# Accessible Form and Validation Developer Guide

This supplemental guide provides additional technical details for developers who are inspecting the [Accessible Form Validation CodePen](https://codepen.io/EOTSS-ACCESS/full/gbaRrNW) example HTML, CSS, and Vanilla JavaScript coding best-practices.

## Semantic form field structure

1. **Required field example sentence** to be put at the beginning of the form when there are required fields:
<p>Fields marked with a red asterisk <span class="ads-asterisk" aria-hidden="true">\*</span> are required.</p>
	* We hide the red asterisk from screen readers using aria-hidden="true" since it is read as “star”.
	* We add aria-required="true" to each required field (or fieldset for radio or checkbox group) to support screen readers.
	* For multi-page forms use this sentence on each screen or page.
2. **Use a red asterisk \*** at the end of each label (or legend tag for a checkbox or radio group) to indicate that it is required.
	1. **Example for <label> tag:**
	<label id="firstNameLabel" for="firstName">First Name <span class="ads-asterisk" aria-hidden="true">\*</span></label>
	2. **Example for <legend> tag:**<legend>How do you want to receive your security code? <span class="ads-asterisk" aria-hidden="true">\*</span></legend>
3. Reference all form elements (input, textarea, select, checkbox, and radio buttons) using the [**“for” attribute**](https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Attributes/for) on the <label> tag to point to the input’s unique ID.
	1. **Example for <input> tag** (this also applies for select and textarea tags):
	<label id="emailLabel" for="email">Email <span class="ads-asterisk" aria-hidden="true">\*</span></label>
	<input type="text" name="email" id="email" aria-required="true" aria-describedby="emailError">
	2. **Example for checkbox and radio button:**
	You can increase the clickable target area by wrapping the checkbox or radio element inside the label tag. You should still include the “for” attribute for enhanced accessibility compatibility.
	<label for="morning"><input type="checkbox" id="morning" name="morning"> Morning</label>

<label for="securityCode\_email"><input type="radio" id="securityCode\_email" name="securityCode" value="email"> Email</label>

1. Use aria-required="true" on **required fields** (using the native “required” HTML5 attribute will result in native HTML5 error messaging which is harder to style and might cover browser native autocomplete options).

	1. Example for **<input> tag** (this also applies for select and textarea tags):
	<input type="text" name="email" id="email" aria-required="true" aria-describedby="emailError">
	2. Example for a **single** **checkbox**:

<input type="checkbox" id="terms\_conditions\_required" aria-required="true" name="terms">
🡪 [Opt-in checkbox CodePen example](https://codepen.io/EOTSS-ACCESS/pen/ZYbxpyo)

* 1. Example for **checkbox** and **radio button group**:
	To properly associate a checkbox or radio button group as a required field you must add the aria-required="true" on the <fieldset> tag.

	**Important:** Don’t add it to each checkbox or radio button, this would result in repetition for screen reader users.

	<fieldset aria-required="true" aria-describedby="securityCodeError" id="securityCode">

Also, for screen reader users to hear the **error message** you must add the aria-describedby pointing to the error message container ID on the <fieldset> tag.

## Error Messages

### Screen Reader Support using ARIA

* Add the aria-label="error:" to the **error icon** to provide clear screen reader output when an error message is announced.
* Use aria-describedby on input, select, or textarea fields to reference the ID of the error message container.
* Use JavaScript to inject the error icon and text message to show it.

When using a **checkbox** or **radio button group** put the aria-describedby onto the <fieldset> tag. The error message will be read by the screen reader once the first element in the group receives focus.
**After correcting the error** and triggering the field’s JavaScript on “blur” event, remove the error icon and error message text from the error container.

Important: don’t use role="alert"
Don’t use role="alert" for inline error message <div> containers like the ones used in this example, doing so would interrupt the screen reader and read all error messages in a row and be very confusing for the user.

## Error Messages

### Showing error messages on form submit

For your form validation to deliver the best user experience, validate your form on submit. Form submit should be triggered by activating the “submit” button.

**Important:** avoid showing error messages using the JavaScript “blur” event which occurs when a field loses keyboard focus or after the user tabs off the form field. This is important especially for larger forms containing more than one input. Using on “blur” validation can be very disorienting and confusing for screen reader users since the error message is shown after the user tabs off the field. Some screen reader users have partial vision and rely on assistive technology to supplement their ability to see. The visual and assistive technology experiences should always match and be consistent.

Removing errors after they have been fixed using the JavaScript “blur” event
After the user fixes the error on an input field and the field loses focus by either clicking elsewhere on the screen or tabbing off the field, the error message should be removed. Also, any ARIA attributes used to indicate that the field has an error should be either removed or set to “false” (i.e., aria-invalid="false").

Developing your form validation JavaScript

1. Your submit button HTML must include type="submit" for robust browser support: <button type="submit">Submit</button>
2. Use JavaScript to build your font-end form validation. The CodePen example includes HTML, CSS, and detailed Vanilla JavaScript to get you started.
Use JavaScript to do the following:
	1. add or remove the error message icon and text on a field
		1. **to add:**
		document.querySelector('[data-error-for="firstName"]').innerHTML = errorIcon + ' <span>First Name is required.</span>';
		2. **to remove:**
		document.querySelector('[data-error-for="firstName"]').innerHTML = "";
	2. aria-invalid="true" attribute to indicate that an input formatting error
		1. **to add:**
		birthDayInput.setAttribute("aria-invalid","true");
		2. **to remove:**
		birthDayInput.setAttribute("aria-invalid","false");
	3. add or remove a CSS class to show an error border color on an input field. In this example we commented out the **CSS class “errorBorder”** in the CSS file. This class is used to add a red border to the input, select, and textarea fields. It is not a WCAG requirement to show a red border on input fields as part of an error indicator. However, we decided to include the JavaScript and CSS for users who might use this style as part of the design system.
		1. **to add the CSS class:**
		lastNameInput.classList.add("ads-errorBorder");
		2. **to remove CSS class:**
		lastNameInput.classList.remove("ads-errorBorder");
3. Show a **descriptive error message** below the field or group (i.e., checkbox or radio) that references the field label name and what type of error it is. The aria-label="Error:" on the error icon will add “Error:” at the beginning of each error message for screen readers.
	1. Empty field error example: “First name is required.”
	Screen reader output: “Error: First name is required.”
	
