Accessibility Design Tools

Second Edition

Table of Contents

Welcome

Foreword

Design exists to serve people. At SAP, this principle anchors our work as we build enterprise software for a diverse, global user base. In an increasingly digital world, accessibility is not optional - it is both a human and business imperative. Our responsibility as designers, developers, and product managers is to ensure our products are inclusive by design, enabling all users to access, understand, and interact with our solutions effectively.

I am pleased to introduce the second edition of the Accessibility Design Tools by SAP, a significant step forward in our commitment to inclusive user experience. This edition reflects insights from continued dialogue with our design and development community, alongside evolving user research. We've deepened the focus on human needs, recognizing the diversity in how people perceive, navigate, and use digital products.

Accessibility goes beyond the visual surface. It requires holistic thinking - considering interaction, alternative modes of operation, and the underlying structures that make a product usable for all. While designing for accessibility may initially seem intangible, it is entirely achievable through practice, iteration, and a user-centered mindset.

This guidance draws on more than 20 years of accessibility expertise at SAP, co-created by our central Accessibility Competence Center and product design experts across the organization. Whether you're just starting your accessibility journey or are already experienced, we believe this resource will support your work and inspire further impact.

At SAP, we believe inclusive design is essential to building a more equitable digital future. Thank you for contributing to this important work. I look forward to seeing how your efforts continue to advance accessibility starting with design.

Nicole Windmann

Head of the SAP Accessibility Competence Center
Vice President Software Accessibility and Inclusive Design at SAP SE

Our Mission

Enabling accurate and effective implementation of accessibility from the beginning.

We, as designers, are committed to providing inclusive and seamless experiences to all users. This includes ensuring easy access to information, clear and efficient workflows, and communication that is both engaging and meaningful.

The goal is to support this Shift-Left approach by addressing accessibility earlier in the software development lifecycle. By embedding accessibility considerations in the design phase, we reduce the challenges of retrofitting during the development phase and ensure that we intentionally design accessible experiences.

This handbook offers guidance and support to annotate accessibility during the design phase. The library of visual elements serves as a resource for informing about annotations for visual, interaction, screen reading, cognitive, and auditory experiences. These annotation elements can be used on pages, screens, and components.

The annotation elements are followed by descriptions and examples so that a designer understands how it impacts the experiences of the users. The goal is to design for a diverse range of users, delivering them an accessible and inclusive experience.

As you explore this handbook, we encourage you to reflect on questions that drive this handbook such as:

- Do we comprehend how users interact with our applications?
- Do we know what accessibility means during the design phase?
- Do we think about what distinguishes user types?
- Do we consider our users' differences in ability or background so that our designs enhance their understanding and ability to work efficiently?

This handbook is not a rigid set of rules. Instead, our goal is to offer practical proposals that enable designers to enhance design practices and create inclusive designs that meet the needs of all users.

Irla Bocianoski Rebelo
User Experience Designer Expert and Accessibility Lead
at SAP SuccessFactors

Nina Krauss
Accessibility Specialist at SAP Product & Engineering

Stefan Schnabel
User Experience Design Expert at SAP Product & Engineering

Our Target Groups

Developers

Implement all annotated designs and requirements.

Product Managers

Plan and prioritize accessibility work and inclusive user research in the roadmap.

Quality Assurance Specialists

Verify visuals and behaviors against specifications and report any deviations.

Researchers

Discover user accessibility needs and evaluate design solutions.

System Designers

Build accessible, reusable components.

UX Designers

Annotate designs to indicate accessibility requirements.

UX Writers

Write and review screen reader announcements.

Visual Designers

Define visual specifications that fulfill accessibility requirements.

What's New in This Release

The first edition of the SAP Accessibility Design Tools was released in December 2023. It was created to help designers focus on, and work with, accessibility annotations in wireframes and components. It also served to ease the communication between designers and developers.

Since then, the annotations have been applied by designers from 160 SAP teams to create numerous accessibility design specifications, with several annotation inserts reaching up to 10,000 in a single week.

Annotations, such as keyboard keys, reading order, focus order, component name, and role are the most popular, and together, they have been used over 90,000 times in the past year.

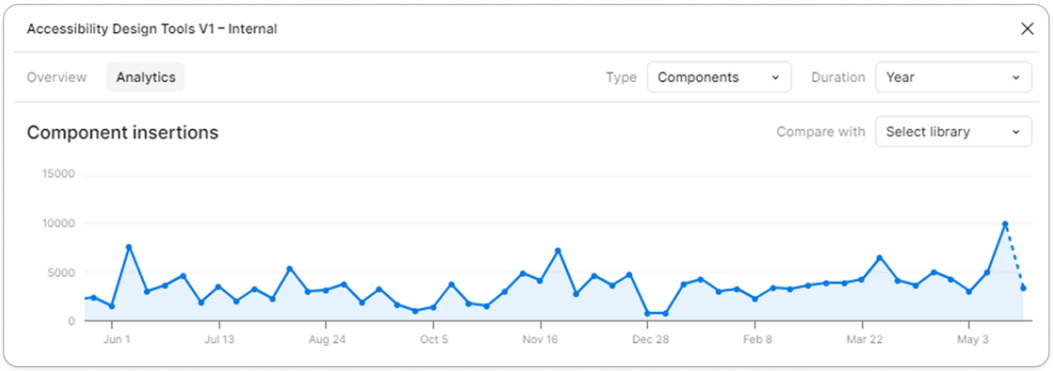


Figure 1: Accessibility Design Tools Usage June 2023 - June 2024

This revised second edition of the SAP Accessibility design tools includes enhancements to existing annotations, introduces numerous new annotations, and now incorporates information for developers. It also features quotes from designers who use the annotations in their work.

A refined and extended documentation structure covers a broader scope of accessibility requirements and user groups.

Grouping annotations per persona represents users with similar disabilities, creates empathy and makes the design annotation work more user- centric and humane.

Annotation examples were enhanced and extended to cover more use cases.

“The Accessibility Design Tools are unrivaled in their ability to facilitate detailed screen reader and keyboard specifications, aiding in both technical exploration and implementation, and promoting effective collaboration with developers to ensure the creation of truly accessible designs.”

Marvin Gille – UX Designer & Accessibility Advocate

Introduction

Shifting the Paradigm

Design beyond the visible

Accessibility is design quality. It means usability under diverse conditions and adaptability for different contexts, bodies, and minds.

Strong usability and interaction principles are not just good design, they are the foundation of accessibility on any platform, desktop or mobile.

A designer truly committed to accessibility thinks beyond the visuals. They annotate designs to ensure that every user, including those with low vision, cognitive challenges, or limited mobility, is considered from the start.

However, designing for the invisible experience, what screen readers and assistive technology interpret, can be tricky.

The same is true for keyboard design, for users who might prefer keyboard over mouse. Here, the entire experience depends on a logical, intuitive focus order, as well as visual cues that guide the user. That is where we come in.

We will show you how to turn visible UI elements into a clear invisible structure that screen readers can interpret. This will support inclusive and intuitive navigation and interaction, whether a user is using a mouse, keyboard, or gestures.

By understanding how users rely on visible cues to navigate and interact, you will demonstrate competency in creating inclusive experiences, from your first design draft to the final delivery.

This thought-provoking introduction invites you to shift your mindset toward more inclusive and accessible design. Embrace accessibility as a core design principle by accepting the challenge to rethink your design approach.

How To Start

Respect the Global Accessibility Market

Although the global accessibility testing market is huge, it is common that teams miss the opportunity to integrate accessibility into development and design due to a lack of awareness, expertise, and prioritization.

Meanwhile, the global accessibility testing market is already quite significant, valued at USD 564.73 million in 2023, and projected to grow over 50% to USD 825.43 million by 2032.[15]

This growth reflects a market which focuses primarily on audits, automated testing, and manual evaluations. Unfortunately, this reactive approach to accessibility is expensive and highlights the ongoing challenges teams may face in proactively embedding accessibility throughout the development process.

Be Aware of Remediation Costs

Design sits at the earliest stage of the software development lifecycle, making it the most effective point to embed accessibility and prevent future defects.

Addressing accessibility during design is not only good practice; it is also strategic. Costly remediation, a poor user experience, and legal risk are the common consequences of waiting until the later stages of development.

Employers account for 92% of all case for lawsuits filed under the Americans with Disabilities Act (ADA) since 1992. With high success rates and the associated costs, it is an essential need that digital products are accessible and inclusive.[1] Industry estimates show that addressing defects early, during the design phase, costs just 1x to fix, while waiting until after release can make corrections up to 100x more expensive.[10]

Early planning for accessibility not only helps avoid costly rework but also ensures compliance with laws such as the Americans with Disabilities Act (ADA), meeting both legal requirements and growing user expectations.

Establish a Shift-Left Strategy

Using a Shift-Left approach ensures that your team creates inclusive experiences from the start. This reduces risk, improves quality, and empowers all users.

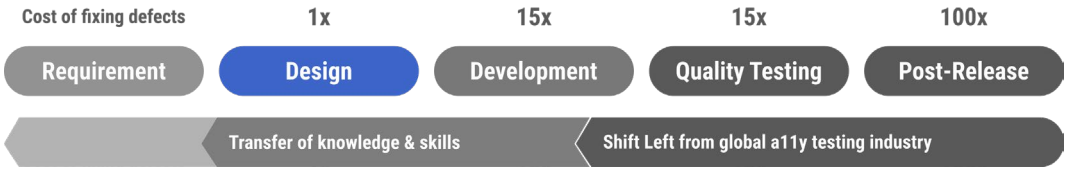


Figure 2: Shifting focus from defect detection to prevention helps teams avoid issues proactively.

A Shift-Left strategy brings accessibility and inclusion to the forefront of design and research, embedding them early in the process rather than waiting until development or post-release phases. By planning inclusive experiences from the beginning, teams can prevent accessibility issues before they arise, avoid costly retroactive fixes, and ultimately create better experiences for all users.

When usability is thoughtfully integrated from the outset, it helps avoiding lengthy discussions later about how to address accessibility gaps. And if questions arise along the way, revisit your usability principles and ask yourself: *What more can I do to enhance accessibility for all users?*

This approach empowers designers and researchers to define inclusive interactions upfront, based on user needs, behavior, and expectations. With clear design and accessibility specifications, developers can implement with greater confidence, efficiency and accuracy, resulting in a more robust and accessible product from the start. The Shift-Left strategy also reduces the burden on developers by preventing accessibility issues earlier in the process, rather than resolving them late in the cycle.

Despite this proactive approach, testing and auditing remain still very important and hold a strong position in the market. Accessibility specialists have a deep and structured understanding of accessibility requirements, international standards, and assistive technologies. Shifting from just detecting defects to proactively avoiding them is a true advancement for accessibility. Adopting a Shift-Left approach for designers is an opportunity to integrate accessibility from the start of the design phase.

To achieve Shift-Left, we need to follow the steps of people who audit systems for accessibility. Testers have undergone rigorous training and have extensive hands-on practice. They translate abstract guidelines, such as the

Web Content Accessibility Guidelines (WCAG), to identify defects and suggest improvements. They have the clarity, and the precision needed to identify, document, and guide teams toward meaningful accessibility enhancements, which is expertise that many designers and developers lack.

To successfully adopt the Shift-Left approach, let's address the basic concepts of accessibility, understand the accessibility life cycle, and prepare to work on accessibility documentation.

Inclusive Design Program Strategies

An inclusive design program does not happen by chance; it grows from deliberate learning, planning, and practice.

This chapter outlines strategies to strengthen accessibility at every stage: first, by improving your skills through historical context, understanding diverse user groups, and applying both testing and training.

Next, by developing strategy with practical blueprints and shared libraries that make accessibility repeatable.

Finally, by establishing a design process that embeds annotations, checklists, and structured reviews into everyday work.

Together, these approaches help teams move from intention to execution, building digital products that are accessible, consistent, and sustainable.

1. Improve Your Skills

Know the Historical Context

Understanding the history of accessibility helps explain today's laws and policies. The timeline below traces key milestones from early US Civil Rights laws to international standards that continue to shape accessibility regulations worldwide.

Table 1: Key milestones in the history of accessibility

Year	Milestone	Significance
1973	Rehabilitation Act (US)	First major federal disability rights law; established foundation for disability civil rights movement globally
1986	First Screen Reader	Jim Thatcher created the first screen reader at IBM which was later released to the public.
1990	Americans with Disabilities Act (US)	Landmark civil rights law serving as model for international disability legislation; covers employment, public accommodations, and government services
1998	Section 508 Amendment to Rehabilitation Act (US)	First law requiring government digital accessibility; influenced global standards for accessible technology
1999	Web Content Accessibility Guidelines (WCAG) 1.0	The Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C) developed a standard to ensure the accessibility of web content. The WCAG 1.0 consisted of 14 guidelines with 65 checkpoints.
2008	Web Content Accessibility Guidelines (WCAG) 2.0	The Web Accessibility Initiative (WAI) introduced with WCAG 2.0 four essential principles: perceivable, operable, understandable, and robust. With the principles came 12 guidelines and 61 success criteria. The success criteria are from now on divided in the three conformance levels A, AA, and AAA.
2009	VoiceOver and TalkBack Screen Reader for Smartphones	Starting with the release of the iPhone 3GS, VoiceOver by Apple was the first screen reader for smartphones, enabling users with visual impairments to navigate on touch devices. The same year TalkBack was released by Google for Android devices.
2016	EU Web Accessibility Directive	With Directive (EU) 2016/2102 the public sector organizations of the European Union (EU) must meet a minimum of accessibility requirements. The member states of the EU had to transpose the directive into national legislation to ensure the accessibility of public sector organizations.
2018	Web Content Accessibility Guidelines (WCAG) 2.1	With version 2.1, the WCAG provided 17 additional success criteria to also address mobile accessibility, people with low vision, and people with cognitive and learning disabilities.
2019	European Accessibility Act (EAA)	With Directive (EU) 2019/882 accessibility requirements also affect the private sector. A selection of important products and services is required to meet a minimum of accessibility. The EAA also covers digital products including for example e-commerce.
2023	Web Content Accessibility Guidelines (WCAG) 2.2	With version 2.2, the WCAG provided nine additional success criteria.
2025	European Accessibility Act (EAA)	Since June 25th, 2025, all requirements of the European Accessibility Act must be implemented for the affected products and services.

Understand User Groups and Disabilities

When designing for accessibility, it is essential to recognize the diverse range of users who benefit. This includes people with disabilities, who are

grouped by shared characteristics such as vision, mobility, hearing, or cognitive differences.

Accessibility benefits not only those with permanent disabilities, but also individuals experiencing temporary challenges such as a broken arm, recovering from surgery, bright sunlight, or a noisy environment.

By considering these overlapping needs, we create experiences that are more usable for everyone.

The SAP Accessibility Design Tools provide a simplified approach to covering the 50+ WCAG requirements by introducing personas that represent groups of people with similar disabilities.

This strategy aims to help designers and other stakeholders build empathy with end users who have specific accessibility needs. By aligning WCAG requirements with key personas, we enable a more efficient design documentation process, supported by checklists and practical guidance associated with these personas.

These personas appear in each of the five main disability groups as they often overlap with permanent conditions.

In this strategy, each persona acts as a bridge to understanding real user needs, helping ensure that accessibility is integrated with intention and consistency.

Table 2: Different disability groups – experience relationships

		
Low or Limited Vision	**Limited Mobility**	**Blind or Very Limited Vision**
The **visual experience** explores crucial factors to keep in mind when designing for people with limited vision, such as being color blind, having short vision, tunnel vision, etc.	The **interactive experience** addresses what to look for when designing for users with limited mobility. They may find it hard to control different input mechanisms like a mouse or touch devices.	The **screen reading experience** delves into what should be considered for screen reader users that may be blind or with very limited vision.
		
Cognitive Disabilities	**Auditory Disabilities**	**Situational and Temporary Disabilities**
The **cognitive experience** highlights important aspects to focus on when designing for people with cognitive issues such as concentrating, remembering or learning.	The **hearing experience** addresses aspects to look for regarding users that are deaf or hard of hearing, such as supporting captions, transcripts or sound alternatives.	These users benefit from accessible features when they find themselves in **situational or temporary** contexts.

Situational & Temporary Disabilities

We must expand what we understand about usability to address invisible experiences. This involves providing conditions for all types of users to enjoy the experience yet perform tasks safely, effectively, and efficiently.

Temporary and situational disabilities differ from permanent disabilities in terms of their duration and impact on the ability of the individual to interact with the environment, including digital products. Including "situational" disabilities in each of the five main disability groups helps designers recognize that situational or temporary limitations often overlap with permanent conditions, requiring special attention during the design phase.

Each chapter includes a note explaining how the disability group can be expanded to cover situational disabilities.

- Permanent Disabilities: These are long-term, or lifelong conditions often present from birth, accident, or illness. They require ongoing

accommodation and assistive technologies (e.g., screen readers, hearing aids, prosthetics).

- Temporary Disabilities: Short-term impairments from injury, illness, or other events. Ability typically returns after recovery.
- Situational Disabilities: Context-specific limitations caused by the environment or circumstances (e.g., poor lighting, noise, stress, fatigue) and are not tied to a permanent condition.

By understanding the differences between permanent, temporary, and situational disabilities, you can design more flexible and inclusive experiences that accommodate a wide range of users, even if their needs change over time or depend on external conditions. The table below shows key contrasts between permanent, temporary, and situational disabilities.

Table 3: Permanent, temporary and situational disabilities

	Permanent disabilities	**Temporary disabilities**	**Situational disabilities**
Duration	Lifelong or long-term	Short-term (recoverable)	Variable, based on context (can be momentary or hours-long)
Impact on Daily Life	Consistent and ongoing adjustments are needed	Short-term adjustments for a limited period	Context-specific, often requiring temporary adjustments for specific tasks
Recoverability	Not recoverable or only partially recoverable	Full recovery possible with time or medical treatment	No long-term recovery needed; situation resolves when context changes
Examples	Blindness, permanent mobility impairments, chronic hearing loss	Broken limb, temporary vision loss, concussion	Sweating hands, fatigue, stress, temporary loud background noise
Design Focus	Assistive technologies and long-term accommodations	Short-term adaptability and accommodations for a fixed period	Context-aware design and flexibility based on situational needs
Remarks	Often require persistent accommodations and assistive technologies (e.g., screen readers, sign language interpretation, alternative input methods).	Require short-term solutions or assistive devices (e.g., voice-to-text for a person with a broken hand, large-button keyboards for people recovering from surgery).	Can be mitigated by adaptive and responsive design, allowing users to adjust settings or interfaces depending on the environment or mental state (e.g., offering high contrast mode in a bright environment or one-handed mode when holding something).

Think Beyond Assistive Tools Support

Popular misunderstandings include:

- Supporting Accessibility is just supporting assistive technology
- All users with disabilities will use tools to access content
- “We do it on top for assistive tech users”

In reality,

- Accessibility is not just about assistive technology compatibility. It is fundamentally **built in** and not **added on after.**
- Not everyone with a disability uses tools. All users benefit from good design.
- "Doing it only for assistive tech users" misses the broader inclusion impact.

When accessibility is discussed in projects, some stakeholders quickly equate it with screen reader compatibility or keyboard navigation, which are tools commonly associated with assistive technologies.

While these are critical, it is a misconception to think accessibility is only about supporting assistive tech users.

Let’s break that down by looking at common assistive technologies and beyond, because not all disabilities require assistive tools, and not all accessibility needs are solved by supporting devices alone.

