second[yellow]
This passage has a yellow background and black text.
```

Download: Twee Code

"CSS and Passage Tags": Snowman (v1.3)

Summary

This example shows how to use CSS selectors to style different passages. Snowman does not support tags in passages. However, the effect can be implemented through using jQuery and the **toggleClass()** function to switch between different pre-defined classes.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
CSS and Passage Tags in Snowman

:: UserStylesheet[stylesheet]
.grey {
    background: grey;
      color: white;
}
.yellow {
    background: yellow;
      color: black;
}

:: Start[grey]
<% $("body").toggleClass("grey") %>
This passage has a grey background and white text.

[[Second]]

:: Second[yellow]
<% $("body").toggleClass("yellow") %>
This passage has a yellow background and black text.
```

Download: Twee Code

"CSS and Passage Tags": Sugarcane (v1.4.2)

Note: The following example is designed for Twine 1.4.2.

Summary

This example shows how to use CSS selectors to selectively style different passages based on how they are tagged. In Sugarcane, the "data-tags" attribute can be used to create different sets of styles and apply them when shown.

Live Example

Download: Live Example

Twee Code

```
:: Start [grey]
This passage has a grey background and white text.

[[Second]]


:: Second [yellow]
This passage has a yellow background and black text.


:: StoryTitle
CSS and Passage Tags in Twine 1.4.2


:: Stylesheet [stylesheet]
[data-tags="grey"] {
    background: grey;
    color: white;
}

[data-tags="yellow"] {
    background: yellow;
    color: black;
}


:: StoryAuthor
Videlais
```

Download: Twee Code

"Cycling Choices": Chapbook (v1.0.0)

Summary

Chapbook provides the `{cycling link}` modifier for creating a cycling link effect.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Cycling Links

:: Start
This cycling link example remembers the choice made:
{cycling link for: 'hair', choices: ["Black", "Brown", "Blo

This cycling-link example will disappear (show empty string
{cycling link for: 'breakfast', choices: ["Two eggs", "One 
```

Download: Twee Code

"Cycling Choices": Harlowe (v2.0)

> **Note:** This example is affected by history changes in the story. Undoing or re-doing back to a passage containing this recipe has the potential to change its saved values.

Summary

"Cycling Choices" demostrates how to create a 'cycling' effect of different choices through clicking on them.

The cycle starts with the use of the `(display:)` macro and the assumption that the 1st element in the *$choices* Array is the currently selected choice.

If the user clicks on the link (created through using the `(link:)` macro), the *$choices* array is updated using the `(rotated:)` macro. This causes the current 1st element to be moved to the end of the array, making the element that was previously 2nd to now be 1st.

At the end of every cycle, the currently selected value is always the 1st element in the *$choices* array.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Cycling Choices in Harlowe

:: Start
(set: $choices to (array: "First", "Second", "Third"))
Click options to cycle: [(display: "Cycling")]<choice|
[[Submit|Results]]

:: Cycling
{
    (link: (text: $choices's 1st) )[
        (set: $choices to (rotated: -1, ...$choices))
        (replace: ?choice)[(display: "Cycling")]
    ]
}

:: Results
Selected choice: (print: $choices's 1st)
```

Download: Twee Code

See Also

Setting and Showing Variables, Modularity

"Cycling Choices": Harlowe (v3.0)

Summary

Starting in Harlowe 3.0.1, the `(cycling-link)` macro was introduced. Clicking on the link it provides allows for cycling though its possibilities. Combined with the `bind` keyword, its selection can be saved to a variable.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe 3: Cycling Links

:: Start
This cycling-link example remembers the choice made:
(cycling-link: bind $hair, "Black", "Brown", "Blonde", "Rec

This cycling-link example does not:
(cycling-link: "Cat", "Dog", "Fish", "Mouse")

This cycling-link example will disappear (show empty string
(cycling-link: "Two eggs", "One egg", "")

[[Show result]]

:: Show result
The choice of hair was $hair.
```

Download: Twee Code

See Also

Setting and Showing Variables, Modularity

"Cycling Choices": SugarCube (v2.18)

Summary

"Cycling Choices" demostrates how to create a 'cycling' effect of different choices through clicking on them.

The cycle starts with the use of the `<<include>>` macro and assumption of *$choicesCount* beginning at the number -1. Within the passage "Cycling", the first 'cycle' begins with testing if the variable *$choices* exists. If it does not, *$choices* and *$choicesCount* are set (created) to their initial values.

Next, *$choicesCount* is then increased by one to the start value of 0 (the first location of an array in SugarCube) and the position of *$choices* is shown based on this.

If the user clicks on the link (created through using the `<<linkreplace>>` and `<<include>>` macros) again, future 'cycles' test if *$choicesCount* increases beyond the number of values in the *$choices* array and resets it to 0.

At the end of every cycle, the currently selected value is stored in the variable *$cyclingResult* for future access and usage.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Cycling Choices in SugarCube

:: Start
Click options to cycle: <<include "Cycling">>
[[Submit|Results]]

:: Cycling
<<silently>>
<<if not $choices>>
    <<set $choicesCount to -1>>
    <<set $choices to ["First", "Second", "Third"]>>
<</if>>

<<set $choicesCount to $choicesCount + 1>>

<<if $choicesCount >= $choices.length>>
     <<set $choicesCount to 0>>
<</if>>

<<set $cyclingResult to $choices[$choicesCount]>>
<</silently>>
\<<linkreplace $choices[$choicesCount]>><<include "Cycling'

:: Results
$cyclingResult
```

Download: Twee Code

See Also

Setting and Showing Variables, Modularity

"Cycling Choices": Snowman (v1.3.0)

Summary

"Cycling Choices" demostrates how to create a 'cycling' effect of different choices through clicking on them.

Starting with iterating over all elements with the class cycle, each element's 'choices' and 'selection' attribute values are saved as global variables not expected to change during the story. Next, through using jQuery, a *.click()* trigger is set for all elements with the class "cycle".

When triggered, the global values of "choices" and "selection" for the element are retrieved and the "selection" updated. The **text()** of the element is set to the selection index of the *choices* array.

Finally, the global variables are updated according to the element's *id* for later access and to prevent changes to the history of the story potentially affecting saved values.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Cycling Choices in Snowman

:: UserScript[script]
$(function() {

    // Create a global object
    window.setup = window.setup || {};

    // Iterate through all elements with the class 'cycle'
    //  For each, save the current 'choices' and 'selection
    //  (This sets all the 'default' values.)
    $('.cycle').each(function() {

        // Create a global object for each 'id'
        var id = $(this).attr('id');
        setup[id] = {};

        // Save the current 'choices' for each
        var choices = JSON.parse($(this).attr("data-cycling
        setup[id].choices = choices;

        // Save the current 'selection' for each
        var selection = $(this).attr("data-cycling-selectio
        setup[id].selection = selection;

    });

    $('.cycle').click(function(){

        // Save the 'id'
        var id = $(this).attr('id');

        // Retrieve the global 'choices'
        var choices = setup[id].choices;

        // Retrieve the global 'selection'
        var selection = setup[id].selection;

        // Update the 'selection' number
        selection++;

        // Check if 'selection' is greater than length of c
        if(selection >= choices.length) {
            selection = 0;
        }

        // Update the 'selection' on the element
        $(this).attr("data-cycling-selection", selection);

        // Update the text of the element with the choice
        $(this).text(choices[selection]);

        // Update the global values of 'choices' and 'selec
        setup[id].choices = choices;
        setup[id].selection = selection;

    });

});

:: Start
<a href='javascript:void(0)' id='cycleOne' class='cycle' da

[[Submit|Results]]

:: Results
<%= setup["cycleOne"].choices[setup["cycleOne"].selection]
```

Download: Twee Code

See Also

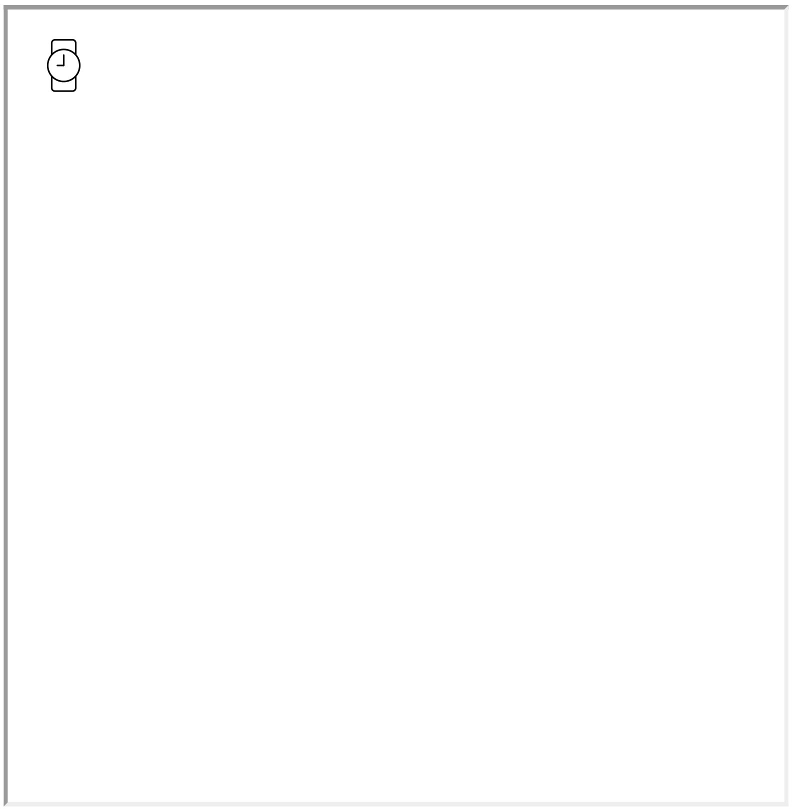
Setting and Showing Variables

"Date and Time": Chapbook (1.0.0)

Summary

Using lookups, Chapbook can easily retrieve the current month, day, and year.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Date and Time in Chapbook

:: Start
The current month number is {now.month}.

The current day number is {now.day}.

The current full-year number is {now.year}.
```

Download: Twee Code

"Date and Time": Harlowe (v2.1)

Summary

"Date and Time" demonstrates how to use the `(current-date:)`, `(current-time:)`, and other time-related macros in Harlowe.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Date and Time in Harlowe

:: Start
The current date is (current-date:).
The current time is (current-time: ).
The current day (of the month) is (monthday: ).
The current day is (weekday: ).
```

Download: Twee Code

"Date and Time": SugarCube (v2.18)

Summary

"Date and Time" demonstrates how to use the JavaScirpt *Date()* functionality in SugarCube.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Date and Time in SugarCube

:: Start
<<print "The current time (in milliseconds since January 1,

<<set $date to new Date()>>
The current month is <<print $date.getMonth() >>.
The current day is <<print $date.getDay() >>.
The current hour is <<print $date.getHours() >>.
The current minute is <<print $date.getMinutes() >>.
The current fullyear is <<print $date.getFullYear() >>.

<<set $originalDate to new Date("October 20, 2018")>>
<<set $timeDifference to Date.now() - $originalDate>>
It has been $timeDifference milliseconds since October 20,
```

Download: Twee Code

"Date and Time": Snowman (v1.3)

Summary

"Date and Time" demonstrates how to use the JavaScirpt *Date()* functionality in Snowman.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Date and Time in Snowman

:: Start
The current time (in milliseconds since January 1, 1970 00:

<%
    window.setup = {};
    window.setup.newDate = new Date();
%>
The current month is <%= setup.newDate.getMonth() %>.

The current day is <%= setup.newDate.getDay() %>.

The current hour is <%= setup.newDate.getHours() %>.

The current minute is <%= setup.newDate.getMinutes() %>.

The current fullyear is <%= setup.newDate.getFullYear() %>.

<% window.setup.originalDate = new Date("October 20, 2018")
It has been <%=  Date.now() - setup.originalDate%> millisec
```

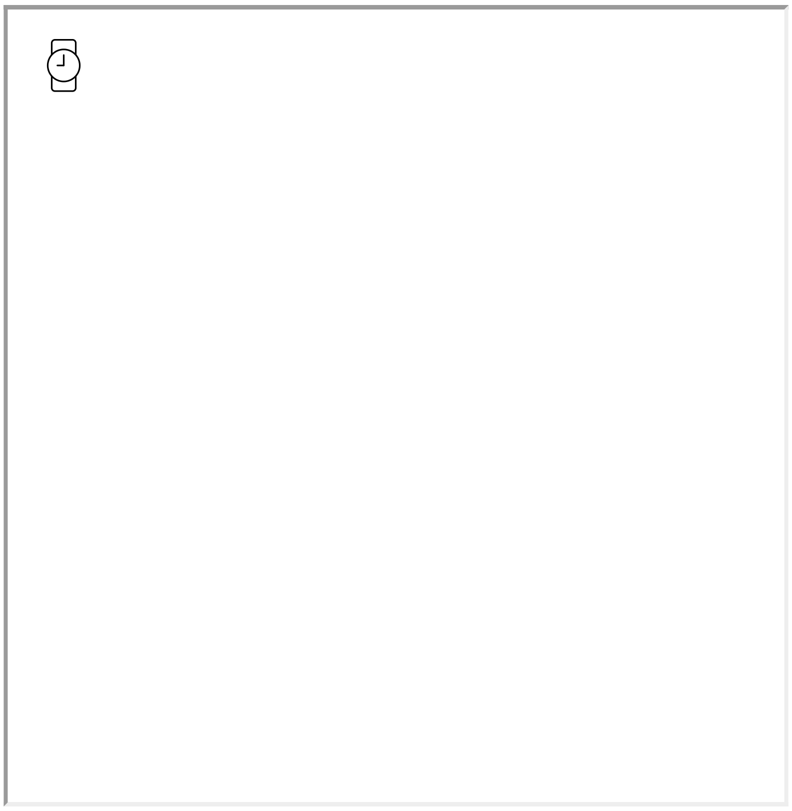
Download: Twee Code

"Delayed Text": Chapbook (v1.0.0)

Summary

In Chapbook, the `[after]` modifier shows text after a set amount of time.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Delayed Text

:: Start
[after 5s; append]
It has been 5 seconds. Show the text!
```

Download: Twee Code

"Delayed Text": Harlowe (v2.0)

Summary

"Delayed Text" uses the `(live:)` and `(stop:)` macros to create a loop that runs only once with a delay of five seconds.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Delayed Text in Harlowe

:: Start
{
    (live: 5s)[
        (stop:)
        It has been 5 seconds. Show the text!
    ]
}
```

Download: Twee Code

See Also

Typewriter Effect

"Delayed Text": SugarCube (v2.18)

Summary

"Delayed Text" uses the `<<timed>>` macro to delay five seconds before showing text.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Delayed Text in SugarCube

:: Start
<<timed 5s>>
It has been 5 seconds. Show the text!
<</timed>>
```

Download: Twee Code

See Also

Typewriter Effect

"Delayed Text": Snowman (v1.3.0)

Summary

"Delayed Text" uses the **delay()** function in Undercore combined with a jQuery selector to target an element with the ID of "results" to change its internal text after five seconds.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Delayed Text in Snowman

:: Start
<div id="results"></div>
<%
    _.delay(
        function()
        {
            $("#results").text("It has been 5 seconds. Show
        },
        5000)
%>
```

Download: Twee Code

See Also

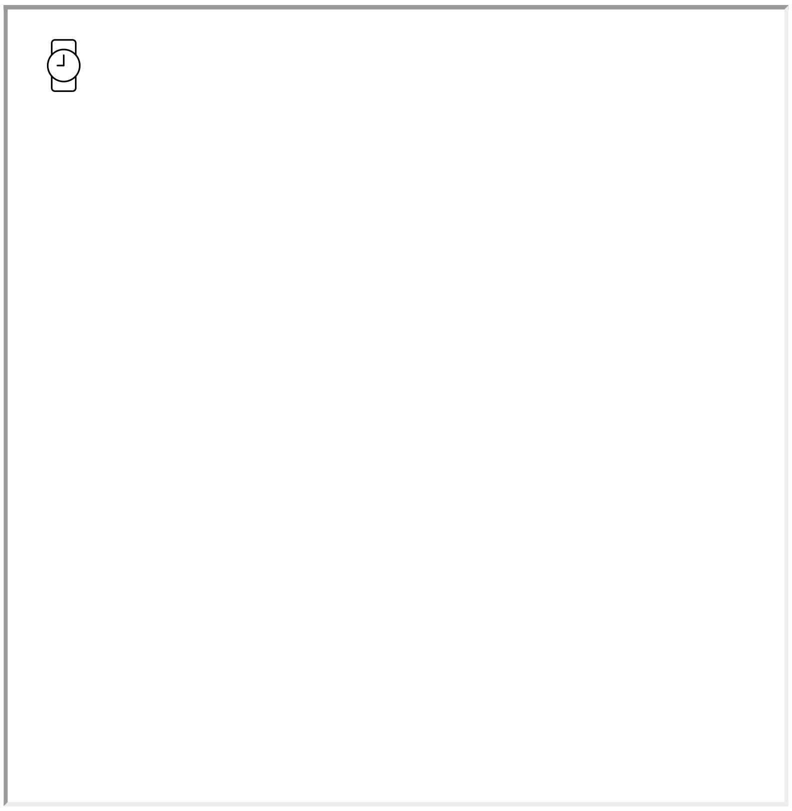
Typewriter Effect

"Deleting Variables": Chapbook (v1.0.0)

Summary

While variables created using the `[JavaScript]` modifier can be deleted within their blocks, any created within the Vars Section can simply be set to `undefined` to make them unable to be shown within an expression using the **engine.state.set()** function.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Deleting Variables

:: Start
color: "red"
--
[JavaScript]
engine.state.set("color", undefined)
[continued]
What is color? {color}
```

Download: Twee Code

"Deleting Variables": SugarCube (v2.18)

Summary

In SugarCube, `<<unset>>` works as a "reverse" to `<<set>>`. Instead of setting a value, it deletes it. `<<unset>>` works on both temporary and story variables.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Unsetting Variables

:: Start
<<set $proof to "hand-written letter">>

[[Accidentally drop the letter]]

:: Accidentally drop the letter
<<unset $proof>>

[[Present the letter to the sheriff]]

:: Present the letter to the sheriff
You present the $proof to the sheriff, not realizing the ra
```

Download: Twee Code

See Also

Conditional Statements, Setting and Showing

"Deleting Variables": Snowman (v1.3)

Summary

Through using the Underscore template library available in Snowman, JavaScript can be used within passages without a `<script>` tag. The **delete** operator can be used in JavaScript to remove a variable.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Deleting Variables

:: UserScript[script]
window.story = {};

window.story.example = "an example!";

:: Start
What is the value of the property "example" of the object w

[[Delete the value!]]

:: Delete the value!
<%

// Delete the variable
delete window.story.example;

%>

[[Test for value]]

:: Test for value
Does "example" still exist as part of the object window.sto
```

Download: Twee Code

"Dice Rolling": Chapbook (v1.0)

Summary

This example demonstrates how to create the same effects of rolling various physical dice through using the *random* global variable in Chapbook.

Inserts are used to display the use of *random* in the passage. For calculations, a var section and temporary variables are used because expressions cannot be used within inserts.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Rolling Dice

:: Start
_example1: random.d4 + 4
_example2: random.d6 - 2
_example3: random.d6 + random.d6 + 10
--

Rolling a 1d4: {random.d4}

Rolling a 1d6: {random.d6}

Rolling a 1d8: {random.d8}

Rolling a 1d10: {random.d10}

Rolling a 1d12: {random.d12}

Rolling a 1d20: {random.d20}

Rolling a 1d100: {random.d100}

Rolling a 1d4 + 4: {_example1}

Rolling a 1d6 - 2: {_example2}

Rolling a 2d6 + 10: {_example3}
```

Download: Twee Code

"Dice Rolling": Harlowe (v2.0)

Summary

"Dice Rolling" demonstrates how to create the same effects of rolling various physical dice through using the `(random:)` macro and adding or subtracting numbers.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Dice Rolling

:: Start
Rolling a 1d4: (random: 1,4)
Rolling a 1d6: (random: 1,6)
Rolling a 1d8: (random: 1,8)
Rolling a 1d10: (random: 1, 10)
Rolling a 1d12: (random: 1, 12)
Rolling a 1d20: (random: 1, 20)
Rolling a 1d100: (random: 1, 100)
Rolling a 1d4 + 4: (text: (random: 1, 4) + 4)
Rolling a 1d6 - 2: (text: (random: 1, 6) - 2)
Rolling a 2d6 + 10: (text: (random: 1, 6) + (random: 1, 6)
```

Download: Twee Code

"Dice Rolling": SugarCube (v2.18)

Summary

"Dice Rolling" demonstrates how to create the same effects of rolling various physical dice through using the **random()** function and the `<<print>>` macro to show the results.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Dice Rolling

:: Start
Rolling a 1d4: <<print random(1,4) >>
Rolling a 1d6: <<print random(1,6) >>
Rolling a 1d8: <<print random(1,8) >>
Rolling a 1d10: <<print random(1, 10) >>
Rolling a 1d12: <<print random(1, 12) >>
Rolling a 1d20: <<print random(1, 20) >>
Rolling a 1d100: <<print random(1, 100) >>
Rolling a 1d4 + 4: <<print random(1, 4) + 4 >>
Rolling a 1d6 - 2: <<print random(1, 6) - 2 >>
Rolling a 2d6 + 10: <<print random(1, 6) + random(1, 6) + 1
```

Download: Twee Code

"Dice Rolling": Snowman (v1.3)

Summary

"Dice Rolling" demonstrates how to create the same effects of rolling various physical dice through using Underscore.js's `_.random()` function and its interpolating functionality.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Dice Rolling

:: Start
Rolling a 1d4: <%= _.random(1,4) %>

Rolling a 1d6: <%= _.random(1,6) %>

Rolling a 1d8: <%= _.random(1,8) %>

Rolling a 1d10: <%= _.random(1, 10) %>

Rolling a 1d12: <%= _.random(1, 12) %>

Rolling a 1d20: <%= _.random(1, 20) %>

Rolling a 1d100: <%= _.random(1, 100) %>

Rolling a 1d4 + 4: <%= _.random(1, 4) + 4 %>

Rolling a 1d6 - 2: <%= _.random(1, 6) - 2 %>

Rolling a 2d6 + 10: <%= _.random(1, 6) + _.random(1, 6) + 1
```

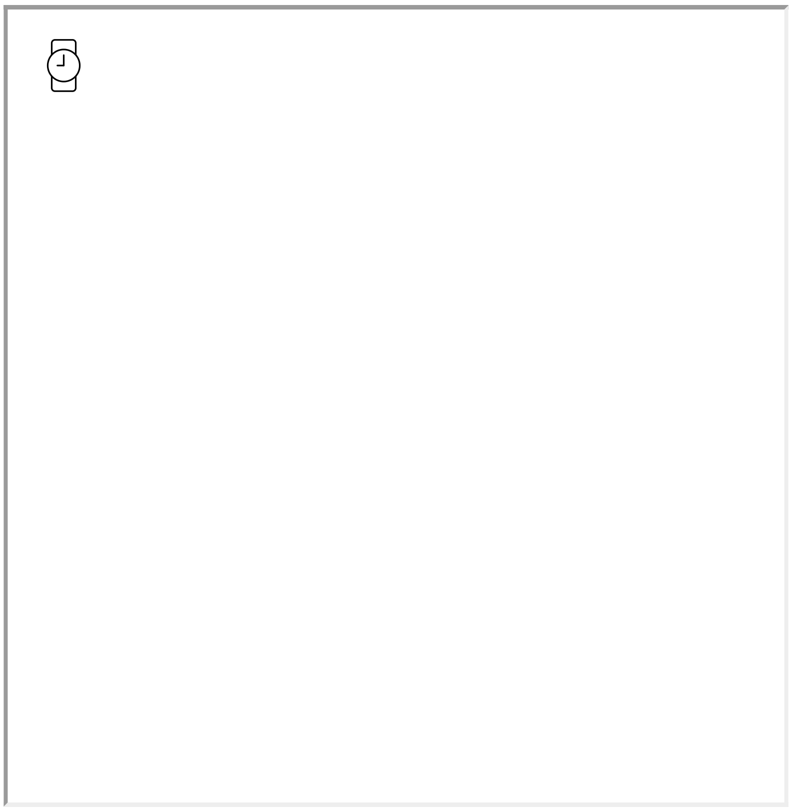
Download: Twee Code

"Dropdown": Chapbook (v1.0)

Summary

The insert `{dropdown}` is used to create a dropdown menu in Chapbook. The `for:` option specifies where to save the selected value and the `choices:` option defines what the choices should be for the user.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Dropdown

:: Start
{dropdown menu for: "chosenValue", choices: ["Up", "Down",

[[Check Choice]]

:: Check Choice
You chose {chosenValue}.
```

Download: Twee Code

"Dropdown": Harlowe (v3.0)

Summary

The macro `(dropdown:)` was introduced with Harlowe 3.0. It creates a drop-down menu based on the options supplied to it. In order to save the outcome of using the drop-down menu, the keyword `bind` is used to save the choice.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe 3: Dropdown

:: Start
Choose direction:
(dropdown: bind $direction, "Up", "Down", "Left", "Right")

[[Show result]]

:: Show result
The direction picked was $direction.
```

Download: Twee Code

"Fairmath System": Chapbook (v1.0.0)

Summary

"Fairmath System" demonstrates how to re-create the Fairmath system found in ChoiceScript. Based on the operation, increasing and decreasing changes the value by a percentage as the difference between the original and adjusted value.

This example defines functions in the Story JavaScript, which are then used in the Vars Section of a passage to set values. These are shown using expressions within the text of the passage itself.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Fairmath

:: UserScript[script]
// Create a global object
window.setup = window.setup || {};

// Create a fairmath global object
window.setup.fairmath = {};

// Create an 'increase' function
setup.fairmath.increase = function(x,y) {
    return Math.round(x+((100-x)*(y/100)));
};

// Create a "decrease" function
setup.fairmath.decrease = function(x,y) {
    return Math.round(x-(x*(y/100)));
};


:: Start
decreaseExample: setup.fairmath.decrease(100, 50)
increaseExample: setup.fairmath.increase(50, 50)
--

Decrease 100 by 50% using Fairmath:
Decrease Example: {decreaseExample}

Increase 50 by 50% using Fairmath:
Increase Example: {increaseExample}
```

Download: Twee Code

"Fairmath System": Harlowe (v2.0)

Summary

"Fairmath System" demonstrates how to re-create the Fairmath system found in ChoiceScript. Based on a percentage operation, increasing and decreasing changes the value by a percentage as the difference between the original and adjusted value.

This example uses the `(display:)` macro in Harlowe to separate operations for increasing and decreasing. Through setting values to adjust, either passage can be included and the *$resultValue* used to track and store changes.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Fairmath System in Harlowe

:: Start
<!-- Fairmath formulas based on http://choicescriptdev.wiki

<!-- Set an initial value for the story -->
(set: $valueToAdjust to 100)
The initial value is $valueToAdjust.

<!-- Set originalValue to the value to adjust -->
(set: $originalValue to $valueToAdjust)
<!-- Set the changeValue (percentage) to adjust -->
(set: $changeValue to 50)
<!-- Display (call) the Fairmath Decrease passage -->
(display: "Decrease")
<!-- The new value will be resultValue -->
The adjusted value is $resultValue.

<!-- Update valueToAdjust -->
(set: $valueToAdjust to $resultValue)
<!-- Set originalValue to the value to adjust -->
(set: $originalValue to $valueToAdjust)
<!-- Set the changeValue (percentage) to adjust -->
(set: $changeValue to 100)
<!-- Display (call) the Fairmath Increase passage -->
(display: "Increase")
The adjusted value is $resultValue.

:: Increase
(set: $resultValue to (round: $originalValue+((100-$origina

:: Decrease
(set: $resultValue to (round: $originalValue-($originalValu
```

Download: Twee Code

"Fairmath System": SugarCube (v2.18)

Summary

"Fairmath System" demonstrates how to re-create the Fairmath system found in ChoiceScript. Based on a percentage operation, increase and decrease changes the value by a percentage as the difference between the original and adjusted value.

