Disclaimer: This resource provides general guidance and is not legal advice. Consult qualified counsel for jurisdiction-specific requirements.

WCAG 2.2 Implementation Guide

Complete reference for Web Content Accessibility Guidelines 2.2

Prepared by: Halytic

Version: 1.0

Date: October 18, 2025

Standards: WCAG 2.2 AA, W3C Recommendations

Table of Contents

Back to TOC

- Introduction to WCAG 2.2 - Understanding the Structure - The Four Principles - Principle 1: Perceivable - Principle 2: Operable - Principle 3: Understandable - Principle 4: Robust - Perceivable Guidelines - 1.1 Text Alternatives - 1.2 Time-based Media - 1.3 Adaptable - 1.4 Distinguishable - Operable Guidelines - 2.1 Keyboard Accessible - 2.2 Enough Time - 2.4 Navigable - Understandable Guidelines - 3.1 Readable - 3.2 Predictable - 3.3 Input Assistance - Robust Guidelines - 4.1 Compatible - Implementation Strategies - 1. Start with Semantic HTML - 2. Progressive Enhancement - 3. Testing Strategy - Testing and Validation - Automated Testing Tools - Manual Testing Checklist - References - Alt-text List

Introduction to WCAG 2.2

Back to TOC

The Web Content Accessibility Guidelines (WCAG) 2.2 is the latest version of the international standard for web accessibility. Published by the World Wide Web Consortium (W3C), WCAG 2.2 builds upon WCAG 2.1 with additional success criteria to address emerging accessibility needs.

Key Updates in WCAG 2.2: New success criteria for focus management, drag-and-drop interactions, and target size requirements, ensuring better accessibility for users with motor impairments and mobile device users.

Understanding the Structure

WCAG 2.2 is organized around four fundamental principles:

1. **Perceivable** - Information must be presentable in ways users can perceive
2. **Operable** - Interface components must be operable by all users
3. **Understandable** - Information and UI operation must be understandable
4. **Robust** - Content must be robust enough for interpretation by assistive technologies

Each principle contains guidelines, which are further broken down into testable success criteria with three levels of conformance: A (minimum), AA (recommended), and AAA (enhanced).

The Four Principles

Back to TOC

Principle 1: Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

Key Guidelines:

- Provide text alternatives for non-text content
- Provide alternatives for time-based media
- Create content that can be presented in different ways
- Make it easier for users to see and hear content

Principle 2: Operable

User interface components and navigation must be operable.

Key Guidelines:

- Make all functionality available from a keyboard
- Provide users enough time to read and use content
- Do not design content in a way that is known to cause seizures
- Provide ways to help users navigate, find content, and determine where they are

Principle 3: Understandable

Information and the operation of user interface must be understandable.

Key Guidelines:

- Make text content readable and understandable
- Make web pages appear and operate in predictable ways
- Help users avoid and correct mistakes

Principle 4: Robust

Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies.

Key Guidelines:

- Maximize compatibility with current and future user agents

Perceivable Guidelines

Back to TOC

1.1 Text Alternatives

Success Criterion 1.1.1 Non-text Content (Level A)

Requirement: All non-text content that is presented to the user has a text alternative that serves the equivalent purpose.

Implementation Examples:

```
<!-- Good: Descriptive alt text -->
<img src="chart.png" alt="Sales increased 25% from Q1 to Q2, r

<!-- Good: Decorative image -->
<img src="decoration.png" alt="" />

<!-- Good: Complex image with long description -->
<img src="infographic.png" alt="Detailed sales performance int
<details>
  <summary>View detailed description</summary>
  <p>This infographic shows quarterly sales data...</p>
</details>
```

Testing Methods:

- Use browser developer tools to inspect alt attributes

- Test with screen readers (NVDA, JAWS, VoiceOver)
- Verify that decorative images have empty alt attributes

1.2 Time-based Media

Success Criterion 1.2.1 Audio-only and Video-only (Level A)

Requirement: For prerecorded audio-only and prerecorded video-only media, alternatives are provided.

Implementation Examples:

```
<!-- Audio with transcript -->
<audio controls>
  <source src="podcast.mp3" type="audio/mpeg" />
  <p>
    Download the <a href="podcast-transcript.txt">transcript</
    podcast.
  </p>
</audio>

<!-- Video with audio description -->
<video controls>
  <source src="presentation.mp4" type="video/mp4" />
  <track
    kind="descriptions"
    src="descriptions.vtt"
    srclang="en"
    label="English descriptions"
  />
</video>
```

1.3 Adaptable

Success Criterion 1.3.1 Info and Relationships (Level A)

Requirement: Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text.