Table 4: Disabilities, needs and assistive tools

	Blind Users	**Low and Limited Vision**	**Limited Mobility**	**Cognitive Impairments**	**Hearing Impairments**
Assistive Tools	Screen readers (e.g., NVDA, JAWS, VoiceOver) Braille displays Voice assistants	Screen magnifiers High contrast settings Zoom and browser tools	Switch devices Alternative keyboards Voice control (e.g., Dragon NaturallySpeaking) On-screen keyboards	Some will use: Text-to-speech Distraction blockers Simple interface overlays	Many will use: Hearing aids Cochlear implants Live captioning or transcripts
Remarks	It is not enough to say: "We support screen readers." If the content structure, headings, labels, and interaction flows are confusing or inconsistent, even the best screen reader cannot create a good experience. And sighted users relying on keyboard navigation (e.g., due to temporary impairments or preferences) are impacted too.	Design choices such as tiny text, low contrast, or hidden focus indicators hurt far more than assistive tech users. Many older users or people using mobile in bright sunlight benefit from accessible visual design, without needing specialized tools.	Someone with a repetitive strain injury might just use a standard keyboard but avoid using a mouse. If buttons cannot be reached via tab, or if timeouts interrupt the task, it is not about technology. It is about usability.	Many users with ADHD, dyslexia, or memory challenges do not use tools at all. They rely on clear layouts, plain language, consistency, and reduced cognitive load, none of which depend on any assistive tech.	Captions help everyone in various situations: in noisy environments, while multitasking, or learning a second language. If alerts rely only on sound without visual cues, you are excluding a lot more people than you think.

Differences between Web, Tablets and Mobile Devices

Users interact with digital products across many devices, including desktops, tablets, and mobile phones. They use a range of input methods, such as touch, keyboard, mouse, or voice, and receiving information through various outputs such as visuals, audio, or haptics. They rely on different senses to understand and interact with what you design.

Input and output devices matter for accessibility. Before adding annotations, designers should understand how users interact with different devices to make thoughtful, inclusive decisions.

It is essential to consider the variety of input mechanisms people use to interact with digital products. While many rely on standard inputs, such as keyboard, mouse, or touchscreen, users with disabilities may depend on alternatives: screen readers with keyboard shortcuts, switch devices, voice commands, eye-tracking systems, or adaptive controllers.

These input methods can differ significantly in speed, precision, and interaction patterns, which means that design choices, such as target sizes, gesture complexity, focus order, and input flexibility directly affect usability. Ensuring that interfaces support multiple input types not only increases accessibility but also improves the overall adaptability of a product across devices.

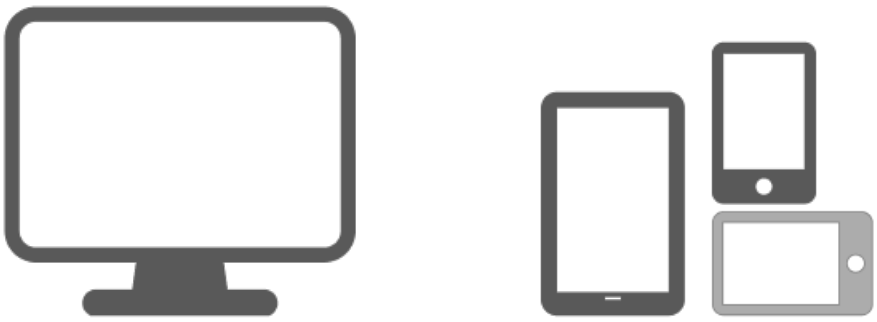


Figure 3: Different challenges on different devices: Web desktop devices are mostly used with a mouse and a keyboard, some of them also with touch and gestures. Common interactions of Tablets and Mobile Devices are based on touch and gestures, but some users also use keyboards to work with them.

Most accessibility annotations apply across web and mobile platforms. While the core principles remain the same, making content perceivable, operable, understandable, and robust, each platform has its unique challenges.

Native mobile devices, for example, introduce specifics such as touch interactions using fingers of all sizes:

- Small screens that need to fit complex content
- Support for both portrait and landscape orientations
- Built-in assistive technologies, such as VoiceOver (iOS) and TalkBack (Android)

Some annotations differ for mobile, especially those related to screen reader behavior. For instance, mobile apps do not use web standards like ARIA landmarks, and heading structures are simpler (usually a single level).

Mobile operating systems rely on gestures and touch for navigation, while screen readers provide audio feedback. Though annotations such as Keyboard Support are more common for web, they also apply to mobile in cases where users connect external keyboards.

Other annotations, such as those for time limits, labels, and roles, apply universally across digital products, regardless of the device.

Perform Automatic and Manual Testing

Accessibility is not complete until the implemented solution is tested. After development, when the testing phase starts, accessibility testers validate that digital products are usable by people with disabilities.

According to our experience, automated testing can catch about 20–30% of accessibility issues. Tools, such as Axe, Lighthouse, or WAVE can detect issues such as missing alt text, low color contrast, missing form labels, ARIA misuse, keyboard traps, and missing document landmarks.

But automated tools cannot evaluate whether alt text is meaningful, if the focus order is logical, or if text in links or buttons is clear and descriptive. Manual testing is essential for the remaining 70–80%.

Manual testing reports usability issues for screen readers, verifies ARIA and semantic markup, confirms captions, labels, and alt text, checks dynamic content updates, and assesses user experience across disabilities.

It also includes keyboard-only navigation, screen reader checks, visual inspection for layout and zoom. Also, people with disabilities should be involved in the test cycle, when possible, to get end user feedback that may sometimes lead to necessary design iterations.

Combining automated and manual testing for full coverage is key to checking all accessibility requirements. Automation gives speed; humans give insight.

Some tools used to conduct manual testing are available to many as a built-in assistive tool such as screen readers, which can be found on desktops and mobile phones.

Design Context

A designer is not expected to conduct all these testing tasks. Instead, accessibility annotations drive implementation and prevent many common issues. If a designer wants to verify how a design experience performs after implementation, some testing tools can support that review.

Some resources for designers to test an implemented application are available online as browser extensions. Instead, accessibility annotations drive implementation and prevent many common issues. If a designer wants to verify how a design experience performs after implementation, some testing tools can support here.

Many checks do not require any tools and can be conducted manually. The annotation design checklists can be used as a reference to plan a design test or heuristic evaluation.

All operating systems (Windows, iOS, Android, etc.) offer a set of configurations to adjust visual elements, including color contrast, theming, and text spacing. Explore these settings and conduct visual testing of your application. Observe how the UI elements behave when adjusting to the selected settings.

Built-in assistive technologies for testing on desktop computers and laptops include the screen reader, Narrator, on Windows and VoiceOver on Mac. For mobile devices, there is the screen reader TalkBack on Android and VoiceOver for iOS.

Chrome supports various automatic accessibility checks, including Lighthouse which is built into its DevTools. Open it on Apple with OPTION+CMD+I and on Windows using F12.

Additional features are available through tools that can be installed separately on Windows, such as the Jaws and NVDA screen readers.

Finally, there are browser extensions for accessibility testing, including tools for Color Contrast or Landmark and Heading structure analysis.

Although developers are the primary users of these tools, designers can and should take advantage of them as well, since an issue flagged by automation may reveal not just a technical bug but also a potential weakness in interaction or navigation that benefits from a designer's manual review.

Figma Support

The collection of annotations for accessibility offers a confident strategy for designers to create accessible designs.

To streamline accessibility work within the design process, SAP developed its own set of SAP Figma plugins dedicated to accessibility annotations. These plugins allow designers to go beyond visual polish and document critical details such as focus order, reading order, heading hierarchy, and landmarks directly within their design files.

By using plugins, annotations can be created faster and more efficiently, reducing repetitive manual effort. Designers can also manage annotations more effectively by switching annotation variants, fixing the sequence, removing unnecessary elements, or reordering items as the design evolves. This ensures accessibility documentation remains accurate, consistent, and easy to maintain.

Explore the SAP A11Y Plugins for Focus Order, Reading Order, Headings and Landmarks at https://www.figma.com/community/plugin/1072563579293318294/accessibility-design-tools-second-edition.

Additionally, the Figma community offers many plugins, including free and paid plugins. Here are some of them:

- Contrast: Checks color contrast between two elements
- Able: Checks color contrast
- Color Blind: Simulates different types of colorblindness
- Text Resizer Accessibility Checker: Simulates readability by increasing or decreasing font sizes
- Accessibility Assistant: Offers support to conduct usability walkthrough and to create annotations for screen reader and keyboard
- Stark: A paid plugin that offers color contrast check, typography, alt-text, focus order, Landmarks and more.
- Evinced: A paid solution that provides full support including annotations, accessibility design report, and design analysis for single elements and full design.

Figma Accessible Prototype

Figma offers accessible prototypes that support screen readers such as VoiceOver, JAWS, and NVDA, but only in the desktop app or web browser, not on mobile or mobile web. The screen reader mode converts your design into HTML, enabling semantic interpretation of elements. Image fills become images labeled with layer names, components turn into landmarks, and interactive “On click” actions become buttons or links.

Importantly, focus outline indicators in Figma remain limited, so keyboard users often do not see visual feedback during tab navigation. It is important

to note that focus outline indicators in Figma remain limited, so keyboard users often do not see visual feedback during tab navigation

Watch Training Videos

All the information you need to learn about accessibility is available online such as WCAG by W3C, which has a comprehensive list of requirements explaining how to prepare a web application to be accessible. This list is also used to validate and measure accessibility in web applications.

You might find the content of W3C dense and not well-tailored for designers, since they avoid any design direction. The instructions focus on the experience of the user and support developers in coding for accessibility. But as designers, we want clear messages and instructions to help us work efficiently and effectively during the design phase.

SAP created a series of training videos to explain how to annotate designs and how interfaces properly designed for accessibility impact the experience of the end user.

Enjoy!

- Design Annotation for Accessibility
- Keyboard Support in UI Components
- Screen Reader Support in UI Components
- Prepare For All Users
- Prepare for Keyboard Usage
- Screen Reader Usage Part 1
- Screen Reader Usage Part 2
- Annotating Mobile Apps

“As a designer focused on creating a transparent user experience, I look for strong examples of how to do so with accessibility in mind - and the Accessibility Design Tools offer exactly that.”

Dominika Zamojska - Senior User Experience Design Specialist

2. Develop a Strategy

List components used on the application, floorplans, and pages as this will set the basis for consistent re-use of structural elements, such as components and floorplans, allowing you to annotate them only once.

An inventory of components reduces the amount of work needed to annotate designs and results in an efficient development of the inclusive experience. The precision of components selected to create the experiences correspond to the same accessible control used by development to code the visuals and interactions.

Create Blueprints as Foundation

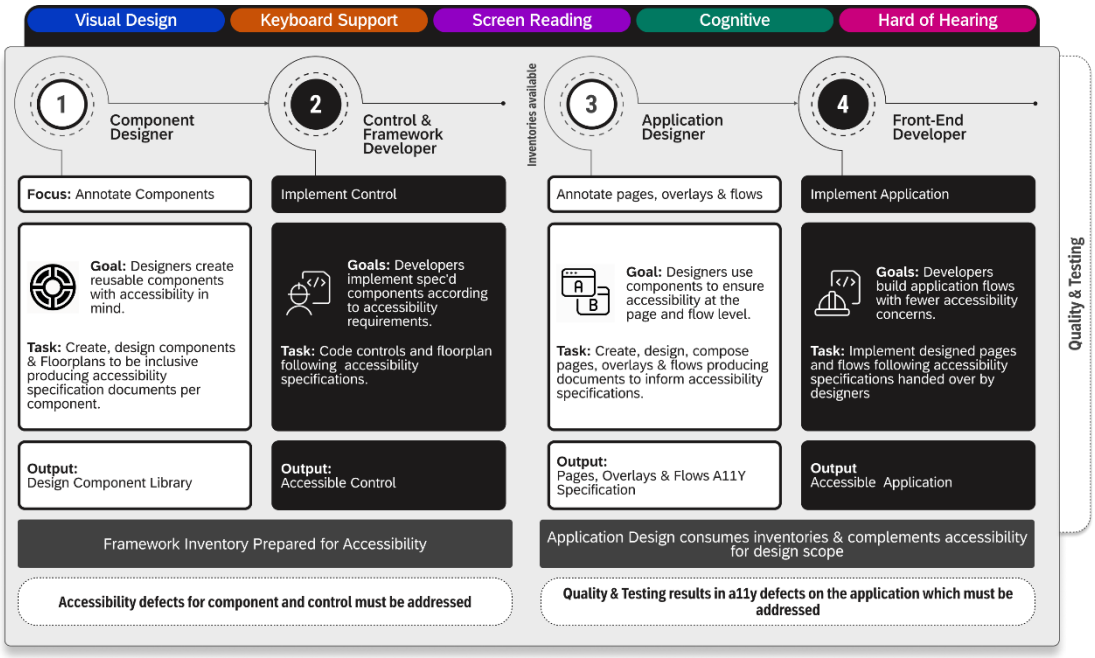


Figure 4: Infographic relationship between design and development

A blueprint is a detailed plan or guide that outlines the structure, components, and relationships of a system before it is built. In product development, specifically design and accessibility, a blueprint serves as a shared source of truth that informs how something should function and how it should be constructed across teams.

A blueprint is not just a static diagram. It is a strategic foundation for collaboration, consistency, and accountability.

Think of accessibility as the electrical wiring plan within a building blueprint. While not immediately visible, it is essential for functionality, safety, and the experience of anyone who enters the space. If it is not

properly designed from the start, making changes later can be complicated, costly, and risky (IBM Sciences Institute, 2023).

Just like an architect collaborates with engineers and electricians early on, designers collaborate with developers and technical writers. Accessibility must be integrated in the design phase. This ensures all components, layouts, and interactions support inclusive use from the ground up.

The accessibility blueprint of your product is the result of a thoughtful process that includes building detailed inventories and aligning closely with engineering. These artifacts are shared across teams ranging from QA and front-end developers to technical writers and product managers, which ensuring everyone is working from the same foundation.

Easy access to well-documented components and controls significantly improves efficiency. When Component Designers define reusable components with accessibility in mind, developers can implement them consistently and confidently. This empowers Application Designers to use those components knowing accessibility has been addressed at the component level and implemented accordingly, allowing them to focus on ensuring accessibility at the page and flow level.

Front-end developers can then build application flows with fewer accessibility concerns and less need for rework or troubleshooting.

Leverage Libraries and Inventories

Plan, Prepare and Share

Extensive libraries of design components and implementation techniques are powerful tools for designers, front-end developers, and engineers. They help create consistent user experiences and significantly improve efficiency in meeting accessibility requirements.

A design component library should include accessibility annotations directly in the documentation, making it easier for designers to incorporate inclusive practices from the start. In parallel, an accessible control library used by developers should offer configurable properties to adapt the controls to specific contexts without compromising accessibility.

The biggest advantage of these libraries is efficiency, the ability to produce high-quality work with less effort. Efficiency in design and development thrives on reusability, and accessible libraries make it possible to work faster while maintaining consistency and quality.

To maximize performance:

- Ensure easy access to component libraries with documented accessibility specifications.
- Make control libraries readily available and confirm they are implemented with accessibility in mind.
- Use consistent visuals, behavior, and terminology. User Assistance Developers can support this by reinforcing shared language across teams.

When component libraries used by designers and control libraries used by developers are aligned have matched naming conventions and consistent interaction patterns, everyone involved can work more confidently and effectively.

To support this alignment and drive efficiency, maintain three key inventories:

- Floorplans: Reusable layout templates for pages
- Components: Reusable UI elements, including core and custom components
- Pages: A list of all implemented pages in the solution, forming the ultimate Information Architecture of your application

"As noted in the Press Release W3C Issues Improved Accessibility Guidance for Websites and Applications, "WCAG 2.1 will be supported by an extensive library of implementation techniques and educational materials."[23] This reinforces the value of well-maintained, accessible

libraries as an essential part of any inclusive design and development workflow.

At SAP, designers and developers are provided with libraries to support their jobs. Designers have access to reusable components in a Figma library and Design guidelines documentation with accessibility reminders. Developers have access to seamless control libraries to implement the experience planned by the designer.

Table 5: SAP libraries and guidelines

Design Library	Control Library	Design Guidelines
	Fundamental	Fiori Design Guidelines
Specification	**Implementation**	**Direction**
To speed up the work while assembling pages, designers use components ready to use and directly available from a central library. Sometimes, more than one library is available (for example, core and customer components). The Material Design by Google is an example of a core library. The component contains visual design specifications. In Figma, reusable components can also contain predefined behaviors, making the creation of functional prototypes more efficient. Designers are owners of these libraries.	This library contains controls (implemented components) carrying all visual aspects needed to fulfill accessibility requirements, such as color contrast. It may include behaviors for interaction and navigation, screen reader, and keyboard support. Engineering is the owner of this library, and designers consult controls to check look and behavior.	Design guidelines are the base of a design system. They provide designers and developers with valuable references to support their design decisions as they build new controls. An example is the SAP Fiori Design Guidelines.

“Using our accessibility design tools enhances clarity and accuracy by using a consistent SAP-wide shared language when collaborating with teams.”

Sup Suh – Senior Product Designer

3. Establish an Accessibility Design Process

Use Annotations

Annotating accessibility in the design phase requires diligence and preparation. Yet, remembering and applying every accessibility aspect relevant to the design phase is a heavy cognitive task.

Start by planning and aligning with product management. Then, plan the accessibility blueprint of your product as illustrated in Figure 4 that describes the relationship between design and development.

It explains why it is essential to understand the difference between the component design library used to produce designs and the control libraries used by front-end developers to code the application.

To help you accelerate accessible designs, we propose an annotation set for components that address accessibility. A designer supported by a library with accessible components contributes to a consistent inclusive product. Use checklists to track accessibility guidelines and ensure all aspects are considered during the design phase.

This guideline aligns with and covers many WCAG standards and international requirements to annotate designs. It covers many aspects needed to ensure that you deliver an accessible product, including the four groups of requirements from the Web Content Accessibility Guidelines (WCAG 2.2): Perceivable, Operable, Understandable, and Robust.

SAP Design Guidelines offers a variety of elements to annotate screens and components to indicate accessibility considerations during the design phase.

To annotate designs and create accessibility specifications, follow the checklists and use the annotation assets from the library for precise design documentation. Use the annotations pointing to an element or highlighting a specific area. This can be done manually or using available plugins for screen reading, focus order, landmarks, and headings.

Work with Checklists

In this guideline, there are six main groups of annotations to support an accessible design, and they are available as checklists.

A checklist is a good tool to guide designers and track the progress of fulfilling accessibility through annotations. Over time, creating annotations will become a natural part of the design process, strengthening your ability to connect accessibility concepts with core usability principles.

The Accessibility Checklists by SAP were created to guide designers and help them observe and consider every accessible requirement during the design phase and when creating documentation.

Plan Design Reviews

One last note! Plan for accessibility design reviews. An A11Y expert provides a critical eye on the design specifications, ensuring all accessibility standards are met and usability is optimized for everyone. Incorporating accessibility into design reviews not only fosters inclusivity but also enhances usability, ensuring a more seamless experience for all users.

Annotating accessibility in the design phase requires diligence and preparation. Yet, remembering and applying every accessibility aspect relevant to the design phase is a complex task.

During the design phase, accessibility requires structured alignment and checkpoints to prevent blockers and barriers. A strong practice is to start with an accessibility blueprint (annotation specification) to capture design intent, followed by informal checks with accessibility experts to validate early decisions.

Teams should then conduct a formal review, where potential issues are documented, and outcomes are tracked. Designers are expected to address the concerns raised, updating annotations and design details as needed to avoid violations.

This step-by-step approach ensures accessibility is integrated into the design review process rather than left as a late correction

1. Alignment

Start aligning with product management, development, and testing teams to ensure a shared understanding of accessibility goals, constraints, and timelines. This early collaboration helps clarify expectations, surface potential blockers and barriers, and ensure accessibility is embedded into user stories, technical planning, and testing coverage right from the start.

This is also the moment to establish a common accessibility vocabulary and agree on how issues will be categorized. For example, issues can be defined as “potential barriers”, which may impact usability for some, and “potential blockers”, which can prevent access entirely. This clear categorization will streamline communication and prioritization throughout the process.

The alignment is also essential to understand the difference between a component design library used to produce designs and the control libraries used by front-end developers to code the application.

2. Accessibility Blueprint

Prepare the design specification using the accessibility checklists. This will be the product blueprint. Checklists help the designer to track every accessibility requirement needed during the design phase. This collection of accessibility annotations and the guidelines align with numerous WCAG standards and international accessibility requirements for design annotation.

To help you accelerate the creation of accessibility specifications consider a central documentation where components also address accessibility and are annotated to indicate possible configurations and usage to meet visual, interactions, and cognitive and screen reader requirements. A designer supported by a library with accessible components contributes to an assertive inclusive product.