This example uses the `<<widget>>` macro in SugarCube to separate operations for increasing and decreasing.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Fairmath in SugarCube

:: Start
<!-- Fairmath formulas based on http://choicescriptdev.wiki

<<set $valueToAdjust to 100>>
The inital value is $valueToAdjust

<!-- Call the decrease widget -->
<<decrease $valueToAdjust 50>>
The adjusted value is $resultValue.

<!-- Save the changed value -->
<<set $valueToAdjust to $resultValue>>
<!-- Call the increase widget -->
<<increase $valueToAdjust 100>>
The adjusted value is $resultValue.

:: Fairmath Operations[widget]
<<widget "increase">>
<<set $resultValue to Math.round($args[0]+((100-$args[0])*(
<</widget>>

<<widget "decrease">>
<<set $resultValue to Math.round($args[0]-($args[0]*($args[
<</widget>>
```

Download: Twee Code

"Fairmath System": Snowman (v1.3.0)

Summary

"Fairmath System" demonstrates how to re-create the Fairmath system found in ChoiceScript. Based on a percentage operation, increase and decrease changes the value by a percentage as the difference between the original and adjusted value.

This example uses functions **increase()** and **decrease()** as part of a created global *window.setup.fairmath*. These can be called through using the Underscore template functionality to define, use, and show the values of the functions in any one passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Fairmath in Snowman

:: UserScript[script]
// Create a global object
window.setup = window.setup || {};

// Create a fairmath global object
window.setup.fairmath = {};

// Create an 'increase' function
setup.fairmath.increase = function(x,y) {
    return Math.round(x+((100-x)*(y/100)));
};

// Create a "decrease" function
setup.fairmath.decrease = function(x,y) {
    return Math.round(x-(x*(y/100)));
};

:: Start
Decrease 100 by 50% using Fairmath:
<%= setup.fairmath.decrease(100, 50) %>

Increase 50 by 50% using Fairmath:
<%= setup.fairmath.increase(50, 50) %>
```

Download: Twee Code

"Geolocation": Chapbook (v1.0.0)

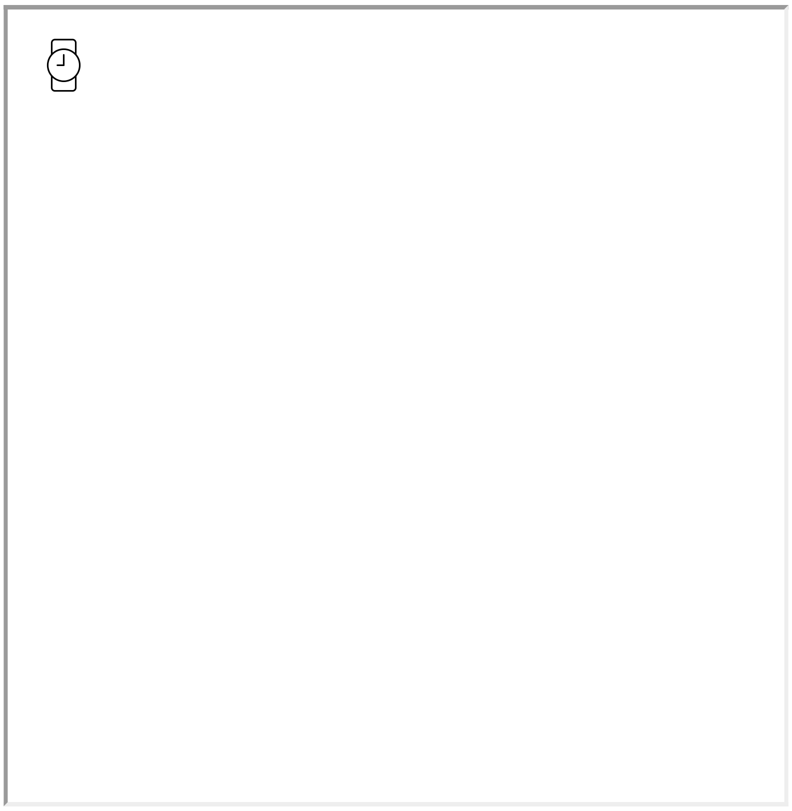
Summary

Many browsers allow access to the current location through the Geolocation property and associated functions. This functionality is subject to the user agreeing to allow access. Until the functionality is unlocked, or if the user declines, default values will be returned.

Functionality availability and their results should always be tested against other location services or information. Most browsers will return results through the fastest and sometimes least-accurate methods possible.

In this example, the `[JavaScript]` modifer is used to test for, run, and show data using an **alert()** from the JavaScript functions.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Geolocation

:: UserScript[script]
(function () {

    window.geolocation = {

        available: function() {
            return ("geolocation" in navigator
                && typeof navigator.geolocation.getCurrentP
        },
        getLocation: function() {

            // Create initial values
            var location = { latitude : 0, longitude : 0 };

            // Create success callback to store values
            var    positionSuccess = function (position) {

                location.latitude = position.coords.latituc
                location.longitude = position.coords.longit

            };

            // Create error callback
            var positionError = function (error) {
                /* Code that handles errors */
            };

            // Create initial options
            var positionOptions = {
                timeout: 31000,
                enableHighAccuracy: true,
                maximumAge : 120000
            };


            // Ask for location based on callbacks and opti
            navigator.geolocation.getCurrentPosition(
                positionSuccess,
                positionError,
                positionOptions
            );

            // Return location found
            // If not location, will return initial (0,0) v
```

```
                return location;

            },
            approximateLocation: function (a, b, allowedDiff) {
                // allowedDiff must always be > 0
                if (a === b) { // handles various "exact" edge
                    return true;
                }

                allowedDiff = allowedDiff || 0.0005;

                return Math.abs(a - b) < allowedDiff;
            }

        };

}());

:: Start
[[GeoLocation]]

:: GeoLocation
[JavaScript]
if(window.geolocation.available() ) {
    var geolocation = window.geolocation.getLocation();
    alert("Latitude: " + geolocation.latitude + " Longitude
}
[continued]
```

Download: Twee Code

"Geolocation": Harlowe (v2.0)

Summary

Many browsers allow access to the current location through the Geolocation property and associated functions. This functionality is subject to the user agreeing to allow access. Until the functionality is unlocked, or if the user declines, default values will be returned.

Functionality availability and their results should always be tested against other location services or information. Most browsers will return results through the fastest and sometimes least-accurate methods possible.

Harlowe does not have an easy way to bridge the gap between its macros and JavaScript. In this example, the `<script>` element is used to test for, run, and show an **alert()** with data from the JavaScript functions.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Geolocation in Harlowe

:: UserScript[script]
(function () {

    window.geolocation = {

        available: function() {
            return ("geolocation" in navigator
                && typeof navigator.geolocation.getCurrentP
        },
        getLocation: function() {

            // Create initial values
            var location = { latitude : 0, longitude : 0 };

            // Create success callback to store values
            var    positionSuccess = function (position) {

                location.latitude = position.coords.latituc
                location.longitude = position.coords.longit

            };

            // Create error callback
            var positionError = function (error) {
                /* Code that handles errors */
            };

            // Create initial options
            var positionOptions = {
                timeout: 31000,
                enableHighAccuracy: true,
                maximumAge : 120000
            };


            // Ask for location based on callbacks and opti
            navigator.geolocation.getCurrentPosition(
                positionSuccess,
                positionError,
                positionOptions
            );

            // Return location found
            // If not location, will return initial (0,0) v
            return location;
        },
        approximateLocation: function (a, b, allowedDiff) {
            // allowedDiff must always be > 0
            if (a === b) { // handles various "exact" edge
                return true;
            }

            allowedDiff = allowedDiff || 0.0005;

            return Math.abs(a - b) < allowedDiff;
        }

    };

}());

:: Start
[[Ask for permission]]

:: Ask for permission
<script>
if(window.geolocation.available() ) {
    var geolocation = window.geolocation.getLocation();
    alert("Latitude: " + geolocation.latitude + " Longitude
}
</script>
```

Download: Twee Code

"Geolocation": SugarCube (v2.18)

Summary

Many browsers allow access to the current location through the Geolocation property and associated functions. This functionality is subject to the user agreeing to allow access. Until the functionality is unlocked, or if the user declines, default values will be returned.

Functionality availability and their results should always be tested against other location services or information. Most browsers will return results through the fastest and sometimes least-accurate methods possible.

This example uses `<<linkreplace>>` and `<<script>>` macros to run JavaScript in passages in SugarCube. The *State.variables* object is used to store the results of running JavaScript functions and using those values in TwineScript.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Geolocation in SugarCube

:: UserScript[script]
(function () {

    window.geolocation = {

        available: function() {
            return ("geolocation" in navigator
                && typeof navigator.geolocation.getCurrentP
        },
        getLocation: function() {

            // Create initial values
            var location = { latitude : 0, longitude : 0 };

            // Create success callback to store values
            var    positionSuccess = function (position) {

                location.latitude = position.coords.latituc
                location.longitude = position.coords.longit

            };

            // Create error callback
            var positionError = function (error) {
                /* Code that handles errors */
            };

            // Create initial options
            var positionOptions = {
                timeout: 31000,
                enableHighAccuracy: true,
                maximumAge : 120000
            };


            // Ask for location based on callbacks and opti
            navigator.geolocation.getCurrentPosition(
                positionSuccess,
                positionError,
                positionOptions
            );

            // Return location found
            // If not location, will return initial (0,0) v
            return location;
        },
        approximateLocation: function (a, b, allowedDiff) {
            // allowedDiff must always be > 0
            if (a === b) { // handles various "exact" edge
                return true;
            }

            allowedDiff = allowedDiff || 0.0005;

            return Math.abs(a - b) < allowedDiff;
        }

    };

}());

:: Start
<<linkreplace "Ask for permission">>
  <<script>>

  State.variables.geoLocationAvailable = window.geolocation

  if(window.geolocation.available()) {
      State.variables.location = window.geolocation.getLoca
  }
  <</script>>
  [[Show results]]
<</linkreplace>>


:: Show results
<<script>>

  State.variables.geoLocationAvailable = window.geolocation

  if(window.geolocation.available()) {
      State.variables.location = window.geolocation.getLoca
  }
<</script>>

Is geolocation available? $geoLocationAvailable

If so, what is the current location?

Latitude: $location.latitude

Longitude: $location.longitude
```

```
Are we in the approximate location of Stonehenge (51.178885

<<set $approxLat to  window.geolocation.approximateLocatior

<<set $approxLong to window.geolocation.approximateLocatior

Latitude: $approxLat
Longitude: $approxLong
```

Download: Twee Code

"Geolocation": Snowman (v1.3.0)

Summary

Many browsers allow access to the current location through the Geolocation property and associated functions. This functionality is subject to the user agreeing to allow access. Until the functionality is unlocked, or if the user declines, default values will be returned.

Functionality availability and their results should always be tested against other location services or information. Most browsers will return results through the fastest and sometimes least-accurate methods possible.

This example uses Underscore template functionality to test for and show the values of properties storied in the *s* global variable in Snowman.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Geolocation in Snowman

:: UserScript[script]
(function () {

    window.geolocation = {

        available: function() {
            return ("geolocation" in navigator
                && typeof navigator.geolocation.getCurrentP
        },
        getLocation: function() {

            // Create initial values
            var location = { latitude : 0, longitude : 0 };

            // Create success callback to store values
            var    positionSuccess = function (position) {

                location.latitude = position.coords.latituc
                location.longitude = position.coords.longit

            };

            // Create error callback
            var positionError = function (error) {
                /* Code that handles errors */
            };

            // Create initial options
            var positionOptions = {
                timeout: 31000,
                enableHighAccuracy: true,
                maximumAge : 120000
            };


            // Ask for location based on callbacks and opti
            navigator.geolocation.getCurrentPosition(
                positionSuccess,
                positionError,
                positionOptions
            );

            // Return location found
            // If not location, will return initial (0,0) v
            return location;
        },
        approximateLocation: function (a, b, allowedDiff) {
            // allowedDiff must always be > 0
            if (a === b) { // handles various "exact" edge
                return true;
            }

            allowedDiff = allowedDiff || 0.0005;

            return Math.abs(a - b) < allowedDiff;
        }

    };

}());

:: Start
[[Ask for permission]]

:: Show results
Is geolocation available? <%= window.geolocation.available(
<%
    if(window.geolocation.available()) {
        s.location = window.geolocation.getLocation();
    }
%>

If so, what is the current location?

Latitude: <%= s.location.latitude %>

Longitude: <%= s.location.longitude %>

Are we in the approximate location of Stonehenge (51.178885

Latitude: <%= window.geolocation.approximateLocation(s.loca

Longitude: <%= window.geolocation.approximateLocation(s.loc

:: Ask for permission
<%
    if(window.geolocation.available()) {
        s.location = window.geolocation.getLocation();
    }
%>
[[Show results]]
```

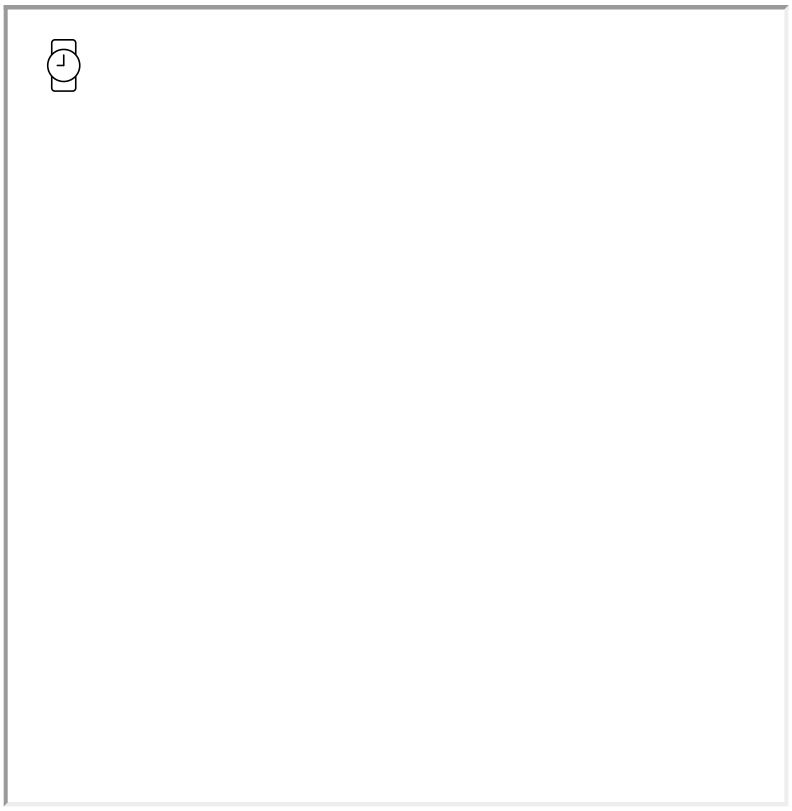
Download: Twee Code

"Google Fonts": Chapbook (v1.0.0)

Summary

Chapbook provides a global variable, *config.style.googleFont*, that can be used within the Vars Section to load and use an external font.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Google Fonts

:: Start
config.style.googleFont: '<link href="https://fonts.googlea
config.style.page.font: 'Roboto'
--
This text is styled by a Google Font.
```

Download: Twee Code

"Google Fonts": Harlowe (v2.0)

Summary

"Google Fonts" uses a Google Font loaded via the CSS **@import** at-rule. The loaded font is then applied to selected text using the `(font:)` macro.

Other Google Fonts could be imported and applied using the same method, creating new class or ID style rules to be applied for and across different HTML elements in the same way.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Google Fonts

:: StoryStylesheet[stylesheet]
@import url('https://fonts.googleapis.com/css?family=Roboto

:: Start
(font:"Roboto")[This text is styled by a Google Font]
```

Download: Twee Code

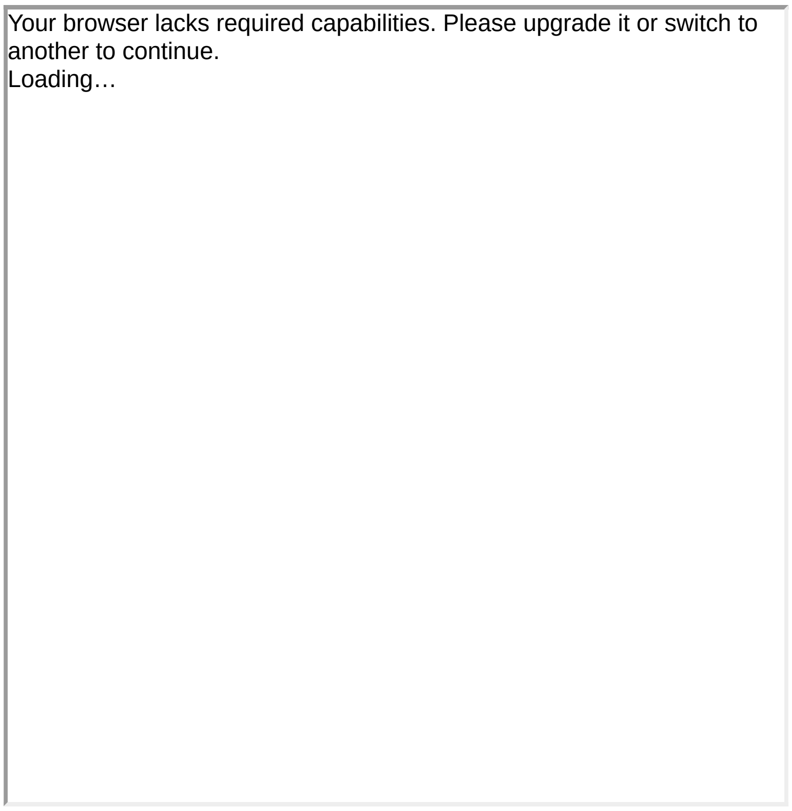
"Google Fonts": SugarCube (v2.18)

Summary

"Google Fonts" uses a Google Font loaded via the CSS **@import** at-rule. A class style rule ("message") is then created using the imported font-family and applied to a `<div>` element within a single passage.

Other Google Fonts could be imported and applied using the same method, creating new class or ID style rules to be applied for and across different HTML elements in the same way.

Live Example



Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Google Fonts

:: StoryStylesheet[stylesheet]
@import url('https://fonts.googleapis.com/css?family=Roboto

.message {
    font-family: 'Roboto', sans-serif;
}

:: Start
<div class="message">This text is styled using a Google For
```

Download: Twee Code

"Google Fonts": Snowman (v1.3)

Summary

"Google Fonts" uses a Google Font loaded via the CSS **@import** at-rule. A class style rule ("message") is then created using the imported font-family and applied to a `<div>` element within a single passage.

Other Google Fonts could be imported and applied using the same method, creating new class or ID style rules to be applied for and across different HTML elements in the same way.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Google Fonts

:: StoryStylesheet[stylesheet]
@import url('https://fonts.googleapis.com/css?family=Roboto

.message {
    font-family: 'Roboto', sans-serif;
}

:: Start
<div class="message">This text is styled using a Google For
```

Download: Twee Code

"Google Fonts": Sugarcane (v1.4.2)

Note: The following example is designed for Twine 1.4.2.

Summary

"Google Fonts" uses a Google Font loaded via the CSS **@import** at-rule. A class style rule ("message") is then created using the imported font-family and applied to a `<div>` element within a single passage.

Other Google Fonts could be imported and applied using the same method, creating new class or ID style rules to be applied for and across different HTML elements in the same way.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Responsive: Google Fonts

:: StoryAuthor
@videlais

:: StoryStylesheet [stylesheet]
@import url('https://fonts.googleapis.com/css?family=Roboto

.message {
    font-family: 'Roboto', sans-serif;
}

:: Start
<div class="message">This text is styled using a Google For
```

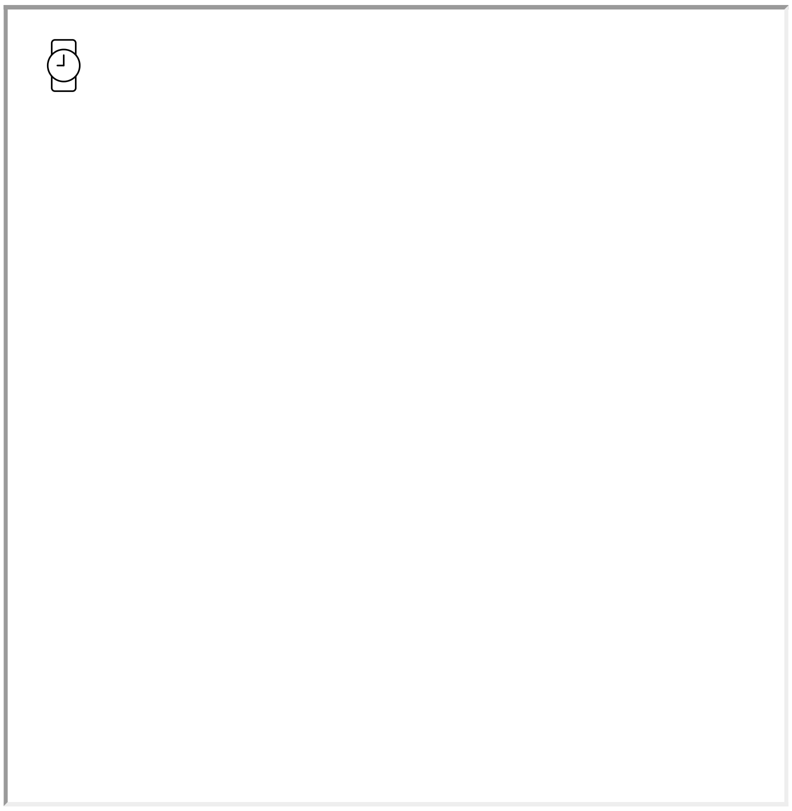
Download: Twee Code

"Headers and Footers": Chapbook (v1.0.0)

Summary

Chapbook provides two global variables, *config.header* and *config.footer*, that each have the properties *left*, *right*, and *center*. When set, these properties show their content as the header or footer in the particular position matching its property name.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Header and Footer

:: Start
config.header.center: "This is the header!"
config.footer.center: "This is the footer!"
--
This is content.
```

Download: Twee Code

"Headers and Footers": Harlowe (v2.0)

Summary

"Headers and Footers" demonstrates the use of the "header" and "footer" passage tags. When used in passages, they are either prepended (header) or appended (footer) to every passage. Multiple passages can have the tags and are loaded alphabetically when detected.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Headers and Footers

:: Start
This is content between the header and the footer.


:: Header[header]
This is the header!


:: Footer[footer]
This is the footer!
```

Download: Twee Code

"Headers and Footers": Sugarcube (v2.18)

Summary

"Headers and Footers" demonstrates the use of "PassageHeader" and "PassageFooter" special names for passages. When these special names are used, the 'Header' is prepended and the 'Footer' name is appended to all passages.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Headers and Footers

:: Start
This is content between the header and the footer.


:: PassageHeader
This is the header!


:: PassageFooter
This is the footer!
```

Download: Twee Code

See Also

Passage Events

"Headers and Footers": Snowman (v1.3)

Summary

"Headers and Footers" demonstrates the use of the **window.story.render()** function to return the HTML contents of another passage. Combined with Underscore template functionality, the content of passages can be "displayed" in others. Because Snowman does not have pre-defined 'header' or 'footer' functionality, using these two different methods together can create the same result. However, the code would need to be included on any additional passages to continue the effect.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Headers and Footers

:: Start
<%= window.story.render("Header") %>

This content is between the header and the footer.

<%= window.story.render("Footer") %>

:: Header
This is the header!

:: Footer
This is the footer!
```

Download: Twee Code

See Also

Passage Events

"Hidden Link": Harlowe (v2.0)

Summary

"Hidden Link" demonstrates how to create a 'hidden' link that is only revealed when the cursor passes over it.

Using CSS and JavaScript, a rule is created for transparent color and applied or removed through using jQuery's **on()** function with 'mouseenter' and 'mouseleave' events.

The use of a "footer" special Passage is also used to run the required JavaScript after each passage is displayed.

Harlowe supports a number of different techniques for creating links and the resulting HTML elements generated is different for each of these techniques. The generated HTML falls into two main groups: those that include a `<<tw-link>>` element, and those that include a ".enchantment-link" classed element. This example supports both groups.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Hidden Link


:: UserStylesheet [stylesheet]
.hidden tw-link, .hidden .enchantment-link {
    color: transparent;
}

tw-include[title="Hidden Link Setup"] {
    display: none;
}


:: Hidden Link Setup [footer]
<script>
    /*
        Hidden links that are always hidden:
            <span class="hidden">[[A hidden link]]</span>
    */
    $('.hidden')
        .addClass('hidden');

    /*
        Hidden links that hide unless you're hovering over
            <span class="hides">[[A hidden link]]</span>
    */
    $('.hides')
        .addClass('hidden')
        .on('mouseenter', function () {
            $(this).removeClass('hidden');
        })
        .on('mouseleave', function () {
            $(this).addClass('hidden');
        });

    /*
        Hidden links that reveal themselves when you hover
            <span class="reveals">[[A hidden link]]</span>
    */
    $('.reveals')
        .addClass('hidden')
        .one('mouseenter', function () {
            $(this).removeClass('hidden');
        });
</script>
```

```
:: Start
''Examples of tw-link element based links''

A hidden link that's always hidden: <span class="hidden">[[

A hidden link that hides unless you're hovering over it: <s

A hidden link that reveals itself when you hover over it: <


''Examples of .enchantment-link CSS class based links''

A hidden link that's always hidden: <span class="hidden">[A

A hidden link that hides unless you're hovering over it: <s

A hidden link that reveals itself when you hover over it: <

(click: ?link)[(go-to: "A hidden link")]


:: A hidden link
You found it!
```

Download: Twee Code

"Hidden Link": SugarCube (v2.18)

Summary

"Hidden Link" demonstrates how to create a 'hidden' link that is only revealed when the cursor passes over it.

Using CSS and JavaScript, a rule is created for transparent color and applied or removed through using jQuery's **on()** function with 'mouseenter' and 'mouseleave' events.