Implementation Examples:

```
<!-- Good: Proper heading structure -->
<h1>Main Page Title</h1>
<h2>Section Title</h2>
<h3>Subsection Title</h3>

<!-- Good: Form labels -->
<label for="email">Email Address</label>
<input type="email" id="email" name="email" required />

<!-- Good: Table headers -->
<table>
  <thead>
    <tr>
      <th scope="col">Product</th>
      <th scope="col">Price</th>
      <th scope="col">Stock</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">Widget A</th>
      <td>$19.99</td>
      <td>In Stock</td>
    </tr>
  </tbody>
</table>
```

1.4 Distinguishable

Success Criterion 1.4.3 Contrast (Minimum) (Level AA)

Requirement: The visual presentation of text and images of text has a contrast ratio of at least 4.5:1, except for large text which has a contrast ratio of at least 3:1.

Implementation Examples:

```css
/* Good: High contrast text */
.text-primary {
  color: #1f2937; /* Dark gray on white = 16.5:1 ratio */
  background-color: #ffffff;
}

.text-secondary {
  color: #4f46e5; /* Indigo on white = 7.1:1 ratio */
  background-color: #ffffff;
}

/* Good: Large text with lower contrast requirement */
.large-text {
  color: #6b7280; /* Gray on white = 4.5:1 ratio */
  font-size: 18pt;
  font-weight: bold;
}
```

Testing Tools:

- WebAIM Color Contrast Checker
- Chrome DevTools Accessibility panel
- axe-core automated testing

Success Criterion 1.4.11 Non-text Contrast (Level AA)

Requirement: The visual presentation of user interface components and graphical objects has a contrast ratio of at least 3:1 against adjacent colors.

Implementation Examples:

```css
/* Good: Button with sufficient contrast */
.button-primary {
  background-color: #4f46e5; /* Indigo */
  color: #ffffff; /* White */
  border: 2px solid #3730a3; /* Darker indigo for border */
}
```

```css
/* Good: Focus indicators */
.focus-visible {
  outline: 2px solid #4f46e5;
  outline-offset: 2px;
}
```

Operable Guidelines

Back to TOC

2.1 Keyboard Accessible

Success Criterion 2.1.1 Keyboard (Level A)

Requirement: All functionality of the content is operable through a keyboard interface without requiring specific timings for individual keystrokes.

Implementation Examples:

```html
<!-- Good: Keyboard accessible button -->
<button onclick="submitForm()" tabindex="0">Submit Form</butto

<!-- Good: Skip link -->
<a href="#main-content" class="skip-link">Skip to main content

<!-- Good: Custom interactive element -->
<div role="button" tabindex="0" onkeydown="handleKeydown(event
  Custom Button
</div>
```

```javascript
// Keyboard event handling
function handleKeydown(event) {
  if (event.key === "Enter" || event.key === " ") {
    event.preventDefault();
```

```
    // Perform button action
  }
}
```

Success Criterion 2.1.2 No Keyboard Trap (Level A)

Requirement: If keyboard focus can be moved to a component of the page using a keyboard interface, then focus can be moved away from that component using only a keyboard interface.

Implementation Examples:

```
// Good: Modal with proper focus management
function openModal() {
  const modal = document.getElementById("modal");
  const firstFocusable = modal.querySelector(
    'button, [href], input, select, textarea, [tabindex]:not(
  );
  const lastFocusable = modal.querySelectorAll(
    'button, [href], input, select, textarea, [tabindex]:not(
  );
  const lastElement = lastFocusable[lastFocusable.length - 1];

  modal.style.display = "block";
  firstFocusable.focus();

  // Trap focus within modal
  modal.addEventListener("keydown", function (e) {
    if (e.key === "Tab") {
      if (e.shiftKey) {
        if (document.activeElement === firstFocusable) {
          lastElement.focus();
          e.preventDefault();
        }
      } else {
        if (document.activeElement === lastElement) {
          firstFocusable.focus();
```

```
          e.preventDefault();
        }
      }
    }

    if (e.key === "Escape") {
      closeModal();
    }
  });
}
```

2.2 Enough Time

Success Criterion 2.2.1 Timing Adjustable (Level A)

Requirement: For each time limit that is set by the content, at least one of the following is true: the user can turn off the time limit, adjust the time limit, or extend the time limit.