The Accessibility blueprint is complete when the designer annotates all the screens and flows using all applicable annotations to cover visual, screen reading, interaction, cognitive and auditory experiences.

3. Informal Check with Experts

Consultations are informal yet valuable checkpoints with accessibility design experts throughout the design process. These quick, collaborative discussions provide timely feedback, help clarify uncertainties and ensure that accessibility considerations are thoughtfully integrated before formal reviews. By making consultations part of the workflow, teams can prevent issues early, build confidence in their decisions, and reinforce a shared understanding of inclusive design best practices.

But most importantly, consultations help guide designers in shaping a clear and complete accessibility design specification, or product blueprint, an essential handoff document for developers and testers.

These expert touchpoints ensure that annotations are meaningful, precise, and aligned with technical expectations, reducing ambiguity and interpretation errors during implementation. This shared clarity between design and development fosters consistency across teams, supports more efficient testing, and ultimately leads to a more robust and accessible end-to-end user experience.

4. Formal Review

Final check of an accessibility specification before implementation.

A formal design review is a second pair of eyes in the design specification to ensure accessibility has been fully addressed. The A11Y design expert can use a card to collect notes from the review to indicate design decisions that could become an acc defect reported by a quality team or in future by customers. These can be used as a plan for future fixes informing dependencies, upcoming requirements or deprecation.

This process is an assessment of the design annotations to identify potential **blockers** or **barriers** that may be reported by the testing team at the end of the software development lifecycle, or worse, by customers after release. This process aims to ensure that the design is usable and navigable by all individuals, adhering to accessibility standards and best practices using the A11Y checklists.

Checklists are also helpful to support the expert during the review as they track accessibility requirements and ensure all aspects were considered in the design specification. By following the checklists, designers can confidently address the critical elements needed to ensure an accessible product, covering all four core principles of WCAG 2.1: Perceivable, Operable, Understandable, and Robust.

Mark the design as ready to initiate the accessibility review. This step typically follows the completion of annotations and core user flow definitions. The operational model you choose, whether through design system labels, workflow tools, or direct communication, should clearly indicate that an A11Y Design Expert is needed to assess the design. Formalizing this step helps teams track how accessibility is being addressed throughout the design phase and builds accountability. It also ensures that potential accessibility issues are identified and resolved early, reducing delays and rework during development.

5. Review Outcome

Once the A11Y Design Expert completes the review, the design is marked as Reviewed, and the assigned Designer should fix any potential violation. This means that the design has undergone an accessibility review, all applicable annotations have been provided, and blockers and barriers have been removed or explained.

However, the reviewed status does not imply that the design is free of issues. Observations in the documentation should inform developers about potential accessibility violations identified during the review. These For Developers are essential for tracking, to support transparency, cross-team coordination, and prioritizing accessibility concerns.

These typically fall into two categories:

- **Potential Blockers** – Issues that would prevent people with disabilities from completing tasks or accessing key content. For example, if keyboard-only users cannot activate or reach a feature, the task becomes fundamentally inaccessible. These must be resolved before development to avoid creating inaccessible features.
- **Potential Barriers** – These refer to accessibility violations but could be confused with usability concerns that may make it difficult or inefficient for users with disabilities to understand or complete tasks. For instance, if screen reader users cannot locate key information due to missing landmarks or inconsistent headings, they may become disoriented or give up. Beyond usability or clarity issues, these may hinder, confuse, or slow down users with disabilities. These are important to address but may be prioritized differently or planned for future releases.

The goal of this reviewed status is to inform all stakeholders, which include product managers, developers, and testers, about where accessibility risks exist so that mitigation strategies can be planned. It also supports shared accountability by aligning expectations and enabling informed decisions about what to fix now and what to log for future releases.

“The clarity it provides helps identify many violations. Its impact is like a domino effect—one finding can prevent a chain of accessibility issues and save significant time and effort.”

Sanket Kundu - UX Design Associate

Foundation

Imagine a developer who does not just implement designs, but enables access. Each design specification is not just layout. It is a set of cues for people navigating by touch, voice, keyboard, or screen reader. Build it like someone depends on it, because someone does.

This chapter helps designers to connect design and development.

Inclusive Design Foundation

Meeting the needs of developers to code a fully accessible experience

If you want to create a culture of accessibility for designers, aim to foster the mindset of accessibility by providing a source of truth for designers to reference components used on the designs. One of the main benefits associated with this strategy is that Designers will trust accessibility is considered in the foundational controls used by developers.

Another benefit is the developer confidence during implementation as they will use known controls to create the accessible experiences. Lastly, users will explore and interact more confidently, knowing they can trust patterns and consistent behaviors across the product.

If you have started an annotation effort in your group, you may have discovered the importance of having one source of truth for designers and developers to follow. Strategic design starts from the base where annotated components are part of a reusable library of components.

The best practices presented in this section focus on basic information to build consistent interaction and navigation styles. The annotation proposed provide basic understanding of the layout and flow.

Principles

Designer

Persona: Dan

"I create the design specification using SAP Design Tools for Accessibility to give a solid basis for an accessibility-ready implementation"

The designer role prepares the specification to bridge the gap between creative vision and engineering. The selection of page templates and UI elements on the design are precisely informed to articulate the vision of designers and enable developers to implement the intended experience.

General Design Tips

1. Make clear component definitions

Ambiguity in component naming or mismatches between design libraries and code libraries can lead to rework or inaccessible implementations.

Design Tips: Use consistent annotation naming conventions for all components (e.g., Button, Dialog, Table Row). Reference design system tokens and document expected behaviors clearly.

2. Use reusable layout structures (floorplans)

Developers waste time reinterpreting layouts when reusable page structures (e.g., navigation, headers, footers) are not identified.

Design Tips: Annotate floorplan names in design specs. Add labelled ARIA landmark annotations (e.g., main, navigation, banner) to simplify implementation and improve accessibility (see chapter Screen Reading Experience)

3. Work with annotation checklists

The **Checklist** component offers two complementary formats that designers can choose from depending on their workflow and purpose.

Design Tips: Use one of the proposed checklists to create the accessibility specification, the unified checklist or the group of specialized checklists.

Annotation Checklists

You can choose to use individual checklists or a complete cheat sheet, during design specification or during the design review.

1. Unified Checklist (Single-Sheet Overview)

- A consolidated, one-page resource that merges the six focus areas, **Foundation, Visual, Screen Reading, Cognitive, Interaction, and Auditory**, into a **single overview sheet**.
- Best suited for **design reviews**, where a quick but comprehensive scan of accessibility considerations is needed.
- Includes **mobile-specific exceptions** directly within the sheet, making it easier to spot issues across platforms.

2. Specialized Checklists (Contextualized Sets)

- Six **dedicated checklists**, each focusing on one accessibility area: **Foundation, Visual, Screen Reading, Cognitive, Interaction, and Auditory**.
- Designed for **deep-dive evaluations** or when working on a **design specification document**, where detailed consideration of a particular aspect is required.
- Enables designers to focus on the nuances of each accessibility dimension, supporting more thorough documentation.

By consolidating accessibility concerns into one list, teams can quickly spot potential issues, maintain consistency across projects, and avoid overlooking key requirements.

This approach not only streamlines the review process but also helps designers build habits of inclusive thinking, ensuring that accessibility is embedded into the workflow rather than treated as an afterthought.

Ultimately, checklists make it easier to produce designs that are more usable, more compliant, and more welcoming to all users.

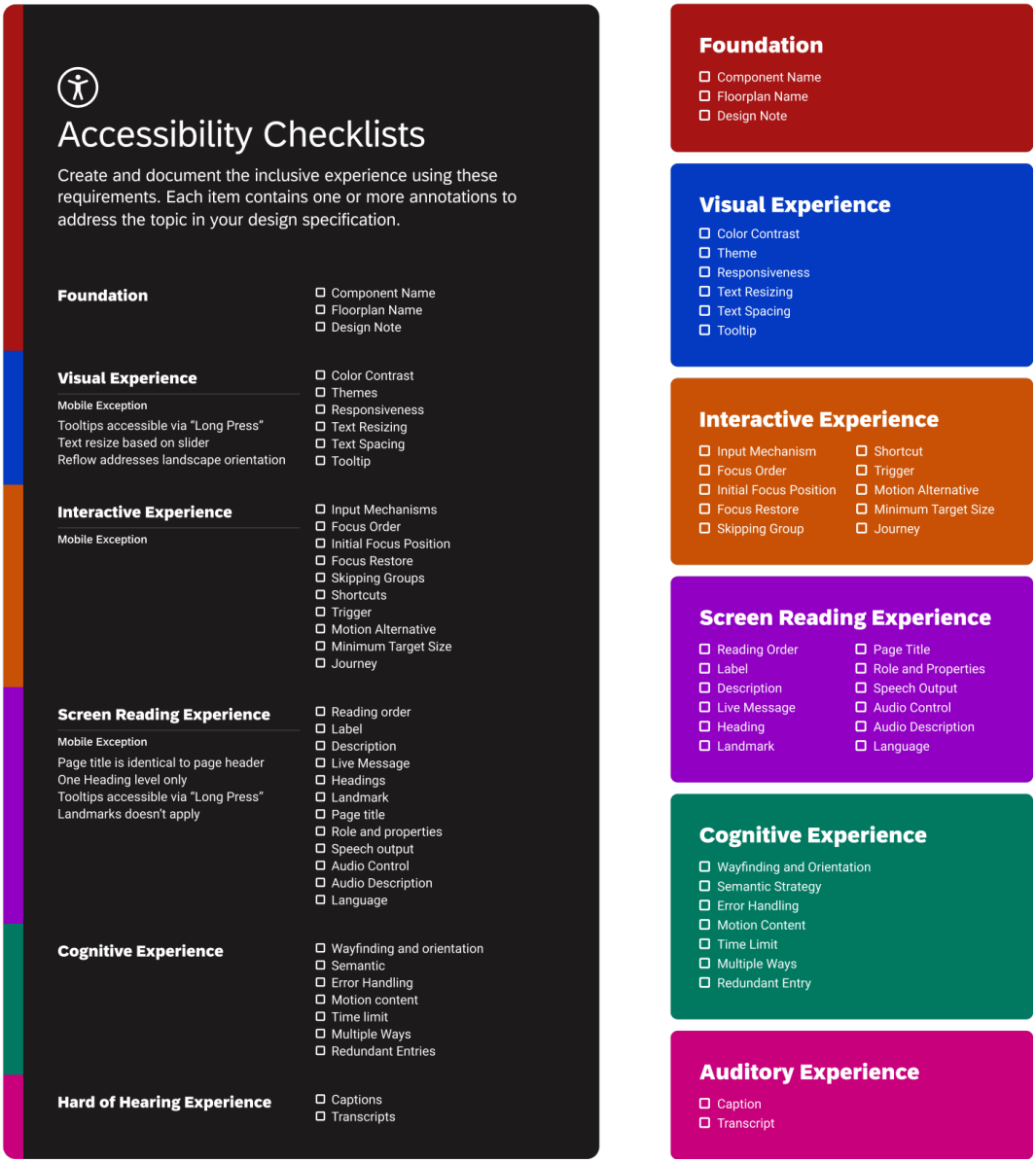


Figure 5: Checklists, which include foundation and one for each of the five groups of people with disabilities, are used to guide designers during the annotation phase or during the design review

Developer

Persona: Denise

"I consult the accessibility specifications in a design to implement the end user experience free of accessibility defects."

The developer role goes far beyond typing lines of code. They bridge the gap between creative vision and real-world use, ensuring every user, regardless of ability, can enjoy a seamless experience. You are the architect of accessibility, translating the vision of the designer into a living, breathing application that welcomes everyone.

Design & Development Collaboration

Design and development collaboration begins with a shared language that includes component names, floorplan names, and external content annotations. These are basic elements defining the design blueprint for consistency and clarity. These foundational elements are not mere labels; they are the guiding framework that ensures every part of the app fits perfectly within the larger structure, bridging creative ideas and technical implementation.

It is also the key task of designers and developers to deeply understand the shared accessibility language. The accessibility annotations function like the architecture of the application, pointing precisely to where every piece fits and should be configured, ensuring usable and inclusive flows, consistency and clarity across the entire digital landscape.

When designers specify components with exact names and link them to floorplans, the reusable page structures, they provide developers with a powerful map. This map guides the implementation, allowing the developer to connect the dots between design intent and code reality without guesswork.

External content annotations point to third-party elements that fall outside your direct control but still affect the user experience: embedded widgets, external libraries, or API-driven content. While the developer may not manage how these elements are built, they are still responsible for

understanding their behavior, ensuring the elements integrate smoothly, and flagging any accessibility risks they introduce.

Prepare the code for testing

But your work as a developer does not stop at translating designs into code. Preparing the app for testing is where your craftsmanship truly shines. By embedding semantic HTML, following ARIA standards, and ensuring proper landmarks and roles, you lay the groundwork for assistive technologies to interpret and interact with your app flawlessly. You are crafting invisible threads that empower screen readers, keyboard navigation, and other assistive tools to deliver a coherent story to every user.

Collaborating closely with designers to clarify annotations and accessibility For Developers means fewer surprises during testing and smoother certification processes overall.

Ultimately, your role is that of an accessibility steward, ensuring that the architecture of the app is both robust and flexible; ready to welcome diverse users without compromise.

The foundation you help build today, with meticulous attention to design alignment and accessibility compliance, becomes the launching pad for inclusive digital experiences that do not just meet standards, but genuinely open doors. In this dance of design and development, your expertise turns vision into vibrant reality, one accessible component at a time.

General Development Tips

When you encounter accessibility annotations in a design:

1. Identify the Annotations

- Find the annotations in the respective subchapters of this documentation

2. Understand the Use Cases

- Review all use cases (or user stories) and all additional linked information for your designs.
- Examine the documentation and examples

3. Implement According to Best Practices

- Clarify missing edge cases in design.
- Find extra implementation information in the “For Developers” annotation subchapters
- Consult developer documentation for respective technology about implementation methods and examples in the “For Developers” annotation subchapters

4. Iterate on your Work

- Showcase proof of concept samples to designers
- Align and harmonize expectations
- Contact subject matter experts for open questions

Annotations

Foundation Annotations

The checklist helps designers remember aspects that need to be addressed during the design phase. Foundation annotations help designers make thoughtful choices about components, page templates, layouts, flow, and navigation. They also clearly communicate these decisions to developers, ensuring the designed experience meets the needs of all users.

Foundation

- [ ] Component Name
- [ ] Floorplan Name
- [ ] Design Note

Component Name

Identify the connection between the design component and the control used for implementation that is aligned with developers to ensure an efficient development process.

Component Variants

Component Name

Figure 6: Indicates the name of a component used on the layout

Component Name (Scope)

Figure 7: Indicates the name of a component used on the layout with a visual outline of its limits

Component Name (External)

Figure 8: Informs that an external component or app is used in the layout

About Component Name

Components that have the same functionality within a set of web pages must be identified consistently.

WCAG 3.2.4 Consistent Identification requires that components with the same functionality are identified consistently across a product or set of pages. This means using the same visual symbols, labels, or roles for elements that perform the same action, such as a “Search” icon or a “Submit” button. Consistency supports recognition, reduces cognitive load, and helps all users, especially those with cognitive or visual disabilities, navigate interfaces more easily and predictably. It is a key principle in building intuitive, inclusive user experiences.

Naming components in the design according to shared component and control libraries is the first step to provide precision and remove doubts from design intention. Mapping design component and floorplans to their

technical UI library used for implementation is the best way to communicate the design construct.

Component Name and Component Name (Scope)

The Component Name annotation identifies the UI element in the design and maps it to a corresponding component in the technical UI library used for implementation. Indicate components by pointing to them or framing its area using the scope annotation.

This ensures developers understand which component to reference, even if the visual design combines multiple UI patterns. This also prevents ambiguity during handoff and improves alignment between design intent and implementation.

- Note: The component name may differ from the internal name used in the design tool. If this happens align with development and design systems to document deviations.

Component Name (External)

Use the Component Name (External) variant when the design includes elements outside the immediate control of the product team. These may come from a third-party source, external design system, or API-based widget.

When to use:

- Embedded 3rd-party components (e.g., a chatbot, calendar picker, video player).
- Shared components not owned by your team but consumed via the UI framework.

Accessibility Notes: **Any accessibility issue must be addressed at the source** to allow your product to rely on how the external content is structured and coded.

Table 6: Component annotation types

Annotation Type	Purpose	Accessibility Relevance
Component Name and Component Name (Scope)	Identify UI elements and their matching components in the implementation library	Helps developers apply correct semantics and interactive roles

Component Name (External)	Annotate 3rd-party or non-owned elements embedded in the design	

Examples

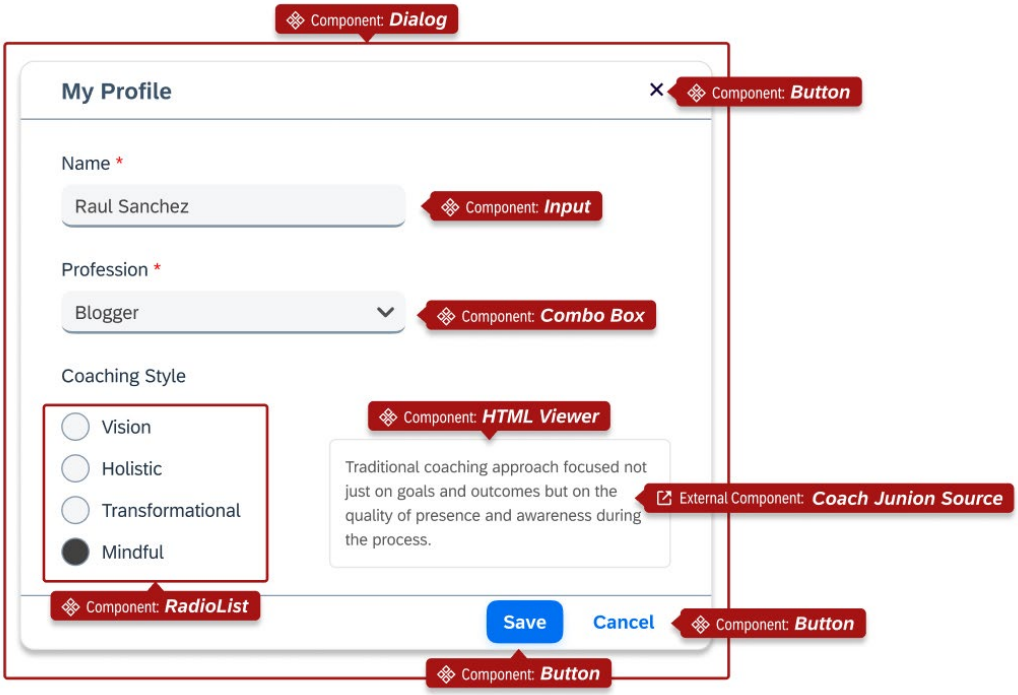


Figure 9: Component, Component (Source) and Component (External) annotations used in a dialog design

For Developers

Developers should check, during the review process, if the names used in Component Name reflect existing component names in the development framework inventories used for implementing the design.

They should also verify if the usage of components in the design correctly reflects all features of the respective framework components since component feature changes (such as interface and functionality modifications) could have been sometimes made without informing the designers.

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

3.2.4 Consistent Identification (AA)

5.4 Statement of Partial Conformance - Third Party Content

Floorplan Name

Indicate the name of a page floorplan template

Component Variants

Floorplan Name

Figure 10: Indicates the name of a page template used to create a new page layout

About Floorplan Name

The Floorplan Name annotation identifies a reusable page-level layout or structure, such as templates that include headers, footers, menus, breadcrumbs, or other navigational elements. It serves as a reference to standard page templates defined in the design system or framework and helps ensure consistency across screens.

Why It Matters

- Design Consistency: Using named floorplans encourages the reuse of familiar patterns, streamlining both design and development.
- Clarity in Handoff: Clear naming reduces ambiguity for developers and content creators, enabling accurate implementation of page structures.
- Efficiency: Annotating with Floorplan Names saves time by eliminating the need to redefine structure or behavior for every page.