The use of the **postdisplay** functionality is also used to run JavaScript after each passage is displayed.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Hidden Link

:: UserScript[script]
postdisplay['hidden-link-setup'] = function () {
    /*
        Hidden links that are always hidden:
            <span class="hidden">[[A hidden link]]</span>
    */
    $('.hidden')
        .addClass('hidden');

    /*
        Hidden links that hide unless you're hovering over
            <span class="hides">[[A hidden link]]</span>
    */
    $('.hides')
        .addClass('hidden')
        .on('mouseenter', function () {
            $(this).removeClass('hidden');
        })
        .on('mouseleave', function () {
            $(this).addClass('hidden');
        });

    /*
        Hidden links that reveal themselves when you hover
            <span class="reveals">[[A hidden link]]</span>
    */
    $('.reveals')
        .addClass('hidden')
        .one('mouseenter', function () {
            $(this).removeClass('hidden');
        });
};


:: UserStylesheet[stylesheet]
.hidden a {
    color: transparent;
}


:: Start
A hidden link that's always hidden: <span class="hidden">[[

A hidden link that hides unless you're hovering over it: <s
```

```
A hidden link that reveals itself when you hover over it: <

:: A hidden link
You found it!
```

Download: Twee Code

"Hidden Link": Snowman (v1.3.0)

Summary

"Hidden Link" demonstrates how to create a 'hidden' link that is only revealed when the cursor passes over it.

Using CSS and JavaScript, a rule is created for transparent color and applied or removed through using jQuery's **on()** function with 'mouseenter' and 'mouseleave' events.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Hidden Link

:: UserScript[script]
$(function () {
    /*
        Hidden links that are always hidden:
            <span class="hidden">[[A hidden link]]</span>
    */
    $('.hidden')
        .addClass('hidden');

    /*
        Hidden links that hide unless you're hovering over
            <span class="hides">[[A hidden link]]</span>
    */
    $('.hides')
        .addClass('hidden')
        .on('mouseenter', function () {
            $(this).removeClass('hidden');
        })
        .on('mouseleave', function () {
            $(this).addClass('hidden');
        });

    /*
        Hidden links that reveal themselves when you hover
            <span class="reveals">[[A hidden link]]</span>
    */
    $('.reveals')
        .addClass('hidden')
        .one('mouseenter', function () {
            $(this).removeClass('hidden');
        });
});



:: UserStylesheet[stylesheet]
.hidden a {
    color: transparent;
      /* By default links in Snowman have a border */
    border-bottom: 0px;
}


:: Start
A hidden link that's always hidden: <span class="hidden">[[
A hidden link that hides unless you're hovering over it: <s

A hidden link that reveals itself when you hover over it: <


:: A hidden link
You found it!
```

Download: Twee Code

"Images": Chapbook (v1.0.0)

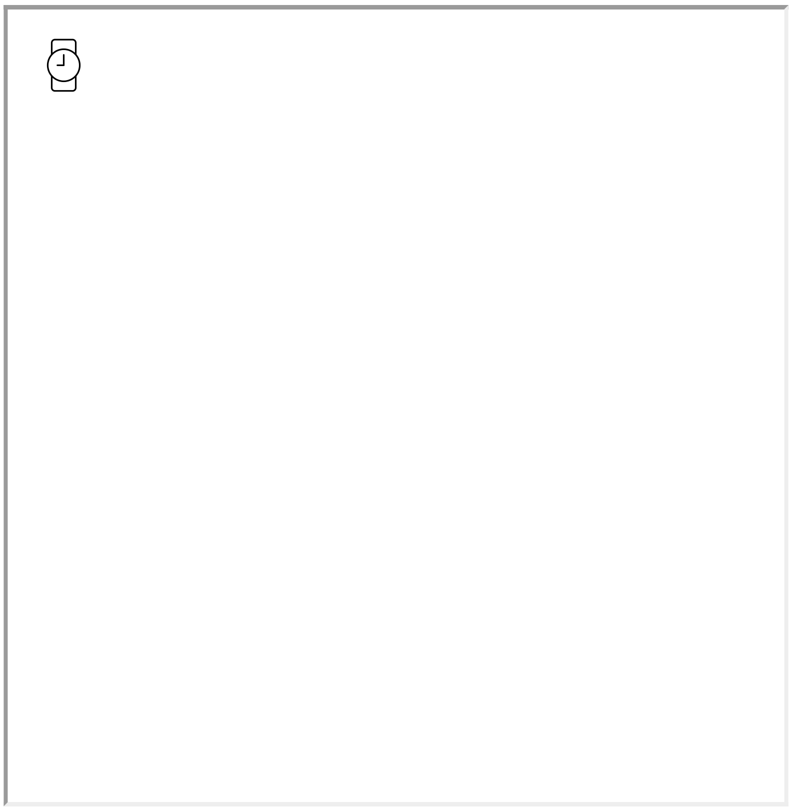
Summary

When using Chapbook, images can be displayed through the image HTML element and **url()** CSS data type when encoded as Base64.

When using an image element, its source is either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the image in relation to the webpage.

Because images are external resources, they need to be included with the webpage as Base64-encoded or in another location. While Base64-encoded images can be embedded in a webpage, it also increases its overall size. External images require additional hosting and are included through their reference in CSS (URL) data type or image (SRC) attribute.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Images in Chapbook

:: UserStylesheet[stylesheet]
.base64image {
  width: 256px;
  height: 256px;
  /* Base64 image truncated for example */
  /* See Twee file for full version. */
  background-image: url('data:image/png;base64...');
}

:: Start
This is an image element:

<img src="https://twinery.org/homepage/img/logo.svg" width=

This is a base-64-encoded CSS image background:

<div class="base64image"></div>
```

Download: Twee Code

"Images": Harlowe (v2.0)

Summary

When using Harlowe, images can be displayed through the image HTML element and **url()** CSS data type when encoded as Base64.

When using an image element, its source is either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the image in relation to the webpage.

Because images are external resources, they need to be included with the webpage as Base64-encoded or in another location. While Base64-encoded images can be embedded in a webpage, it also increases its overall size. External images require additional hosting and are included through their reference in CSS (URL) data type or image (SRC) attribute.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Images in Harlowe

:: UserStylesheet[stylesheet]
.base64image {
  width: 256px;
  height: 256px;
  /* Base64 image truncated for example */
  /* See Twee file for full version. */
  background-image: url('data:image/png;base64...');
}

:: Start
This is an image element:

<img src="https://twinery.org/homepage/img/logo.svg" width=

This is a base-64-encoded CSS image background:

<div class="base64image"></div>
```

Download: Twee Code

"Images": SugarCube (v2.18)

Summary

When using SugarCube, images can be displayed through the image HTML element and **url()** CSS data type when encoded as Base64.

When using an image element, its source is either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the image in relation to the webpage.

Because images are external resources, they need to be included with the webpage as Base64-encoded or in another location. While Base64-encoded images can be embedded in a webpage, it also increases its overall size. External images require additional hosting and are included through their reference in CSS (URL) data type or image (SRC) attribute.

Many macros also support using images in Sugarcube and their location can be used within wiki syntax. Base64-encoded images are not supported in wiki image syntax.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Images in SugarCube

:: UserStylesheet[stylesheet]
.base64image {
  width: 256px;
  height: 256px;
  /* Base64 image truncated for example */
  /* See Twee file for full version. */
  background-image: url('data:image/png;base64...');
}

:: Start
This is an image element:

<img src="https://twinery.org/homepage/img/logo.svg" width=

This is a Base64-encoded CSS image background:

<div class="base64image"></div>
```

Download: Twee Code

"Images": Snowman (v1.3.0)

Summary

When using Snowman, images can be displayed through the image HTML element and **url()** CSS data type when encoded as Base64.

When using an image element, its source is either absolutely or relatively located. An absolute reference starts with HTTP or another protocol; a relative reference describes the location of the image in relation to the webpage.

Because images are external resources, they need to be included with the webpage as Base64-encoded or in another location. While Base64-encoded images can be embedded in a webpage, it also increases its overall size. External images require additional hosting and are included through their reference in CSS (URL) data type or image (SRC) attribute.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Images in Snowman

:: UserStylesheet[stylesheet]
.base64image {
  width: 256px;
  height: 256px;
  /* Base64 image truncated for example */
  /* See Twee file for full version. */
  background-image: url('data:image/png;base64...');
}

:: Start
This is an image element:

<img src="https://twinery.org/homepage/img/logo.svg" width=

This is a Base64-encoded CSS image background:

<div class="base64image"></div>
```

Download: Twee Code

"Importing External JavaScript": Chapbook (v1.0.0)

Note: This example uses code from Three.js, a library for creating 3D graphics in the browser. It is include only for demonstrational purposes and its own documentation should be consulted to understand its functionality.

Summary

To include external JavaScript, it must first be loaded. This example uses code from the Mozilla Developer Network to dynamically import scripts. The example library loaded is Three.js.

Once loaded, a callback function is used to create a simple Three.js example and, if the browser supports it, shows a rotating 3D cube within the passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Importing External JS

:: UserScript[script]
// The following code is used from MDN for
// dynamically importing scripts
// https://developer.mozilla.org/en-US/docs/Web/API/HTMLScr

window.setup = {};

setup.loadError = function(oError) {
  throw new URIError("The script " + oError.target.src + "
};

setup.loadScript = function(url, onloadFunction) {
  var newScript = document.createElement("script");
  newScript.onerror = setup.loadError;
  if (onloadFunction) { newScript.onload = onloadFunction;
  document.head.appendChild(newScript);
    newScript.async = true;
  newScript.src = url;
};


:: Start
<div id="drawArea"></div>
[JavaScript]
setup.loadScript("https://ajax.googleapis.com/ajax/libs/thr
    var scene = new THREE.Scene();
    var camera = new THREE.PerspectiveCamera(
        75,
        1,
        0.1,
        1000 );

    var renderer = new THREE.WebGLRenderer();
    renderer.setSize( 250, 250 );
    document.getElementById("drawArea").appendChild( render

    var geometry = new THREE.BoxGeometry( 1, 1, 1 );
    var material = new THREE.MeshBasicMaterial( { color: 0x
    var cube = new THREE.Mesh( geometry, material );
    scene.add( cube );

    camera.position.z = 5;

    var animate = function () {
        requestAnimationFrame( animate );

        cube.rotation.x += 0.01;
        cube.rotation.y += 0.01;

        renderer.render( scene, camera );
    };

    animate();

});

[continued]
```

Download: Twee Code

"Importing External JavaScript": Harlowe (v2.0)

Summary

"Importing External JavaScript" demonstrates how to import an externally stored JavaScript library, jQuery UI.

This example uses the built-in jQuery **$.getScript()** function to load the library and demonstrates a short example of how to use it.

Note: The successful loading of an external JavaScript file or library commonly produces no visual output. The code within the example passage is not required for the successful loading of an external file or library.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Importing External JavaScript


:: UserScript [script]
/* import jQuery UI library. */
$(function () {
    $.getScript("https://ajax.googleapis.com/ajax/libs/jque
        function (data, textStatus, jqxhr) {
            console.log('jquery ui file loaded');
        }
    );
});


:: Start
<p>Click on the grey box below to see it bounce.</p>
<div id="box" style="width: 100px; height: 100px; backgrour

<script>
$("#box").click(function () {
  $("#box").toggle("bounce", {times: 3}, "slow");
});
</script>
```

Download: Twee Code

"Importing External JavaScript": SugarCube (v2.18)

Summary

"Importing External JavaScript" demonstrates how to import an externally stored JavaScript library, jQuery UI.

This example uses the SugarCube **importScripts()** function to load and integrate the script file's contents.

Note: The successful loading of an external JavaScript file or library commonly produces no visual output. The code within the example passage is not required for the successful loading of an external file or library.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Importing External JavaScript


:: UserScript [script]
/* Import the jQuery UI library. */
importScripts("https://ajax.googleapis.com/ajax/libs/jquery


:: Start
<p>Click on the grey box below to see it bounce.</p>
<div id="box" style="width: 100px; height: 100px; backgrour

<<script>>
$(document).one(':passagerender', function (ev) {
    $(ev.content)
        .find("#box")
        .click(function () {
            $("#box").toggle("bounce", {times: 3}, "slow");
        });
});
<</script>>
```

Download: Twee Code

"Importing External JavaScript": Snowman (v1.3.0)

Summary

"Importing External JavaScript" demonstrates how to import an externally stored JavaScript library, jQuery UI.

This example uses the built-in jQuery **$.getScript()** function to load the library and demonstrates a short example of how to use it.

Note: The successful loading of an external JavaScript file or library commonly produces no visual output. The code within the example passage is not required for the successful loading of an external file or library.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Importing External JavaScript


:: UserScript [script]
/* import jQuery UI library. */
$(function () {
    $.getScript("https://ajax.googleapis.com/ajax/libs/jque
        function (data, textStatus, jqxhr) {
            console.log('jquery ui file loaded');
        }
    );
});


:: Start
<p>Click on the grey box below to see it bounce.</p>
<div id="box" style="width: 100px; height: 100px; backgrour

<script>
$("#box").click(function () {
  $("#box").toggle("bounce", {times: 3}, "slow");
});
</script>
```

Download: Twee Code

"Keyboard Events": Chapbook (v1.0.0)

Summary

"Keyboard Events" demonstrates how to capture keyboard events and then how to associate individual keys with activities within a story.

The example uses **addEventListener()** to monitor for all "keyup" events. Once a "keyup" event has occurred, two values are available:

- The *keyCode* property: the numerical value representing the key presented in its decimal ASCII code supported by effectively all browsers.
- The *key* property: the string value of the key presented supported by most modern web-browsers.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Keyboard

:: UserScript[script]
(function () {
    document.addEventListener('keyup', function (ev) {
        /* the ev variable contains a keyup event object.
         *
         * ev.keyCode - contains the ASCII code of the key
         * ev.key     - contains the key value of the key t
         *
         */

        /* the following shows an alert when the 'a' key is
        if (ev.key === 'a') {
            alert("the 'a' key was released.");
        }
    });
}());


:: Start
Press and release the ''a'' key to show an Alert dialog.
```

Download: Twee Code

"Keyboard Events": Harlowe (v2.0)

Summary

"Keyboard Events" demonstrates how to capture keyboard events and then how to associate individual keys with activities within a story.

The example uses jQuery's **on()** function to monitor for all *keyup* events. Once a "keyup" event has occurred, two values are available:

- The *keyCode* property: the numerical value representing the key presented in its decimal ASCII code supported by effectively all browsers.
- The *key* property: the string value of the key presented supported by most modern web-browsers.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Keyboard


:: UserScript[script]
(function () {
    $(document).on('keyup', function (ev) {
        /* the ev variable contains a keyup event object.
         *
         * ev.keyCode - contains the ASCII code of the key
         * ev.key     - contains the key value of the key t
         *
         */


        /* the following shows an alert when the 'a' key is
        if (ev.key === 'a') {
            alert("the 'a' key was released.");
        }
    });
}());


:: Start
Press and release the ''a'' key to show an Alert dialog.
```

Download: Twee Code

"Keyboard Events": SugarCube (v2.18)

Summary

"Keyboard Events" demonstrates how to capture keyboard events and then how to associate individual keys with activities within a story.

The example uses jQuery's **on()** function to monitor for all "keyup" events. Once a "keyup" event has occurred, two values are available:

- The *keyCode* property: the numerical value representing the key presented in its decimal ASCII code supported by effectively all browsers.
- The *key* property: the string value of the key presented supported by most modern web-browsers.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Keyboard

:: UserScript[script]
(function () {
    $(document).on('keyup', function (ev) {
        /* the ev variable contains a keyup event object.
         *
         * ev.keyCode - contains the ASCII code of the key
         * ev.key     - contains the key value of the key t
         *
         */


        /* the following shows an alert when the 'a' key is
        if (ev.key === 'a') {
            UI.alert("the 'a' key was released.");
        }
    });
}());


:: Start
Press and release the ''a'' key to show an Alert dialog.
```

Download: Twee Code

"Keyboard Events": Snowman (v1.3.0)

Summary

"Keyboard Events" demonstrates how to capture keyboard events and then how to associate individual keys with activities within a story.

The example uses jQuery's **on()** function to monitor for all "keyup" events. Once a "keyup" event has occurred, two values are available:

- The *keyCode* property: the numerical value representing the key presented in its decimal ASCII code supported by effectively all browsers.
- The *key* property: the string value of the key presented supported by most modern web-browsers.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Keyboard


:: UserScript[script]
(function () {
    $(document).on('keyup', function (ev) {
        /* the ev variable contains a keyup event object.
         *
         * ev.keyCode - contains the ASCII code of the key
         * ev.key     - contains the key value of the key t
         *
         */


        /* the following shows an alert when the 'a' key is
        if (ev.key === 'a') {
            alert("the 'a' key was released.");
        }
    });
}());


:: Start
Press and release the ''a'' key to show an Alert dialog.
```

Download: Twee Code

"Left Sidebar": Harlowe (both v1.x and v2.x series)

Summary

Harlowe has a built-in left sidebar. Before v2.1.0, however, there was no functionality to add dynamic content. It is possible to re-purpose a "footer" tagged passage to act as a custom sidebar to display dynamic content below Harlowe's own sidebar.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Left Sidebar in Harlowe


:: UserStylesheet [stylesheet]
/*
    Reposition the Sidebar 'footer' tagged passage.
*/
tw-include[title="Sidebar"] {
    position: fixed;
    top: 0;
    left: 0;
    width: 20%;                       /* padding-right of
    max-height: 100%;
    margin-top: calc(5% + 171px);    /* padding-top of the
    padding: 0.5em;
    background-color: transparent;
    text-align: right;
}


:: Start
(set: $name to "Jane Doe", $location to "Work")\
[[Another passage]]


:: Sidebar [footer]
Name: $name
Location: $location


:: Another passage
(set: $name to "John Smith", $location to "Shop")\
[[Start]]
```

Download: Twee Code

See Also

CSS and Passage Tags

"Left Sidebar": Harlowe (only v2.1.0 or later)

Summary

Harlowe v2.1.0 or later includes a built-in named hook named *?Sidebar*. When combined with the `(append:)` macro, dynamic content can be added to the left, blank area containing the default Undo and Redo links. A "footer" tagged passage is used to update the dynamic content after each passage transition, and CSS is used to resize and position the existing `<tw-sidebar>` element.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Left Sidebar in Harlowe (v2.1.0 or later)

:: UserStylesheet [stylesheet]
/*
    Reposition the Sidebar 'footer' tagged passage.
*/
tw-sidebar {
    position: fixed;
    top: 0;
    left: 0;
    width: 20%;                        /* padding-right of
    max-height: 100%;
    margin-top: 5%;                    /* padding-top of th
    padding: 0 0.5em 0.5em 0.5em;
    text-align: right;
    background-color: transparent;
}
tw-icon {
    text-align: right;
    padding-right: 0.75em;
}

:: Start
(set: $name to "Jane Doe", $location to "Work")\
[[Another passage]]

:: Sidebar [footer]
(append: ?SideBar)[\
Name: $name
Location: $location
]

:: Another passage
(set: $name to "John Smith", $location to "Shop")\
[[Start]]
```

Download: Twee Code

See Also

CSS and Passage Tags

"Left Sidebar": SugarCube (v2.18)

Summary

SugarCube has a built-in left sidebar whose contents can be changed by adding one of several special passages to your story.

The following list describes each of the special passages in the order that they appear vertically within the sidebar:

- StoryBanner appears directly above the story's Title. One use is to show the story's icon/image.
- StorySubtitle appears directly below the story's Title. One use is to show the story's version information.
- StoryAuthor is used to show the Author's information.
- StoryCaption is generally used to show dynamic information about the main character or the story's progress.
- StoryMenu appears directly above the *Save* button and is used to show custom menu items.
- StoryShare appears directly below the *Restart* button and is used to access a dialog containing Author's social media or web-site links.

The sidebar can be manually stowed (hidden) and unstowed (revealed) by selecting the **<** or **>** icon in the sidebar's top right corner. The same effect can be achieved programatically by using the UIBar global object and its **UIBar.stow()** and **UIBar.unstow()** functions.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Left Sidebar in SugarCube

:: Start
<<set $name to "Jane Doe", $location to "Work">>\
[[Another passage]]


<<link "Stow the sidebar!">>
    <<run UIBar.stow() >>
<</link>>
<<link "Unstow the sidebar!">>
    <<run UIBar.unstow() >>
<</link>>


:: StoryBanner
<img src="https://twinery.org/homepage/img/logo.svg" width=


:: StorySubtitle
Version: 0.2.1


:: StoryAuthor
by Anonymous


:: StoryCaption
Name: $name
Location: $location


:: StoryMenu
[[New story link!|Start]]


:: StoryShare
[[Twinery|https://twinery.org/]]


:: Another passage
<<set $name to "John Smith", $location to "Shop">>\
[[Start]]
```

Download: Twee Code

"Left Sidebar": Snowman (v1.3.0)

Summary

Snowman does not have a built-in sidebar, but one can be created using JavaScript, jQuery, and CSS.

The **createElement()** function is used to create a new DIV element into which the generated output of the Sidebar passage will later be added. This new DIV is assigned an ID of "sidebar" using the **attr()** function and then inserted into the story's Document Object Model (DOM) using the **insertBefore()** function.

Snowman triggers a "showpassage:after" event after each passage is shown. The **on()** function can be used to monitor for this event. Once it has occurred, a combination of the **html()** and **window.story.render()** functions can be used to display the dynamic contents of the Sidebar passage within the "sidebar" DIV element.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Left Sidebar in Snowman


:: UserScript [script]
/*
    Create the element to display the contents of the Sideb
*/
$(document.createElement('div'))
    .attr('id', 'sidebar')
    .insertBefore('#passage');

/*
    Monitor for the event that is triggered after the curre
*/
$(window).on('showpassage:after', function () {
    $('#sidebar').html(window.story.render("Sidebar"));
});


:: UserStylesheet[stylesheet]
#passage {
    margin-left: 20%;
}

#sidebar {
    position: fixed;
    top: 0;
    left: 0;
    width: 18%;
    height: 100%;
    margin: 0;
    padding: 0.5em;
    background-color: black;
    color: white;
}


:: Start
<% s.name = "Jane Doe"; s.location = "Work" %>
[[Another passage]]


:: Sidebar
Name: <%= s.name %><br>
Location: <%= s.location %>
```

```
:: Another passage
<% s.name = "John Smith"; s.location = "Shop" %>
[[Start]]
```

Download: Twee Code

"Limiting the Range of a Number": Chapbook (v1.0.0)

Summary

This example demonstrates how to limit a numeric variable to a value between a set range. This process is commonly known as clamping.

The example adds a **clamp()** function to the existing built-in *Math* global JavaScript object, which can then be called to achieve the desired result.

This added function is then used within the Vars Section to set a value before using an expression to show its value within the passage itself.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Limiting Range of Number

:: UserScript[script]
(function () {

    Object.defineProperty(Number.prototype, 'clamp', {
        configurable : true,
        writable     : true,

        value(/* min, max */) {
            if (this == null) { // lazy equality for null
                throw new TypeError('Number.prototype.clamp
            }

            if (arguments.length !== 2) {
                throw new Error('Number.prototype.clamp cal
            }

            var min = Number(arguments[0]);
            var max = Number(arguments[1]);

            if (min > max) {
                var tmp = min;
                min = max;
                max = tmp;
            }

            return Math.min(Math.max(this, min), max);
        }
    });


    /*
        Returns the given numerical clamped to the specifie

        Usage:
            → Limit numeric variable to a value between 1 a
            example: Math.clamp(variable, 1, 10)

            → Limit result of mathematical operation to a v
            variable: Math.clamp(variable + 5, 1, 10)
    */
    Object.defineProperty(Math, 'clamp', {
        configurable : true,
        writable     : true,

        value(num, min, max) {
            var value = Number(num);
            return Number.isNaN(value) ? NaN : value.clamp(
        }
    });

})();

:: Start
exampleNumber: 11
exampleResult: Math.clamp(exampleNumber, 1, 10)
--

Despite *exampleNumber* being {exampleNumber}, it will be '
```

Download: Twee Code

"Limiting the Range of a Number": Harlowe (v2.0)

Summary

This example demonstrates how to limit a numeric variable to a value between a set range. This process is commonly known as clamping.

The example adds a **clamp()** function to the existing built-in *Math* global JavaScript object, which can then be called to achieve the desired result.

Note: The example also adds a **clamp()** function to the built-in *Number* global JavaScript object, which the new **Math.clamp()** function uses internally. However, due to how Harlowe implements variables, the **.clamp()** function can't be used directly.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Limiting the range of a number in Harlowe


:: UserScript [script]
(function () {

    /*
        Returns the number clamped to the specified bounds.

        WARNING:
            Due to how Harlowe implements variables you car
            you must use the Math.clamp() function instead.
    */
    Object.defineProperty(Number.prototype, 'clamp', {
        configurable : true,
        writable     : true,

        value(/* min, max */) {
            if (this == null) { // lazy equality for null
                throw new TypeError('Number.prototype.clamp
            }

            if (arguments.length !== 2) {
                throw new Error('Number.prototype.clamp cal
            }

            var min = Number(arguments[0]);
            var max = Number(arguments[1]);

            if (min > max) {
                var tmp = min;
                min = max;
                max = tmp;
            }

            return Math.min(Math.max(this, min), max);
        }
    });


    /*
        Returns the given numerical clamped to the specifie

        Usage:
            → Limit numeric variable to a value between 1 a
            (set: $variable to Math.clamp($variable, 1, 10)
            → Limit result of mathematical operation to a v
            (set: $variable to Math.clamp($variable + 5, 1,
    */
    Object.defineProperty(Math, 'clamp', {
        configurable : true,
        writable     : true,

        value(num, min, max) {
            var value = Number(num);
            return Number.isNaN(value) ? NaN : value.clamp(
        }
    });

})();


:: Start
Initialise the numeric variable to a value with the range y
eg. between ''1'' and ''10'' inclusive.
&#40;note: You don't need to use the //Math.clamp()// funti

(set: $valueToClamp to 5)
''Current value'': $valueToClamp

Increase the number to a value that is ''within'' the desir
eg. Add 1 to the current value.\

(set: $valueToClamp to Math.clamp($valueToClamp + 1, 1, 10)
''New value'': $valueToClamp

Try to increase the number to a value that is ''outside'' t
eg. Add 100 to the current value.\

(set: $valueToClamp to Math.clamp($valueToClamp + 100, 1, 1
''New value'': $valueToClamp

Decrease the number to a value that is ''within'' the desir
eg. Minus 5 from the current value.\

(set: $valueToClamp to Math.clamp($valueToClamp - 5, 1, 10)
''New value'': $valueToClamp

Try to decrease the number to a value that is ''outside'' t
eg. Minus 100 from the current value.\

(set: $valueToClamp to Math.clamp($valueToClamp - 100, 1, 1
''New value'': $valueToClamp
```

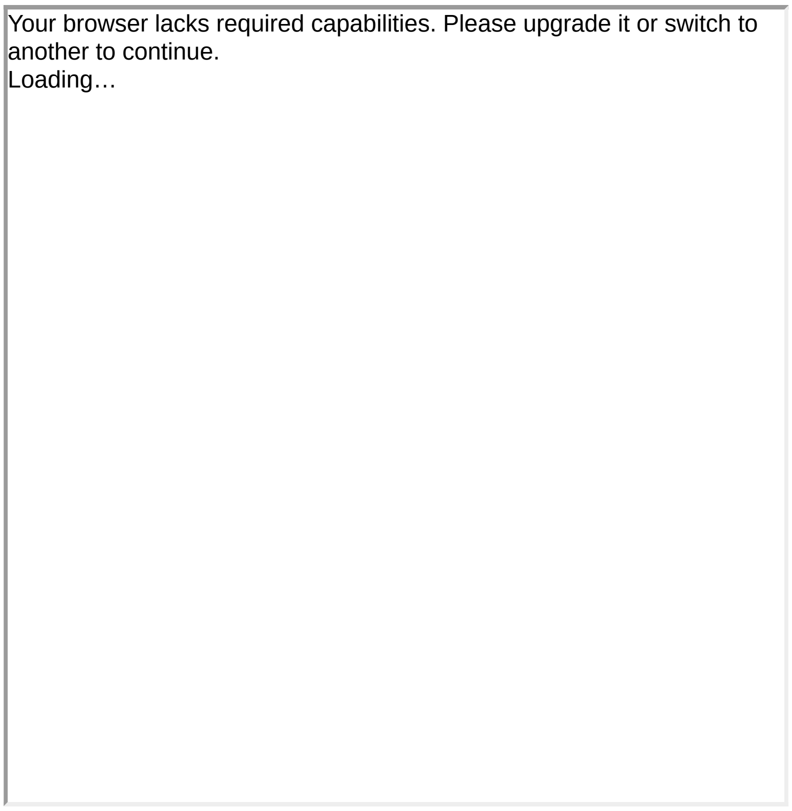
Download: Twee Code

"Limiting the Range of a Number": SugarCube (v2.18)

Summary

This example demonstrates how to limit a numeric variable to a value between a set range, this process is commonly known as clamping. It uses the **Math.clamp()** function in SugarCube to achieve the desired result.