Implementation Examples:

```
<!-- Good: Session timeout with warning -->
<div id="timeout-warning" style="display: none;">
  <p>Your session will expire in <span id="countdown">5</span>
  <button onclick="extendSession()">Extend Session</button>
  <button onclick="logout()">Logout Now</button>
</div>
```

```
// Session timeout management
let sessionTimeout;
let warningTimeout;

function startSessionTimer() {
  sessionTimeout = setTimeout(logout, 30 * 60 * 1000); // 30 m
  warningTimeout = setTimeout(showWarning, 25 * 60 * 1000); //
}
```

```
function extendSession() {
  clearTimeout(sessionTimeout);
  clearTimeout(warningTimeout);
  hideWarning();
  startSessionTimer();
}
```

2.4 Navigable

Success Criterion 2.4.1 Bypass Blocks (Level A)

Requirement: A mechanism is available to bypass blocks of content that are repeated on multiple Web pages.

Implementation Examples:

```
<!-- Good: Skip links -->
<a href="#main-content" class="skip-link">Skip to main content
<a href="#navigation" class="skip-link">Skip to navigation</a>

<!-- Good: Landmark regions -->
<nav role="navigation" id="navigation">
  <!-- Navigation content -->
</nav>

<main role="main" id="main-content">
  <!-- Main content -->
</main>

<aside role="complementary">
  <!-- Sidebar content -->
</aside>
```

```css
/* Skip link styling */
.skip-link {
  position: absolute;
  top: -40px;
  left: 6px;
  background: #4f46e5;
  color: white;
  padding: 8px;
  text-decoration: none;
  z-index: 1000;
}

.skip-link:focus {
  top: 6px;
}
```

Understandable Guidelines

Back to TOC

3.1 Readable

Success Criterion 3.1.1 Language of Page (Level A)

Requirement: The default human language of each Web page can be programmatically determined.

Implementation Examples:

```html
<!-- Good: Language declaration -->
<html lang="en">
  <head>
    <title>My Website</title>
  </head>
  <body>
    <!-- Content in English -->
```

```
  </body>
</html>

<!-- Good: Language changes -->
<p>
  This is in English, but here's some <span lang="es">texto er
</p>
```

3.2 Predictable

Success Criterion 3.2.1 On Focus (Level A)

Requirement: When any component receives focus, it does not initiate a change of context.

Implementation Examples:

```
<!-- Bad: Context change on focus -->
<input type="text" onfocus="window.location.href='other-page.h

<!-- Good: No context change on focus -->
<input type="text" placeholder="Search..." />
<button onclick="performSearch()">Search</button>
```

3.3 Input Assistance

Success Criterion 3.3.1 Error Identification (Level A)

Requirement: If an input error is automatically detected, the item that is in error is identified and the error is described to the user in text.

Implementation Examples:

```
<!-- Good: Form with error handling -->
<form novalidate>
  <div class="form-group">
```

```
    <label for="email">Email Address</label>
    <input
      type="email"
      id="email"
      name="email"
      required
      aria-describedby="email-error"
    />
    <div id="email-error" role="alert" aria-live="polite"></di
  </div>

  <button type="submit">Submit</button>
</form>
```

```
// Form validation with accessibility
function validateForm(event) {
  event.preventDefault();
  const email = document.getElementById("email");
  const errorDiv = document.getElementById("email-error");

  if (!email.validity.valid) {
    email.setAttribute("aria-invalid", "true");
    errorDiv.textContent = "Please enter a valid email address
    email.focus();
  } else {
    email.setAttribute("aria-invalid", "false");
    errorDiv.textContent = "";
    // Submit form
  }
}
```

Robust Guidelines

Back to TOC

4.1 Compatible

Success Criterion 4.1.1 Parsing (Level A)

Requirement: In content implemented using markup languages, elements have complete start and end tags, elements are nested according to their specifications, and elements do not contain duplicate attributes.