Accessibility Relevance

Floorplan annotations are especially critical for accessibility. They help ensure:

- Consistent Landmarks: Reusable templates can define key landmark regions such as main, banner, navigation, and content info (footer), making navigation easier for screen reader users.
- Structured Headings: Templates often dictate where <h1> and other headings should appear, supporting a logical and predictable heading hierarchy.
- Labeling of Repeating Elements: Floorplans can include labels for repeating regions, improving clarity and orientation within complex pages.

Inclusive Design Benefits

Standardizing floorplan names across the design process helps ensure that accessibility best practices are not applied ad hoc. Without this structure:

- Heading levels may be inconsistent.
- Landmark roles could be missed or misused.
- Navigational regions could appear unpredictably across screens.

By naming and referencing floorplans clearly, teams reinforce semantic consistency and help assistive technologies interpret content more effectively, thereby creating a more inclusive user experience.

Examples

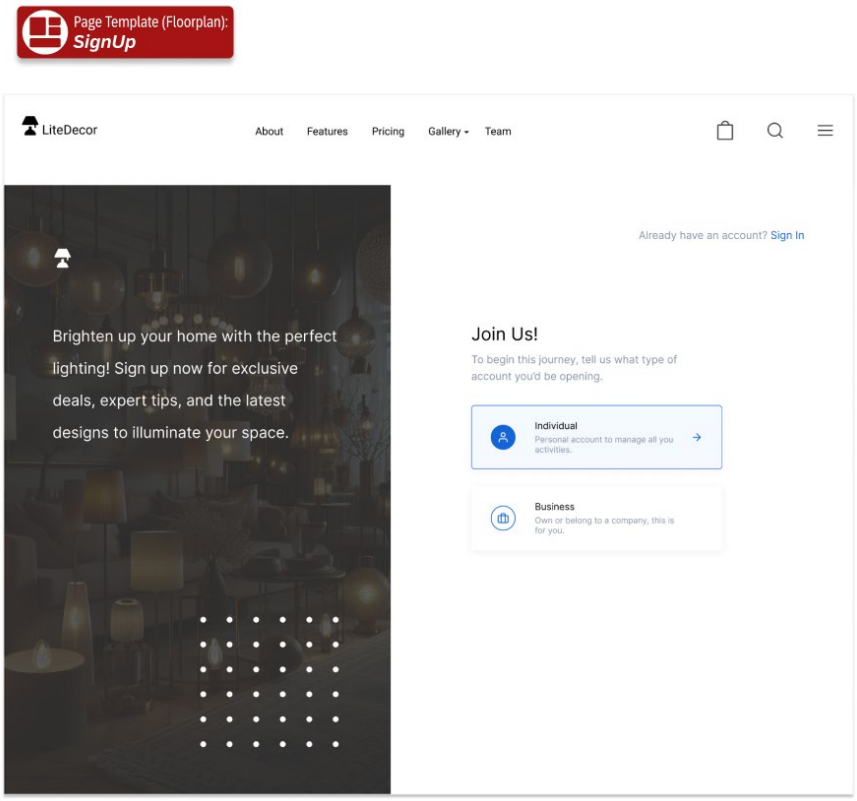


Figure 11: Floorplan name annotation used to identify a Sign-Up page floorplan from design and framework inventories

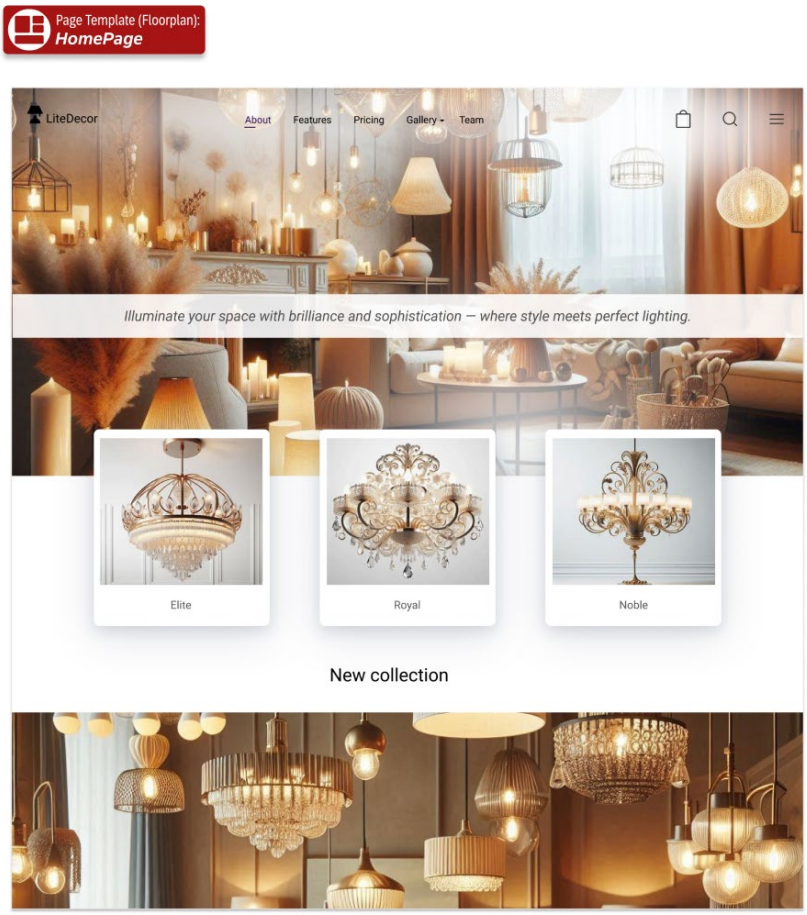


Figure 12: Floorplan name annotation used to identify a home page floorplan from design and framework inventories

For Developers

Once designers create accessibility-focused design specs, developers should review them before implementation. This review ensures that annotations, focus order, interaction behaviors, and semantic elements are clearly documented and feasible to implement.

By checking the specs early, developers can flag potential issues, suggest improvements, and align on technical constraints, helping the team deliver accessible and usable products efficiently.

References

WCAG 2.2 Reference

3.2.3 Consistent Navigation (AA)

Design Note

Add information for alignment with others to complete your design

The set of annotations proposed in the document will help designers clarify how the accessible experience has been addressed during the design phase. But as presented in the introduction chapter, not all WCAG success criteria are covered.

Component Variants

Design Note

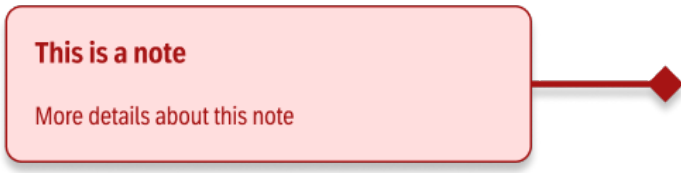


Figure 13: Alignments with other stakeholders may result in added details to the design specification. Use a Design Note to inform consumers of the specification about such details.

About Design Note

There are also edge cases not addressed in this guide which may become critical to delivering a fully accessible experience. Sometimes these scenarios can be solved with user research; other times, it requires alignment with engineers to discuss technical challenges. The results of such strategies may require a more elaborated explanation of a design decision that affects accessibility.

For this, you may find the Design Note annotation helpful to elaborate scenarios of the Visual Design experience, Screen Reading experience, Interactive and Cognitive experience. The annotation is intended to inform developers about something that should be considered during development that cannot be informed using existing annotations. Use the note when discussing design details during design reviews that are worth mentioning to developers. It can also be used to indicate user research results that would be available to all stakeholders. It is helpful to summarize the result of long comment threads in Figma with various stakeholders in the design note. As a result, the decision is also available for anyone who may have access to the document but not to resolved comments.

The design note annotation is also useful for supporting informal design reviews.

Examples

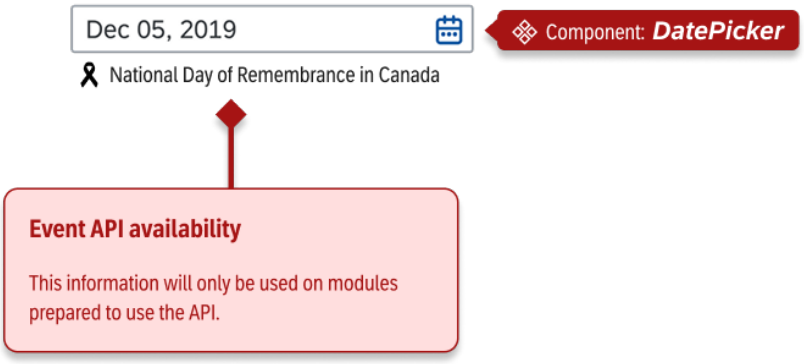


Figure 14: Design note used to indicate dynamic information under an input field that will be handled by and API event

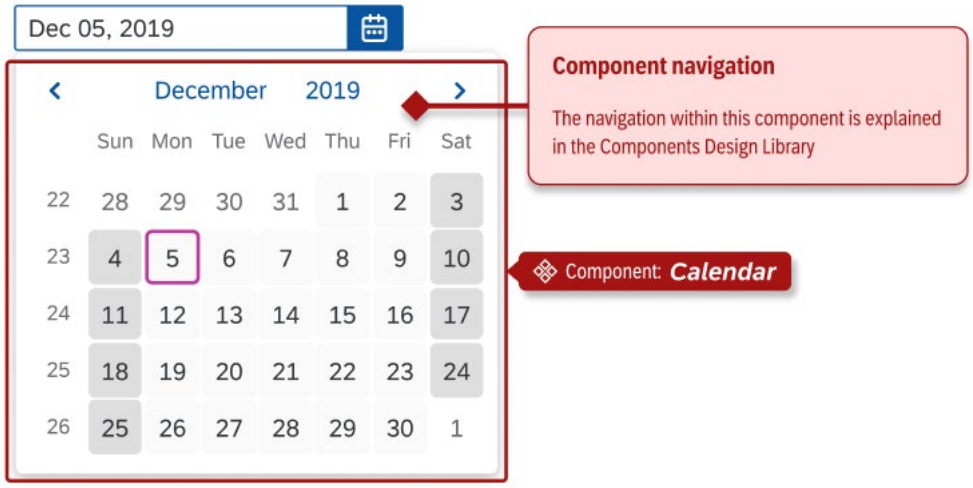


Figure 15: Design note used to indicate that component navigation and exploration is explained in the library of components

For Developers

Developers should review all provided annotations and pay close attention to Design Notes.

These notes often include recommendations for meeting accessibility requirements or guidance on implementing inclusive experiences, highlighting areas where code needs to be handled with extra care.

References

(no references)

“The Accessibility Design Tools have significantly boosted my productivity by streamlining how I identify and address accessibility needs early in the design process - saving time, reducing rework, and helping me deliver more inclusive solutions with greater efficiency.”

Hanna Klymenko – Product Design Expert

Visual Experience

Imagine looking at a screen where the text is clear and easy to read, and important information stands out. What turns visual content into a clear, comfortable experience? The answer is visual content that allows all users to perceive and understand information clearly, regardless of their visual abilities or viewing conditions.

This chapter shows designers how to create perceivable designs that provide an inclusive experience for everyone, especially people with visual disabilities.

The Inclusive Visual Experience

Design your application and UI components with clear, legible, and comfortable visuals that everyone can perceive.

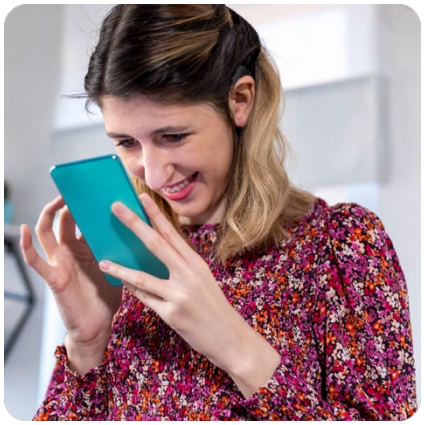
This supports users with visual impairments by enhancing clarity and reducing cognitive load. An inclusive visual design benefits everyone, from those using devices in bright sunlight to anyone who needs a more readable interface.

Sufficient color contrast ensures text and interactive elements are distinguishable for all users. Theme options like light, dark, and high contrast give users essential control over their visual experience. While critical for those with visual impairments, these choices also help people with sensory sensitivities. Text resizing and adjustable spacing enable users with visual or cognitive disabilities to follow lines of text more easily.

Communicate information through multiple methods, such as supplementing color with text labels or icons and using tooltips to clarify purpose. This approach helps critical information reach all users, regardless of how they perceive visual content.

Principles

Visual Impairment



Persona: Victoria

"I'm color blind, have myopia and I'm sensitive to light. I use a screen magnifier, and screen reading at times. It's a challenge to identify object contours, distinguish certain colors and interpret information effectively. Theming provides appropriate screen contrast."

There are degrees of vision impairment that people may experience. This can range from mild vision loss through various levels of reduced visual ability, including difficulty perceiving details, light, or movement.

Permanent: Conditions such as color blindness, astigmatism, long-sightedness. Low vision can happen due to diabetes, retinopathy, macular degeneration, retinitis pigmentosa, glaucoma, and cataracts.

The persona above, Victoria, represents individuals with color blindness, myopia, low vision, light sensitivity, and other conditions that limit or hinder the visibility of digital products presented in digital screens (tunnel vision or peripheral field loss, central vision loss, floaters and obstructions, double vision or diplopia, visual field distortions).

According to the World Health Organization (WHO), 2.2 billion people globally have a visual impairment. Therefore, visibility, a fundamental principle in design, must be addressed with care to support those experiencing visual impairments in perceiving information and conducting tasks.[20]

Some statistics:

- About 2.2 billion people worldwide have some form of vision impairment or blindness (World Health Organization, 2023)[20]
- Approximately 217 million people have moderate to severe vision impairment, meaning they can see but often need aids such as magnifiers, enhanced contrast, or larger text.[7]

- Color vision deficiency affects millions of people across all regions and communities. Based on prevalence rates (~8% of men, ~0.5% of women), this amounts to about 300 million people globally.[11]
- Vision impairments increase with age; age-related macular degeneration (AMD) affects millions of adults aged 50 and older worldwide. In 2021, about 8 million people, roughly 1 in every 1,000 adults globally, experienced vision loss, making even simple, daily tasks a challenge.[9]

Situational and Temporary Disabilites

Accounting for situational and temporary disabilities enables design teams to address challenges that may not be permanent but still impact user experience under certain conditions, such as:

- Glare sensitivity in bright light: This could make it hard for users to see screens or navigate in certain lighting conditions (e.g., outdoors in daylight, or in fluorescent lighting).
- Short-sightedness (not wearing corrective lenses): A person may struggle with vision if they forget, cannot access, or even have broken their glasses or contacts.
- Fatigue-induced vision challenges such as extended screen time may worsen vision temporarily.

General Design Tips

1. **Adjustable Text and Element Size (Text Resizing):**
   When text or interface elements are too small or cannot be adjusted users may have difficulty reading them and using them effectively.

   Design Tip: Support scalable text and UI components without breaking layouts. Ensure resizing is smooth without content overlap or truncation.

2. **High Contrast and Clear Visual Separation (Color Contrast):**
   Low contrast reduces the ability for users to distinguish interface elements and text.

   Design Tip: Use high contrast color schemes and ensure sufficient contrast ratios for text and key UI elements. Avoid relying on color alone to convey information.

3. **Simplified and Responsive Layouts (Responsiveness):**
   Wide or cluttered layouts can be difficult to navigate for users with reduced visual fields or those using magnification.

   Design Tip: Design flexible and well-structured layouts that adapt well to smaller or magnified viewports, minimizing horizontal scrolling and preserving hierarchy.

4. **Support for Zoom and Magnification (Responsiveness):**
   Users who rely on zooming or magnification tools may encounter distorted or broken interfaces if designs are not flexible and well-structured.

   Design Tip: Test designs at various zoom levels to ensure text readability and interface element functionality remain intact.

5. **Support Multiple Themes (Theme):**
   Designs that only work in a single-color scheme can limit accessibility and personalization.

   Design Tip: Design with flexibility for light, dark, and high contrast themes. Ensure brand colors adapt and maintain contrast requirements across all modes.
6. **Consistent Text Spacing (Text Spacing):**
   Fixed text spacing can make it difficult for users with cognitive

disabilities such as dyslexia to read text effectively.

Design Tip: Allow line height, letter spacing, and word spacing to adjust without cutting off text or breaking layouts. Ensure readability when users apply custom spacing settings.

7. **Accessible Tooltips (Tooltip):**
   Minimal or icon-only designs can leave sighted users uncertain about meaning or function. Tooltips help provide clarity on mouse hover and keyboard focus - covering the needs of those with limited mobility who cannot perform precise pointer actions.

   b Use tooltips to clarify the meaning of icons or complex actions for sighted users. Ensure they can be activated and dismissed with both mouse and keyboard and remain visible long enough to be read comfortably by the user. Never rely on tooltips as the only way to convey critical information—always provide context in text or labels as well.

Annotations

Visual Experience Annotations

This checklist helps designers remember key accessibility aspects that need to be addressed during the design phase.

The visual experience scope helps designers check color contrast, ensure theming works, validate semantics, and address truncation or wrapping through simulations for reflow, responsiveness, text resizing, and text spacing.

The visual experience category also incorporates visual tooltips which should be used according to best practices for labelling components.

Visual Experience Checklist

- [ ] Color Contrast
- [ ] Theme
- [ ] Responsiveness
- [ ] Text Resize
- [ ] Text Spacing
- [ ] Tooltip

Color Contrast

Describe the color contrast values used in your design.

Component Variants

Color Contrast on Standard Theme

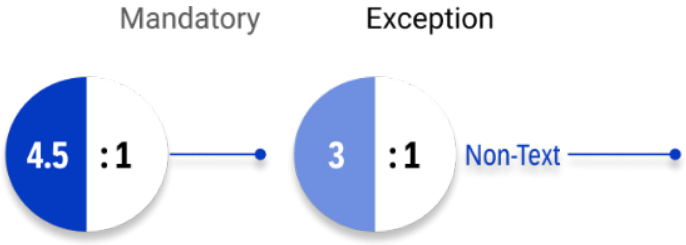


Figure 16: The color contrast annotations 4.5:1 (for text) and 3.1 (for non-text elements) indicate that each chosen color meets the minimum color contrast requirements with its background

Color Contrast on High Contrast Theme

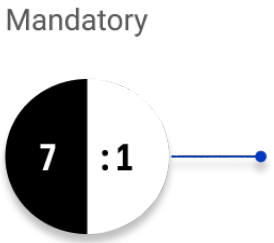


Figure 17: The color contrast annotation 7:1 indicates that the chosen color meets the minimum color contrast requirements with its background

About Color Contrast

Color contrast is the contrast ratio between two values, foreground and background color. It is mandatory for any UI element conveying relevant information to meet the minimum contrast ratio. It applies to text and graphical elements, such as controls and icons.

Color Contrast Requirements

Ensure a minimum contrast ratio of **4.5:1** for all text (including hover, focus, and active states), placeholders, icons, and images of text. This standard guarantee readability and supports users with low vision or contrast sensitivity.

The following cases require a sufficient color contrast:

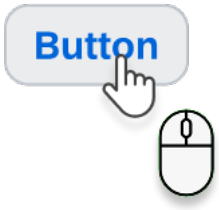


Figure 18: Text in hover state of a button

Figure 19: Icons

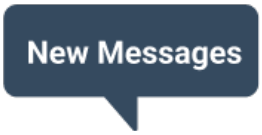


Figure 20: Images of text

Color Contrast Exemptions

Certain elements are exempt from contrast requirements: disabled controls, purely decorative borders or patterns, and non-informative graphics such as illustrations, photographs, and logos. While exempt, designers should still aim for clarity whenever possible.

The following cases are exemptions:

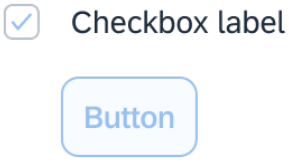


Figure 21: Disabled controls

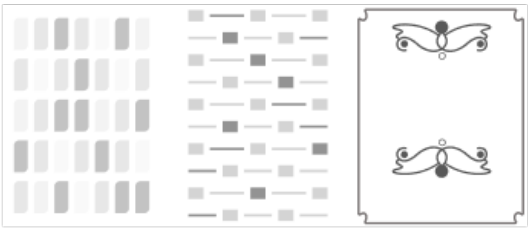


Figure 22: Patterns, borders and decorations

Figure 23: Illustrations, photographs and logos

High contrast theming aims to meet 7:1 color contrast by default, with a minimum of 4.5:1 for exceptions. Users with low vision, particularly those working in bright conditions, use this theming to compensate for moderate visual challenges such as floaters, cloudy vision, or early macular degeneration. This helps them work more efficiently and without exhaustion.