Live Example



Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Limiting the range of a number in SugarCube


:: Start
Initialise the numeric variable to a value with the range y
eg. between ''1'' and ''10'' inclusive.
(note: You don't need to use the //Math.clamp()// funtion a

<<set $valueToClamp to 5>>
''Current value'': $valueToClamp

Increase the number to a value that is ''within'' the desir
eg. Add 1 to the current value.\

<<set $valueToClamp to Math.clamp($valueToClamp + 1, 1, 10)
''New value'': $valueToClamp

Try to increase the number to a value that is ''outside'' t
eg. Add 100 to the current value.\

<<set $valueToClamp to Math.clamp($valueToClamp + 100, 1, 1
''New value'': $valueToClamp

Decrease the number to a value that is ''within'' the desir
eg. Minus 5 from the current value.\

<<set $valueToClamp to Math.clamp($valueToClamp - 5, 1, 10)
''New value'': $valueToClamp

Try to decrease the number to a value that is ''outside'' t
eg. Minus 100 from the current value.\

<<set $valueToClamp to Math.clamp($valueToClamp - 100, 1, 1
''New value'': $valueToClamp
```

Download: Twee Code

"Limiting the Range of a Number": Snowman (v1.3)

Summary

This example demonstrates how to limit a numeric variable to a value between a set range, this process is commonly known as clamping.

The example adds a **clamp()** function to the existing built-in *Math* global JavaScript object, which can then be called to achieve the desired result. It also adds a **clamp()** function to the built-in *Number* global JavaScript object that the new *Math.clamp()* function uses internally.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Limiting the range of a number in Snowman


:: UserScript [script]
(function () {

    /*
        Returns the number clamped to the specified bounds.

        Usage:
            → Limit numeric variable to a value between 1 a
            <% s.variable = s.variable.clamp(1, 10) %>
    */
    Object.defineProperty(Number.prototype, 'clamp', {
        configurable : true,
        writable     : true,

        value(/* min, max */) {
            if (this == null) { // lazy equality for null
                throw new TypeError('Number.prototype.clamp
            }

            if (arguments.length !== 2) {
                throw new Error('Number.prototype.clamp cal
            }

            var min = Number(arguments[0]);
            var max = Number(arguments[1]);

            if (min > max) {
                var tmp = min;
                min = max;
                max = tmp;
            }

            return Math.min(Math.max(this, min), max);
        }
    });


    /*
        Returns the given numerical clamped to the specifie

        Usage:
            → Limit numeric variable to a value between 1 a
            <% s.variable = Math.clamp(s.variable, 1, 10) %
            → Limit result of mathematical operation to a v
            <% s.variable = Math.clamp(s.variable + 5, 1, 1
    */
    Object.defineProperty(Math, 'clamp', {
        configurable : true,
        writable     : true,

        value(num, min, max) {
            var value = Number(num);
            return Number.isNaN(value) ? NaN : value.clamp(
        }
    });

})();


:: Start
Initialise the numeric variable to a value with the range y
eg. between <b>1</b> and <b>10</b> inclusive.<br>
&#40;note: You don't need to use the <i>Math.clamp()</i> fu

<% s.valueToClamp = 5 %>
<b>Current value</b>: <%= s.valueToClamp %>

Increase the number to a value that is <b>within</b> the de
eg. Add 1 to the current value.

<% s.valueToClamp = Math.clamp(s.valueToClamp + 1, 1, 10) %
<b>New value</b>: <%= s.valueToClamp %>

Try to increase the number to a value that is <b>outside</b
eg. Add 100 to the current value.

<% s.valueToClamp = Math.clamp(s.valueToClamp + 100, 1, 10)
<b>New value</b>: <%= s.valueToClamp %>

Decrease the number to a value that is <b>within</b> the de
eg. Minus 5 from the current value.

<% s.valueToClamp = Math.clamp(s.valueToClamp - 5, 1, 10) %
<b>New value</b>: <%= s.valueToClamp %>

Try to decrease the number to a value that is <b>outside</b
eg. Minus 100 from the current value.

<% s.valueToClamp = Math.clamp(s.valueToClamp - 100, 1, 10)
<b>New value</b>: <%= s.valueToClamp %>
```

Download: Twee Code

"Loading Screen": SugarCube (v2.18)

Summary

"Loading Screen" demonstrates how the **LockScreen.lock()** and **LockScreen.unlock()** functions work in SugarCube. (This example also uses the **setTimeout()** JavaScript function.)

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Loading Screen in SugarCube

:: UserScript[script]
// Lock the screen and save the ID
var lockID = LoadScreen.lock();

// Pause for 5 second before unlocking the screen
setTimeout(function(){
        LoadScreen.unlock(lockID);
}, 5000);

:: Start
You can now see this after the long pause!
```

Download: Twee Code

"Lock and Key: Variable": Chapbook (v1.0.0)

Summary

"Lock and Key: Variable" demonstrates how to create the effect of picking up a key and unlocking a door. In this example, the key is a variable (*key*) and is initially set to the value "false" in the Start passage.

When the link "Pick up key" is clicked in the "Back Room" passage, *key* is changed to the value "true" via embedding the passage "Key". When the passage is visited and *key* is set to the value of "true", door link changes from its initial response of "Locked Door" to "Unlock the door".

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Lock and Key: Variable

:: Start
key: false
--

Rooms:

[[Front Room]]

[[Back Room]]


:: Front Room
[if key == true]
    [[Unlock the door->Exit]]
[else]
    *Locked Door*
[continued]

Rooms:

[[Back Room]]


:: Back Room
[if key == false]
    Items:
    {reveal link: 'Pick up key', passage: 'Key'}
[else]
    There is nothing here.
[continued]

Rooms:

[[Front Room]]

:: Exit
You found the key and went through the door!

:: Key
key: true
--
You picked up the key!
```

Download: Twee Code

"Lock and Key: Variable": Harlowe (v2.0)

Note: This recipe is affected by history changes in the story. Undoing or re-doing back to a passage containing this recipe has the potential to change its saved values.

Summary

"Lock and Key: Variable" demonstrates how to create the effect of picking up a key and unlocking a door. In this example, the key is a variable (*$key*) and is initially set to the value "false" in the Start passage.

When the link "Pick up the key" is clicked, *$key* is changed to the value "true" and the door link changes from its initial response of "Locked Door" to a link to the passage Exit.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Lock and Key: Variable in Harlowe

:: Start
(set: $key to false)

Rooms:
[[Front Room]]
[[Back Room]]

:: Front Room
(if: $key is true)[
    [[Exit]]
]
(else:)[
    *Locked Door*
]

Rooms:
[[Back Room]]

:: Back Room
(if: $key is false)[
    Items:
    (link: "Pick up key")[(set: $key to true)You have a key
]
(else:)[
    There is nothing here.
]

Rooms:
[[Front Room]]

:: Exit
You found the key and went through the door!
```

Download: Twee Code

See Also

Setting and Showing Variables

"Lock and Key: Variable": SugarCube (v2.18)

Note: This recipe is affected by history changes in the story. Undoing or re-doing back to a passage containing this recipe has the potential to change its saved values.

Summary

"Lock and Key: Variable" demonstrates how to create the effect of picking up a key and unlocking a door. In this example, the key is a variable (*$key*) and is initially set to the value *false* in the Start passage.

When the link (created using a `<<linkreplace>>` macro) "Pick up the key" is clicked, *$key* is changed to the value *true* and the door link changes from its initial response of "Locked Door" to a link to the passage Exit.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Lock and Key: Variable in SugarCube

:: Start
<<set $key to false>>

Rooms:
[[Back Room]]
[[Front Room]]

:: Back Room
<<if $key is false>>
    Items:
    <<linkreplace "Pick up the key">><<set $key to true>>Yo
<<else>>
    There is nothing here.
<</if>>

Rooms:
[[Front Room]]

:: Front Room
<<if $key is true>>
    [[Exit]]
<<else>>
    Locked Door
<</if>>

Rooms:
[[Back Room]]

:: Exit
You found the key and went through the door!
```

Download: Twee Code

See Also

Setting and Showing Variables

"Lock and Key: Variable": Snowman (v1.3)

Summary

"Lock and Key: Variable" demonstrates how to create the effect of picking up a key and unlocking a door. In this example, the key is a variable (*s.key*) and does not initially exist in the Start passage.

When the link "Pick up the key" is clicked, *s.key* is changed to the value "true" and the door link changes from its initial response of "Locked Door" to a link to the passage Exit.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Lock and Key: Variable in Snowman

:: Start
Rooms:
- [[Front Room]]
- [[Back Room]]

:: Front Room
<% if (s.key) { %>
[[Exit]]
<% } else { %>
*Locked Door*
<% } %>

Rooms:
- [[Back Room]]

:: Back Room
<% if (!s.key) { %>
Items:
- <a href="javascript:void(0)" class="key-item">Pick up key
<% } else { %>
There is nothing here.
<% } %>

<%
$(function() {
    $('.key-item').click(function() {
        s.key = true;
        $(this).replaceWith('<span>You have a key.</span>')
    });
});
%>

Rooms:
- [[Front Room]]

:: Exit
You found the key and went through the door!
```

Download: Twee Code

See Also

Setting and Showing Variables

"Looping": Chapbook (v1.0.0)

Summary

In programming terminology, a "loop" is a common technique for iterating, moving through one by one, some type of data.

In Chapbook, the modifier `[JavaScript]` allows for using JavaScript inside a passage. Through using the **forEach()** function of Arrays and the **write()** function supplied by Chapbook, each entry within an array can be shown.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Looping

:: Start
exampleArray: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
--

The values of the array are:
<ul>
[JavaScript]
exampleArray.forEach(function(value, index){
    write("<li>" + value + "</li>");
});
[continued]
</ul>
```

Download: Twee Code

"Looping": Harlowe (v2.0)

Summary

In programming terminology, a "loop" is a common technique for iterating, moving through one by one, some type of data. In Harlowe, the macros `(loop:)` and `(for:)` provide this functionality. Combined with the keywords `each`, to move through all entries, or "where," to specify some condition, they allow for "looping" through data structures like arrays or datamaps.

In this example, the variable *arrayInventory* is set to the value of an array containing the strings "Bread", "Pan", and "Book". The `(for:)` macro is used with the keyword `each` to set the values contained in the array to the temporary variable *_temp* for each value of the spread out array. The text contained in the associated hook to the `(for:)` macro is shown each loop with the value of *_temp* changed for each value in the array.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Looping in Harlowe

:: Start
<!-- Create an array of the strings "Bread", "Pan", "Book"
(set: $arrayInventory to (a: "Bread", "Pan", "Book") )

<!-- For each entry in the expanded array in turn, -->
<!--  set the entry to the temporary variable _temp -->
(for: each _temp, ...$arrayInventory)[
You have _temp.]
```

Download: Twee Code

"Looping": SugarCube (v2.18)

Summary

In programming terminology, a "loop" is a common technique for iterating, moving through one by one, some type of data. In SugarCube, the control macro `<<for>>` provides this functionality. It acts like the **for** keyword in JavaScript and its usage works in a similar way.

In this example, the array *arrayInventory* is set to the series of strings "Bread", "Pan", and "Book". Using the `<<for>>` macro, a temporary variable is set to 0 and increased for each loop until its value is no longer less than the length (number of entries) in the array. Inside the macro, the text is shown each time with the value of the entry matching the position of the value of *_i* in the array substituted.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Looping in SugarCube

:: Start
<!-- Set the variable $arrayInventory to the array containi
<<set $arrayInventory to ["Bread", "Pan", "Book"]>>

<!-- Set the temporary variable _l to 0 and increase it unt
<<for _i to 0; _i lt $arrayInventory.length; _i++>>
You have $arrayInventory[_i]
<</for>>
```

Download: Twee Code

"Looping": Snowman (v1.3.0)

Summary

In programming terminology, a "loop" is a common technique for iterating, moving through one by one, some type of data. Because Snowman does not provide macros, the existing JavaScript **for** keyword can be used to create loops. Since Snowman also includes the Underscore.js and jQuery libraries, the **_.each()** and **jQuery.each()** functions can also be used.

In this example, the *s* global shortcut to the *window.story.state* variable used. A new property called "arrayInventory" is set to the series of values "Bread", "Pan", and "Book". The first example uses the JavaScript **for** keyword to move through the values. The second example uses the **_.each()** function in Underscore.js, and the third uses the **jQuery.each()** function for the same purpose.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Looping in Snowman

:: Start
<%
// An array of the strings "Bread", "Pan", "Book"
s.arrayInventory = ["Bread", "Pan", "Book"];

// An example using JavaScript
for (var i = 0; i < s.arrayInventory.length; i++){
 %>You have <%= s.arrayInventory[i] %>.<br> <%
}
%>

<hr>
<%
// An example using Underscore.js
_.each(s.arrayInventory, function(item) {
    %>You have <%= item %>.<br> <%
});
%>

<hr>
<%
// An example using jQuery
jQuery.each(s.arrayInventory, function( index, value ) {
   %>You have <%= value %>.<br> <%
});

%>
```

Download: Twee Code

"Modal (Pop-up Window)": Harlowe (v2.0)

Summary

This example creates a re-usable modal window. It can be opened using the combination of `(link-repeat:)` and `(replace:)` to create the window in an existing hook, and be 'closed' using the same macros to remove the window. CSS rules are applied with `(css:)` to style the modal, and to change an enclosing hook into a "dimmer" which obscures the rest of the page.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Modal

:: Header[header]
|modalhooks>[]

:: Modal code
(replace: ?modalhooks)[{
  (css:"
    position: fixed;
    display:block;
    z-index: 1;
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: rgba(0,0,0,0.4);
  ")[
    (css:"
      display:block;
      margin: 15% auto;
      padding: 20px;
      width: 80%;
      border: 1px solid white;
    ")|modal>[
      (css:"float:right")+(link-repeat:"×")[(replace: ?moda
    ]
  ]
}]

:: Start
(link-repeat:"Open Modal!")[(display:"Modal code")(append:?
```

Download: Twee Code

"Modal (Pop-up Window)": SugarCube (v2.18)

Summary

This example uses the built-in *Dialog* object to **setup()**, add content (**wiki()**), and finally **open()** the dialog window. SugarCube also comes with additional functionality to adjust other settings.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Modal

:: Start
<<link "Open dialog!">>
    <<script>>
        Dialog.setup("Dialog");
        Dialog.wiki("Text within the dialog window");
        Dialog.open();
    <</script>>
<</link>>
```

Download: Twee Code

"Modal (Pop-up Window)": Snowman (1.3.0)

Summary

This example uses the jQuery **click()** and **show()/hide()** functions to watch for the user clicking on a button (to open) or "X" (to close). Additional CSS rules are used to create the effect of having the content 'behind' the modal window be darkned and unusable until the modal itself is closed.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Modal

:: UserScript[script]
$(function(){

        // When the user clicks the button, open the modal
        $("#myBtn").click(function() {
              $("#myModal").show();
        });

        // When the user clicks on <span> (x), close the mo
        $(".close").click(function() {
              $("#myModal").hide();
        });

});


:: UserStylesheet[stylesheet]
/* The Modal (background) */
.modal {
    display: none; /* Hidden by default */
    position: fixed; /* Stay in place */
    z-index: 1; /* Sit on top */
    padding-top: 100px; /* Location of the box */
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: rgb(0,0,0); /* Fallback color */
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity
}

/* Modal Content */
.modal-content {
    background-color: #fefefe;
    margin: auto;
    padding: 20px;
    border: 1px solid #888;
    width: 80%;
}

/* The Close Button */
.close {
    color: #aaaaaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}

.close:hover,
.close:focus {
    color: #000;
    text-decoration: none;
    cursor: pointer;
}


:: Start
<button id="myBtn">Open Modal</button>

<div id="myModal" class="modal">
  <div class="modal-content">
    <span class="close">×</span>
    <p>Example text in the modal</p>
  </div>
</div>
```

Download: Twee Code

"Modularity": Harlowe (v2.0)

Summary

In programming terminology, modularity refers to dividing software into different sections related to their purpose or to better organize the whole. In Harlowe, this technique can be used through the `(display:)` macro to print the contents of one passage in another. Parts of a story can often be re-used in this way.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Modularity in Harlowe

:: Start
(set: $lineOne to "Give us a verse")
(set: $lineTwo to "Drop some knowledge")

(display: "showLineOne")
(display: "showLineTwo")

:: showLineOne
$lineOne

:: showLineTwo
$lineTwo
```

Download: Twee Code

"Modularity": SugarCube (v2.18)

Summary

In programming terminology, modularity refers to dividing software into different sections related to their purpose or to better organize the whole. In SugarCube, this technique can be used through the `<<include>>` macro to print the contents of one passage in another. Parts of a story can often be re-used in this way.

The `<<widget>>` macro offers a simplified way of creating new, custom macros using other SugarCube macros and TwineScript instead of JavaScript. When compared to `<<include>>`, widgets have the advantage of accepting arguments and expressions similar to the way other SugarCube macros can. New widgets must be added through passages with the tag `widget`.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Modularity in SugarCube

:: Start
<<set $lineOne to "Give us a verse">>
<<set $lineTwo to "Drop some knowledge">>

<<include "showLineOne">>
<<include "showLineTwo">>

<<showLine 1>>
<<showLine 2>>

:: showLineWidget [widget]
<<widget 'showLine'>>\
    <<nobr>>
        <<if $args[0] is 1>>
            $lineOne
        <<elseif $args[0] is 2>>
            $lineTwo
        <</if>>
    <</nobr>>\
<</widget>>

:: showLineOne
$lineOne

:: showLineTwo
$lineTwo
```

Download: Twee Code

"Modularity": Snowman (v1.3.0)

Summary

In programming terminology, modularity refers to dividing software into different sections related to their purpose or to better organize the whole. In Snowman, this technique can be used through the **window.story.render()** function to print the contents of one passage in another. Parts of a story can often be re-used in this way.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Modularity in Snowman

:: Start
<%
    s.lineOne = "Give us a verse";
    s.lineTwo = "Drop some knowledge";
%>

<%= window.story.render("showLineOne") %>
<%= window.story.render("showLineTwo") %>


:: showLineOne
<%= s.lineOne %>

:: showLineTwo
<%= s.lineTwo %>
```

Download: Twee Code

"Moving through a 'dungeon'": Harlowe (v2.0)

Summary

"Moving through a 'dungeon'" uses the `(array:)` macro to create a multidimensional array. Movement positions are then tracked through X and Y variables for a grid system. Each movement subtracts or adds to its cooresponding X or Y position and is compared to those same positions within the array. Different directions are shown if movement is possible in that direction.

A map of the array is created by iterating through temporary variables and placing different symbols matching walls, movement spaces, and the player herself.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Moving through Dungeons


:: User Style [stylesheet]
tw-include[type="startup"], tw-hook[name="workarea"] {
    display: none;
}
tw-hook[name="map"] {
    font-family: monospace;
    line-height: 1.75;
    font-size: 16pt;
}


:: Start
|map>[(display: "Map")]


:: Map
(set: $map to "")\
|workarea>[
    (for: each _y, ...(range: 1, $dungeon's length))[
        (for: each _x, ...(range: 1, $dungeon's (_y)'s leng
            (if: _x is $positionX and _y is $positionY)[
                (set: $map to it + "P ")
            ]
            (else-if: $dungeon's (_y)'s (_x) is 0)[
                (set: $map to it + "&num; ")
            ]
            (else-if: $dungeon's (_y)'s (_x) is 1)[
                (set: $map to it + ". ")
            ]
            (else-if: $dungeon's (_y)'s (_x) is 2)[
                (set: $map to it + "E ")
            ]
        ]
        (set: $map to it + " <br>")
    ]
]\
$map

{
    (set: $seperator to "")\
    (set: _north to $dungeon's ($positionY - 1)'s ($positio
    (set: _east to $dungeon's ($positionY)'s ($positionX +
    (set: _south to $dungeon's ($positionY + 1)'s ($positio
    (set: _west to $dungeon's ($positionY)'s ($positionX -

    (if: _north is 1)[
        $seperator
        (link: "North")[
            (set: $positionY to it - 1)
            (replace: ?map)[(display: "Map")]
        ]
        (set: $seperator to " | ")
    ]
    (else-if: _north is 2)[
        $seperator[[Exit]]
        (set: $seperator to " | ")
    ]

    (if: _east is 1)[
        $seperator
        (link: "East")[
            (set: $positionX to it + 1)
            (replace: ?map)[(display: "Map")]
        ]
        (set: $seperator to " | ")
    ]
    (else-if: _east is 2)[
        $seperator[[Exit]]
        (set: $seperator to " | ")
    ]

    (if: _south is 1)[
        $seperator
        (link: "South")[
            (set: $positionY to it + 1)
            (replace: ?map)[(display: "Map")]
        ]
        (set: $seperator to " | ")
    ]
    (else-if: _south is 2)[
        $seperator[[Exit]]
        (set: $seperator to " | ")
    ]

    (if: _west is 1)[
        $seperator
        (link: "West")[
            (set: $positionX to it - 1)
            (replace: ?map)[(display: "Map")]
        ]
    ]
    (else-if: _west is 2)[
        $seperator[[Exit]]
    ]
}


:: Startup [startup]
{
    (set: $dungeon to (array:
            (a: 0,0,0,0,0,0,0,0,0,0,0),
            (a: 0,1,1,1,0,1,1,1,1,1,0),
            (a: 0,0,0,1,0,0,0,0,0,1,0),
            (a: 0,1,0,1,1,1,1,1,0,1,0),
            (a: 0,1,0,0,0,0,0,1,0,1,0),
            (a: 0,1,1,1,1,1,1,1,0,1,0),
            (a: 0,0,0,0,0,0,0,1,0,1,0),
            (a: 0,1,0,1,1,1,1,1,1,1,0),
            (a: 0,1,0,1,0,0,0,1,0,0,0),
            (a: 0,1,1,1,0,1,1,1,1,2,0),
            (a: 0,0,0,0,0,0,0,0,0,0,0)
        )
    )
    (set: $positionX to 2)
    (set: $positionY to 2)
}


:: Exit
You have exited the map.
```

Download: Twee Code

See Also

Setting and Showing Variables, Conditional Statements, Modularity

"Moving through a 'dungeon'": SugarCube (v2.0)

Summary

"Moving through a 'dungeon'" uses a two-dimensional array for the "map" and two variables, X and Y, to track movement through the space. The 'Map System' passage checks the positions of X and Y relative to the "map" and writes the available directional movement options. Once a direction is clicked, the X and Y values are added or subtracted corresponding to the direction and the map is re-drawn again. Symbols are then placed on the map matching the walls, movement space, and player.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Moving through a Dungeon

:: UserStylesheet[stylesheet]
#map {
    font-family: monospace;
}

:: StoryInit
<<set $mapArray to
[[0,0,0,0,0,0,0,0,0,0,0],
[0,1,1,1,0,1,1,1,1,1,0],
[0,0,0,1,0,0,0,0,0,1,0],
[0,1,0,1,1,1,1,1,0,1,0],
[0,1,0,0,0,0,0,1,0,1,0],
[0,1,1,1,1,1,1,1,0,1,0],
[0,0,0,0,0,0,0,1,0,1,0],
[0,1,0,1,1,1,1,1,1,1,0],
[0,1,0,1,0,0,0,1,0,0,0],
[0,1,1,1,0,1,1,1,1,2,0],
[0,0,0,0,0,0,0,0,0,0,0]]>>

<<set $positionX to 1>>
<<set $positionY to 1>>

:: Location
<span id="map">
<<nobr>>
<<for $i to 0; $i lt $mapArray.length; $i++>>
    <<for $k to 0; $k lt $mapArray[$i].length; $k++>>
        <<if $k eq $positionX and $i eq $positionY>>
            <<print "P">>
        <<elseif $mapArray[$i][$k] eq 1>>
            <<print ".">>
        <<elseif $mapArray[$i][$k] eq 0>>
            <<print "#">>
        <<elseif $mapArray[$i][$k] eq 2>>
            <<print "E">>
        <</if>>
    <</for>>
    <<print "<br>">>
<</for>>
<</nobr>>
</span>

:: East
<<set $positionX += 1>>
<<include "Map System">>
```

```
:: West
<<set $positionX -= 1>>
<<include "Map System">>

:: South
<<set $positionY += 1>>
<<include "Map System">>

:: North
<<set $positionY -= 1>>
<<include "Map System">>

:: Map System
<<include "Location">>
<<nobr>>
<<if $mapArray[$positionY-1][$positionX] eq 1>>
[[North]] |
<<elseif $mapArray[$positionY-1][$positionX] eq 2>>
[[Exit]] |
<</if>>
<<if $mapArray[$positionY][$positionX+1] eq 1>>
[[East]] |
<<elseif $mapArray[$positionY][$positionX+1] eq 2>>
[[Exit]] |
<</if>>
<<if $mapArray[$positionY+1][$positionX] eq 1>>
[[South]] |
<<elseif $mapArray[$positionY+1][$positionX] eq 2>>
[[Exit]] |
<</if>>
<<if $mapArray[$positionY][$positionX-1] eq 1>>
[[West]] |
<<elseif $mapArray[$positionY][$positionX-1] eq 2>>
[[Exit]] |
<</if>>
<</nobr>>

:: Exit
Double-click this passage to edit it.
```

Download: Twee Code

See Also

Setting and Showing Variables, Conditional Statements, Modularity

"Moving through a 'dungeon'": Snowman (1.3.0)

Summary

"Moving through a 'dungeon'" creates a multidimensional array. Movement positions are then tracked through X and Y variables for a grid system. Each movement subtracts or adds to its cooresponding X or Y position and is compared to those same positions within the array. Different directions are shown if movement is possible in that direction.

Live Example

Download: Live Example

Twee Code

```
:: Start
<div class="maze"></div>

<button type="button" data-move="n">North</button>
<button type="button" data-move="s">South</button>
<button type="button" data-move="e">East</button>
<button type="button" data-move="w">West</button>

<%
/* 0s are walls, 1 are spaces, 2 is the goal. */
var maze =
[
[0,0,0,0,0,0,0,0,0,0,0],
[0,1,1,1,0,1,1,1,1,1,0],
[0,0,0,1,0,0,0,0,0,1,0],
[0,1,0,1,1,1,1,1,0,1,0],
[0,1,0,0,0,0,0,1,0,1,0],
[0,1,1,1,1,1,1,1,0,1,0],
[0,0,0,0,0,0,0,1,0,1,0],
[0,1,0,1,1,1,1,1,1,1,0],
[0,1,0,1,0,0,0,1,0,0,0],
[0,1,1,1,0,1,1,1,1,2,0],
[0,0,0,0,0,0,0,0,0,0,0]
];

/* Where the player starts. The top left is (0, 0). */

var positionX = 1, positionY = 1;

function renderMaze() {
    /* Transform the maze into ASCII art. */

    /* What characters we use to display the maze. */
    var displayChars = ['#', '.', 'E'];

    $('.maze').html(maze.map(function(row, renderY) {
        return row.reduceRight(function(html, cell, renderX
            if (renderX === positionX && renderY === positi
                return 'P' + html;
            }

            return displayChars[cell] + html;
        }, '<br>');
    }));
}

function updateMoves() {
    /*
    Enable/disable buttons to move based on what's allowed.
    We take advantage of the fact that both 0 and undefinec
    (outside the maze) are converted to false by JavaScript
    ! operator.
    */

    $('[data-move="n"]').attr('disabled', !maze[positionY -
    $('[data-move="s"]').attr('disabled', !maze[positionY +
    $('[data-move="e"]').attr('disabled', !maze[positionY][
    $('[data-move="w"]').attr('disabled', !maze[positionY][
}

