Stimulus.js Cheatsheet

A comprehensive cheat sheet for Stimulus.js, covering its core concepts, usage, and best practices to help you build maintainable JavaScript frontends.

Core Concepts

Controllers

Controllers are the fundamental building blocks of Stimulus applications. They connect behavior to your HTML.

- A controller is a JavaScript object that encapsulates the behavior for a DOM element and its descendants.
- Controllers are defined as ES classes, extending from `Stimulus.Controller`.

Example:

```
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  connect() {
    console.log("Hello, Stimulus!")
  }
}
```

Actions

Actions connect user events (e.g., clicks, form submissions) to controller methods.

- Actions are declared using the `data-action` HTML attribute.
- `data-action` values consist of an event type, an optional target identifier, and a method name.

Example:

```
<button data-action="click->my-
controller#greet">Greet</button>
```

This connects the `click` event to the `greet` method in the `my-controller`.

Targets

Targets provide a way for controllers to reference specific elements within their element.

- Targets are declared using the `data-target` HTML attribute.
- Controllers automatically generate getter methods for accessing target elements.

Example:

```
<div data-controller="my-controller">
  <input type="text" data-target="my-
controller.name">
  <button data-action="click->my-
controller#showName">Show Name</button>
</div>


// my_controller.js
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  static targets = [ "name" ]

  showName() {
    alert(`Hello, ${this.nameTarget.value}!`);
  }
}
```

Values

Values allow you to associate typed data with your controller.

- Values are declared in the controller's `static values` definition.
- Stimulus automatically converts HTML `data-▯▯▯-▯▯` attributes to values.

Example:

```
<div data-controller="my-controller" data-my-
controller-name-value="World">
  <button data-action="click->my-
controller#greet">Greet</button>
</div>


// my_controller.js
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  static values = {
    name: String
  }

  greet() {
    alert(`Hello, ${this.nameValue}!`);
  }
}
```

Advanced Usage

Lifecycle Callbacks

Stimulus provides lifecycle callbacks that are invoked at different points in a controller's lifecycle.

- `connect()` : Called when the controller is connected to the DOM.
- `disconnect()` : Called when the controller is disconnected from the DOM.
- `initialize()` : Called only once, when the controller is created.

Example:

```
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  initialize() {
    console.log("Controller initialized");
  }

  connect() {
    console.log("Controller connected");
  }

  disconnect() {
    console.log("Controller disconnected");
  }
}
```

Using `this.element`

`this.element` refers to the controller's root DOM element. Accessing attributes, adding classes, and manipulating the DOM is all possible through `this.element`.

Example:

```
import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  connect() {
    this.element.classList.add("active");
  }

  disconnect() {
    this.element.classList.remove("active");
  }
}
```

Data Attributes

Stimulus relies heavily on `data-` attributes. Besides `data-controller`, `data-action`, and `data-target`, you can add custom `data-` attributes and access them in your controller.

Example:

```
<div data-controller="my-controller" data-
custom-value="some-value">
</div>


import { Controller } from
"@hotwired/stimulus"

export default class extends Controller {
  connect() {

console.log(this.element.dataset.customValue);
  }
}
```

Working with Forms

Submitting Forms

Stimulus can be used to enhance form submissions, handling them asynchronously with AJAX.

Example:

```
<form data-controller="form-controller" data-action="submit->form-
controller#submit">
  <input type="text" name="name">
  <button type="submit">Submit</button>
</form>


// form_controller.js
import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
  submit(event) {
    event.preventDefault();
    const formData = new FormData(this.element);

    fetch(this.element.action, {
      method: this.element.method,
      body: formData
    }).then(response => {
      // Handle the response
    });
  }
}
```

Form Input Binding

Stimulus can be used to bind form inputs to controller values, allowing real-time updates and validation.

Example:

```
<input type="text" data-controller="input-controller" data-action="input-
>input-controller#updateValue" data-input-controller-value="">
<div data-target="input-controller.output"></div>


// input_controller.js
import { Controller } from "@hotwired/stimulus";

export default class extends Controller {
  static values = { value: String };
  static targets = ["output"];

  updateValue() {
    this.valueValue = this.element.value;
    this.outputTarget.textContent = `Value: ${this.valueValue}`;
  }
}
```

Best Practices

Keep Controllers Small

Controllers should be focused on specific behaviors. Avoid creating large, monolithic controllers.

If a controller becomes too complex, consider breaking it down into smaller, more manageable controllers.

Use Targets Wisely

Targets are a powerful way to reference elements, but avoid overusing them. Only use targets when you need to directly manipulate an element from the controller.

Consider using CSS classes and event delegation for simpler interactions.

Leverage Values for Configuration

Use values to configure the behavior of your controllers. This allows you to customize controllers without modifying the code.

Values also provide a way to type-check the data passed to your controllers.