Use the mandatory color contrast ratio of 4.5:1 (or 7:1 for high contrast theming) or higher whenever possible. If this is not possible, check if the element qualifies as an exception or exemption. Use a minimum contrast of 3:1 for object borders of UI elements.

Examples

Font Size and Icons in Reference to Color Contrast

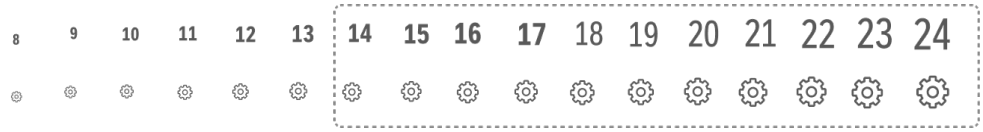


Figure 24: Large text definition starts at 14 pt or larger when bold or regular 18 pt text or larger when not bold

The same color contrast rule applies to icons. Aim for a minimum contrast ratio of 4.5:1 on the default theme. For large text, an exception allows a minimum contrast of is 3:1 on the default theme.

Figure 25: Meaningful non-text elements, such as an input field, require a contrast ratio of at least 3:1 against the background

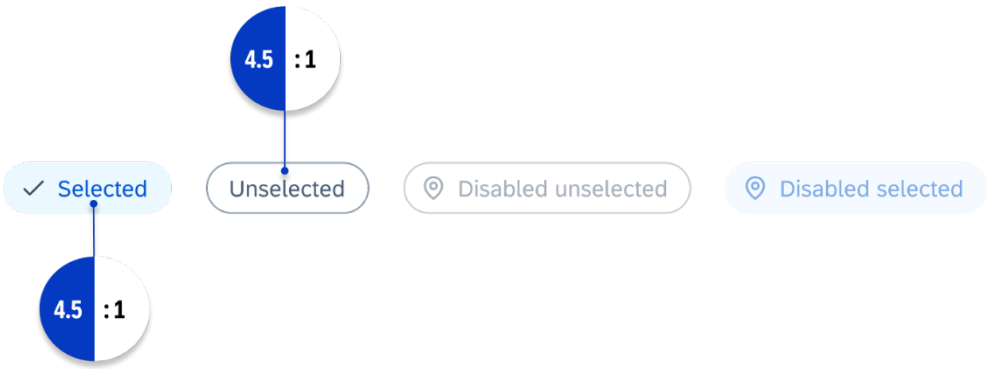


Figure 26: The visual presentation of text must have a contrast ratio of at least 4.5:1. Text within an inactive user interface is exempt from this contrast requirement.

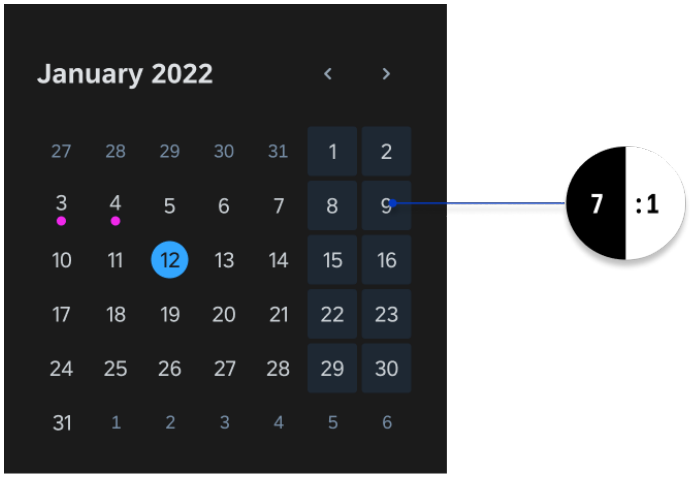


Figure 27: In high contrast themes, text must have a contrast ratio of at least 7:1.

Shared Benefits

Temporary Disabilities

All users

A sufficient color contrast benefits more than just users with low vision. Individuals vary greatly in their ability to perceive color and contrast. Environmental factors, such as intense sunlight shining on a display, can make it difficult to read content.

Limited Cognition

Cognitive

For some users with cognitive disabilities, sufficient color contrast is crucial. Conditions that affect the ability to process information visually benefit from improved readability and clarity. Choosing colors that can be easily perceived can also reduce stress and cognitive load.

For Developers

Developers should check during the implementation if foreground and background color values for all user interface component parts in their frameworks meet the required contrast ratios for the default theme.

References

WCAG 2.2

1.4.1 Use of Color (A)

1.4.3 Contrast (Minimum) (AA)

1.4.11 Non-text Contrast (AA)

Theme

Ensure your designs account for both light and dark contrast themes and include high contrast themes in your designs.

Component Variants

Theme

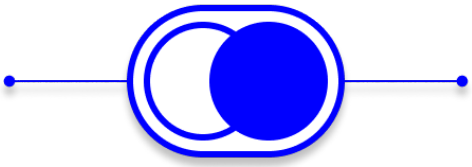


Figure 28: This annotation is used to draw a parallel between a screen designed for the standard theme and its corresponding version in the high contrast theme.

About Theme

The annotation featured above illustrates the relationship between a screen designed in the standard theme and its counterpart in high contrast, which is available as an optional setting.

Both standard and high contrast themes must be addressed in the design.

Reduce color saturation and avoid or limit pure white and black colors. When using pure black (#000) as a background for dark mode, the text and content can be slightly dimmed to reduce eye bleed due to extreme contrast.

If possible, design the light mode version first and ensure all visual elements and cues are preserved when creating the dark mode version. Maintain the colors in dark mode and adjust them as needed within the theme implementation to ensure the intended design remains clear and effective.

Plugins to help adjust colors:

- Appearance (Figma plugin).
- Dark mode magic (Figma plugin).
- Camilo (Sketch plugin).

Keep consistency in mind by applying the color guidance of your design system.

An application should have a color contrast setting control for users who wish to alternate themes. The setting offers the option to switch from a default light theming contrast ratio of 4.5:1 to a high contrast theming, respecting the 7:1 contrast ratio.

Examples

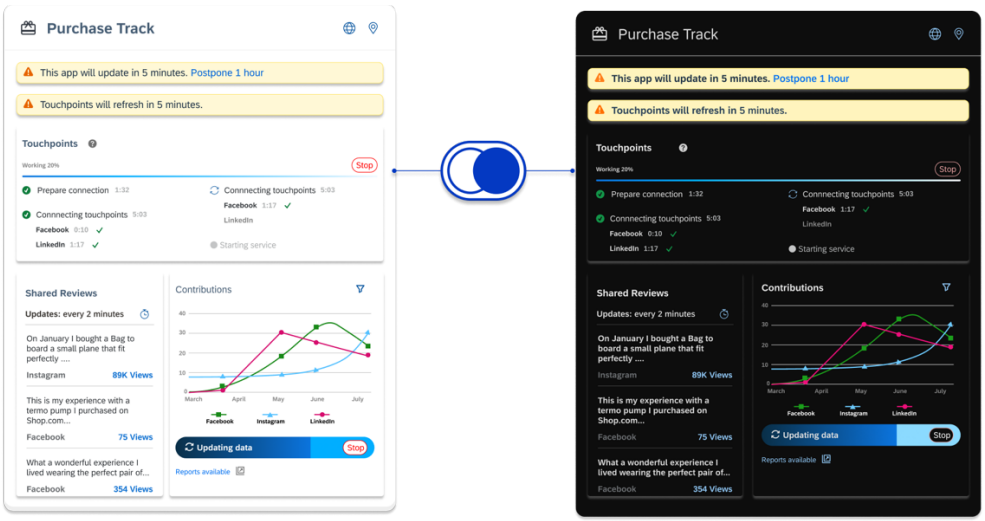


Figure 29: This theme annotation is used to show a parallel between a screen designed for the standard theme and its corresponding screen designed in the high contrast theme.

Shared Benefits

Temporary Disabilities

All users

Themes also support users with temporary disabilities such as an eye strain or an injury. Switching themes for better contrast, reduced glare, or easier readability can ensure the usage is more comfortable until recovery. High contrast themes are not limited to specific user groups; people without disabilities may also prefer more visually distinct interfaces and find them easier to read or navigate.

Limited Cognition

Cognitive

Themes like high contrast primarily support users with visual impairments but can also be beneficial for users with cognitive disabilities. This affects, for example, people who struggle with focusing on tasks due to sensory sensitivities. However, it is crucial to understand that preferences will vary, and some users with cognitive disabilities may prefer a lighter color scheme over high contrast.

For Developers

Developers should verify during the implementation if they:

- provide end users multiple themes as part of the UI framework, one of them being a theme with 4.5:1 contrast ratio support and another with 7:1 high contrast support.
- provide an option for end users to switch between these themes.

If no theme configuration is provided as part of the UI framework, a fallback must exist that allows users to apply different OS themes on the platform, while still complying with the given requirements.

References

WCAG 2.2

1.4.1 Use of Color (A)

1.4.11 Non-text Contrast (AA)

Responsiveness

Design the layout so that UI elements are reorganized dynamically, adjusting and responding to different device orientations and screen sizes.

Component Variants

Screen Resize

Figure 30: Simulate a web screen seen from a mobile device or tablet to ensure content visibility is preserved.

Orientation

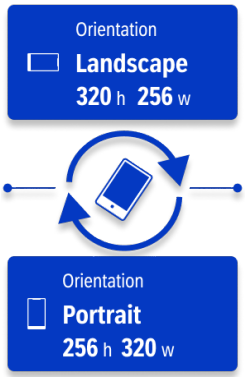


Figure 31: Simulate the mobile layout orientation to explore how content adjusts to portrait and landscape orientations.

Content Visibility

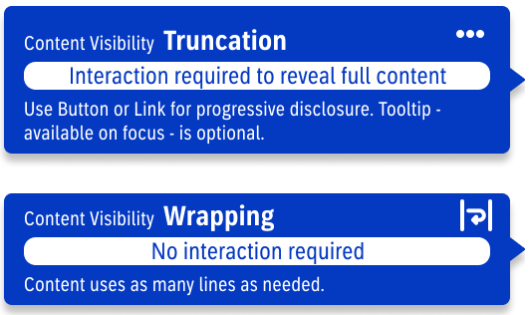


Figure 32: Provide instructions on how to address content visibility issues by proposing truncation or wrapping.

About Responsiveness

Prepare your designs to simulate how layouts and content behave across different screen sizes and orientations, including web, tablet, and mobile. The Screen Size Reflow and Orientation annotations help designers ensure the user interface adapts correctly and remains usable across devices.

This design consideration is supported by two WCAG requirements. The 1.3.4 Orientation (AA) requires that content remains fully operable and understandable in both portrait and landscape modes (or any orientation), unless a specific orientation is essential for the functionality. This ensures that users can maintain consistent spatial cues and mental maps as they switch between device orientations, preventing disorientation and supporting a stable navigation experience.

The 1.4.10 Reflow (AA) requires that content can be presented without loss of information or functionality, and without the need for two-dimensional scrolling (both horizontal and vertical). This ensures that users who need to enlarge text or content, can still follow the layout and understand the page structure without excessive scrolling. This supports continuity, helps users keep their place, and reduces cognitive load during navigation.

Screen Resize

Use the screen size reflow annotation to show how the layout and content adapt when the viewport changes, for example when resizing a web browser or adjusting a design for tablet and mobile screens.

Purpose: This annotation ensures that the UI is responsive and that it maintains usability and content clarity across a wide range of screen sizes. It focuses on layout reflow, element wrapping, and content scaling during viewport resizing.

What to Specify:

- Reflow Behavior: Describe how content and layout adapt when transitioning from large to smaller screens (e.g., multi-column to single column).
- Element Wrapping or Truncation: Indicate which elements are allowed to wrap and which should truncate, following design system guidance.
- Minimum Supported Size: Ensure the design supports on desktop a width of 320 CSS px for vertical scrolling content and a height of 256 CSS px for horizontal scrolling content.
- No 2D Scrolling: Confirm that content does not require horizontal scrolling in addition to vertical scrolling, unless it is essential (e.g., data tables, maps).
- Zoom and Resize Compatibility: Ensure designs remain operable at 200% zoom.

Why it Matters:

- Supports accessibility and flexible usage across devices.
- Improves design handoff clarity for developers implementing responsive behavior.
- Helps users who rely on zooming, screen magnifiers, and custom device settings.

Orientation (Mobile/Tablet)

Use the orientation annotation to define how the design behaves in portrait vs. landscape orientations, particularly on mobile and tablet devices.

Purpose: This annotation ensures that the design is not restricted to a single orientation unless necessary, and that it works reliably in both modes.

What to Specify:

- Support for Both Orientations: Indicate whether the interface adapts properly in portrait and landscape modes without loss of content or functionality.
- Layout Adjustments: Note any key layout shifts (e.g., navigation bar repositioning or text scaling) that occur between orientations.
- Essential Orientation: If one orientation is required for a functional reason, clearly document it as a design note (e.g., landscape-only video editing tool).
- Minimum Size Support: As with reflow, verify that orientation changes do not reduce the layout below accessibility thresholds of 320×256 CSS pixels, depending on the mode.

- No Content Lockout: Ensure users are not blocked from accessing key functions based on how they hold the device.

Why it Matters:

- Orientation flexibility accommodates users with different device preferences and physical needs.
- Prevents critical functionality and content from becoming inaccessible in any orientation, not just the preferred mode.
- Aligns with WCAG requirements and inclusive design standards, especially for mobile-first experiences.

The takeaway of this chapter is: Prepare your screens to support content and layout adjustments for smaller screens. This requires careful attention to layout adjustments, content reorganization, and decisions about wrapping or truncation.

Table 7: Comparison of responsiveness annotation variants

Annotations	Focus	Consideration
Screen Size Reflow	From large to small screens	Reflow logic, wrapping/truncation, zoom, 2D scroll prevention, min size support
Orientation	Portrait vs. Landscape modes	Mode adaptability, layout shifts, restrictions, functional requirements

Content Visibility

Loss of visibility is a violation of WCAG 1.4.4. Text Resize. When content needs to adjust to different screen sizes, the design should plan for potential content visibility issues. These can be addressed with truncation or wrapping.

Wrapping

This is the simplest solution to guarantee full content remains visible when a layout adjusts to text resizing, but also text spacing and responsiveness. This solution is ideal for content that is not related to an interactive UI element, such as page headers.

Truncation

Try using line wrapping whenever possible, truncation is acceptable as long as the full content is available on focus or drill in (after user activation where truncated content is presented). An indication that this information can be accessed, should be provided to the user in some way, such as a button or a link. This trigger allows users to activate a flow to visualize the full content. A button can open a dialog, popup, and expand text or a section. A link can transport the user to a new page where the full content is available.

Examples

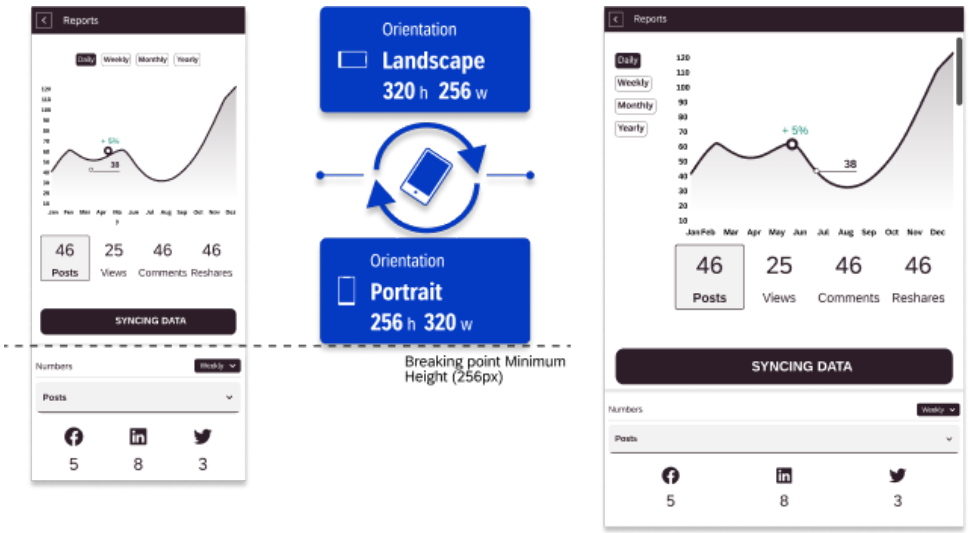


Figure 33: Simulation of a mobile layout in both portrait and landscape orientations to assess how content adjusts

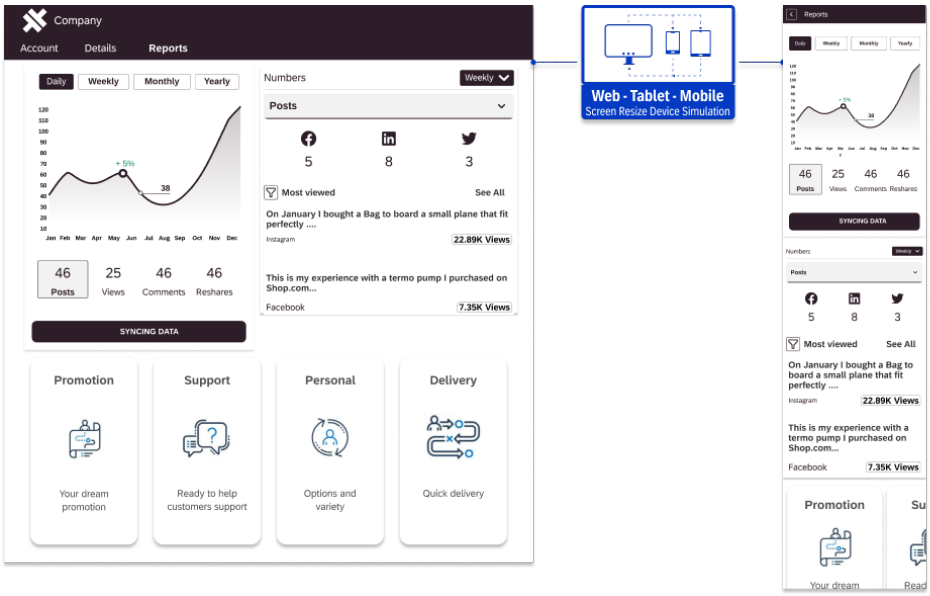


Figure 34: Simulation of a web, tablet, and mobile layout in portrait and landscape orientations to assess how content adjusts to different screen formats

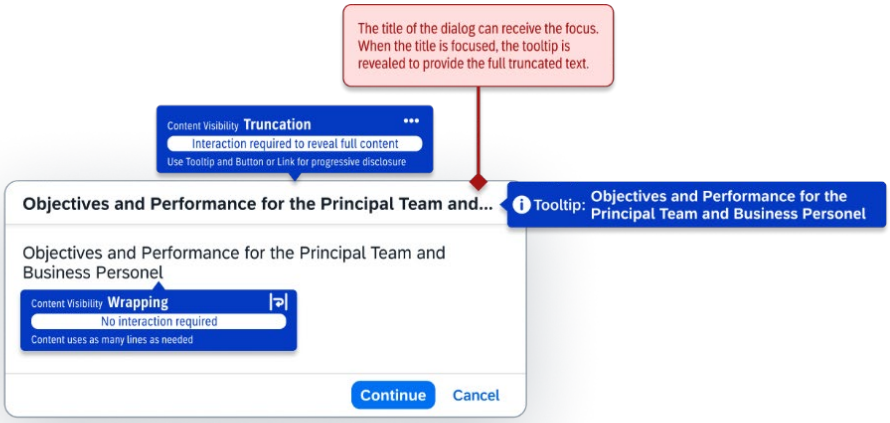


Figure 35: Illustration of different truncation and wrapping scenarios using the content visibility annotation variant

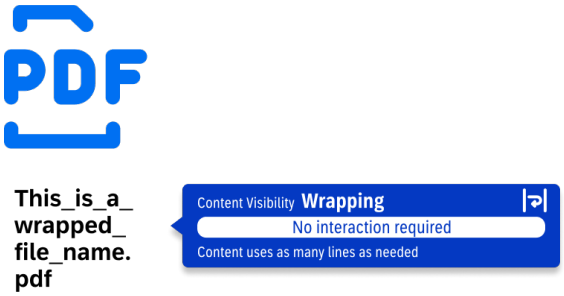


Figure 36: Illustration of a wrapping scenario using the content visibility annotation variant

Shared Benefits

Temporary Disabilities

All users

Responsive design and adaptable orientation, such as resizing a browser while multitasking or rotating a phone, benefit everyone, including people with temporary disabilities such as a broken arm or eye strain, by reducing effort and preventing loss of content or functionality.