$(function() {
    renderMaze();
    updateMoves();

    /*
    Change position when the user clicks an appropriate lir
    We depend on updateMoves() to prevent the user from wal
    through a wall.
    */

    $('[data-move]').click(function() {
        var direction = $(this).data('move');

        switch (direction) {
            case 'n':
                positionY--;
                break;
            case 's':
                positionY++;
                break;
            case 'e':
                positionX++;
                break;
            case 'w':
                positionX--;
                break;
            default:
                throw new Error('Don\'t know how to move '
        }

        if (maze[positionY][positionX] === 2) {
            story.show('Exit');
        }
        else {
            renderMaze();
            updateMoves();
        }
    });
});

%>


:: Exit
You've escaped this fiendish maze!
```

Download: Twee Code

Setting and Showing Variables, Conditional Statements

"Moving through a 'dungeon'": Sugarcane (v2.0)

Note: The following example is designed for Twine 1.4.2.

Summary

"Moving through a 'dungeon'" uses an array of arrays to track positions 'moved' through using X and Y variables. It also creates a `<<navigate>>` macro that handles the showing of directions.

Live Example

Download: Live Example

Twee Code

```
:: North
<<set $posy = $posy - 1>>
<<navigate>>
<<if $North eq 1>>
[[North]]
<<endif>>
<<if $South eq 1>>
[[South]]
<<endif>>
<<if $West eq 1>>
[[West]]
<<endif>>
<<if $East eq 1>>
[[East]]
<<endif>>
<<if $Exit eq 1>>
[[Exit]]
<<endif>>


:: StoryTitle
Sugarcane: Moving through a 'Dungeon'


:: Start
<<display "Maze Addon">>

[[Enter Dungeon]]


:: Maze Addon
<<silently>>
<<set $MazeAddon =
function()
{

   var maze = [[0,0,0,0,0,0,0,0,0,0,0],
[0,1,1,1,0,1,1,1,1,1,0],
[0,0,0,1,0,0,0,0,0,1,0],
[0,1,0,1,1,1,1,1,0,1,0],
[0,1,0,0,0,0,0,1,0,1,0],
[0,1,1,1,1,1,1,1,0,1,0],
[0,0,0,0,0,0,0,1,0,1,0],
[0,1,0,1,1,1,1,1,1,1,0],
[0,1,0,1,0,0,0,1,0,0,0],
[0,1,1,1,0,1,1,1,1,2,0],
[0,0,0,0,0,0,0,0,0,0,0]];
```

```
  var x = 1;
  var y = 1;

  $posx = 1;
  $posy = 1;

  macros['navigate'] =
    {
        handler: function(obj, fnc, val)
        {
            x = $posx; y = $posy;
         if(maze[y-1][x] eq 1)
         { $North = 1; }
         else if(maze[x][y+1] eq 2) {$Exit = 1;}
         else {$North = 0;}

         if(maze[y+1][x] eq 1)
         { $South = 1; }
         else if(maze[x][y-1] eq 2) {$Exit = 1;}
         else {$South = 0;}

         if(maze[y][x-1] eq 1)
         { $West = 1; }
         else if(maze[x-1][y] eq 2) {$Exit = 1;}
         else {$West = 0;}

         if(maze[y][x+1] eq 1)
         { $East = 1; }
         else if(maze[x+1][y] eq 2) {$Exit = 1;}
         else {$East = 0;}
        }
    }


}
>>
<<print $MazeAddon()>>
<<endsilently>>


:: StoryAuthor
@videlais



:: West
<<set $posx = $posx - 1>>
<<navigate>>
<<if $North eq 1>>
```

```
[[North]]
<<endif>>
<<if $South eq 1>>
[[South]]
<<endif>>
<<if $West eq 1>>
[[West]]
<<endif>>
<<if $East eq 1>>
[[East]]
<<endif>>
<<if $Exit eq 1>>
[[Exit]]
<<endif>>


:: East
<<set $posx = $posx + 1>>
<<navigate>>
<<if $North eq 1>>
[[North]]
<<endif>>
<<if $South eq 1>>
[[South]]
<<endif>>
<<if $West eq 1>>
[[West]]
<<endif>>
<<if $East eq 1>>
[[East]]
<<endif>>
<<if $Exit eq 1>>
[[Exit]]
<<endif>>


:: Enter Dungeon
<<navigate>>
<<if $North eq true>>
[[North]]
<<endif>>
<<if $South eq true>>
[[South]]
<<endif>>
<<if $West eq true>>
[[West]]
<<endif>>
<<if $East eq true>>
```

```
[[East]]
<<endif>>


:: South
<<set $posy = $posy + 1>>
<<navigate>>
<<if $North eq 1>>
[[North]]
<<endif>>
<<if $South eq 1>>
[[South]]
<<endif>>
<<if $West eq 1>>
[[West]]
<<endif>>
<<if $East eq 1>>
[[East]]
<<endif>>
<<if $Exit eq 1>>
[[Exit]]
<<endif>>


:: Exit
```

Download: Twee Code

"Passage Events": SugarCube (v2.18)

Summary

SugarCube triggers different events as they happen to passages. Through using jQuery and its own JavaScript event handling, code can be added to work with other, existing functionality.

In this example, the "passagestart" and "passagerender" events are shown. In the event progression, the 'passagestart' event occurs before the PassageHeader prepending its content and the 'passagerender' happens after the PassageFooter. Through using event handling, content is added before the PassageHeader and after the PassageFooter as an example of when events occur in presenting passages.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Passage Events in SugarCube

:: UserScript[script]
/*
    Prepend the content of the passage "New Header" to ever

        This demonstrates that the 'passagestart' event com
         the PassageHeader prepending.
*/
$(document).on(':passagestart', function (eventObject) {
      var headerContent = Story.get("New Header").processTe
    $(eventObject.content).wiki(headerContent);
});

/*
    Append the content of the passage "New Footer" to every

        This demonstrates that the 'passagerender' event cc
         the PassageFooter appending.
*/
$(document).on(':passagerender', function (eventObject) {
      var footerContent = Story.get("New Footer").processTe
    $(eventObject.content).wiki(footerContent);
});

:: Start
[[Another passage]]

:: Another passage
[[Back to beginning|Start]]

:: New Header
This is added before the PassageHeader!


:: PassageHeader
This is the PassageHeader.


:: PassageFooter

This is the PassageFooter!


:: New Footer
This is added after the PassageFooter!
```

Download: Twee Code

See Also

Headers and Footers

"Passage Events": Snowman (v1.4.0)

Summary

Snowman triggers different events as they happen to passages.

In this example, a header and footer is created by listening for the 'shown.sm.passage' event with a jQuery event handle and then prepending the content of the passage "Header" and appending the content of the passage "Footer" to the current passage after it has been initially rendered.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Passage Events

:: UserScript[script]
/*
    Prepend the content of the passage "Header" to every pa
        Append the content of the passage "Footer" to every
*/
$(window).on('shown.sm.passage', function (eventObject, pas
        var headerContent = window.story.render("Header");
        var currentContent = passageObject.passage.render()
        var footerContent = window.story.render("Footer");

        $('#main').html(headerContent + currentContent + fo
});

:: Start
[[Another Passage]]

:: Another Passage
[[Back to Beginning|Start]]

:: Header
This is the header!

:: Footer
This is the footer!
```

Download: Twee Code

See Also

Headers and Footers

"Passage Events": Snowman (v2.0.0)

Summary

Snowman triggers different events as they happen to passages.

In this example, a header and footer is created by listening for the 'sm.passage.shown' event with a jQuery event handle and then prepending the content of the passage "Header" and appending the content of the passage "Footer" to the current passage after it has been initially rendered.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman 2: Passage Events

:: UserScript[script]
/*
    Prepend the content of the passage "Header" to every pa
        Append the content of the passage "Footer" to every
*/
$(window).on('sm.passage.shown', function (eventObject, pas
        var headerContent = window.story.render("Header");
        var currentContent = passageObject.passage.render()
        var footerContent = window.story.render("Footer");

        $('tw-passage').html(headerContent + currentContent
});

:: Start
<p>[[Another Passage]]</p>

:: Another Passage
[[Back to beginning->Start]]

:: Header
This is the header!

:: Footer
This is the footer!
```

Download: Twee Code

See Also

Headers and Footers

"Passages in Passages": Chapbook (v1.0.0)

Summary

Using the insert `{embed passage: }`, Chapbook can include one passage in another.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Passages in Passages

:: Start
This is the Start passage!
{embed passage: 'Another'}

:: Another
And this is another passage!
```

Download: Twee Code

"Passages in Passages": Harlowe (v1.0)

Summary

The Harlowe story format allows for content in one passage to be displayed in another through the use of the `(display:)` macro. Given the name of an existing passage, its contents will added wherever the macro is called.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Passages in Passages

:: Start
This is the Start passage!
(display: "Another")

:: Another
And this is another passage!
```

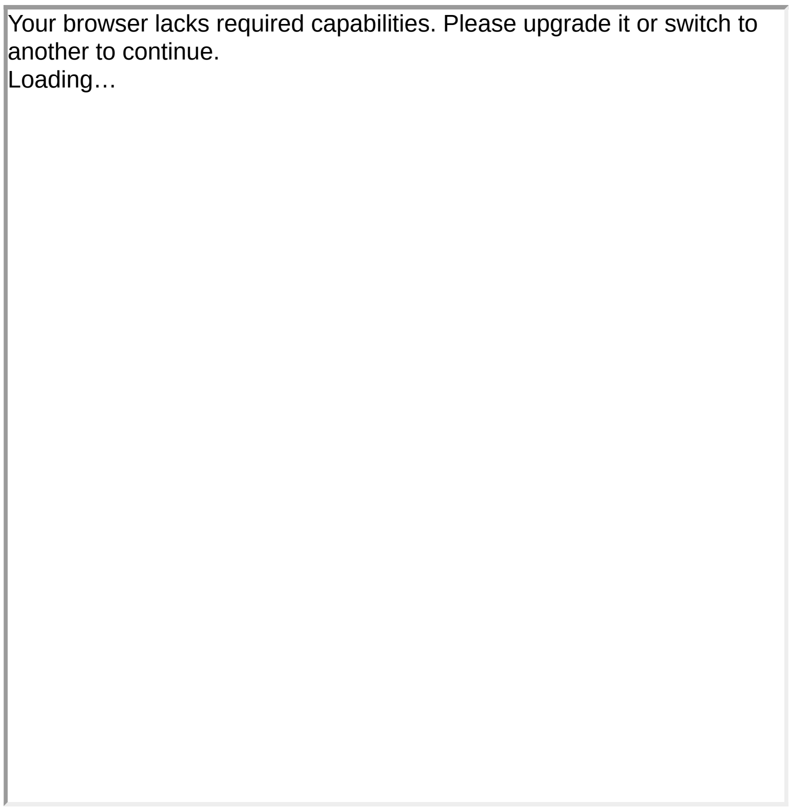
Download: Twee Code

"Passages in Passages": SugarCube (v2.2)

Summary

In SugarCube, the contents of a passage can be shown in another through the use of the `<<include>>` macro. Through using the name of an existing passage, its contents will be included where the macro is called.

Live Example


Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Passages in Passages

:: Start
This is the Start passage!
<<include "Another">>

:: Another
And this is Another passage!
```

Download: Twee Code

"Passages in Passages: Snowman (v1.3)

Summary

In Snowman, the contents of one passage can be included in another through using the **window.story.render()** function. This will find and return the source of an existing passage in the story. Combined with the use of Underscore's template system, the returned value can be included directly where the function is used in a passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Passages in Passages

:: Start
This is the Start passage!
<%= window.story.render("Another") %>

:: Another
And this is Another passage!
```

Download: Twee Code

"Passage Transitions": Harlowe (v3.0)

Summary

When using the `(transition-arrive:)` or `(transition-depart:)` macros with links, the resulting transition effect will be shown to the player upon activating the link (depart) or as it is shown (arrive).

Possible transition effects include:

- "instant" (causes the passage to instantly vanish)
- "dissolve" (causes the passage to gently fade out)
- "flicker" (causes the passage to roughly flicker in - don't use with a long (transition-time:)))
- "shudder" (causes the passage to disappear while shaking back and forth)
- "rumble" (causes the passage to instantly appear while shaking up and down)
- "slide-right" (causes the passage to slide in from the right)
- "slide-left" (causes the passage to slide in from the left)
- "pulse" (causes the passage to disappear while pulsating rapidly)

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe 3: Passage Transitions

:: Start
(transition-depart: "dissolve")[[Dissolve Passage]]

:: Dissolve Passage
(transition-arrive: "slide-right")[[Slide-right Passage]]

:: Slide-right Passage
Double-click this passage to edit it.
```

Download: Twee Code

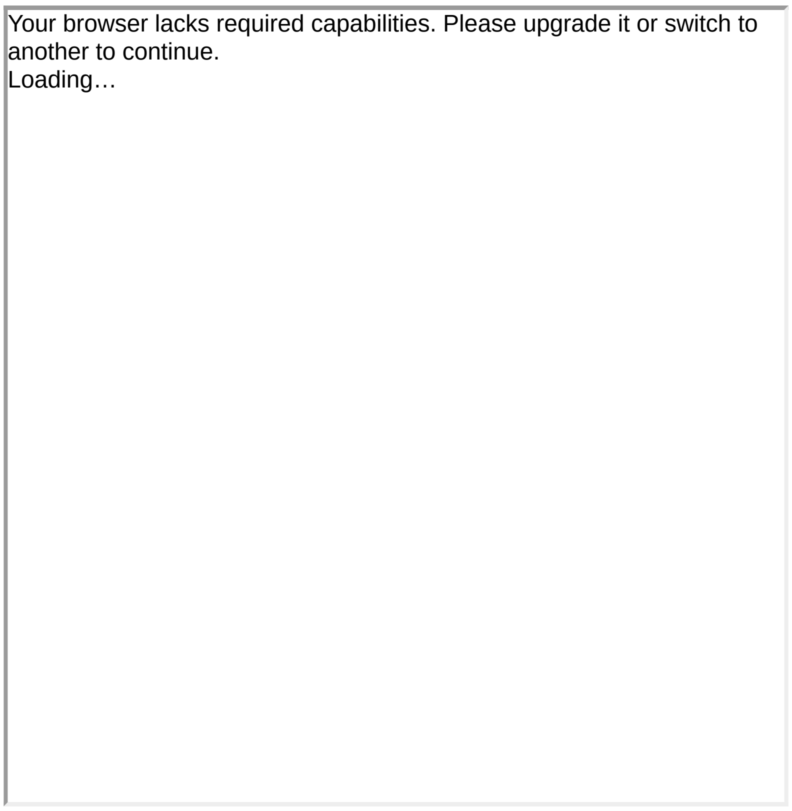
"Passage Transitions": SugarCube (v2.18)

Summary

SugarCube provides multiple ways to address passage transitions. To work with the existing functionality, the *Config.passages.transitionOut* variable can be set to change the property (height, width, opacity, etc) of the passage element or the amount of time to retain the outgoing passage before removing it with the default transition effect.

Through using jQuery event listeners, different functionality can be triggered as part of normal passage events such as rendering or displaying content. Acting on the ":passagerender" event, for example, could be used to apply a jQuery effect on the passage element after it has been loaded and rendered.

Live Example



Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Passage Transitions

:: UserScript[script]
// Fade in content using jQuery Effects
// Hide and then fade in the content over 2000ms (2s)
$(document).on(':passagerender', function (event) {
    $(".passage").hide(0).fadeIn(2000);
});

:: Start
[[Another passage]]

:: A third passage
No more content!

:: Another passage
[[A third passage]]
```

Download: Twee Code

"Passage Transitions": Snowman (v1.3)

Summary

In Snowman, the "showpassage" event is triggered once a passage is loaded. By listening for this event, jQuery effects can be applied to the passage element to produce a transition.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Passage Transitions

:: UserScript[script]
$(document).on('showpassage', function(event, passage) {

    $("#passage").hide(0).fadeIn(2000);

});

:: Start
[[Another Passage]]

:: Another Passage
[[A Third Passage]]

:: A Third Passage
Double-click this passage to edit it.
```

Download: Twee Code

"Passage Transitions": Snowman (v2.0)

Summary

In Snowman 2.X, the `sm.passage.showing` event is triggered once a passage is loaded. By listening for this event, jQuery effects can be applied to the passage element to produce a transition. (In Snowman 2.X, the passage is the `tw-passage` element.)

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman 2: Passage Transitions

:: UserScript[script]
$(document).on('sm.passage.showing', function(event, passag

    $("tw-passage").hide(0).fadeIn(2000);

});

:: Start
[[Another Passage]]

:: Another Passage
[[A Third Passage]]

:: A Third Passage
Double-click this passage to edit it.
```

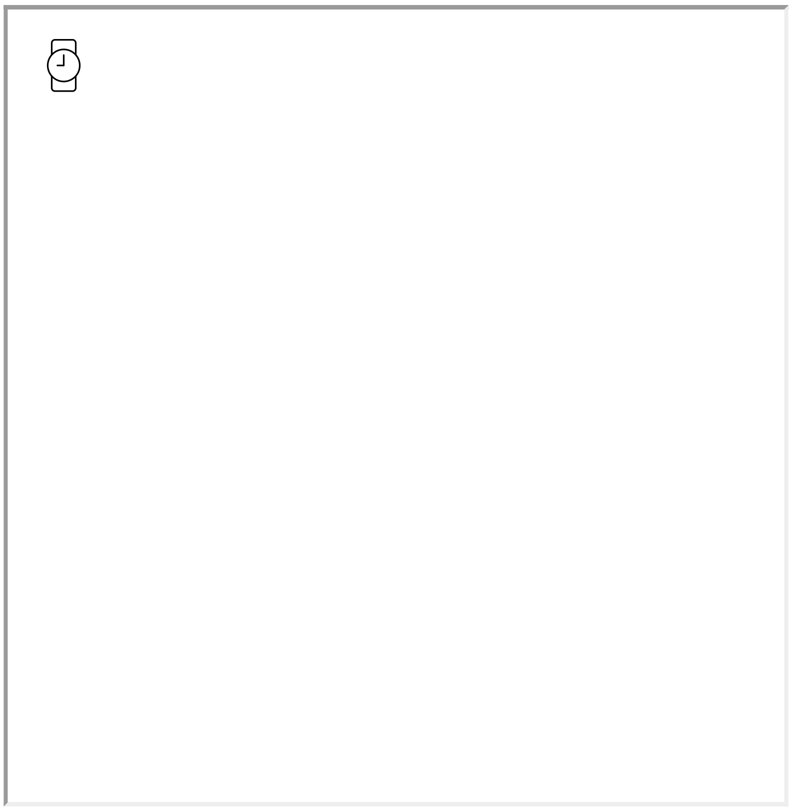
Download: Twee Code

"Passage Visits": Chapbook (v1.0.0)

Summary

In Chapbook, the lookup *passage.visits* variable contains the number of times the current passages has been visited by the user.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Passage Visits

:: Start

[[Another Passage]]

:: Another Passage
How many times has the current passage been visited? {passa

[[Start]]
```

Download: Twee Code

"Passage Visits": Harlowe (v1.0)

Summary

In Harlowe, the macro `(count:)` returns the number of times a value appears in an array. The macro `(history:)` returns an array of any passages visited as part of the story.

Combined together, the macro `(count:)` with the `(history:)` can return the number of times a certain passage has been visited during the course of the story.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Passage Visits

:: Start
How many times has the passage "Another Passage" been visit

[[Another Passage]]

:: Another Passage
[[Start]]
```

Download: Twee Code

"Passage Visits": SugarCube (v2.2)

Summary

In SugarCube, the global function **visited()** returns the number of times a passage has been visited during the course of the story. Combined with the `<<= >>` macro expression, the result of a global function can be written to a passage.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Passage Visits

:: Start
How many times has the passage "Another Passage" been visit

[[Another Passage]]

:: Another Passage
[[Start]]
```

Download: Twee Code

"Passage Visits": Snowman (v2.0)

Summary

Starting with Snowman 2.0, the global function **visited()** returns the number of times one or more passages have been visited during the course of the story. Combined with the use of Underscore template interpolation, the result of a function can be written to a passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Passage Visits

:: Start
How many times has the passage "Another Passage" been visit

[[Another Passage]]

:: Another Passage
[[Start]]
```

Download: Twee Code

"Player Statistics": Chapbook (v1.0.0)

Summary

One of the most popular mechanics of table-top role-playing games are those where the player must determine their in-game statistics and then use them to make decisions.

In Chapbook, the values of variables can only be changed as part of the Vars Section or using JavaScript. This example combines the two and uses the `{embed passage}` modifier to use different passages as sections of code and adjust values based on user interactions.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Player Statistics

:: Start
{embed passage: "Setup"}
{embed passage: "Statistics"}

:: IncreaseEmpathy
empathy: empathy + 1
points: points - 1
--
{embed passage: "Statistics"}

:: DecreaseEmpathy
empathy: empathy - 1
points: points + 1
--
{embed passage: "Statistics"}

:: IncreaseIntelligence
intelligence: intelligence + 1
points: points - 1
--
{embed passage: "Statistics"}

:: DecreaseIntelligence
intelligence: intelligence - 1
points: points + 1
--
{embed passage: "Statistics"}

:: Setup
empathy: 10
intelligence: 10
points: 5
--

:: Statistics
{embed passage: "CheckValues"}

[if points > 0 && empathy > 0]
Empathy:
[[[+]->IncreaseEmpathy]] [[[-]->DecreaseEmpathy]]
[continued]

[if points > 0 && intelligence > 0]
Intelligence:
[[[+]->IncreaseIntelligence]] [[[-]->DecreaseIntelligence]]
```

```
[continued]

Empathy: {empathy}

Intelligence: {intelligence}

Remaining Points: {points}

[[Reset Points->Start]]

:: CheckValues
[JavaScript]
    let empathy = engine.state.get('empathy');
    let intelligence = engine.state.get('intelligence');
    let points = engine.state.get('points');

    if(empathy > 20){empathy = 20;}
    if(intelligence > 20){intelligence = 20;}
    if(points < 0){points = 0;}
    if(points > 25){points = 25;}

    engine.state.set('empathy', empathy);
    engine.state.set('intelligence', intelligence);
    engine.state.set('points', points);
```

Download: Twee Code

"Player Statistics": Harlowe (v2.0)

Summary

One of the most popular mechanics of table-top role-playing games are those where the player must determine their in-game statistics and then use them to make decisions.

In this example, the `(link-repeat:)` macro is used multiple times with `(replace:)` and `(set:)` macros based on if they are higher than a target value. In a second passage, these values are used in combination with a random number between 1 to 6, mimicking a common 1d6 mechanic to check if a value is above a target number.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Player Statistics in Harlowe

:: Start
Empathy: {
  (link-repeat: "|+|")[
      (if: $totalPoints > 0)[
        (set: $empathy to it + 1)
        (set: $totalPoints to it - 1)
        (replace: ?empathyStat)[|empathyStat>[$empathy]]
        (replace: ?pointsStat)[|pointsStat>[$totalPoints]]
      ]
  ]

  (link-repeat: "|-|")[
      (if: $empathy > 0)[
        (set: $empathy to it - 1)
        (set: $totalPoints to it + 1)
        (replace: ?empathyStat)[|empathyStat>[$empathy]]
        (replace: ?pointsStat)[|pointsStat>[$totalPoints]]
      ]
  ]
}
Intelligence: {
  (link-repeat: "|+|")[
      (if: $totalPoints > 0)[
        (set: $intelligence to it + 1)
        (set: $totalPoints to it - 1)
        (replace: ?intelligenceStat)[|intelligenceStat>[$ir
        (replace: ?pointsStat)[|pointsStat>[$totalPoints]]
      ]
  ]

  (link-repeat: "|-|")[
      (if: $intelligence > 0)[
        (set: $intelligence to it - 1)
        (set: $totalPoints to it + 1)
        (replace: ?intelligenceStat)[|intelligenceStat>[$ir
        (replace: ?pointsStat)[|pointsStat>[$totalPoints]]
      ]
  ]
}
{
  (link-repeat: "|Reset Points|")[
      (set: $empathy to 10)
      (set: $intelligence to 10)
      (set: $totalPoints to 5)
      (replace: ?empathyStat)[|empathyStat>[$empathy]]
```

```
        (replace: ?intelligenceStat)[|intelligenceStat>[$inte
        (replace: ?pointsStat)[|pointsStat>[$totalPoints]]
    ]
  }

  Empathy: |empathyStat>[10]
  Intelligence: |intelligenceStat>[10]
  Remaining Points: |pointsStat>[5]

  [[Test Stats]]

  :: Test Stats
  (link: "Make an intelligence check?")[
      (set: _result to (random: 1, 6) + $intelligence)
      (if: _result >= 15)[
      Intelligence Success! (_result >= 15)
      ](else:)[
      Intelligence Failure! (_result < 15)
      ]
  ]
  (link: "Make an empathy check?")[
      (set: _result to (random: 1, 6) + $empathy)
      (if: _result >= 15)[
      Empathy Success! (_result >= 15)
      ](else:)[
      Empathy Failure! (_result < 15)
      ]
  ]

  :: Startup[startup]
  (set: $empathy to 10)
  (set: $intelligence to 10)
  (set: $totalPoints to 5)
```

Download: Twee Code

See Also

Conditional Statements, Setting and Showing

"Player Statistics": SugarCube (v2.18)

Summary

One of the most popular mechanics of table-top role-playing games are those where the player must determine their in-game statistics and then use them to make decisions.