Implementation Examples:

```
<!-- Good: Valid HTML structure -->
<div class="container">
  <h1>Page Title</h1>
  <p>This is a paragraph with <strong>bold text</strong>.</p>
  <ul>
    <li>First item</li>
    <li>Second item</li>
  </ul>
</div>

<!-- Good: Proper form structure -->
<form action="/submit" method="post">
  <fieldset>
    <legend>Contact Information</legend>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required />
  </fieldset>
</form>
```

Success Criterion 4.1.2 Name, Role, Value (Level A)

Requirement: For all user interface components, the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies.

Implementation Examples:

```
<!-- Good: Custom button with proper ARIA -->
<div
  role="button"
  tabindex="0"
  aria-label="Close dialog"
  aria-pressed="false"
  onkeydown="handleKeydown(event)"
  onclick="closeDialog()"
>
  <span aria-hidden="true">×</span>
</div>

<!-- Good: Progress indicator -->
<div
  role="progressbar"
  aria-valuenow="65"
  aria-valuemin="0"
  aria-valuemax="100"
  aria-label="Download progress"
>
  65%
</div>
```

Implementation Strategies

Back to TOC

1. Start with Semantic HTML

Use proper HTML elements for their intended purpose:

```
<!-- Good: Semantic structure -->
<header>
  <nav role="navigation">
    <ul>
```

```
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
  </nav>
</header>

<main>
  <article>
    <h1>Article Title</h1>
    <p>Article content...</p>
  </article>
</main>

<footer>
  <p>&copy; 2025 Company Name</p>
</footer>
```

2. Progressive Enhancement

Build accessible functionality first, then enhance:

```
// Base accessible functionality
function toggleMenu() {
  const menu = document.getElementById("menu");
  const isExpanded = menu.getAttribute("aria-expanded") === "t

  menu.setAttribute("aria-expanded", !isExpanded);
  menu.classList.toggle("open");
}

// Enhanced functionality
function toggleMenuWithAnimation() {
  toggleMenu(); // Base functionality

  // Add animation if supported
  if (window.matchMedia("(prefers-reduced-motion: no-preferenc
```

```
    // Add smooth animation
  }
}
```

3. Testing Strategy

Implement a comprehensive testing approach:

1. **Automated Testing**
   - axe-core integration
   - Lighthouse accessibility audits
   - HTML validation
2. **Manual Testing**
   - Keyboard navigation
   - Screen reader testing
   - Color contrast verification
3. **User Testing**
   - Testing with actual users with disabilities
   - Usability testing with assistive technologies

Testing and Validation

Back to TOC

Automated Testing Tools

```
// axe-core integration example
import axe from "axe-core";

// Run accessibility tests
axe.run(document, (err, results) => {
  if (err) throw err;
```

```
  if (results.violations.length > 0) {
    console.log("Accessibility violations found:", results.vio
  } else {
    console.log("No accessibility violations found");
  }
});
```

Manual Testing Checklist

- [ ] All interactive elements are keyboard accessible
- [ ] Focus indicators are visible and clear
- [ ] Color contrast meets WCAG 2.2 AA standards
- [ ] Images have appropriate alt text
- [ ] Form labels are properly associated
- [ ] Headings create a logical structure
- [ ] Language is properly declared
- [ ] Error messages are clear and helpful

References

Back to TOC

1. Web Content Accessibility Guidelines (WCAG) 2.2 - Official W3C guidelines
2. WCAG 2.2 Understanding Documents - Detailed explanations of success criteria
3. WebAIM Screen Reader Testing - Manual testing procedures
4. axe-core Documentation - Automated accessibility testing
5. ARIA Authoring Practices Guide - Best practices for accessible components
6. Color Contrast Analyzer - Tool for verifying contrast ratios
7. HTML Validator - W3C markup validation service
8. Accessibility Testing Tools - Comprehensive list of testing tools

Alt-text List

Back to TOC

Figure	Alt Text	Context

Halytic Logo	"Halytic logo - Website accessibility compliance platform"	Brand identification
WCAG 2.2 Badge	"WCAG 2.2 AA compliance certification badge"	Credibility indicator
Accessibility Testing Flowchart	"Flowchart showing the process of accessibility testing from automated tools to user testing"	Process visualization
Color Contrast Example	"Side-by-side comparison showing text with sufficient contrast versus insufficient contrast"	Educational example

---

> **Implementation Support:** Halytic provides comprehensive WCAG 2.2 compliance auditing and implementation guidance. Our platform automates testing while our experts provide strategic consultation for complex accessibility challenges.

This guide is prepared by Halytic and follows WCAG 2.2 AA guidelines for accessibility compliance.