Limited Cognition

Cognitive

A clear, consistent layout across screen sizes and orientations reduces mental effort, supports predictable navigation, and prevents confusion caused by shifting or hidden content. Consistent placement of navigation and controls supports memory and recall, while flexible device and orientation options allow users to choose the setup that best supports focus, making the experience more intuitive and less overwhelming.

Limited Mobility

Interactions

Responsive and adaptive design helps users with reduced mobility navigate and interact across devices. Layouts that adjust appropriately prevent interactive elements from becoming too small or hard to reach, and consistent placement of controls minimizes physical effort. Flexible device and orientation options allow users to choose the setup that best supports their motor abilities.

For Developers

Developers should verify during the implementation whether they:

- use relative units
- use media queries
- use responsive layout grids
- do not intentionally suppress responsivity (e.g., by setting special flags)
- do not limit device orientation capabilities

References

WCAG 2.2

1.3.4 Orientation (AA)

1.4.10 Reflow (AA)

Text Resize

Simulate text resizing to verify how the text remains visible on the screen and recommend strategies to wrap or extend elements as needed.

Component Variants

Text Resize Annotation with Recipe for Success

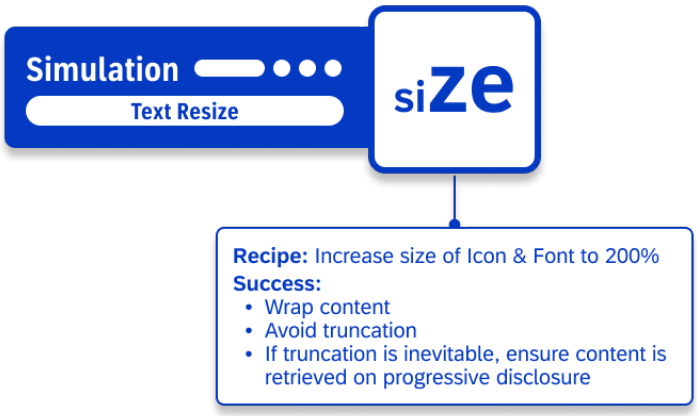


Figure 37: Demonstrates how text resizing is applied while maintaining content visibility. The annotation links the standard screen to a screen with a 200% font size increase, showing that content remains visible, with acceptable wrapping or truncation where necessary.

About Text Resize

Accommodate text content by reorganizing the layout within the same screen size for when users resize text up to 200%. This accessibility support benefits low-vision users who rely on built-in capability, rather than magnification tools and assistive technologies.

Prepare your screen to allow the text to resize for all UI elements. This will require the layout to adjust and reorganize accordingly.

Users that magnify text up to 200% can do so using a mouse and keyboard. Using the CTRL key while scrolling the mouse wheel up or down resizes content in Chrome, Firefox and IE. Some browsers also provide configuration options to control text resizing.

The screen should provide visual feedback as the layout scales, ensuring all functions remain usable on the page, including text alternatives for non-text content. Horizontal scrollbars should be avoided. Some issues may occur during content rearrangement.

Here are some ways to solve them:

- Give a variable amount of space that adapts to the text length
- Wrap the text.
- Add a data tip with the full text that becomes active when the text is truncated and make sure the tooltip is triggered on keyboard focus.
- Display the full text on focus; overlapping with nearby text is acceptable in this case.
- Make the text rollable on focus.

Content Visibility

Loss of visibility is a violation of WCAG 1.4.4. Text Resize (AA). When content needs to adjust to the screen, the design should anticipate potential content visibility issues, which can be addressed with truncation or wrapping.

Wrapping

This is the simplest solution to guarantee full content remains visible when a layout adjusts to text resizing, but also text spacing and responsiveness. Wrapping is ideal for text that is not part of an interactive element, such as page headers, to maintain readability without affecting usability.

Truncation

Avoid truncation whenever possible. If line wrapping is not possible; truncation is acceptable provided the full content is available on focus or drill in (after user activation of trigger where truncated content is presented). An indication that the content can be accessed, should be provided to the user in some way, for example a button or a link. This indication will enable users to trigger a flow to visualize the full content. A button can open a dialog, popup, expand text or expand a section of text, while a link will transport the user to a new page where the full content is available.

Examples

Figure 38: Example of a component demonstrating how content adjusts when the component is resized

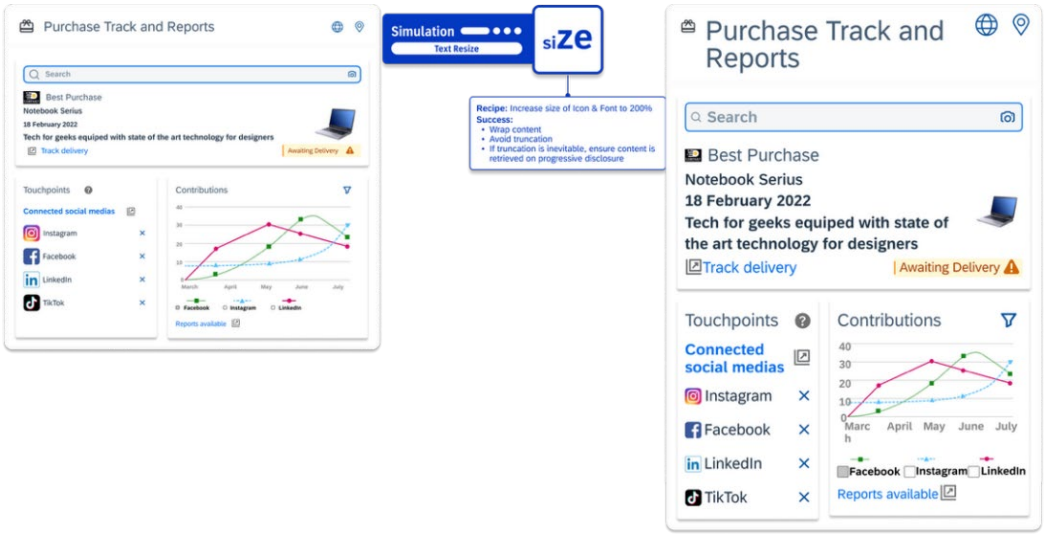


Figure 39: Example of a screen demonstrating how the layout changes when content is resized

Shared Benefits

Temporary Disabilities

All users

Adjusting text size is not only important for users with visual impairments. Enlarged text may also simply reflect personal preference or make reading easier, and external conditions can make larger text desirable. Therefore, designs should ensure that all content remains usable at up to 200% font size.

Limited Cognition

Cognitive

Increasing the text size can also support users with cognitive disabilities such as dyslexia to ease legibility.

For Developers

Web Development: Developers should verify during the implementation whether they:

- Use a responsive layout approach (see previous chapter)
- Set the preferred browser font size
- Decide on a minimum font size
- Set font-size: 100% on the HTML tag
- Use em/rem units for font sizes
- Observe how it behaves in practice by checking for truncation, wrapping, and other display issues

Do not suppress any zooming functionality. In particular, avoid the following:

```
<meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1, user-scalable=no">
```

Native Mobile Development: Verify that all app text and object borders adapt to font resizing settings of the platform without truncation or improper wrapping.

References

WCAG 2.2

1.4.4 Resize text (AA)

1.4.8 Visual Presentation (AAA)

Text Spacing

Simulate the layout with text spacing applied to the content and observe how it affects the organization of UI elements on the screen.

Component Variants

Text Spacing Annotation with Recipe for Success

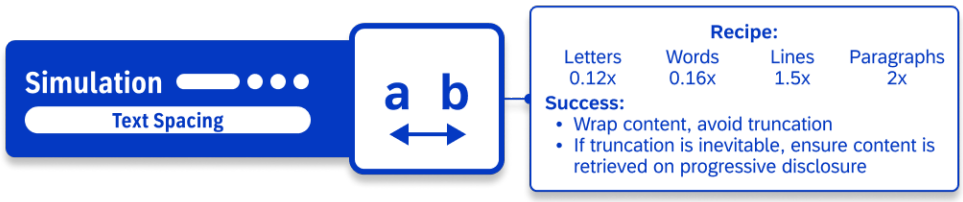


Figure 40: Simulate the web screen layout shifts when customizing the text spacing, ensuring content visibility is maintained using wrapping or truncation where acceptable. Annotation includes instructions to address text spacing.

About Text Spacing

Many people with visual deficiencies benefit from increased text spacing, as single-spaced blocks can make it difficult to track lines and distinguish characters. By allowing users to adjust line height, letter spacing, and word spacing, the layout can expand naturally without breaking content, which improves readability and reduces visual strain. It is essential to note that when text space increases, the screen area requires more space to display the information, preferably without truncation. All information should remain visible.

A designer can test how additional test spacing affects the layout to ensure that the content remains readable and operable when text spacing is activated. As a result, UI elements may need to be enlarged to fit the content or wrap it to the next line. Here are the rules applied to text spacing:

- Line height (line spacing) to at least 1.5 times the font size
- Spacing paragraphs to at least 2 times the font size
- Letter spacing (tracking) to at least 0.12 times the font size
- Word spacing to at least 0.16 times the font size.

These rules apply to small or large pieces of text, including labels, read-only fields, header and sub header, small strings of text, and text used in tables.

Avoid truncation if possible. Explore wrapping the text and adding an extra line, such as on a table or cards. Content cut off with no way to access to the hidden information demonstrates poor design. Truncation should provide access to the full content, either through an additional click to read the whole content or a tooltip triggered on keyboard focus.

Examples

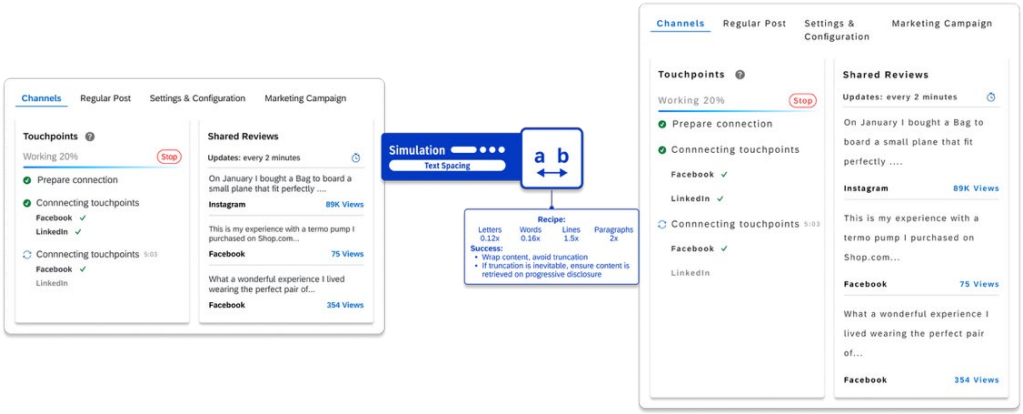


Figure 41: Example for simulating how a web screen layout shifts by text spacing, ensuring content visibility is maintained including recommendations to fulfill the requirement

Shared Benefits

Limited Cognition

Cognitive

Many people with cognitive disabilities benefit from adjustable text spacing, as it helps track lines of text more easily and reduces mental effort when reading dense content. Allowing line height, letter spacing, and word spacing to expand naturally supports comprehension, prevents confusion, and makes the reading experience more manageable and less overwhelming.

For Developers

Web Development: Developers should check during the implementation that they:

- Use a responsive layout approach (see previous chapter)
- Set the preferred browser font size

- Decide on a minimum font size
- Set font-size: 100% on the html tag
- Use em/rem units for font sizes

Additionally, Developers should provide an option to override default text spacing by defining corresponding CSS as follows:

```
*{line-height: 1.5em !important;
    letter-spacing: 0.12em !important;
    word-spacing: 0.16em !important;
  }
p{margin-bottom: 2em !important;}
```

This can be configured, for example, in the application settings.

Native Mobile Development: Developers should verify whether all app text and object borders adapt to extra text spacing settings of the OS without causing truncation or unwanted wrapping.

Alternatively, Developers should ensure the ability to select a font with larger spacing, which has the same effect.

References

WCAG 2.2

1.4.12 Text Spacing (AA)

Tooltip

Use tooltips to clarify icons, map elements, and graph data points. Ensure they work with both mouse and keyboard and do not exclusively hold critical information.

Component Variants

Tooltip Annotation

Figure 42: Tooltip reminder pointing to the element that will reveal information on mouse over or keyboard focus.

About Tooltip

A tooltip is a short piece of text that appears on mouse hover or keyboard focus, providing an additional layer of information in the UI. Tooltips are commonly used with icons, links, graphics, charts, maps, and other interactive elements. They are especially helpful for people with vision, but it is important to note that some blind users disable tooltip narration in assistive technologies. Because screen reader users can switch off the announcement of tooltips, they should never be the sole way of communicating essential information. Read more about this topic in the Screen Reader Experience chapter.

Some users also rely on screen magnifiers to read the content. To support them, tooltips should:

- Stay visible long enough to be read comfortably.
- Scale correctly when zoomed.
- Avoid overlapping or hiding other important interface elements.

Since tooltips appear and disappear with keyboard focus or mouse hover, the interaction should be designed so that users can perceive the additional content and dismiss it without disruption.

Tooltips must be:

- Dismissible (e.g., by pressing Escape).
- Persistent long enough for comfortable reading.
- Positioned carefully so they do not obscure other pertinent information.

Tooltips are usually not associated with a visual design. They are rendered by the system. Tooltips are dismissed when users move the mouse out of the triggered element or continue tabbing through the interface.

Recommended Usages

Because tooltips can be easily missed, they should not be the only way of providing vital information to complete a task. Use a tooltip only for secondary or supplemental information, not for repeating visible labels or obvious interactions. Some appropriate use cases that would require an invisible description for screen reading users:

- Onboarding guidance: introduce new users to a flow (typically no more than 3 or 4 tooltips).
- Featured adoption: highlight and explain new capabilities.
- Taxonomy support: clarify terminology and remind users about rarely used features.

Tooltips can be used to inform users about available shortcuts, such as hotkeys in native tooltip on the screen. However, they should only be attached to interactive UI elements. Placing tooltips on non-interactive elements prevents users with limited mobility, such as keyboard-only users, from accessing information. For screen reader users, a tooltip should function as an invisible label, so it is announced once, minimizing verbosity for blind users. This also ensures the information remains available when users disable tooltip announcements in their assistive technology.

Tooltips can also be triggered through gestures and short presses on mobile devices. In mobile applications, they are typically revealed with a long press or by tapping an info/help icon, since a short press is reserved for the primary action. While these patterns are common, there is no universally adopted gesture standard across industries to open tooltips in mobile apps.

Beyond Tooltip

The WCAG requirement 1.4.13 Content on Hover or Focus (AA) covers any content that appears when a user hovers with a mouse or focuses with a keyboard, such as tooltips, popovers, dialogs, or custom dropdowns. The requirement ensures that this content is usable for everyone by specifying that it must be dismissible without moving the pointer, remain visible while being hovered over or focused, and not disappear unexpectedly. This helps users with limited mobility, low vision, or those using keyboard navigation to fully access and interact with additional content.

Tooltips and popovers are useful because they reduce the need for users to interrupt their workflow and consult documentation. However, when

more detail is required, it is often better to use an overlay popup, a popover, or to present the information in context menus.

The tooltip annotation was moved to the Visual Experience category because it enhances interface clarity for sighted users by displaying additional information without cluttering the main layout. For sighted users, a tooltip can serve as an alternative to a label for example when an icon button is used. For blind users, however, tooltips are often disabled in screen readers, so they should not be relied upon as the only method of conveying vital information.

Examples

Figure 43: Example tooltip for an Edit button

Keep in Mind Content for Sighted vs Blind Users

Do not confuse content intended to support screen reader users and content meant to support sighted users.

Tooltips provide secondary information for sighted users, helping them understand context through visual proximity and building confidence about the action that will be triggered.

Content that appears on hover or keyboard focus may also be repeated by the screen reader if the user has not disabled the tooltips in the settings. However, it is likely that users who do hear them will often skip the repeated text. Besides tooltips, invisible labels should also be provided to support blind users, rather than solely relying on tooltips.

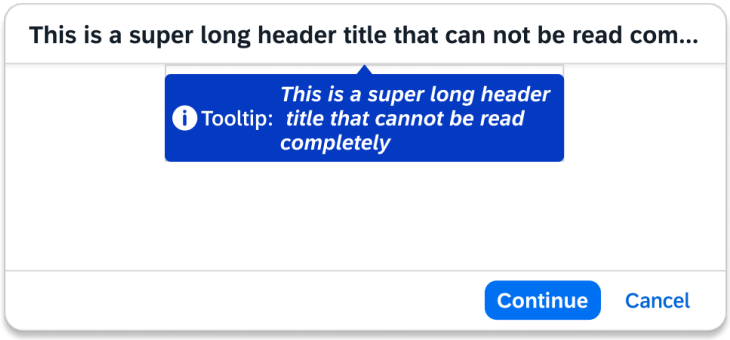


Figure 44: Tooltip added to truncated text, shown on hover or keyboard focus, only to be used when text wrapping is not possible and can be triggered on focus

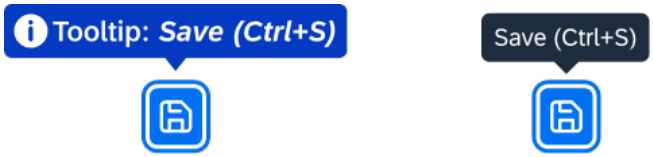


Figure 45: Annotation in a design specification showing the proposed content of the tooltip and its final rendered appearance

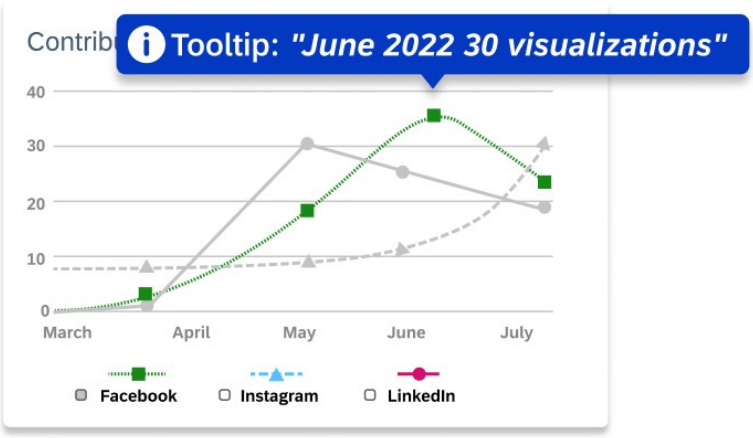


Figure 46: Design specification showing proposed content for implementing a tooltip as a data tip

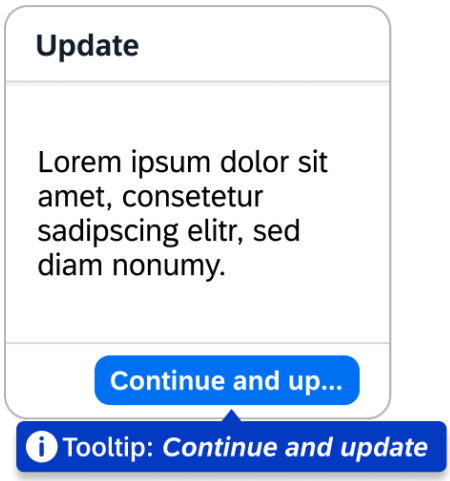


Figure 47: Button with a tooltip that shows the full label which is helpful for screen magnifier users

Shared Benefits

Temporary Disabilities

All users

Sighted users must be supported with tooltips to understand the purpose of icons or buttons that lack a permanent visible text. Tooltips provide additional context on hover or focus, helping users avoid confusion, especially in complex interfaces or when icons alone are ambiguous.