In this example, the `<<link>>` macro is used multiple times to replace content and adjust values based on `<<if>>` they are greater than a target number. In a second passage, these values are used in combination with a random number between 1 to 6, mimicking a common 1d6 mechanic to check if a value is above a target number.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Player Statistics in SugarCube

:: Start
Empathy: \
<<link "[+]">>
    <<if $totalPoints gt 0>>
      <<set $empathy++>>
      <<set $totalPoints-->>
      <<replace "#empathyStat">><<print $empathy>><</replac
      <<replace "#pointsStat">><<print $totalPoints>><</rep
    <</if>>
<</link>>\
<<link "[-]">>
    <<if $empathy gt 0>>
      <<set $empathy-->>
      <<set $totalPoints++>>
      <<replace "#empathyStat">><<print $empathy>><</replac
      <<replace "#pointsStat">><<print $totalPoints>><</rep
    <</if>>
<</link>>
Intelligence: \
<<link "[+]">>
    <<if $totalPoints gt 0>>
      <<set $intelligence++>>
      <<set $totalPoints-->>
      <<replace "#intelligenceStat">><<print $intelligence>
      <<replace "#pointsStat">><<print $totalPoints>><</rep
    <</if>>
<</link>>\
<<link "[-]">>
    <<if $intelligence gt 0>>
      <<set $intelligence-->>
      <<set $totalPoints++>>
      <<replace "#intelligenceStat">><<print $intelligence>
      <<replace "#pointsStat">><<print $totalPoints>><</rep
    <</if>>
<</link>>
<<link "[Reset Points]">>
    <<set $empathy to 10>>
    <<set $intelligence to 10>>
    <<set $totalPoints to 5>>
    <<replace "#empathyStat">><<print $empathy>><</replace>
    <<replace "#intelligenceStat">><<print $intelligence>><
    <<replace "#pointsStat">><<print $totalPoints>><</repla
<</link>>

Empathy: <span id="empathyStat">10</span>
```

```
Intelligence: <span id="intelligenceStat">10</span>
Remaining Points: <span id="pointsStat">5</span>

[[Test Stats]]

:: StoryInit
<<set $empathy to 10>>
<<set $intelligence to 10>>
<<set $totalPoints to 5>>

:: Test Stats
<<linkreplace "Make an intelligence check?">>
    <<set _result to random(1, 6) + $intelligence >>
    <<if _result gte 15>>
    Intelligence Success! (_result >= 15)
    <<else>>
    Intelligence Failure! (_result < 15)
    <</if>>
<</linkreplace>>
<<linkreplace "Make an empathy check?">>
    <<set _result to random(1, 6) + $empathy >>
    <<if _result gte 15>>
    Emaphy Success! (_result >= 15)
    <<else>>
    Empathy Failure! (_result < 15)
    <</if>>
<</linkreplace>>
```

Download: Twee Code

See Also

Conditional Statements, Setting and Showing

"Player Statistics": Snowman (v1.3.0)

Summary

One of the most popular mechanics of table-top role-playing games are those where the player must determine their in-game statistics and then use them to make decisions.

In this example, the jQuery event handler **click()** is used to bind to multiple buttons. Depending on what was clicked, the content is replaced or values adjusted based on if a conditional statement is true. Values are then tested when combined with a random number between 1 to 6, mimicking a common 1d6 mechanic to check if a value is above a target number.

Note: Elements must exist *before* the attempt to bind to them in order to be successful. This example uses the *ready()* function to achieve this with the first, starting passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Player Statistics in Snowman

:: UserScript[script]
$(function() {

    // Create a global setup object
    window.setup = window.setup || {};

    // Create a global propety on the setup object
    window.setup.stats = {};

    // Create (and overwrite) the use of 's'
    var s = window.setup.stats;

    s.empathy = 10;
    s.intelligence = 10;
    s.totalPoints = 5;

  $("#empathyIncrease").click(function(){
        if(s.totalPoints > 0) {
            setup.stats.empathy++;
            s.totalPoints--;
            $("#empathyStat").text(s.empathy);
            $("#pointsStat").text(s.totalPoints);
        }
    });

    $("#empathyDecrease").click(function(){
        if(s.empathy > 0) {
            s.empathy--;
            s.totalPoints++;
            $("#empathyStat").text(s.empathy);
            $("#pointsStat").text(s.totalPoints);
        }
    });

    $("#intelligenceIncrease").click(function(){
        if(s.totalPoints > 0) {
            s.intelligence++;
            s.totalPoints--;
            $("#intelligenceStat").text(s.intelligence);
            $("#pointsStat").text(s.totalPoints);
        }
    });

    $("#intelligenceDecrease").click(function(){
        if(s.intelligence > 0) {
            s.intelligence--;
            s.totalPoints++;
            $("#intelligenceStat").text(s.intelligence);
            $("#pointsStat").text(s.totalPoints);
        }
    });

    $("#pointsReset").click(function(){
        s.empathy = 10;
        s.intelligence = 10;
        s.totalPoints = 5;
        $("#empathyStat").text(s.empathy);
        $("#intelligenceStat").text(s.intelligence);
        $("#pointsStat").text(s.totalPoints);
    });

    // Add a randomInt function to the Math global
    Math.randomInt = function(min, max) {
      min = Math.ceil(min);
      max = Math.floor(max);
      return Math.floor(Math.random() * (max - min)) + min;
    };

    $("#testIntelligence").click(function() {

        var result = s.intelligence + Math.randomInt(1,6);

        if(result >= 15) {
            $("#intelligenceResult").text("Success! (" + re
        } else {
            $("#intelligenceResult").text("Failure! (" + re
        }

        console.log("Test!");
    });

    $("#testEmpathy").click(function() {

        var result = s.empathy + Math.randomInt(1,6);

        if(result >= 15) {
            $("#empathyResult").text("Success! (" + result
        } else {
            $("#empathyResult").text("Failure! (" + result
        }

    });
```

```
});

:: Start
Empathy: <button id="empathyIncrease">[+]</button> <button

Intelligence: <button id="intelligenceIncrease">[+]</button

<button id="pointsReset">[Reset Points]</button>

Empathy: <span id="empathyStat">10</span>

Intelligence: <span id="intelligenceStat">10</span>

Remaining Points: <span id="pointsStat">5</span>

<button id="testIntelligence">Make an intelligence check?</
<div id="intelligenceResult"></div>

<button id="testEmpathy">Make an empathy check?</button>
<div id="empathyResult"></div>
```

Download: Twee Code

See Also

Conditional Statements, Setting and Showing

"Programmatic Undo": Harlowe (v2.0)

Summary

While Harlowe supports allowing the user to undo and redo moves, the "undo" operation can also be accessed through the `(undo:)` macro. Through its use, changes from the most current action can be "undone."

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Programmatic Undo in Harlowe

:: Start
[[Enter the darkness]]

:: Enter the darkness
(link: "You are not ready. Go back!")[(undo:)]
```

Download: Twee Code

"Programmatic Undo": SugarCube (v2.18)

Summary

While SugarCube supports allowing the user to undo and redo moves, the "undo" operation can also be accessed through the `<<back>>` macro. Through its use, changes from the most current action can be "undone."

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Programmatic Undo in SugarCube

:: Start
[[Enter the Darkness]]

:: Enter the Darkness
<<back "You are not ready! Go back!">>
```

Download: Twee Code

"Programmatic Undo": Snowman (v1.3.0)

Summary

Snowman comes with no user-facing functionality for undoing and re-doing actions. However, though using jQuery and a combination of the **window.story.checkpoint()** and **window.history.back()*** functions, this can be emulated.

Note: Checkpoints will only affect properties of the 's' (state) global variable.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Programmatic Undo in Snowman

:: UserScript[script]
$(window).on('showpassage:after', function (e, data)
{
    window.story.checkpoint(data.passage.name);
});

:: Start
[[Enter the Darkness]]

:: Enter the Darkness
<a href="javascript: window.history.back();">You are not re
```

Download: Twee Code

"Render Passage to Element": SugarCube (v2.0.0)

Summary

In SugarCube, the function **setPageElement()** renders the contents of a passage into an element based on its *id*.

The event ":passagedisplay" is used in this example to guarantee that the passage has been rendered before acting. Calling the function **setPageElement()** inside the jQuery event listener then renders another passage into an existing element.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Render Passage to Element

:: Start
<div id="hudID"></div>
<<script>>
    // Wait for the passage to be displayed
    $(document).one(':passagedisplay', function (ev) {
        // Render the passage named HUD into the element wi
        setPageElement("hudID", "HUD");
    });
<</script>>

:: HUD
<h1>This is the heads-up display!</h1>
```

Download: Twee Code

"Render Passage to Element": Snowman (v2.0.0)

Summary

In Snowman, the function **renderToSelector()** renders the contents of a passage into an element based on its jQuery selector.

The event "sm.passage.shown" is used in this example to guarantee that the passage has been rendered before acting. Calling the function **renderToSelector()** inside the jQuery event listener then renders another passage into an existing element.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Snowman: Render to Element

:: Start
<div id="hudID"></div>
<script>
$(document).one('sm.passage.shown', function (ev) {
    // Render the passage named HUD into the element with i
        renderToSelector("#hudID", "HUD");
    });
</script>

:: HUD
<h1>This is the HUD!</h1>
```

Download: Twee Code

"Saving Games": Harlowe (v2.0)

Summary

Harlowe provides macros like `(save-game:)` and `(load-game:)` to store and retrieve game "saves" of the variables and current passage. `(saved-games:)` can also be used to check if a certain game save exists.

Game saves are stored as cookies in the user's browser. If cookies cannot be stored for some reason, game saves will fail. It is recommended to always check if the game save was stored before trying to retrieve it later.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Saving Games in Harlowe

:: Start
(link:"Save game?")[
  (if:(save-game:"Slot A"))[
    (if: (saved-games:) contains "Slot A")[
        Slot A is in the saved-games datamap!
    ]
    (link: "Load Slot A?" )[
          (load-game: "Slot A")
    ]
  ](else: )[
    Sorry, I couldn't save your game.
  ]
]
```

Download: Twee Code

"Saving Games": SugarCube (v2.18)

Summary

SugarCube provides built-in functionality for saving, viewing, and deleting game saves through its sidebar. However, the Save API also provides programmable access for re-creating this for users through functions like **Save.slots.has()***, **Save.slots.save()**, and ***Save.slots.load()**.

This example also demonstrates the use of the *State.variables* object to access variables in JavaScript and use them.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Saving Games in SugarCube

:: Start
<<script>>
if (Save.slots.has(0)) {
    State.variables.slotA = true;
} else {
    State.variables.slotA = false;
}
<</script>>

<<if $slotA is true>>
    The first game slot exists! (This session was most like
<</if>>

<<link "Save to the first slot?">>
    <<script>>
        if (Save.slots.ok()) {
            Save.slots.save(0);
        }
    <</script>>
<</link>>

<<link "Load from the first slot?">>
    <<script>>
        if (Save.slots.has(0)) {
            Save.slots.load(0);
        }
    <</script>>
<</link>>

<<link "Delete first slot and restart story?">>
    <<script>>
        if (Save.slots.has(0)) {
            Save.slots.delete(0);
            Engine.restart();
        }
    <</script>>
<</link>>
```

Download: Twee Code

"Saving Games": Snowman (v1.3.0)

Summary

Snowman provides the **window.story.saveHash()** and **window.story.restore()** functions to produce a hash of the current story state and then recover it. However, it does not provide a mechanism for saving the hash between sessions. Through using the *window.localStorage* global variable, this can be accomplished.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Saving Games in Snowman

:: UserScript[script]
window.storage = {
    ok: function() {
        try {
                var storage = window["localStorage"],
                    x = '__storage_test__';
                storage.setItem(x, x);
                storage.removeItem(x);
                return true;
            }
            catch(e) {
                return e instanceof DOMException && (
                // everything except Firefox
                e.code === 22 ||
                // Firefox
                e.code === 1014 ||
                // test name field too, because code might
                // everything except Firefox
                e.name === 'QuotaExceededError' ||
                // Firefox
                e.name === 'NS_ERROR_DOM_QUOTA_REACHED') &&
                // acknowledge QuotaExceededError only if t
                storage.length !== 0;
            }
    },
    save: function(hash) {
        window.localStorage.setItem('hash', hash);
    },
    restore: function() {
        return window.localStorage.getItem('hash');
    },
    delete: function() {
        window.localStorage.removeItem("hash");
    }
}

:: Start
<%
    if(window.storage.ok()) { %>
    Window storage works!
<% } %>
<%
    if(window.storage.restore()) { %>
    There is a session saved!
<% } %>
```

```
[[Save the session hash?]]

[[Restore from previous session?]]

[[Delete previous session?]]

:: Save the session hash?
The hash is <%= window.story.saveHash() %>. It has been sav

<% if(window.storage.ok()) {
    window.storage.save(window.story.saveHash())
}%>

[[Go back?|Start]]

:: Restore from previous session?
<% if(window.storage.ok()) {
    if(window.story.restore(window.storage.restore()) ) { %
    The restore was successful!
<% }
}
%>

[[Go back?|Start]]

:: Delete previous session?
<% window.storage.delete() %>

[[Go back?|Start]]
```

Download: Twee Code

"Setting and Showing Variables": Chapbook (v1.0.0)

Summary

In Chapbook all variables are initialised and updated within the Vars Section of a passage, this section is always added at the beginning of the passage and there can only be one such section per passage.

The Vars Section is separated from the passage's normal text by two dashes (`--`); The value of a variable can be displayed using the `{insert}` insert.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Setting and Showing Variables in Chapbook

:: Start
numberVariable: 5
wordVariable: "five"
phraseVariable: "The value"
--

{phraseVariable} is {numberVariable} and {wordVariable}.

{embed passage: "Increment number"}

The number variable was incremented by one to {numberVariab

:: Increment number
numberVariable: numberVariable + 1
--
```

Download: Twee Code

"Setting and Showing Variables": Harlowe (v2.0)

Summary

Variables, symbols starting with `$` (for story-wide) or `_` (for temporary), can be "set" using the `(set:)` macro in Harlowe.

`$` is used for storing data throughout the story, and `_` should be used for data only needed in the current passage. Using `_` is useful for not wanting to accidentally overwrite variables elsewhere in the story. They can also help with debugging through not cluttering up the variables list of future passages.

In Harlowe, the keyword **it** can also be used as a shortcut for changing and saving a value in reference to itself. The **it** refers to the first variable named in the macro.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Setting and Showing Variables in Harlowe

:: Start
(set: $numberVariable to 5)
(set: $textVariable to "five")
(set: _textVariable to "The values")

_textVariable are $numberVariable and $textVariable.

(set: $numberVariable to it + 1)

_textVariable are $numberVariable and $textVariable.
```

Download: Twee Code

"Setting and Showing Variables": SugarCube (v2.18)

Summary

Variables, symbols starting with `$` (for normal) or `_` (for temporary), can be "set" using the `<<set>>` macro in SugarCube.

`$` is used for storing data throughout the story, and `_` should be used for data only needed in the current passage. Using `_` is useful for not wanting to accidentally overwrite variables elsewhere in the story. They can also help with debugging through not cluttering up the variables list of future passages.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Setting and Showing Variables in SugarCube

:: Start
<<set $numberVariable to 5>>
<<set $wordVariable to "five">>
<<set $phraseVariable to "The value">>

$phraseVariable is $numberVariable and $wordVariable.

<<set $numberVariable to $numberVariable + 1>>

$phraseVariable is $numberVariable and $wordVariable.
```

Download: Twee Code

"Setting and Showing Variables": Snowman (v1.3.0)

Summary

In Snowman, the *s* global variable can be used to store and retrieve values. Properties can be created and assigned freely. The Underscore template functionality can be used to define, change, and show the values of variables.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Setting and Showing Variables in Snowman

:: Start
<%
    s.numberVariable = 5;
    s.wordVariable = "five";
    s.phraseVariable = "The value";
%>

<%= s.phraseVariable %> is <%= s.numberVariable %> and <%=

<%
    s.numberVariable++;
%>

<%= s.phraseVariable %> is <%= s.numberVariable %> and <%=
```

Download: Twee Code

"Space Exploration": Chapbook (v1.0.0)

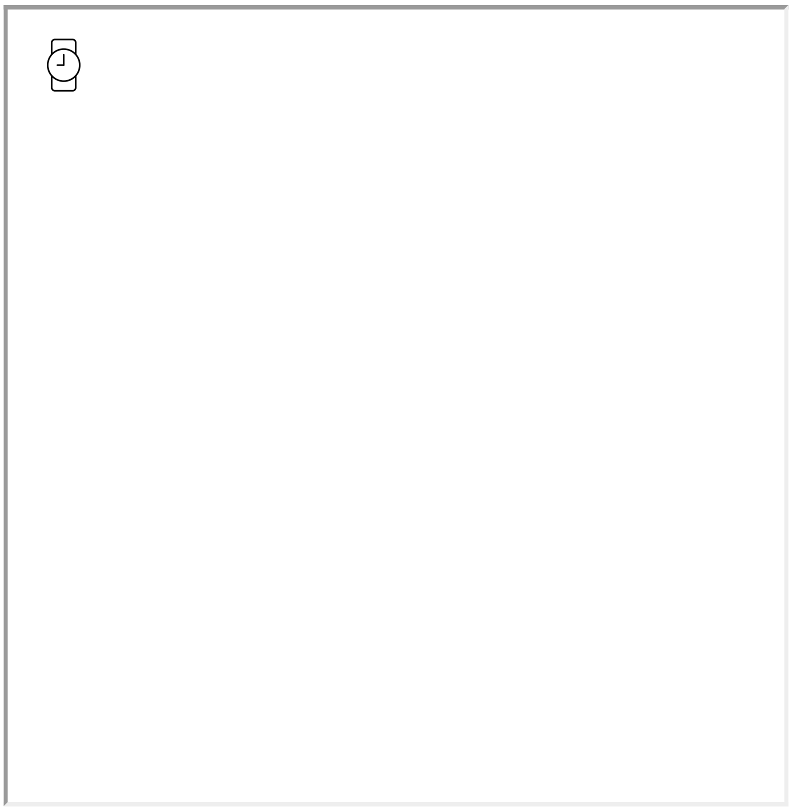
Summary

Games in the rogue-like genre often have random events that influence player choices. Frequently, decisions can have lasting impact or even lead to an ending of play in that session or run depending on these random outcomes.

Heavily inspired by *FTL: Faster Than Light* (2012), this example generates a system of four planets consisting of either RED, more risk and more reward, or GREEN, less risk and less reward. Upon entering a system of planets, the player can "Scan System" and then choose to visit these planets for different outcomes based on a series of choices and the use of the **Math.random()** function. At the same time, the player must also balance the health of the ship, the number of jumps left, and the current fuel. These are displayed when visiting the "Statistics" passage. If any of them drop below 0, the game ends with an ending matching that statistic.

This example uses a complex combination of the Vars Section, `[JavaScript]` modifier, and internal JavaScript functionality in Chapbook, including the use of the function **window.go()**.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Space Exploration

:: UserScript[script]
// Add a randomRange
// Used from https://developer.mozilla.org/en-US/docs/Web/J
Math.randomRange = (min, max) => {
        min = Math.ceil(min);
      max = Math.floor(max);
      return Math.floor(Math.random() * (max - min)) + min;
};

:: Start
health: 20
fuel: 4
system: []
numberOfJumpsLeft: 10
--
{embed passage: 'GenerateSystem'}

[[Explore Space]]

:: Explore Space
{embed passage: "CheckStatus"}
Current System:
{embed passage: 'DisplaySystem'}
<hr>
[[Hyperjump]]

[[Statistics]]

:: GenerateSystem
[JavaScript]

    // There will always be four (4) planets
    let planets = 4;

    // Reset global system
    let system = engine.state.get('system');
    system = [];

    // Add 0 (Red) or 1 (Green) planets
    for(let i = 0; i < planets; i++) {
        // Get a random number from 0 to 2
        // 0 = RED
        // 1 = GREEN
        // 2 = EMPTY
        system.push(Math.randomRange(0,3));
    }

    // Update the new 'system'
    engine.state.set('system', system);



:: CheckStatus
[JavaScript]
    let health = engine.state.get("health");
    let fuel = engine.state.get("fuel");
    let numberOfJumpsLeft = engine.state.get("numberOfJumps

    if(health <= 0) {
        // Introduce a micro-delay before transition
        setTimeout(() => {
            go("Destroyed");
        }, 10);
    }

    if(fuel <= 0) {
        // Introduce a micro-delay before transition
        setTimeout(() => {
            go("Lost in Space");
        }, 10);
    }

    if(numberOfJumpsLeft <= 0) {
        // Introduce a micro-delay before transition
        setTimeout(() => {
            go("Safe");
        }, 10);
    }



:: Destroyed
The ship exploded in flight.

Game Over.

:: Lost in Space
Without fuel, the ship tumbled and spun in the endless blac

Game Over

:: Safe
```

```
After 10 hyperjumps, the ship left the hazardous area and c

Success!

:: Hyperjump
fuel: fuel - 1
numberOfJumpsLeft: numberOfJumpsLeft - 1
--
{embed passage: "CheckStatus"}
{embed passage: "GenerateSystem"}

[[Scan System->Explore Space]]


:: DisplaySystem
[if system[0] == 0]
    {reveal link: 'RED', passage: 'RED'}
[if system[0] == 1]
    {reveal link: 'GREEN', passage: 'GREEN'}
[if system[0] == 2]
    <br>
[if system[1] == 0]
    {reveal link: 'RED', passage: 'RED'}
[if system[1] == 1]
    {reveal link: 'GREEN', passage: 'GREEN'}
[if system[1] == 2]
    <br>
[if system[2] == 0]
    {reveal link: 'RED', passage: 'RED'}
[if system[2] == 1]
    {reveal link: 'GREEN', passage: 'GREEN'}
[if system[2] == 2]
    <br>
[if system[3] == 0]
    {reveal link: 'RED', passage: 'RED'}
[if system[3] == 1]
    {reveal link: 'GREEN', passage: 'GREEN'}
[if system[3] == 2]
    <br>


:: RED
REPORT
[JavaScript]

    let percentage = Math.randomRange(0, 11);
    let foundHealth;
    let foundFuel;
    let health = engine.state.get('health');
    let fuel = engine.state.get('fuel');

    if(percentage >= 6) {

        foundHealth = Math.randomRange(1, 6);
        foundFuel = Math.randomRange(1, 3);

        write(`The hostile environment damaged the ship, bu

        health -= foundHealth;
        fuel += foundFuel;

    }

    if(percentage >= 3 && percentage < 6) {

          foundHealth = Math.randomRange(2, 8);

          write(`A hostile ship attacked. (-${foundHealth}

          health -= foundHealth;

    }

    if(percentage < 3) {
        write("Nothing happened.");
    }

    engine.state.set('health', health);
    engine.state.set('fuel', fuel);
[continued]
{embed passage: "CheckStatus"}

:: GREEN
REPORT
[JavaScript]
    let percentage = Math.randomRange(0, 11);
    let foundFuel = 0;
    let foundHealth = 0;
    let fuel = engine.state.get('fuel');
    let health = engine.state.get('health');


    if(percentage < 2) {

        foundFuel = Math.randomRange(1,3);
        write(`Fuel was found in some wreckage. (+${foundFu
        fuel += foundFuel;

    }

    if(percentage > 6) {

        foundHealth = Math.randomRange(1,4);
        write(`During a brief pause, the ship was able to b
        health += foundHealth;

    }

    if(percentage > 2 && percentage < 6)
    {
        write(`Nothing happened.`);
    }

    engine.state.set('health', health);
    engine.state.set('fuel', fuel);
[continued]
{embed passage: "CheckStatus"}

:: Statistics
Health: {health}

Fuel: {fuel}

Number of Jumps Left: {numberOfJumpsLeft}

{reveal link: 'Statistics', passage: 'Statistics'}
```

Download: Twee Code

"Space Exploration": Harlowe (v2.0)

Summary

Games in the rogue-like genre often have random events that influence player choices. Frequently, decisions can have lasting impact or even lead to an ending of play in that session or run depending on these random outcomes.

Heavily inspired by *FTL: Faster Than Light* (2012), this example uses the `(random:)` macro to generate a system of planets consisting of either RED, more risk and more reward, or GREEN, less risk and less reward. Upon entering a system of planets, the player can choose to visit these planets for different outcomes based on a series of choices and an additional use of the (random:) macro. While traveling, the player must also balance the health of the ship, the number of jumps left, and the current fuel that are all displayed using the `(display:)` macro. Finally, to capture the permanence of many rogue-like games, the `(go-to:)` macro is used to prevent the use of the normal undo/redo operations in Harlowe.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Space Exploration in Harlowe

:: Start
[[Explore Space!|Explore Space 1]]

:: Startup[startup]
(set: $health to 20)
(set: $fuel to 3)
(set: $system to (a:) )
(set: $numberOfJumpsLeft to 10)

:: Generate System
{
  <!-- Save a range from 0 to a max of 3 (total of max 4) -
  (set: _planets to (range: 0, (random: 1, 3) ) )

  <!-- Reset system -->
  (set: $system to (a:) )

  <!-- Create a new system based on the previous random rar
  (for: each _i, ..._planets )[
      <!-- Add to the new system, setting either RED or GRE
      (set: $system to it + (a: (either: "RED", "GREEN") )
  ]
 }

:: HUD
Health: $health
Fuel: $fuel
Number of Jumps Left: $numberOfJumpsLeft
(display: "Check Status")

:: Display System
{
    (for: each _planet, ...$system)[
        (if: _planet is "RED")[
          (link: _planet)[
              (display: "Show Outcome - Red")
          ]
        ]
        (if: _planet is "GREEN")[
          (link: _planet)[
              (display: "Show Outcome - Green")
          ]
        ]
        <br>
    ]
```

```
}

:: Explore Space 1
(link: "Hyperjump")[
    (set: $fuel to it - 1)
    (set: $numberOfJumpsLeft to it - 1)
    (goto: "Explore Space 2")
]
[(display: "HUD")]<HUD|
(display: "Generate System")
(display: "Display System")

:: Show Outcome - Green
{
    (set: _percentage to (random: 1, 10) )

    (if: _percentage is 1)[

        (set: _foundFuel to (random: 1, 2) )

        Fuel was found in some wreckage. (+_foundFuel to fu
        (set: $fuel to it + _foundFuel)

    ] (else-if: _percentage is >= 6)[

        (set: _foundHealth to (random: 1, 3) )

        During a brief pause, the ship was able to be repai

        (set: $health to it + _foundHealth )

    ] (else:) [
        Nothing happened.
    ]

    (replace: ?HUD)[(display: "HUD")]
}

:: Show Outcome - Red
{
    (set: _percentage to (random: 1, 10) )

    (if: _percentage is >= 6)[

        (set: _foundHealth to (random: 1, 5) )
        (set: _foundFuel to (random: 1, 3) )

        The hostile environment damaged the ship, but extra
        (set: $health to it - _foundHealth )
        (set: $fuel to it + _foundFuel )

    ] (else-if: _percentage <= 3)[

        (set: _foundHealth to (random: 2, 7) )

        A hostile ship attacked. (-_foundHealth to health)

        (set: $health to it - _foundHealth )

    ] (else:)[
        Nothing happened.
    ]

    (replace: ?HUD)[(display: "HUD")]
}

:: Explore Space 2
(link: "Hyperjump")[
    (set: $fuel to it - 1)
    (set: $numberOfJumpsLeft to it - 1)
    (goto: "Explore Space 1")
]
[(display: "HUD")]<HUD|
(display: "Generate System")
(display: "Display System")

:: Check Status
{
    (if: $health <= 0)[
        (goto: "Destroyed")
    ]
    (if: $fuel <= 0)[
        (goto: "Lost in space")
    ]
    (if: $numberOfJumpsLeft <= 0)[
        (goto: "Safe")
    ]

}

:: Destroyed
The ship exploded in flight.

###Game Over.
```

```
:: Lost in space
Without fuel, the ship tumbled and spun in the endless blac

###Game Over

:: Safe
After 10 hyperjumps, the ship left the hazardous area and c

###Success!
```

Download: Twee Code

"Space Exploration": SugarCube (v2.18)

Summary

Games in the rogue-like genre often have random events that influence player choices. Frequently, decisions can have lasting impact or even lead to an ending of play in that session or run depending on these random outcomes.

Heavily inspired by *FTL: Faster Than Light* (2012), this example uses the **random()** function to generate a system of planets consisting of either RED, more risk and more reward, or GREEN, less risk and less reward. Upon entering a system of planets, the player can choose to visit these planets for different outcomes based on a series of choices and an additional use of the **random()** function. While traveling, the player must also balance the health of the ship, the number of jumps left, and the current fuel that are all displayed using the `<<include>>` macro. Finally, to capture the permanence of many roguelike games, the `<<goto>>` macro is used to prevent the use of the normal undo/redo in SugarCube.