	2. Formatting error example: “Enter a 10-digit phone number.”
	Screen reader output: “Error: Enter a 10-digit phone number.”
	
4. On form submit, move the focus indicator to the first field (input, select, textarea) or element (i.e., checkbox or radio button) that has a visible error message.
	1. **The first Input, select, or textarea element** that has an error will receive focus
	
	2. **For checkbox or radio groups**, the first checkbox or radio button in the group receives focus.
	
	3. **Focus indicators** must meet the [WCAG 2.1 Success Criterion 1.4.11 Non-text Contrast **(Level AA)**](https://www.w3.org/WAI/WCAG21/Understanding/non-text-contrast.html) requirement of 3:1.
	Typically a focus indicator looks like a border around the element. This is achieved by using [CSS outline properties](https://developer.mozilla.org/en-US/docs/Web/CSS/outline). To distance the outline from the element you can also use [outline-offset](https://developer.mozilla.org/en-US/docs/Web/CSS/outline-offset).

Keyboard-only user accessibility and visible focus indicator

The recommended CSS used for focus outline is using box-shadow in addition to outline and creates a border around elements that meets the 3:1 contrast requirement against all light and dark backgrounds. This example doesn’t use outline-offset to distance the outline from the element instead it uses a light color outline that is 2px wide and a dark color box shadow.

Keyboard-only users must always see a visible outline around elements that can receive focus (i.e., links, buttons, form elements).

#### Keyboard-only testing

To test that the form is keyboard accessible, tab through the form, fill it out using only a keyboard (don’t use a mouse), and verify that you always see a **visible focus** indicator.

Core Navigation Keys

* **Tab:** Moves the keyboard focus forward through all interactive elements on a page, such as links, form fields, and buttons.
* **Shift + Tab:** Moves the keyboard focus backward through the interactive elements.
* **Enter:** Activates links and buttons, similar to a mouse click.
* **Spacebar:** Used to select or deselect checkboxes and activate buttons.

##### Keys for Specific Controls

* **Arrow Keys (Up, Down, Left, Right):** Navigate through options in dropdown menus, radio button groups, and other similar controls.
* **Escape:** Closes pop-up windows, menus, or dismisses dialog boxes.

CSS for global focus indicator that works with light and dark backgrounds:

\*:focus {

/\* inner indicator \*/

 outline: 2px #F9F9F9 solid;

 outline-offset: 0;

/\* outer indicator \*/
box-shadow: 0 0 0 4px #0088ff;

}

## CSS Styles, Reflow, and Resize Text

### CSS Styles

We applied specific CSS techniques to this example form that can be found in the CSS section of the CodePen.

Use the following CSS best practices:

* Use at least 16px (1rem) font-size for all text, use a non-serif font, and use at least 1.5 line-height for labels and text
* Use global box-sizing CSS to create equal width on select and input elements
\*, \*::before, \*::after {

box-sizing: border-box;

}

* Use “flexbox” layout CSS to structure your form elements by group
.ads-flexLayout {display: flex; gap:2rem; align-items: flex-start;}
* Use media queries to stack your form elements in a single column to support smaller viewports (i.e., browser zoom, tablets, and mobile devices). Set the breakpoint earlier than needed to support “text resize” 200%.

**Always test** your implementation using browser zoom, text resize, and mobile emulation.

@media screen and (max-width: 1023px) {

 .ads-flexLayout {flex-direction: column;gap:.5rem;}

}

* **Use** “min-width” CSS only on form input fields - **don’t use** “width” or “max-width”
select, input[type="text"], input[type="email"], input[type="file"], input[type="date"], input[type="search"], input[type="month"], input[type="url"], input[type="tel"], textarea {min-width:280px;}
* **Use** “padding” and “line-height” on your input fields - **don’t** use “height”
select, input[type="text"], input[type="email"], input[type="file"], input[type="date"], input[type="search"], input[type="month"], input[type="url"], input[type="tel"], textarea {border:2px solid #949494;padding:.5rem;line-height:1.4;min-width:260px;border-radius:.5rem;}
* **Use** a “border-color” that meets the 3:1 contrast requirement
select, input[type="text"], input[type="email"], input[type="file"], input[type="date"], input[type="search"], input[type="month"], input[type="url"], input[type="tel"], textarea {border:2px solid #949494;padding:.5rem;line-height:1.2;min-width:260px;border-radius:.5rem;}
* **Optional:** change the input field border color to indicate an error that meets the 3:1 contrast requirement. This style is currently commented out in the CodePen example and uses the class name “ads-errorBorder”.

**Note:** It is not a WCAG requirement to show a red border on input fields as part of the error indicator. Make sure that the error border style is the same width as the default border to prevent layout shifts.

select.ads-errorBorder, input.ads-errorBorder {border-color: #cd0d0d;}