Limited Mobility

Interactions

Keyboard users must be supported with tooltips that appear on focus, not just hover, so they can access the same contextual information as mouse users and blind users using the screen reader. This ensures equal understanding of icon-only controls or complex actions without needing a mouse.

Limited Cognition

Cognitive

Tooltips support users cognitively by reducing memory load and clarifying the purpose of icons, buttons, or data points without requiring users to guess.

For Developers

Web Development: Developers should ensure that tooltips are implemented either by using the native `title` attribute of the respective HTML element or, if they want to support custom styled tooltips, develop their own tooltip widget following established best practices, such as those outlined in the Tooltip Pattern | APG | WAI | W3C.

Native Mobile development: Use the native tooltip support in the Software Development Kits (SDKs) of the given platforms:

iOS: ToolTip | Apple Developer Documentation

Android: Tooltip | Jetpack Compose | Android Developers

References

WCAG 2.2

1.1.1 Non-Text Content (A)

1.4.5 Images of Text (AA)

1.4.13 Content on Hover or Focus (AA)

“A simple, well-structured tool that makes designing with accessibility in mind easier and faster.”

Adelina Todorova – UX Designer

Interaction Experience

Imagine navigating an application with just your keyboard, where each keystroke moves you logically and efficiently toward your goal. What turns a complex interface into an efficient, navigable experience? The answer is logical focus, flow, and clear interaction patterns.

This chapter shows designers how to implement keyboard support and create an inclusive experience for everyone, especially people with limited mobility.

The Inclusive Interactive Experience

Prepare your application and UI components so anyone can navigate and interact with them using multiple input modalities.

This is vital for individuals with limited mobility, who may struggle with or cannot use a mouse. Supporting keyboard interactions also empowers expert users with faster, more efficient navigation and provides an essential interaction method for screen reader users.

Ensure a logical focus order that matches the visual layout of the page**.** When a user presses the Tab key, focus should move through links, buttons, and form fields in a predictable sequence. This is essential for keyboard-only and screen reader users to navigate without getting lost, and it creates a more intuitive and efficient experience for everyone.

Manage focus within complex components to prevent disorientation. Enable standard keyboard patterns, such as using arrow keys to move between internal options, to let them explore its contents. This approach reduces cognitive load for all users by making complex interactions feel predictable and manageable.

Efficient shortcuts let expert users perform common actions instantly, while standard keys offer a predictable way to activate focused elements. This consistency across the application reduces the mental effort required to learn the interface, making it feel intuitive and responsive for everyone.

Principles

Limited Mobility

Persona: Monique

“With age I’ve developed arthritis and tremors, and I also live with Multiple Sclerosis, which brings stiffness, muscle weakness, and fatigue that vary from day to day. These symptoms make precise control with a mouse challenging, and I sometimes struggle with repetitive or time-sensitive tasks.

To stay productive, I rely on alternatives like keyboard shortcuts, touch interactions, and speech-to-text, and I work best when interfaces are forgiving, consistent, and don’t demand precise or rushed input.”

People with mobility limitations may find it difficult to use traditional input devices such as a mouse, keyboard, or touchscreen due to various conditions affecting movement and dexterity. This diverse group ranges from people born with congenital conditions to those who sustain injuries or develop progressive diseases.

This user group includes people of all ages with various conditions affecting movement and dexterity, such as adults with cerebral palsy, middle-aged individuals with Multiple Sclerosis, and older adults with tremors like Monique. These conditions affect people across all age groups, from children born with cerebral palsy to adults who sustain spinal cord injuries.

Conditions that may limit mobility:

- Cerebral palsy, Spina Bifida
- Muscular dystrophy, Parkinson's disease
- Multiple sclerosis, arthritis, essential tremor
- Spinal cord injury, lost or damaged limb(s)

Some statistics:

- The prevalence of mobility limitations increases significantly with age, affecting around 35% of adults by age 70 and rising to more than half for those over 85.[8]
- With increasing age, the likelihood of developing arthritis, a chronic inflammation of the joints, also rises. According to the CDC, it is the leading cause of disability in the United States and limits 24 million Americans in their daily activities.[6]

Situational & Temporary Disabilities

By integrating situational and temporary disabilities in the discussion of limited mobility, we help design teams address challenges that, while not be permanent, can still impact user experience under certain conditions.

Examples of situational and temporary disabilities:

- Sweaty hands (as mentioned earlier): Temporary difficulty with touchscreens or physical devices due to sweating.
- Temporary injuries (e.g., sprained wrist): Limited range of motion in the hands or arms, making it hard to use touch devices or keyboards.
- Carrying objects: When holding heavy objects or bags, a person may find it hard to interact with devices, for example, texting one-handed while holding groceries.

General Design Tips

1. **Keyboard over mouse navigation:**
   Shaky hand movements and limited range of motion make mouse use unreliable, especially when targeting small buttons or moving quickly across the screen, leading to strain and frequent errors.

   Design tip: Ensure all functionality is accessible via keyboard, such as Tab to navigate, Enter/Space to interact. Maintain a logical tab order and avoid focus traps.

2. **Large and easy-to-target controls:**
   Small clickable areas are hard to activate accurately with limited dexterity, often leading to missed or accidental clicks, especially when using touch devices or a shaky pointer.

   Design tip: Design buttons and interactive elements with a minimum touch target of 44×44 px. Increase spacing between adjacent clickable items to reduce accidental activation.

3. **Simple interactions over complex gestures or shortcuts:**
   Multi-step gestures, such as pinch-to-zoom, swipe, or long press, and complex keyboard shortcuts, such as Ctrl + Alt + Shift, can be physically challenging or impossible for some users to perform.

   Design tip: Simplify touch interactions by offering accessible button alternatives. Use single-key commands or customizable shortcuts where possible.

4. **Interfaces that reduce fatigue:**
   Poorly designed flows with long navigation paths or excessive interactions increase physical effort and cause fatigue, especially during extended or repetitive use.

   Design tip: Prioritize task efficiency by minimizing steps. Provide shortcuts and customizable UI settings (e.g., skip links, collapsible sections) to reduce effort and preserve user stamina.

5. **Visible focus cues for orientation:**
   Lack of visible focus indicators makes it difficult for keyboard users to track their position, increasing the risk of activating the wrong element or losing their place in the interface.

   Design tip: Always show a clear, high contrast focus outline for all interactive elements. Ensure the focus order follows a logical, predictable pattern.

6. **Customizable input and interaction settings:**
   Standard input speeds, such as key repeat rate or pointer speed, may not suit all users, and lack of control over scaling or input behavior can increase effort and reduce usability.

   Design tip: Provide adjustable settings such as pointer speed, key repeat rates, and scaling. Allow users to personalize interaction to match their physical abilities.

There are other factors that impact users with physical restrictions. Fixed screen orientations can force uncomfortable positions for those with restricted neck or limb movement. Limited input controls, such as the lack of integrated voice support, and poorly designed flows with long navigation paths or excessive interactions, can increase physical effort and fatigue, especially during extended or repetitive use.

To address these issues, designs should:

- Support flexible screen orientations, ensuring responsiveness in both portrait and landscape modes.
- Ensure compatibility with speech or voice recognition and other assistive technologies.
- Prioritize task efficiency by minimizing steps and offering shortcuts or customizable UI features that reduce physical effort.

Annotations

Interaction Annotations

This checklist helps designers to remember the important aspects of accessibility that should be addressed in the design phase.

Interaction Experience Checklist

- [ ] Input Mechanism
- [ ] Focus Order
- [ ] Initial Focus Position
- [ ] Focus Restore
- [ ] Skipping Group
- [ ] Shortcut
- [ ] Trigger
- [ ] Motion Alternative
- [ ] Minimum Target Size
- [ ] Journey

Input Mechanism

Understand how users trigger actions and navigate pages using a mouse, keyboard, and touch devices. Follow established patterns to define keyboard interactions and navigation within components.

Component Variants

Input Mechanisms - Touch Actions

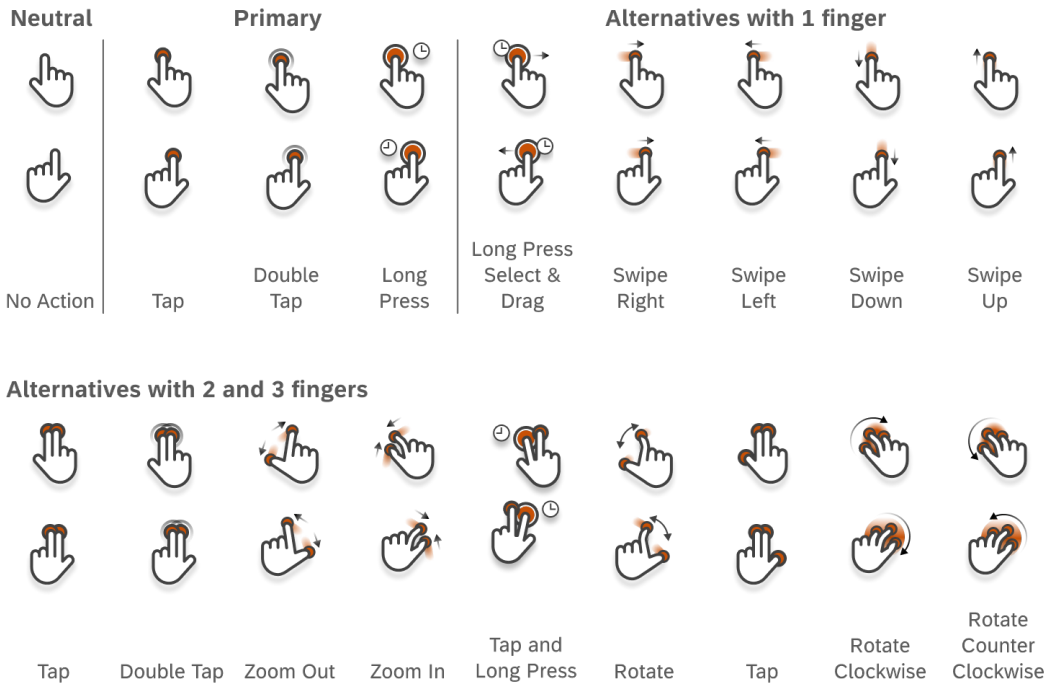


Figure 48: Touch actions performed with the right and left hands

Input Mechanisms - Keyboard Keys

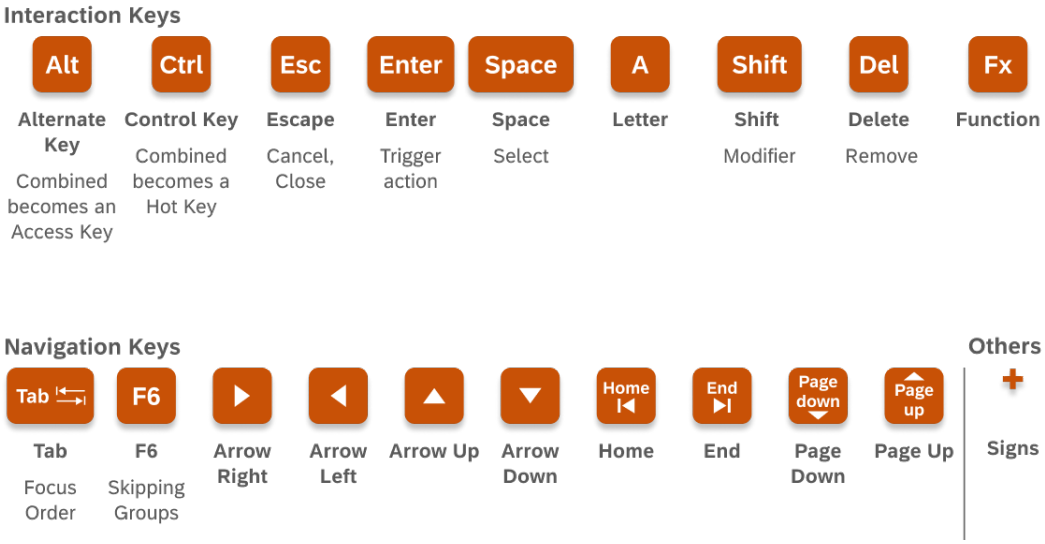


Figure 49: Conventional keyboard keys

Input Mechanisms - Mouse Actions

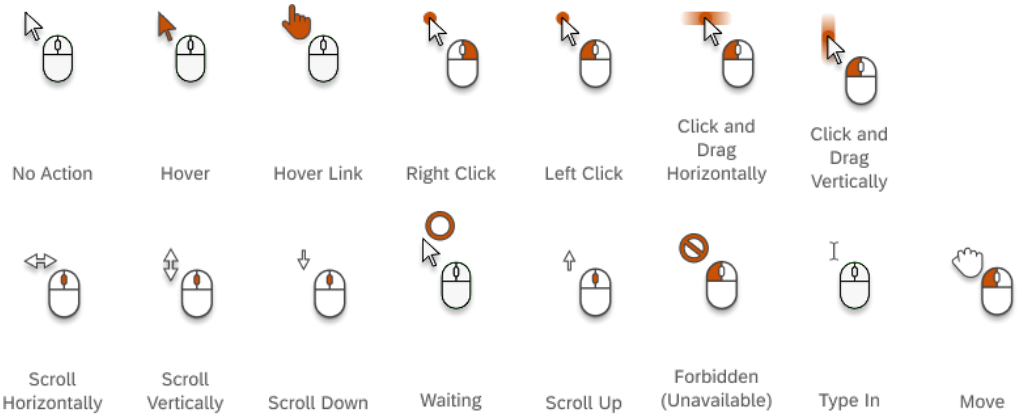


Figure 50: Mouse actions

About Input Mechanisms

Input mechanisms refer to the tools and methods users employ use to interact with digital interfaces. These can include a wide range of technologies, such as voice input, eye tracking, or assistive switches, but this section focuses on the most common: mouse, touch, and keyboard. Each mechanism presents unique interaction patterns and accessibility considerations.

Designing with all three in mind ensures users can navigate, operate, and complete tasks regardless of their preferred or required input method. This

approach benefits not only users with disabilities but also those facing situational or device-based constraints, such as using a touchscreen with gloves or navigating solely by keyboard due to motor limitations.

A mouse is a typical desktop device for interaction and navigation, and it can also extend to mobile capabilities. It is one of the simplest devices to interact with visual elements due to its limited controls and combinations.

Touch input is a standard on mobile devices and tablets. It can become complex when multiple gestures or key combinations are needed to execute a shortcut.

The keyboard is essential for interaction and navigation, and it similarly extends mobile capabilities. Since many users rely on the keyboard alone, all actions available by mouse or touch must also be accessible with keyboard shortcuts or commands. Designing for keyboard interaction involves selecting keys that allow users to act quickly (e.g., shortcuts) and navigate controls smoothly (e.g., arrows keys to move, Enter to open, Esc to close).

Following common keyboard patterns helps users leverage existing knowledge. For example, Enter or Space keys activates links, buttons, and actions such as saving, canceling, editing, or checking a box. The F4 key opens select lists, and ESC closes dialogs, dropdowns, or popovers and returns focus to the trigger element.

Keyboard Keys

Figure 51: Alt key keyboard annotation indicates expected keyboard support patterns in the designed interaction

Mouse Patterns

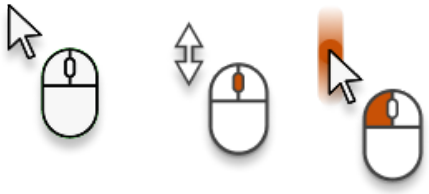


Figure 52: Mouse annotations symbolizing no action, vertical scrolling, and click-and-drag vertically, showing expected mouse patterns in the designed interaction

Touch Patterns

Figure 53: Touch annotations symbolizing long press and zoom out showing expected touch patterns in the designed interaction

Pointer Gesture & Pointer Cancellation

According to WCAG 2.2 Success Criteria 2.5.1 (A) Pointer Gestures, design should support a primary single-pointer gesture before offering alternatives that require multiple fingers or complex paths. A single point of contact should be the primary interaction pattern.

The goal is to enable users to operate touchscreens with one finger and minimal gestures. In other words, “all functionality that uses multipoint or path-based gestures for operation can be operated with a single pointer without a path-based gesture, unless a multipoint or path-based gesture is essential."(WCAG 2.2, Understanding SC 2.5.1)

This requirement recommends that all functionality requiring complex pointer gestures, such as swiping, pinching, or multi-finger input, must also

be operable through simple, single-pointer alternatives. This approach benefits users who cannot perform precise or coordinated gestures due to mobility or motor impairments, allowing them to still fully interact with the interface.

Complex gestures may be difficult or impossible for users with limited dexterity, tremors, or those using assistive devices such as a mouth stick or head pointer. By providing simple alternatives, such as buttons or keyboard commands, designers can make interfaces more accessible and inclusive.

On the other hand, WCAG Success Criteria 2.5.2 Pointer Cancellation (A) ensures that actions triggered by pointer input, such as mouse clicks, screen taps, or stylus presses, are intentional and not accidental. To meet this requirement, the interaction must follow one of several safe patterns:

- the action occurs only on release, not on press,
- the action can be canceled by moving the pointer away before releasing,
- the system provides an option to undo or confirm the action, or
- the interaction is essential to the functionality (e.g., in certain game mechanics).

Pointer cancellation protects users, especially those with motor impairments or using touchscreens, from unintentionally triggering critical functions. It improves usability by ensuring that users can interact with interfaces safely and confidently.

Table 8: Different interaction actions and their corresponding input variants

Action	Input Variants
Hover or Focus	
Select or Activate	
Scroll	

Dragging Movement

Creating accessible input mechanisms requires recognizing that not all users are able to interact with digital interfaces through precise gestures or fluid motion. WCAG Success Criteria 2.5.7 Dragging Movements (AA) addresses this by recommending that any functionality relying on dragging, such as reordering, drawing, or sliding, must also offer an alternative method that does not require a continuous drag gesture.

This benefits users with motor impairments or those relying on assistive technologies that do not support dragging easily so they can complete essential actions. For example, instead of requiring users to drag list items to reorder them, Designers should also offer simple tap or keyboard accessible “move up/down” controls.

By accounting for diverse input needs, including touch, mouse, keyboard, and assistive devices, application teams can ensure that all users are able to perform essential actions without unnecessary physical strain or exclusion.

Design Example:

- Not enough: Only allowing users to rearrange a list by drag-and-drop.
- Accessible: Providing an additional “move up/down” button to reorder items without dragging.