To cleanly present the text, this example also uses both the `<<silently>>` macro, to disregard all output, and `<<nobr>>` macro, to collapse the whitespace.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Space Exploration in SugarCube

:: Start
[[Explore Space|Explore Space 1]]

:: StoryInit
<<set $health to 20>>
<<set $fuel to 4>>
<<set $system to [] >>
<<set $numberOfJumpsLeft to 10>>

:: Explore Space 1
<<link "Hyperjump">>
    <<set $fuel to $fuel - 1>>
    <<set $numberOfJumpsLeft to $numberOfJumpsLeft - 1>>
    <<goto "Explore Space 2">>
<</link>>

<div id="HUD">
    <<include "HUD">>
</div>

<<include "Generate System">>
<<include "Display System">>

:: Explore Space 2
<<link "Hyperjump">>
    <<set $fuel to $fuel - 1>>
    <<set $numberOfJumpsLeft to $numberOfJumpsLeft - 1>>
    <<goto "Explore Space 1">>
<</link>>

<div id="HUD">
    <<include "HUD">>
</div>

<<include "Generate System">>
<<include "Display System">>

:: Generate System
<<silently>>
    <<set _planets to random(1, 4) >>

    <<set $system to new Array(_planets) >>

  <<for _i to 0; _i lt _planets; _i++>>
      <<set $system[_i] to either("RED", "GREEN") >>
  <</for>>

<</silently>>

:: Display System
<<nobr>>
  <<for _i to 0; _i lt $system.length; _i++>>
      <<if $system[_i] eq "RED">>
           <<linkreplace $system[_i]>>
               <<include "Show Outcome - Red">>
           <</linkreplace>>
      <</if>>
      <<if $system[_i] eq "GREEN">>
           <<linkreplace $system[_i]>>
               <<include "Show Outcome - Green">>
           <</linkreplace>>
      <</if>>
      <br>
  <</for>>
<</nobr>>

:: Show Outcome - Red
<<nobr>>
  <<set _percentage to random(1, 10) >>

  <<if _percentage gte 6>>

      <<set _foundHealth to random( 1, 5) >>
      <<set _foundFuel to random( 1, 3) >>

      The hostile environment damaged the ship, but extra f

      <<set $health to $health - _foundHealth >>
      <<set $fuel to $fuel + _foundFuel >>

  <<elseif _percentage lte 3>>

      <<set _foundHealth to random(2, 7) >>

      A hostile ship attacked. (-_foundHealth to health)

      <<set $health to $health - _foundHealth >>

  <<else>>
      Nothing happened.
  <</if>>

  <<replace "#HUD">>
```

```
      <<include "HUD">>
  <</replace>>

<</nobr>>

:: Show Outcome - Green
<<nobr>>
  <<set _percentage to random(1, 10)>>

  <<if _percentage eq 1>>

      <<set _foundFuel to random( 1, 2)>>

      Fuel was found in some wreckage. (+_foundFuel to fuel

      <<set $fuel to $fuel + _foundFuel>>

  <<elseif _percentage gte 6>>

      <<set _foundHealth to random( 1, 3) >>

      During a brief pause, the ship was able to be repaire

      <<set $health to $health + _foundHealth>>

  <<else>>
      Nothing happened.
  <</if>>

  <<replace "#HUD">>
      <<include "HUD">>
  <</replace>>

<</nobr>>

:: HUD
Health: $health
Fuel: $fuel
Number of Jumps Left: $numberOfJumpsLeft
<<include "Check Status">>

:: Destroyed
The ship exploded in flight.

!!!Game Over.

:: Lost in space
Without fuel, the ship tumbled and spun in the endless blac
```

```
!!!Game Over

:: Safe
After 10 hyperjumps, the ship left the hazardous area and c

!!!Success!

:: Check Status
<<nobr>>
    <<if $health lte 0>>
        <<goto "Destroyed">>
    <</if>>
    <<if $fuel lte 0>>
        <<goto "Lost in space">>
    <</if>>
    <<if $numberOfJumpsLeft lte 0>>
        <<goto "Safe">>
    <</if>>
<</nobr>>
```

Download: Twee Code

"Space Exploration": Snowman (v1.3.0)

Summary

Games in the roguelike genre often have random events that influence player choices. Frequently, decisions can have lasting impact or even lead to an ending of play in that session or run depending on these random outcomes.

Heavily inspired by *FTL: Faster Than Light* (2012), this example uses the `_`**.random()** function to generate a system of planets consisting of either RED, more risk and more reward, or GREEN, less risk and less reward. Upon entering a system of planets, the player can choose to visit these planets for different outcomes based on a series of choices and an additional use of the `_`**.random()** function. While traveling, the player must also balance the health of the ship, the number of jumps left, and the current fuel that are all displayed using the **window.story.render()** function in combination with Underscore.js templating.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Space Exploration in Snowman

:: UserScript[script]
// Create a global setup object
window.setup = window.setup || {};

// Add 'variables' object to setup
window.setup.variables = {};
var _vars = window.setup.variables;
_vars.health = 20;
_vars.fuel = 4;
_vars.system = [];
_vars.numberOfJumpsLeft = 10;

// Create global functions
window.setup.functions = {};
var _functions = window.setup.functions;

_functions.redOutcome = function() {
        var _vars = window.setup.variables;
        var _percentage = _.random(1, 10);
        var response = "";
        if( _percentage >= 6) {
             var _foundHealth = _.random( 1, 5);
             var _foundFuel = _.random( 1, 3);
            response = "The hostile environment damaged the
              _vars.health -= _foundHealth;
              _vars.fuel += _foundFuel;

        } else {

              if( _percentage <= 3) {
                  var _foundHealth = _.random(2, 7);
                  response = "A hostile ship attacked. -" +
                 _vars.health -= _foundHealth;

            } else {
                response = "Nothing happened."
            }
        }

        return response;
    };

    _functions.greenOutcome = function() {
        var _vars = window.setup.variables;
        var _percentage = _.random(1, 10);
        var response = "";
        if( _percentage == 1) {
             var _foundFuel = _.random( 1, 2);
            response = "Fuel was found in some wreckage. +
              _vars.fuel += _foundFuel;

        } else {

              if( _percentage >= 6) {
                  var _foundHealth = _.random(1, 3);
                response = "During a brief pause, the ship
                 _vars.health += _foundHealth;

            } else {
                response = "Nothing happened."
            }
        }

        return response;
    };

:: Start
[[Explore Space|Explore Space 1]]

:: Explore Space 1
<div class="gameScreen">

  <% var _vars = window.setup.variables; %>
  <% _vars.fuel-- %>
  <% _vars.numberOfJumpsLeft-- %>

  [[ Hyperjump |Explore Space 2]]

  <div id="HUD">
      <%= window.story.render("HUD") %>
  </div>

  <%= window.story.render("Generate System") %>

  <div id="display"></div>

  <%= window.story.render("Display System") %>

  <%= window.story.render("Check Status") %>

</div>

:: Explore Space 2
```

```
<div class="gameScreen">

  <% var _vars = window.setup.variables; %>
  <% _vars.fuel-- %>
  <% _vars.numberOfJumpsLeft-- %>

  [[ Hyperjump |Explore Space 1]]

  <div id="HUD">
      <%= window.story.render("HUD") %>
  </div>

  <%= window.story.render("Generate System") %>

  <div id="display"></div>

  <%= window.story.render("Display System") %>

  <%= window.story.render("Check Status") %>

</div>

:: Generate System
<script>
    var _vars = window.setup.variables;

    var planets =_.random(1, 4);

    _vars.system = new Array(planets);

    for(var i = 0; i < _vars.system.length; i++) {
        _vars.system[i] = _.sample(["RED", "GREEN"]);
      }
</script>

:: Display System
<script>
    var _vars = window.setup.variables;

    // Wipe out current contents
    $("#display").html("");

    for(var i = 0; i < _vars.system.length; i++) {

      if(_vars.system[i] == "RED") {
        var link = $("<a href='#'>RED</a>")
        .click(function(e) {
            $( this )
            .replaceWith( window.story.render("Show Outcome
            return false;
        });
        $("#display").append( link );
        $("#display").append( "<br>" );
      }

      if(_vars.system[i] == "GREEN") {
          var link = $("<a href='#'>GREEN</a>")
        .click(function(e) {
            $( this )
            .replaceWith( window.story.render("Show Outcome
            return false;
        });
        $("#display").append( link );
        $("#display").append( "<br>" );
      }
    }

</script>

:: Show Outcome - Red
<%= window.setup.functions.redOutcome() %>
<% $("#HUD").html(window.story.render("HUD")) %>
<%= window.story.render("Check Status") %>

:: Show Outcome - Green
<%= window.setup.functions.greenOutcome() %>
<% $("#HUD").html(window.story.render("HUD")) %>
 <%= window.story.render("Check Status") %>

:: HUD
Health: <%= window.setup.variables.health %> <br>
Fuel: <%= window.setup.variables.fuel %> <br>
Number of Jumps Left: <%= setup.variables.numberOfJumpsLeft

:: Destroyed
The ship exploded in flight.

<h3>Game Over</h3>

:: Lost in space
Without fuel, the ship tumbled and spun in the endless blac

<h3>Game Over</h3>

:: Safe
After 10 hyperjumps, the ship left the hazardous area and c
```

```
<h3>Success!</h3>

:: Check Status
<script>

    var _vars = window.setup.variables;

    var status = "";

    if(_vars.health <= 0) {
        status = window.story.render("Destroyed");
    }
    if(_vars.fuel <= 0) {
        status = window.story.render("Lost in space");
    }
    if(_vars.numberOfJumpsLeft <= 0) {
        status = window.story.render("Safe");
    }

    if(status != "") {
        $(".gameScreen").html(status);
    }
</script>
```

Download: Twee Code

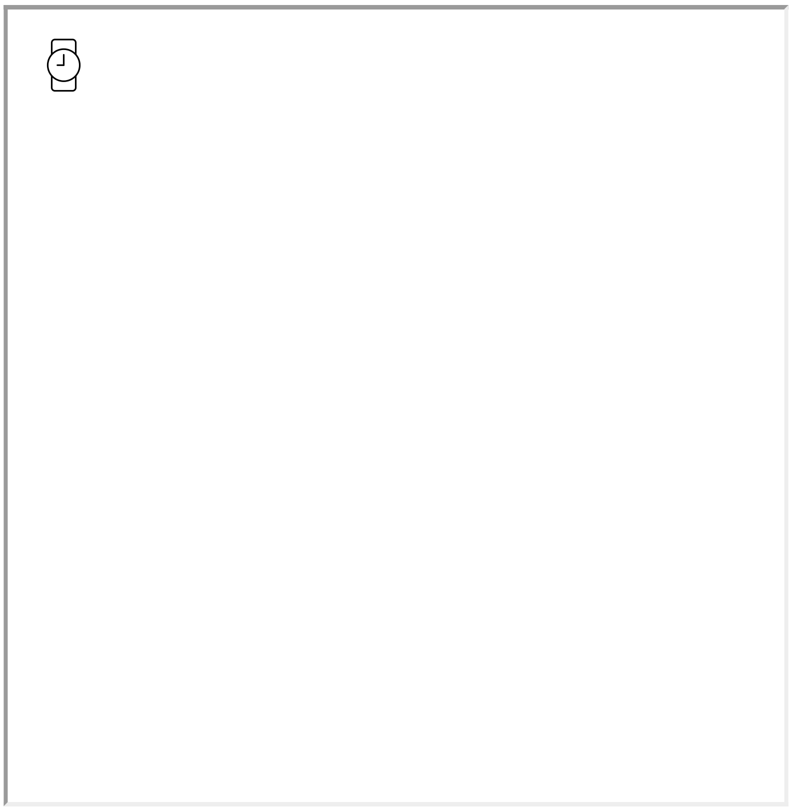
"Static Healthbars": Chapbook (v1.0.0)

Summary

"Static Healthbars" demonstrates how to write HTML elements using JavaScript in Chapbook. Through the `[JavaScript]` can be included in a passage. Chapbook also provides the **engine.state.get()** and **engine.state.set()** functions for getting and setting story variables.

Using these and the **write()** function, dynamic values can be created, accessed, and combined to produce static "healthbars" using the `<progress>` and `<meter>` HTML elements.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Static Healthbars

:: Start
[JavaScript]
    // Create a global variable, health
    engine.state.set('health', 80);

    // Get the current value of 'health'
    let health = engine.state.get('health');

    // Write description
    write("Show a healthbar using a Progress element:<br>")

    // Write the progress element with dynamic value
    write('<progress value="' + health + '" max="100"></pro

    // Write description
    write("Show a healthbar using a Meter element:<br>");

    // Write the meter element with dynamic value
    write('<meter value="' + health + '" min="0" max="100">
```

Download: Twee Code

"Static Healthbars": Harlowe (v2.0)

Summary

"Static Healthbars" demonstrates how to write HTML elements using variable values. In this example, the `(print:)` macro is used to create `<progress>` and `<meter>` elements. A `(text:)` macro is also used to temporarily convert the current Numeric value of the *$heath* story variable into a String value.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Static Healthbars for Harlowe

:: Start
(set: $health to 80)

Show a healthbar using a Progress element:
(print: '<progress value="' + (text: $health) + '" max="100

Show a healthbar using a Meter element:
(print: '<meter value="' + (text: $health) + '" min="0" max
```

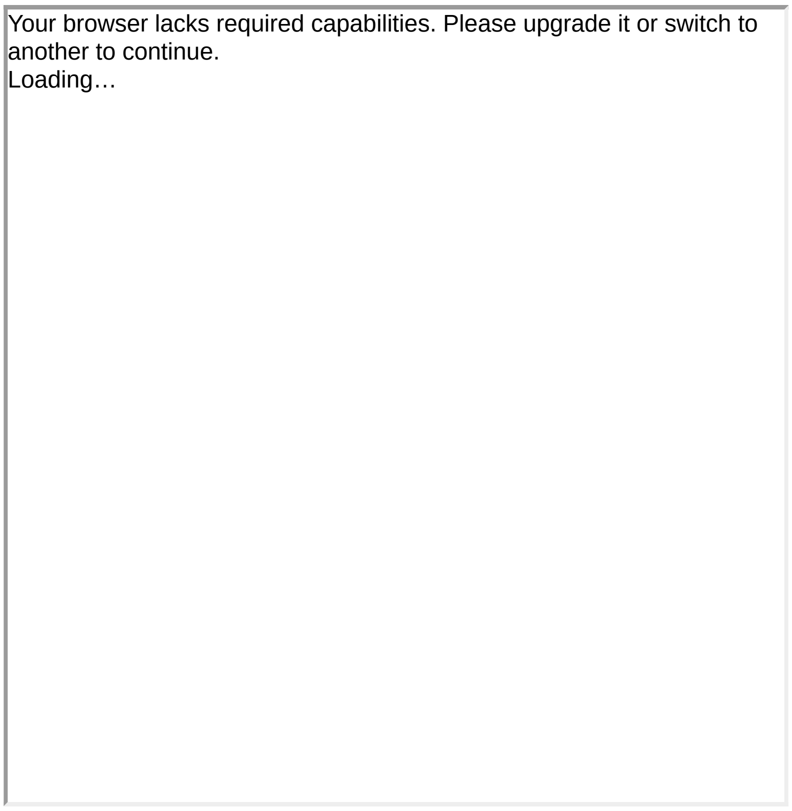
Download: Twee Code

"Static Healthbars": SugarCube (v2.18)

Summary

"Static Healthbars" demonstrates how to write HTML elements using variable values. In this example, Attribute Directive markup is used to inject the current value of the *$heath* story variable into the `<progress>` and `<meter>` elements.

Live Example



Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
Static Healthbars for SugarCube

:: Start
<<set $health to 80>>

Show a healthbar using a Progress element:
<progress @value="$health" max="100"></progress>

Show a healthbar using a Meter element:
<meter @value="$health" min="0" max="100"></meter>
```

Download: Twee Code

"Static Healthbars": Snowman (v1.3)

Summary

"Static Healthbars" demonstrates how to write HTML elements using variable values. In this example, Underscore template functionality is used to create `<progress>` and `<meter>` elements.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Static Healthbars in Snowman

:: Start
<%
    window.setup = {};
    window.setup.health = 80;
%>

Show a healthbar using a Progress element:
<%= '<progress value="' + window.setup.health + '" max="10

Show a healthbar using a Meter element:
<%= '<meter value="' + window.setup.health + '" min="0" ma
```

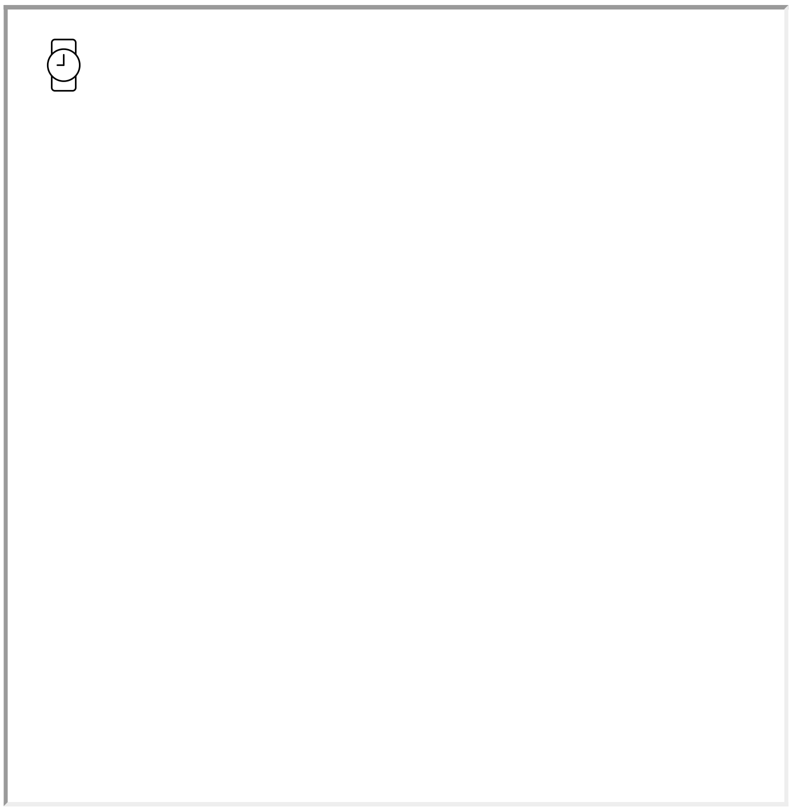
Download: Twee Code

"Story and Passage API": Chapbook (v1.0.0)

Summary

Often, it can be useful to access information about the story or other passages while it is running. In Chapbook, the *engine.story* object provides two functions, **passageNamed()** and **passageWithId()**, for accessing other passages. Combined with the `[JavaScript]` modifier, these functions and values they return when given existing passage names, can be used to show the name and source of one passage in another.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Story and Passage API

:: Start
The title of this story is "{story.name}".

The title of this passage is "{passage.name}".

[JavaScript]
    // Find a passage by name
    let storagePassage = engine.story.passageNamed("Storage

    // Write the name
    write('The name of the passage is "' + storagePassage.r

    // Write the source
    write('The name of the passage is "' + storagePassage.s

:: Storage
This is content in the storage passage!
```

Download: Twee Code

"Story and Passage API": SugarCube (v2.18)

Summary

Often, it can be useful to access information about the story or other passages while it is running. The Story and Passage APIs in SugarCube allow for getting this type of information.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Story and Passage API in SugarCube

:: Start
The title of this story is "<<print Story.title >>."

<<set $passage to Story.get("Storage")>>

The title of the passage is "<<print $passage.title>>."

The text of the passage is "<<print $passage.text >>"


:: Storage
This is content in the storage passage!
```

Download: Twee Code

"Story and Passage API": Snowman (v1.4)

Summary

Often, it can be useful to access information about the story or other passages while it is running. The **window.story.passage()** function and *window.passage.name* propertiy in Snowman allow for getting this type of information.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Story and Passage API in Snowman

:: Start
The title of this story is "<%= window.story.name %>."

<%
    window.setup = {};
    window.setup.passage = window.story.passage("Storage");
%>

The name of the passage is "<%= setup.passage.name %>."

The source of the passage is "<%= setup.passage.source %>"

:: Storage
This is content in the storage passage!
```

Download: Twee Code

"Style Markup": Chapbook (v1.0.0)

Summary

Chapbook uses a customized sub-set of Markdown to support its style formatting.

Notes: The Text Formatting section of the Chapbook Guide states you can also use a # character to indicate a *Numbered List* item. This currently (as of v1.0.0-beta) isn't correct, as that character actually results in a *Level 1 Header*.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Style Markup in Chapbook

:: Start
*Emphasis (aka Italics)* or _using single underscores_ <br>
**Strong Emphasis (aka Bold)** or __using double underscore
Combined Emphasis with **asterisks and _underscores_** <br>
<del>Strikethrough text</del> <br>
Super<sup>script</sup> <br>
Sub<sub>script</sub> <br>
`Monospaced Type (aka Code Block)` <br>
~~Small Caps~~
<blockquote>Quote</blockquote>

* A Bulleted list item (using asterisk)
* Another Bulleted list item (using asterisk)
- A Bulleted list item (using minus)
- Another Bulleted list item (using minus)
+ A Bulleted list item (using plus)
+ Another Bulleted list item (using plus)

1. A Numbered list item
2. Another Numbered list item

[align left]
Text is left-aligned.
[align center]
Text is centered / centred.
[align right]
Text is right-aligned.
[Continue]
Text-alignment has been reset to the default.

Ignoring of \*Formatting\* Characters <br>
More ignoring of \_\_Formatting\_\_ Characters



Above Section Break is Chapbook specific or standard HTML..
***
Below Section Break is supported Markdown extras...

# Level 1 Heading
## Level 2 Heading
### Level 3 Heading
#### Level 4 Heading
##### Level 5 Heading
###### Level 6 Heading
```

```
Alternative Level 1 Heading
======
Alternative Level 2 Heading
------

| Table mark-up    | with             | alignment    |
| ---            | :---:             | ---:        |
| column 1 is    | left-aligned    | 1            |
| col 2 is         | centered       | 10        |
| col 3 is         | right-aligned    | 100        |
```

Download: Twee Code

"Style Markup": Harlowe (v2.0)

Summary

In Harlowe, style markup can take many forms. Covering italics and boldface as basic examples, Harlowe also provides markup for creating alignment and columns as well.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Style Markup in Harlowe

:: Start
//Italics//
''Boldface''
~~Strikethrough text~~
*Emphasis*
**Strong emphasis**
Super^^script^^
``[[Escaped double square brackets]]``
#Level 1 heading
##Level 2 heading
###Level 3 heading
####Level 4 heading
#####Level 5 heading
######Level 6 heading
* Bulleted item
    *    Bulleted item 2
  ** Indented bulleted item
 0. Numbered item
   0. Numbered item 2
 0.0. Indented numbered item
 ==>
This is right-aligned
  =><=
This is centered
 <==>
This is justified
<==
This is left-aligned (undoes the above)
===><=
This has margins 3/4 left, 1/4 right
  =><=====
This has margins 1/6 left, 5/6 right.
|==
This is in the leftmost column, which has a right margin of
    =|||=
This is in the next column, which has margins of 1 letter w
 =====||
This is in the right column, which has a right margin of ab
  |==|
This text is not in columns, but takes up the entire width,
```

Download: Twee Code

"Style Markup": SugarCube (v2.18)

Summary

In SugarCube, style markup follows initial rules established in earlier versions of Twine while also adding many new ones.

Live Example

```
Your browser lacks required capabilities. Please
upgrade it or switch to another to continue.
Loading...
```

Download: Live Example

Twee Code

```
:: StoryTitle
Style Markup in SugarCube

:: Start
//Emphasis//
''Strong Emphasis''
==Strikethrough==
Super^^script^^
Sub~~script~~
> Quote
>> Nested quote
* A list item
* Another list item
# A list item
# Another list item
"""No //format//"""
@@Highlight Inline@@
!Level 1 Heading
!!Level 2 Heading
!!!Level 3 Heading
!!!!Level 4 Heading
!!!!!Level 5 Heading
!!!!!!Level 6 Heading
```

Download: Twee Code

"Style Markup": Snowman (v1.3.0)

Summary

Snowman uses a sub-set of markdown for in-line styling. Snowman does not support multi-line markdown variations.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Style Markup in Snowman

:: Start
*Emphasis* or _Emphasis_.

**Strong emphasis** or __Strong emphasis__.

~~Strikethrough~~

1. First ordered list item
2. Another item

# H1
## H2
### H3
#### H4
##### H5
###### H6

Escaped code line.
Another line of code.

|    Tables     |    Are        |     Cool      |
| ------------- |:-------------:| -------------:|
| col 3         |   is          | right-aligned |
| col 2         |   is          |   centered    |
| col 1         |   is          |  left-aligned |

> Blockquotes are useful.
> This line is part of the same quote.
```

Download: Twee Code

"Style Markup": Sugarcane (v1.4.2)

Note: The following example is designed for Twine 1.4.2.

Summary

Style formatting in Twine 1.4.2 is supported across all story formats and includes both more basic markup such as italics and bolds and the more advanced construction of HTML tables.

Live Example

Download: Live Example

Twee Code

```
:: Start
Italics: //text//
Boldface: ''text''
Underline: __text__
Strikethrough: ==text==
Subscript: H~~2~~O
Superscript: meters/second^^2^^
Comment: /%a comment%/
Error: @@error@@
Inline styling: @@font-weight:bold;text@@
Bulleted list:
* one
* two
Numbered list:
# one
# two
!Heading 1
!!Heading 2
!!!Heading 3
!!!!Heading 4
!!!!!Heading 5
!!!!!!Heading 6
|!table header |!table header |!table header |
|row 1|row 1|row 1|
|row 2|row 2|row 2|
|>|row 3|row 3|
|>|>|row 4|
|rows 5 and 6|row 5|row 5|
|~|row 6|row 6|
|rows 7, 8 and 9|>|row 7|
|~|>|row 8|
|~|row 9|row 9|
|table caption|c


:: StoryTitle
Style Markup


:: StoryAuthor
Videlais
```

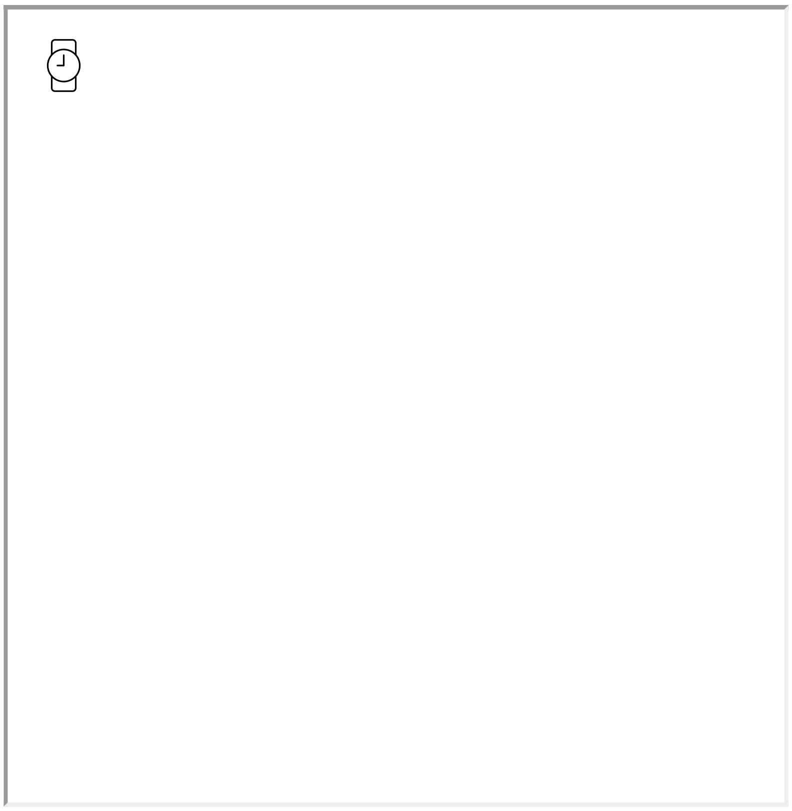
Download: Twee Code

"Timed Passages": Chapbook (v1.0.0)

Summary

Made famous in *Queers in Love at the End of the World* (2013), "Timed Passages" uses the JavaScript function **setTimeout()** and Chapbook's internal **go()** function to reload a passage every second. It combines the use of the Vars Section with multiple modifiers.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Timed Passages

:: Start
timeLeft: 11
--
[[Start->Timer]]

:: World End
The world ended.

:: Timer
timeLeft: timeLeft - 1
--
[if timeLeft > 0]
  There are {timeLeft} seconds left.
[else]
  {embed passage: 'World End'}
[JavaScript]
  window.setTimeout(() => go('Timer'), 1000);
```

Download: Twee Code

"Timed Passages": Harlowe (v2.0)

Summary

Made famous in *Queers in Love at the End of the World* (2013), "Timed Passages" uses the the `(live:)` macro to count seconds while checking if the timer has reached zero. If so, the `(goto:)` macro will immediately go to another passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Harlowe: Timed Passages

:: StoryStylesheet[stylesheet]
tw-include[type="startup"]{
    display: none;
}
tw-sidebar {
      display:none;
}


:: Start
[[Start Timer|First Passage]]

:: World End
The world ended.

:: First Passage
(display: "Timer")

[[Second Passage]]

:: Timer
{
    (live: 1s)[
        (if: $timer is 0)[
            (stop:)
            (goto: "World End")
        ]
        (else: )[
            (set: $timer to it - 1)
            The world will end in $timer seconds
        ]
    ]
}

:: Second Passage
(display: "Timer")

[[First Passage]]

:: Startup[startup]
(set: $timer to 10)
```

Download: Twee Code

See Also

Delayed Text, Typewriter Effect

"Timed Passages": SugarCube (v2.18)

Summary

Made famous in *Queers in Love at the End of the World* (2013), "Timed Passages" uses the the `<<repeat>>` macro to repeat while checking if the timer has reached zero. If so, the `<<goto>>` macro will immediately transition to another passage.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading…

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Timed Passages

:: StoryJavaScript[script]
UIBar.destroy();

:: Start
[[Start Timer|First Passage]]

:: Timer
<span id="countdown">The world will end in $seconds seconds
<<silently>>
    <<repeat 1s>>
        <<set $seconds to $seconds - 1>>
        <<if $seconds gt 0>>
            <<replace "#countdown">>The world will end in $
        <<else>>
            <<replace "#countdown">><</replace>>
            <<goto "World End">>
            <<stop>>
        <</if>>
    <</repeat>>
<</silently>>

:: First Passage
<<include "Timer">>

[[Second Passage]]

:: Second Passage
<<include "Timer">>

[[First Passage]]

:: World End
The world has ended.

:: StoryInit
<<set $seconds to 10>>
```

Download: Twee Code

See Also

Delayed Text, Typewriter Effect

"Timed Passages": Snowman (1.4)

Summary

Made famous in *Queers in Love at the End of the World* (2013), "Timed Passages" uses the the `_`**.delay()** function to count seconds while checking if the timer has reached zero. If so, the **window.story.show()** function will immediately transition to another passage.

Live Example

Download: Live Example

Twee Code

```
:: Start
There are <span class="time-left">10</span> seconds left.

<%
$(function() {
    var timeLeft = parseInt($('.time-left').text());

    function tick() {
        if (--timeLeft === 0) {
            story.show('World End');
        }
        else {
            $('.time-left').text(timeLeft);
        }

        _.delay(tick, 1000);
    }

    /* Start ticking. */

    _.delay(tick, 1000);
});
%>

:: World End
The world ended.
```

Download: Twee Code

See Also

Delayed Text, Typewriter Effect

"Timed Progress Bars": SugarCube (v2.18)

Summary

"Timed Progress Bars" uses the **Macro.Add()** function in SugarCube to introduce a new macro. Using jQuery within the definition, the new macro records arguments passed to it and creates outer and inner `<div>` elements with classes defined in the Story Stylesheet. Using a combination of **setInterval()** and **setTimeout()**, a timer is created based on the argument passed to the macro. The length and color of an inner `<div>` element is adjusted based on the remaining time each loop.

When the timer runs out, the *payload* of the macro is run and the length of the inner `<div>` element is reduced to 0.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Timed Progress Bars in SugarCube

:: UserScript[script]
/*
        Macro: timedprogressbar

        Description: Show a dynamically-created "progress b
        that changes colors as its timer runs down.

        Original design: Akjosch (https://github.com/Akjosc

        Arguments:
            [0]: The time to run in seconds
            [1]: The length of the progress bar in CSS unit
*/
Macro.add("timedprogressbar", {
    isAsync : true,
    tags: null,
    handler: function() {
        // Filter the payload for newlines and save it for
        var payload = this.payload[0].contents.replace(/\n$

        // Save or generate a default duration
        var duration = (Number(this.args[0]) || 60) * 1000;

        // Save or generate a width
        var width = this.args[1] || "100%";

        // Generate a unique hash
        var hash = Math.floor(Math.random() * 0x100000000).

        //  Create an outer ID
        var outerId = "outer_" + hash;

        // Create an inner ID
        var innerId = "inner_" + hash;

        // Create an outer div,
        // add an ID,
        // add a class,
        // change the CSS width, and
        // append to the output
        var progressbar = $("<div>")
        .attr("id", outerId)
        .addClass("progress-bar")
        .css('width', width)
        .appendTo(this.output);

        // Create an inner div,
        // add an ID,
        // add a class,
        // change the CSS width, and
        // append to the progressbar
        var progressvalue = $("<div>")
        .attr("id", innerId)
        .addClass("progress-value")
        .css('width', "100%")
        .appendTo(progressbar);

        // Create a function to convert into hexadecimal
        var toHex = function(num) {
            var res = Math.round(Number(num)).toString(16);
            return (res.length === 1 ? "0" + res : res);
        };

        // Save a reference to possible payload content
        var functionToRun = this.createShadowWrapper(
            payload
                ? function() { Wikifier.wikifyEval(payload)
                : null
        );

        // Watch for the :passagedisplay event once
        jQuery(document).one(":passagedisplay", function()

            // Get the current time
            var timeStarted = (new Date()).getTime();

            // Save a reference to the setInterval functior
            var workFunction = setInterval(function() {

                // Check if the element is still 'connectec
                if(! progressbar.prop("isConnected") ) {

                    // Navigated away from the passage
                    clearInterval(workFunction);
                    return;
                }

                // Figure out how much time has passed
                var timePassed = (new Date()).getTime() - t

                // Check if the timer has run out
                if(timePassed >= duration) {
                    // Reduce the inner width to 0
                    progressvalue.css('width', "0");

                    // Clear interval
                    clearInterval(workFunction);

                    // Run the inner function (if set)
                    setTimeout(functionToRun, 40);
                    return;
                }

                // Update the progress percentage
                var percentage = 100 - 100 * timePassed / d

                // Save the new color
                var color = "#"
                    + toHex(Math.min(255, 510 -  5.1 * perc
                    + toHex(Math.min(255, 5.1 * percentage)

                // Update the background color of the inner
                progressvalue.css("backgroundColor", color)

                // Update the inner div width
                progressvalue.css("width", (100 - 100 * tim

            }, 40);

        });
    },
});


:: UserStylesheet[stylesheet]
.progress-bar {
    position: relative;
    border: 1px solid #777;
    background: black;
    height: 1em;
}

.progress-value {
    position: absolute;
    top: 0;
    left: 0;
    height: 100%;
    background: #00ff00;
}
```

```
:: Start
<<timedprogressbar 5 20em>>
    <<run UI.alert("Too late!")>>
<</timedprogressbar>>
```

Download: Twee Code

See Also

Adding Functionality

"Timed Progress Bars": Snowman (v1.3.0)

Summary

"Timed Progress Bars" creates a global *window.setup* object and function, **timedprogressbars()**. Using jQuery within the definition, the function creates outer and inner `<div>` elements with CSS classes defined in the Story Stylesheet. Using a combination of **setInterval()** and **setTimeout()**, a timer is created. The length and color of an inner `<div>` element is adjusted based on the remaining time each loop.

When the timer runs out, the function argument is run and the length of the inner `<div>` element is reduced to 0.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Timed Progress Bars in Snowman

:: UserScript[script]
// Use or create window.setup
window.setup = window.setup || {};

/*

        Description: Show a dynamically-created "progress b
        that changes colors as its timer runs down.

        Arguments:
            duration: time in seconds
            width: CSS width
            functionToRun: the function to execute when the

*/

setup.timedprogressbars = function(duration, width, functic

        // Save or generate a default duration
        var duration = (Number(duration) || 60) * 1000;

        // Save or generate a width
        var width = width || "100%";

        // Generate a unique hash
        var hash = Math.floor(Math.random() * 0x100000000).

        //  Create an outer ID
        var outerId = "outer_" + hash;

        // Create an inner ID
        var innerId = "inner_" + hash;

        // Create an outer div,
        // add an ID,
        // add a class,
        // change the CSS width, and
        // append to the output
        var progressbar = $("<div>")
        .attr("id", outerId)
        .addClass("progress-bar")
        .css('width', width)
        .appendTo("#passage");

        // Create an inner div,
        // add an ID,
        // add a class,
        // change the CSS width, and
        // append to the progressbar
        var progressvalue = $("<div>")
        .attr("id", innerId)
        .addClass("progress-value")
        .css('width', "100%")
        .appendTo(progressbar);

        // Create a function to convert into hexadecimal
        var toHex = function(num) {
            var res = Math.round(Number(num)).toString(16);
            return (res.length === 1 ? "0" + res : res);
        };

        // Watch for the :passagedisplay event once
        jQuery(document).one("showpassage:after", function(

            // Get the current time
            var timeStarted = (new Date()).getTime();

            // Save a reference to the setInterval functior
            var workFunction = setInterval(function() {

                // Check if the element is still 'connectec
                if(! progressbar.prop("isConnected") ) {

                    // Navigated away from the passage
                    clearInterval(workFunction);
                    return;
                }

                // Figure out how much time has passed
                var timePassed = (new Date()).getTime() - t

                // Check if the timer has run out
                if(timePassed >= duration) {

                    // Reduce the inner width to 0
                    progressvalue.css('width', "0");

                    // Clear interval
                    clearInterval(workFunction);

                    // Run the inner function (if set)
                    setTimeout(functionToRun, 40);
                    return;
                }

                // Update the progress percentage
                var percentage = 100 - 100 * timePassed / d

                // Save the new color
                var color = "#"
                    + toHex(Math.min(255, 510 -  5.1 * perc
                    + toHex(Math.min(255, 5.1 * percentage)

                // Update the background color of the inner
                progressvalue.css("backgroundColor", color)

                // Update the inner div width
                progressvalue.css("width", (100 - 100 * tim

            }, 40);

        });
};




:: UserStylesheet[stylesheet]
.progress-bar {
    position: relative;
    border: 1px solid #777;
    background: black;
    height: 1em;
}
.progress-value {
    position: absolute;
    top: 0;
    left: 0;
    height: 100%;
    background: #00ff00;
}



:: Start
<script>

    setup.timedprogressbars(5, "20em", function(){
        // Hide the progress bar
        $(".progress-bar").css("display", "none");
        // Display the result
        $("#results").text("Too late!");
    });

</script>

<div id="results"></div>
```

Download: Twee Code

See Also

Adding Functionality

"Turn Counter": Chapbook (v1.0.0)

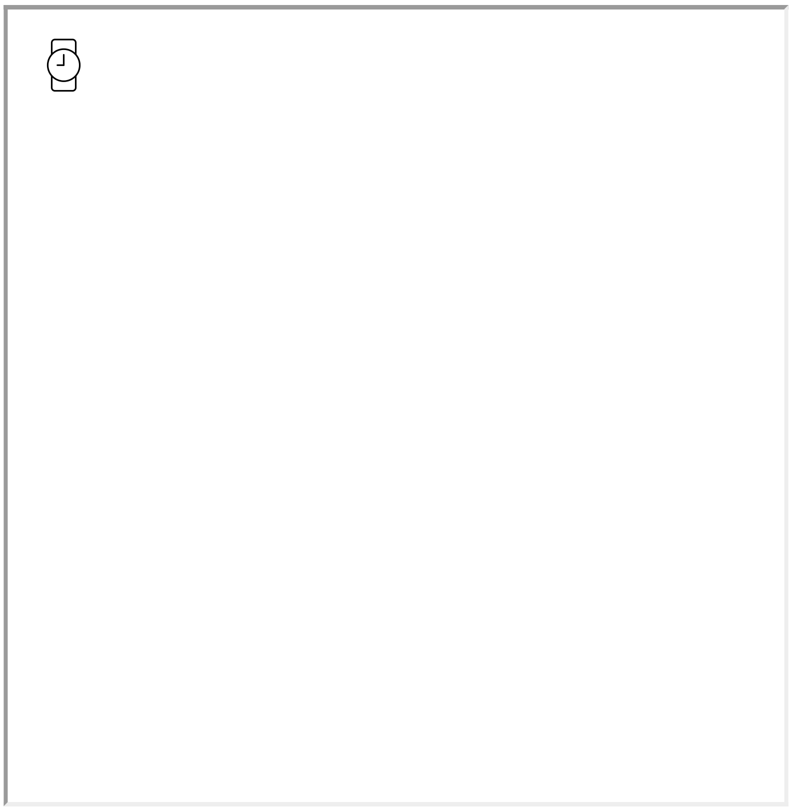
Summary

In Chapbook, the global variable *trail* stores all of the passages visited as an increasing array. For each passage visited, a new entry is added.

Sometimes known as "wrap around," the modulus operator (%) is used to get the remainder of the number of "turns" (number of passages) divided by 24. This creates a clock where its value shows one of a series of strings representing "morning", "mid-morning", "afternoon", or "night."

By visiting other passages, the turn count is increased and the hour reaches 23 before being reset back to 0 before increasing again.

Live Example



Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Turn Counter

:: Start
hour: trail.length % 24
--
{embed passage: "Turn Counter"}
Rooms:

[[Back Room]]

[[Left Room]]

[[Right Room]]


:: Back Room
{embed passage: "Turn Counter"}
Rooms:

[[Left Room]]

[[Right Room]]

[[Front Room|Start]]


:: Left Room
{embed passage: "Turn Counter"}
Rooms:

[[Right Room]]

[[Back Room]]

[[Front Room|Start]]


:: Right Room
{embed passage: "Turn Counter"}
Rooms:

[[Left Room]]

[[Back Room]]

[[Front Room|Start]]
```

```
:: Turn Counter
hour: trail.length % 24
--
[if hour <= 8]
  It is morning.

[if hour > 8 && hour <= 12]
  It is mid-morning.

[if hour > 12 && hour <= 16]
  It is afternoon.

[if hour > 16]
  It is night.
```

Download: Twee Code

"Turn Counter": Harlowe (v2.0)

Summary

"Turn Counter" demonstrates the use of the `(history:)` macro in keeping track of "turns" (number of passages visited).

In this example, the *length* of the array returned by using the `(history:)` macro is compared to its modulo 24 value. Sometimes known as "wrap around," the modulus operator (%) is used to get the remainder of the number of "turns" (passages) divided by 24. This creates a clock where its value shows one of a series of strings representing "morning", "mid-morning", "afternoon", or "night."

By visiting other passages, the turn count is increased and the hour reaches 23 before being reset back to 0 before increasing again.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Turn Counter in Harlowe

:: Start
Rooms:
[[Back Room]]
[[Left Room]]
[[Right Room]]

:: Turn Counter[header]
{
    (set: $hour to (history:)'s length % 24 )
    (if: $hour <= 8)[It is morning.]
    (if: $hour > 8 and $hour <= 12)[It is mid-morning.]
    (if: $hour > 12 and $hour <= 16)[It is afternoon.]
    (if: $hour > 16)[It is night.]
}



:: Back Room
Rooms:
[[Left Room]]
[[Right Room]]
[[Front Room|Start]]

:: Left Room
Rooms:
[[Right Room]]
[[Back Room]]
[[Front Room|Start]]

:: Right Room
Rooms:
[[Left Room]]
[[Back Room]]
[[Front Room|Start]]
```

Download: Twee Code

"Turn Counter": SugarCube (v2.18)

Summary

"Turn Counter" demonstrates the use of the *State.turns* attribute of the *State* object to keep track of the "turns" (moments within the story).

In this example, the *State.turns* property is compared to its modulo 24 value. Sometimes known as "wrap around," the modulus operator (%) is used to get the remainder of the number of "turns" (moments) divided by 24. This creates a clock where its value shows one of a series of strings representing "morning", "mid-morning", "afternoon", or "night."

This example also uses the special name "PassageHeader" as a named passage that is prepended to each passage in the story. The results of the modulo 24 calculation and clock string is displayed on every passage. By visiting other passages, the turn count is increased and the hour reaches 23 before being reset back to 0 before increasing again.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
SugarCube: Turn Counter

:: Start

Rooms:
[[Back Room]]
[[Left Room]]
[[Right Room]]

:: Back Room

Rooms:
[[Left Room]]
[[Right Room]]
[[Front Room|Start]]

:: Left Room

Rooms:
[[Right Room]]
[[Back Room]]
[[Front Room|Start]]

:: Right Room

Rooms:
[[Left Room]]
[[Back Room]]
[[Front Room|Start]]

:: PassageHeader
<<set $hour to State.turns % 24>>
<<if $hour <= 8>>It is morning.<</if>>
<<if $hour > 8 and $hour <= 12>>It is mid-morning.<</if>>
<<if $hour > 12 and $hour <= 16>>It is afternoon.<</if>>
<<if $hour > 16>>It is night.<</if>>
```

Download: Twee Code

"Turn Counter": Snowman (v1.3.0)

Summary

"Turn Counter" demonstrates the use of the *window.story.history* array in keeping track of "turns" (number of passages visited). The ***window.story.render()*** function is used to "display" or otherwise include another passage at the start of each.

In this example, the *length* of the array *window.story.history* is compared to its modulo 24 value. Sometimes known as "wrap around," the modulus operator (%) is used to get the remainder of the number of "turns" (passages) divided by 24. This creates a clock where its value shows one of a series of strings representing "morning", "mid-morning", "afternoon", or "night."

By visiting other passages, the turn count is increased and the hour reaches 23 before being reset back to 0 before increasing again.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Turn Counter in Snowman

:: Start
<%=    window.story.render("Turn Counter") %>
Rooms:

[[Back Room]]

[[Left Room]]

[[Right Room]]

:: Back Room
<%=    window.story.render("Turn Counter") %>
Rooms:

[[Left Room]]

[[Right Room]]

[[Front Room|Start]]

:: Left Room
<%=    window.story.render("Turn Counter") %>
Rooms:

[[Right Room]]

[[Back Room]]

[[Front Room|Start]]


:: Right Room
<%=    window.story.render("Turn Counter") %>
Rooms:

[[Left Room]]

[[Back Room]]

[[Front Room|Start]]


:: Turn Counter
<%
    var hour = window.story.history.length % 24;
    if(hour <= 8){%>
        It is morning.
    <%}
    if(hour > 8 && hour <= 12){%>
        It is mid-morning.
    <%}
    if(hour > 12 && hour <= 16){%>
        It is afternoon.
    <%}
    if(hour > 16){%>
        It is night.
    <%}
%>
```

Download: Twee Code

"Typewriter Effect": Chapbook (v1.0.0)

Summary

"Typewriter Effect" demonstrates how to create a delayed character-by-character effect. In Chapbook, new modifiers can be added through the *engine.extend()* function. This examples creates a new modifier called `[typewriter]` that accepts a time in milliseconds.

The `[typewriter]` modifier creates a series of `<span>` elements for each character found within the output of the modifier and sets an **animation-delay** equal to the time given to the modifier multiplied by the position of the character within the total length of the text output. When used, each character will appear within the passage as if "typed" based on the time given to the modifier.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Typewriter

:: Start
[JavaScript]
engine.extend('1.0.0', () => {
    config.template.modifiers = [{
        match: /^typewriter\s/i,
        process(output, {invocation}) {
            // Get the time
            let time = invocation.replace(/^typewriter\s/i,

            // Save original text
            let text = output.text;

            // Get length of original text
            let length = text.length;

            // Set initial index
            let index = 0;

            // Wipe out output to start
            output.text = "";

            // Loop through the text
            //  -- Add a new <span> for each chracter
            //  -- Set the class "fade-in"
            //  -- Set the delay as equal to time multiplie
            for(let i = 0; i < length; i++) {
                output.text += `<span class='fade-in' style
            }

        }
    }, ...config.template.modifiers];
});
[continued]
[[Start TyperWriter]]

:: Start TyperWriter
[typewriter 1000]
Hello, world!
[continued]
```

Download: Twee Code

"Typewriter Effect": Harlowe (v2.0)

Summary

"Typewriter Effect" demonstrates how to create a delayed character-by-character effect. In Harlowe, this is achieved using the *(live:)* macro for delayed showing and the *(append:)* macro to append text to a hook.

Note: Additional Harlowe code will not be run within the $typewriterText variable and will all be printed as-is. This code can only be used once per passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Typewriter Effect in Harlowe

:: Start
<!-- Set the text to show -->
(set: $typewriterText to "Hello, world!")
<!-- Display (call) the Typewriter passage -->
(display: "Typewriter")

:: Typewriter
{
    <!-- Create a variable to track the position within the
    (set: $typewriterPos to 1)

    <!-- Create a hook to hold the typed text -->
    |typewriterOutput>[]

    <!-- Set a delay of 20ms seconds per loop -->
    (live: 20ms)[

        <!-- Add the next character to the hook -->
        (append: ?typewriterOutput)[(print: $typewriterText

        <!-- Update the position -->
        (set: $typewriterPos to it + 1)

        <!-- If it's gone past the end, stop -->
        (if: $typewriterPos is $typewriterText's length + 1
            (stop:)
        ]
    ]
}
```

Download: Twee Code

See Also

Delayed Text

"Typewriter Effect": SugarCube (v2.18)

Summary

"Typewriter Effect" demonstrates how to create a delayed character-by-character effect. In SugarCube, a `<<widget>>` macro named "typewriter" is created that uses the `<<repeat>>` and `<<stop>>` macros internally to show one character every one second.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Typewriter Effect in Sugarcube

:: Start
<<typewriter "Hello, world">>

:: Typewriter[widget]
\<<widget typewriter>>
\    <!-- Create a SPAN with an ID -->
\  <span id="typewriter"></span>
\    <!-- In SugarCube, arrays start at 0 -->
\  <<set _textArrayLength to 0>>
\    <!-- Repeat every second -->
\  <<repeat 1s>>
\      <!-- Test if textArrayLength is greater than length
\      <<if _textArrayLength gte $args[0].length>>
\        <<stop>>
\      <<else>>
\      <!-- Append the current position to the existing cha
\      <<append "#typewriter">>$args[0][_textArrayLength]<<
\      <!-- Update the length -->
\      <<set _textArrayLength++>>
\      <</if>>
\   <</repeat>>
\<</widget>>
```

Download: Twee Code

See Also

Delayed Text

"Typewriter Effect": Snowman (v1.3.0)

Summary

"Typewriter Effect" demonstrates how to create a delayed character-by-character effect. In Snowman, this is achieved using recursive calls to the *setTimeout()* function to repeat calls once every one second. A jQuery selector is used to find an element with the ID "typewriter" whose HTML content is updated with the text every second until it is fully shown.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Typewriter Effect in Snowman

:: UserScript[script]
// Create a global setup object
window.setup = window.setup || {};

// Add a 'typewriter' object
setup.typewriter = {};

// Save an index of the string.
//     Start at -1 because it will be increased
//  once (to 0) before the first chracter is shown.
setup.typewriter.index = -1;

// Allow users to set global text
setup.typewriter.text = "";

// Save a reference to the setTimeout call
setup.typewriter.timerReference = 0;

// Write text character by character to an element
//  with the ID "typewriter"
setup.typewriter.write = function(){
    // Test if the index is less than the text length
        if(setup.typewriter.index < setup.typewriter.text.l
            // Update the current text character-by-charact
        $("#typewriter").html(
            $("#typewriter").html() + setup.typewriter.text
        );
            // Increase the index
        setup.typewriter.index++;
            // Save the timeout reference
        setup.typewriter.timerReference = setTimeout(setup.
    } else {
        // Clear out the timeout once index is greater than
        clearTimeout(setup.typewriter.timerReference);
        // Reset the index
        setup.typewriter.index = -1;
    }

}

:: Start
<div id="typewriter"></div>
<%
    setup.typewriter.text = "Hello, world!";
    setup.typewriter.write();
%>
```

Download: Twee Code

See Also

Delayed Text

"Variable Story Styling": Chapbook (v1.0.0)

Summary

Using the `[CSS]` modifier in Chapbook, it is possible to combine expressions with variables and change the text and background colors dynamically. This examples creates a variable *color* and changes its value in the Vars Section of two passages.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Chapbook: Variable Story Styling

:: Start
color: "green"
--

[CSS]
#page article {
    color: {color};
}

[continued]
This text will be in green.

[[Switch to red text]]

:: Switch to red text
color: "red"
--
[CSS]
#page article {
    color: {color};
}

[continued]
This text will be in red.

[[Switch to green text->Start]]
```

Download: Twee Code

"Variable Story Styling": Harlowe (v2.0)

Summary

"Variable Story Styling" demonstrates how to combine the `(background:)` and `(color:)` macros as storied in a variable. Combined with the `(enchant:)` macro, the named hook `?Page` is used to select the entire page for the application of the background and color changes in each passage.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Variable Story Styling in Harlowe

:: Start
(set: $storyStyle to (background: white) + (color: green) )
(enchant: ?Page, $storyStyle)
This text is green on a white background.
[[Next Passage]]

:: Next Passage
(set: $storyStyle to (background: black) + (color: white) )
(enchant: ?Page, $storyStyle)
This text is white on a black background.
```

Download: Twee Code

"Variable Story Styling": SugarCube (v2.18)

Summary

"Variable Story Styling" demonstrates how to use the `<<toggleClass>>` macro to switch between two pre-defined style rulesets. Combined with the "body" selector, the entire page is selected and the classes are switched when the macro is used.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Variable Story Styling in SugarCube

:: UserStylesheet[stylesheet]
.green {
    background: white;
      color: green;
}
.white {
    background: black;
      color: white;
}

:: Start
<<set $classToShow to "green">>
This text is green on a white background.
<<toggleclass "body" $classToShow>>
[[Next Passage]]

:: Next Passage
<<set $classToShow to "white">>
This text is white on a black background.
<<toggleclass "body" $classToShow>>
```

Download: Twee Code

"Variable Story Styling": Snowman (v1.3.0)

Summary

"Variable Story Styling" demonstrates how to use the **toggleClass()** jQuery function to switch between two pre-defined style rulesets. Used with the “body” selector, the entire page is selected and the classes toggled when the function is called.

Live Example

Download: Live Example

Twee Code

```
:: StoryTitle
Variable Story Styling in Snowman

:: UserStylesheet[stylesheet]
.green {
    background: white;
      color: green;
}
.white {
    background: black;
      color: white;
}

:: Start
This text is green on a white background.
<%
    s.styling = "green";
    $("body").toggleClass(s.styling)
%>
[[Next Passage]]

:: Next Passage
This text is white on a black background.
<%
    s.styling = "white";
    $("body").toggleClass(s.styling);
%>
```

Download: Twee Code

"Using Add-ons": SugarCube (v2.18)

Summary

Many people have developed external add-ons for use in story formats like SugarCube. Often, these add-ons will come with instructions that should be followed to incorporate them into a story.

This example uses the `<<cyclinglink>>` macro created by Thomas Michael Edwards based on the work done by Leon Arnott for Twine 1. It's code was copied into the Story JavaScript for use in the story.

Live Example

Your browser lacks required capabilities. Please upgrade it or switch to another to continue.
Loading...

Download: Live Example

Twee Code

```
:: StoryTitle
Using Add-ons in SugarCube

:: UserScript[script]
/*! <<cyclinglink>> macro for SugarCube 2.x */
!function(){"use strict";if("undefined"==typeof version||"u

:: Start
<<cyclinglink "First" "Second" "Third">>
```

Download: Twee Code