Table 9: Different dragging actions and their corresponding input variants

Action	Mouse	Touch	Keyboard
Select and drag to move		Use interactive buttons to select and move to a selectable destination	
Scroll			

Examples

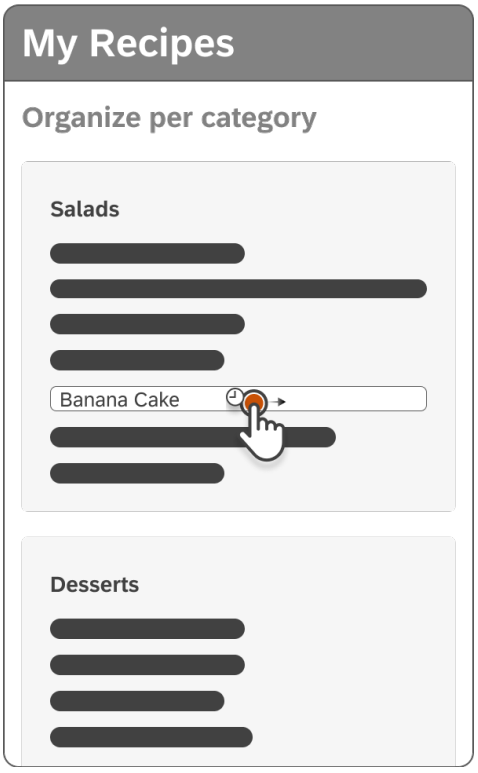


Figure 54: Long press, select, and drag

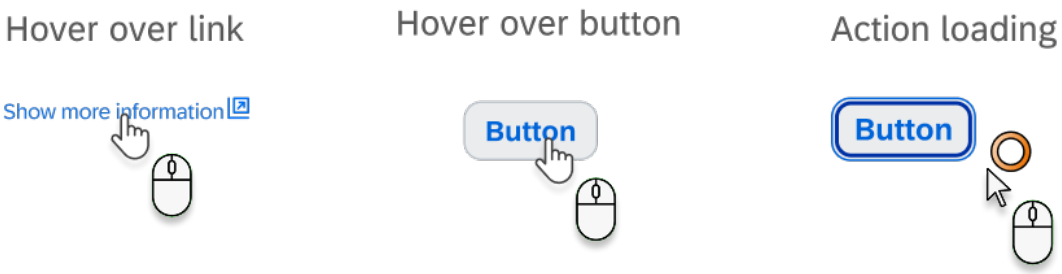


Figure 55: Mouse interactions when hovering over a link, a button, or if an action is loading

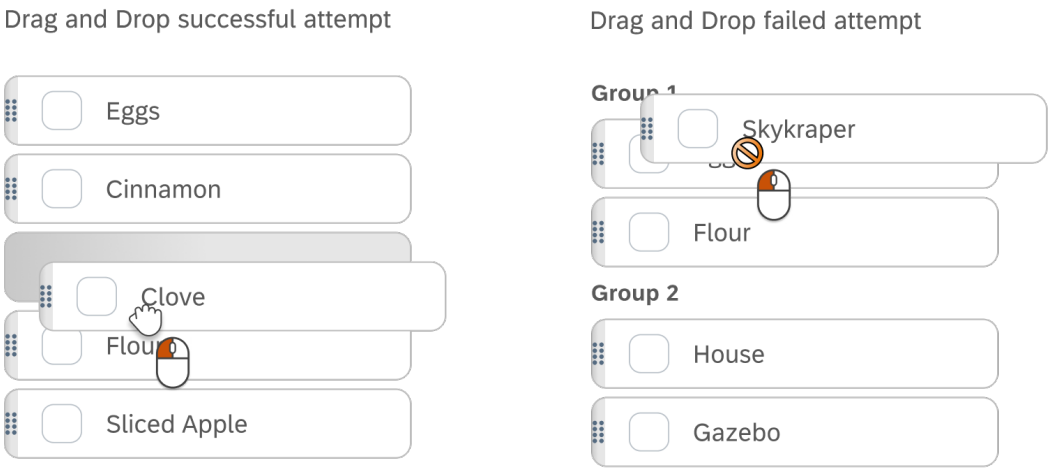


Figure 56: Drag and drop attempt

Figure 57: Mouse cursor when hovering an input field

Shared Benefits

Temporary Disabilities

All users

Users with temporary disabilities, such as a broken arm, eye strain, or post-surgery limitations, also benefit from alternative input methods. For example, a person with a wrist injury may rely on voice input or switch controls, while someone experiencing blurred vision may benefit from screen magnification or keyboard shortcuts. Providing multiple input options ensures that users can continue to interact effectively, regardless of their current physical or sensory condition, supporting broader accessibility and inclusive design.

Without Vision

Screen Reading

Blind users benefit greatly from a range of input mechanisms, especially keyboard navigation, screen readers, and voice commands. The keyboard provides precise, structured navigation without relying on visual cues, while screen readers convert text and semantic structure into spoken output. Voice input offers hands-free interaction, useful on mobile devices or multi-tasking contexts. These mechanisms allow blind users to interact with digital interfaces independently and efficiently when interfaces are properly designed with accessibility in mind.

For Developers

Web Development: Developers should check the relevant online documentation for foundational guidance, such as Developing a Keyboard Interface | APG | WAI | W3C.

Native Mobile Development: Developer should consult the documentation for the SDKs of the given platforms:

iOS:
Support Full Keyboard Access in your iOS app - WWDC21 - Videos - Apple Developer

Android:
Handle keyboard input | Views | Android Developers

Across all technologies, make sure that the visible keyboard focus is fully displayed and not obscured or truncated by other controls or control containers.

References

WCAG 2.2

2.1.1 Keyboard (A)

2.1.3 Keyboard (No Exception) (AAA)

2.4.7 Focus Visible

2.4.11 Focus Not Obscured

2.5.1 Pointer Gestures (A)

2.5.2 Pointer Cancellation (A)

2.5.6 Concurrent Input Mechanisms (AAA)

Focus Order

Define the logical sequence of tab order and focus flow for interactive controls in the layout. Also, identify elements that create focus branching.

Component Variants

Single Tab Stop

Figure 58: Variant indicates interactive elements that receive focus during tab navigation

Inner Tab Stops

Figure 59: Variant with a tab symbol indicates an unknown number of additional control-inner tab stops that are not included in actual order numbering (branched item).

Scope Horizontal, Vertical and Spatial

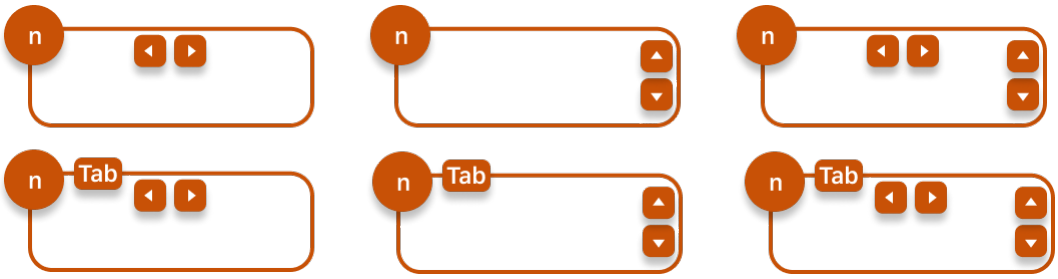


Figure 60: Represents a single tab stop within a branched control scope, where navigation inside the UI element requires additional use of arrow keys to navigate vertically, horizontally, or spatially within the control. A tab symbol may be used to indicate additional Tab stops.

Focus Flow

Figure 61: Indicates the expected focus order flow “left to right” or “top to bottom” (not depicted here) when individual focus order annotations may not be required

About Focus Order

The keyboard focus is a critical accessibility feature that applies exclusively to interactive UI elements, such as buttons, links, or checkboxes. Making this focus visible is vital, as it allows users to identify the element they can interact with, enabling them to trigger actions or explore further to access nested functionalities.

Focus order indicates the sequence of interactive controls. Also called tab stops, focus order is the preferred method for keyboard users to navigate between interactive elements on the screen bi directionally by pressing Tab for forward navigation and Shift + Tab for backward navigation.

Keyboard users perceive focus visually through a consistent style applied to interactive element when it is focused. Blind users, on the other hand, rely on screen reader announcements that describe the control, its current state, and available actions. When the element is focused, the user can activate it using the Space or Enter key.

An efficient keyboard navigation experience relies on predictable tab stops and focus order. Keyboard users can navigate to the next element using the tab key, but when arriving at a complex component, such as a list or table, they often need to use additional keys, such as arrows or specific shortcuts, to navigate within that component before continuing.

Designers should define the logical sequence of interactive controls by leveraging the visual layout and structure used to display UI controls and groups of elements. To document focus order in the design documentation, use the element that best represents the interactive UI control on the screen.

Simple tab stops are depicted using an outlined orange circle and a number that represents its place in the tab order. Additional tab stops inside the control are represented using a line and a tab symbol to indicate their presence.

A branched item is a new starting point, a switch in strategy to continue navigation and interaction within a complex component visible on the screen, such as a tab, a breadcrumb, tables, a combo box, or a list inside a popover. After reaching the branched control using the Tab key, the user navigates within it using Arrow keys and Home/End keys and eventually moves out using the Tab key.

SAP Figma Plugin

The SAP Figma Plugin “Focus Order” helps designers annotate their screens by identifying the relevant focus order sequence.

Examples

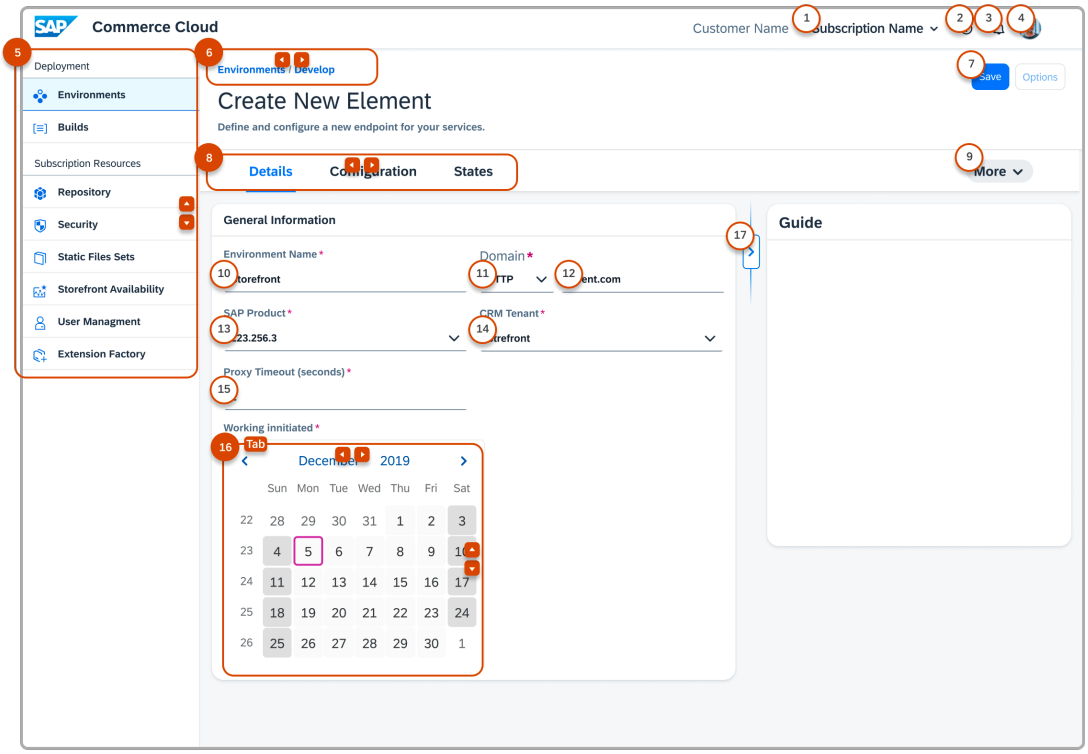


Figure 62: Focus order sequence includes components with inner arrow key navigation, such as 5, 6, and 8. Note that disabled elements are not included in the sequence notation, for example “Options” button after 7. Additional inner tabs of unknown number within components are indicated by an additional “Tab” symbol (16), with numbering continuing the next control annotation (17).

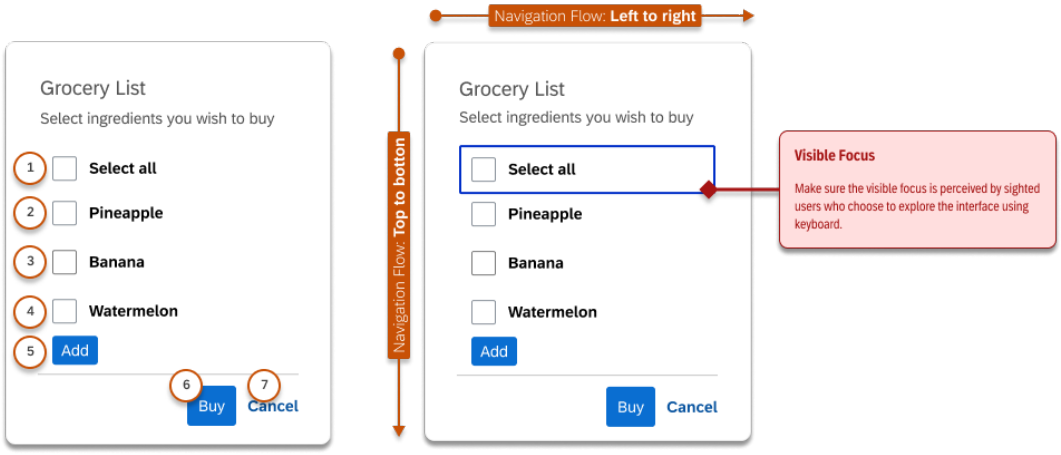


Figure 63: Focus order flow indicates the direction of tab order. The visual focus blue line border shows which element on the screen is currently active.

Shared Benefits

Temporary Disabilities

All users

Tab stops or focus order are not just for accessibility, they are also used by many people who rely on the keyboard to move quickly through forms and interfaces, making navigation faster and more efficient without needing a mouse.

Without Vision

Screen Reading

For blind users using screen readers, tab stops and focus order are essential for navigating and understanding content in a logical, predictable sequence ensuring they do not miss important fields, actions, or instructions.

For Developers

For Web Development, tab order can be programmed using two different concepts:

1. By Document Object Model (DOM) sequence plus nesting in layout groups

```
<div>
    <div id="group1"
class="vertical">
          <button>A</button>
          <button>B</button>
          <button>C</button>
    </div>
    <div id="group2"
class="vertical">
          <button>D</button>
          <button>E</button>
    </div>
</div>
```

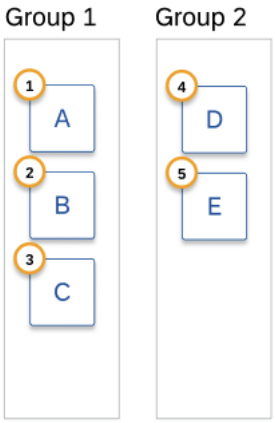


Figure 64: DOM sequence nesting in layout groups

2. By tabindex property

```
<button tabindex="1">Save</button>
<button
tabindex="3">Delete</button>
<button
tabindex="2">Cancel</button>
```

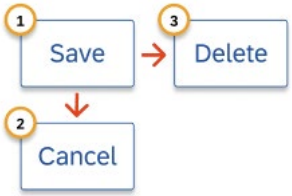


Figure 65: Tab sequence different from default by applying individual tab indices

Notes:

- With tabindex="0" any element can be made to receive keyboard focus. Use this ONLY for interactive elements, with action handlers associated.
- With tabindex="-1" any element can receive focus **only** by code, by JavaScript focus setters
- tabindex > 0 is to be avoided as it can be redundant or may cause confusion

To understand the basics, Developers should also check online documentation that gives an introduction (such as Developing a Keyboard Interface | APG | WAI | W3C) and then study the best practices for applying correct focus order in WCAG 2.2 Success Criterion 2.4.3 Focus Order.

In native mobile development, platform dependent procedures must be applied:

iOS: Focus order depends on the structure of the view.

```
var body: some View {
VStack {
TextField("First name", text: $firstName)
.modifier(InputModifier())
.focusable()
TextField("Last name", text: $lastName)
.modifier(InputModifier())
.focusable()

HStack(alignment: .top) {
Button(action: {}) { Text("Save") }
.accessibility(label: Text( "Save user data and continue
order form"))
Spacer(minLength: 10)
Button(action: {}) { Text("Cancel") }
.accessibility(label: Text("Cancel order form"))
}
}
}
}
```

See [Building layouts with stack views | Apple Developer Documentation]

Android: Implement tab navigation by layout

```
<RelativeLayout ...>
<Button
android:id="@+id/button1"
android:layout_alignParentTop="true"
android:layout_alignParentRight="true"
android:nextFocusForward="@+id/editText1"
... />
<Button
android:id="@+id/button2"
android:layout_below="@id/button1"
android:nextFocusForward="@+id/button1"
... />
<EditText
android:id="@id/editText1"
android:layout_alignBottom="@+id/button2"
android:layout_toLeftOf="@id/button2"
android:nextFocusForward="@+id/button2"
... />
...
</RelativeLayout>
```

Implement directional navigation by property, such as Directional Pad (D-Pad) or arrow keys.

```
android:nextFocusUp
android:nextFocusDown
android:nextFocusLeft
android:nextFocusRight
```

To understand the basics, also consult the documentation for the SDKs of the given platforms:

iOS:

Support Full Keyboard Access in your iOS app - WWDC21 - Videos - Apple Developer

Android:

View | API reference | Android Developers

References

WCAG 2.2

1.3.2 Meaningful Sequence (A)

2.4.3 Focus Order (A)

2.4.7 Focus Visible (AA)

Initial Focus Position

Specify the UI element that receives initial focus when the page loads or when entering a skip block or a control.

Component Variants

Page or Dialog

Figure 66: Indicates which interactive UI element on the screen will receive focus when page loads or a dialog is displayed

Skipping Group

Figure 67: Indicates which interactive UI element inside a skipping group will receive focus when exploring groups on the page [F6]

Control

Figure 68: Indicates which part of a focusable control, simple or complex, will initially receive focus

Focus

Figure 69: Shows a plain focus annotation, useful if you just want to display a focus ring somewhere in your design

About Initial Focus

Initial focus is the position highlight of a single interactive UI element, typically by a visual outline to show which element is active. The initial focus annotation informs the Developer which interactive element should receive focus to prepare the keyboard experience.

When the system communicates the current focus, keyboard users feel confident navigating the page and interacting with elements. Focus serves as a visual cue that an interactive element is active and ready to respond to input, whether from a keyboard or other input devices. This state ensures that users can navigate and interact with the interface seamlessly, enhancing usability and accessibility. In essence, focus represents the state in which an interactive UI element is primed to receive user input, facilitating efficient and inclusive digital experiences.

The initial focus is given to the most essential element of a workflow in several contexts:

- On Page Load: When a user arrives at a new page, initial focus is set on the first most important interactive UI element to allow the user to start the and proceed with keyboard navigation using the Tab key.
- During Skipping Group: F6 is a keyboard shortcut that moves the focus between regions or groups of an application such as the address bar, tabs, the content area, or panels. For efficient group skipping define where initial focus should be to support users in skipping large sections.
- Controls and Complex Components: Initial focus is also useful for groups of components such as tables, toolbars, lists, carousels and calendars. Focus is set on the selected item for single selectable items from a list, menu, or tabs. For multi-selectable items, the focus goes to the first selected item.

The most essential element of a workflow marked as initial focus is also the first element announced when a blind user arrives on a page or performs skipping group navigation using the keyboard shortcut F6.

The concept of initial focus serves as a visual cue to that communicates "You are here" on a page, providing essential orientation for users, particularly those who rely on keyboards to navigate digital products.

The initial or default focus position on the screen and when entering groups is typically set at the framework level. However, in many UI technologies, applications can override the default focus position and assign it to a logical starting point on the screen.

Focus Appearance

The WCAG Success Criterion 2.4.13 Focus Appearance (Minimum) (AA) ensures that the keyboard focus indicator is clearly visible and meets minimum design requirements, allowing users to reliably see which element is currently focused.

The appearance of the focus is typically shown as an outline, border or highlight around the focused element. It is automatically provided by web

browsers for interactive elements. Default style and configuration options may vary depending on the browser vendor.

In contrast to that, the appearance of the visual focus on controls in a brand theme is defined by a design system to align with the general theming AND to comply with WCAG Success Criterion 2.4.13. This ensures that focus indicators meet both visual style and contrast requirements.

Focus Visibility

While the WCAG Success Criterion 2.4.7: Focus Visible (AA) requires a visible focus indicator to appear whenever a user navigates to an element using the keyboard, the WCAG Success Criterion 2.4.11 Focus Not Obscured (Minimum) (AA) requires that when a user tabs to an element, at least part of it must be visible on the screen and not hidden behind sticky headers, modals, or overlays. The key concerns are:

1. Visibility of Focus: Ensure that the keyboard focus is clearly visible to users. Is the focus style visible, for example, as a border, outline, or highlight?
2. Access and Interaction: Ensure users can access and interact with focused elements without confusion or being lost. Is the focused element obstructed or off-screen?

If you make the entire element fully visible on focus without requiring the user to scroll or adjust the view, you are fulfilling the WCAG Success Criterion 2.4.12 Focus Not Obscured (Enhanced) (AAA).

Examples

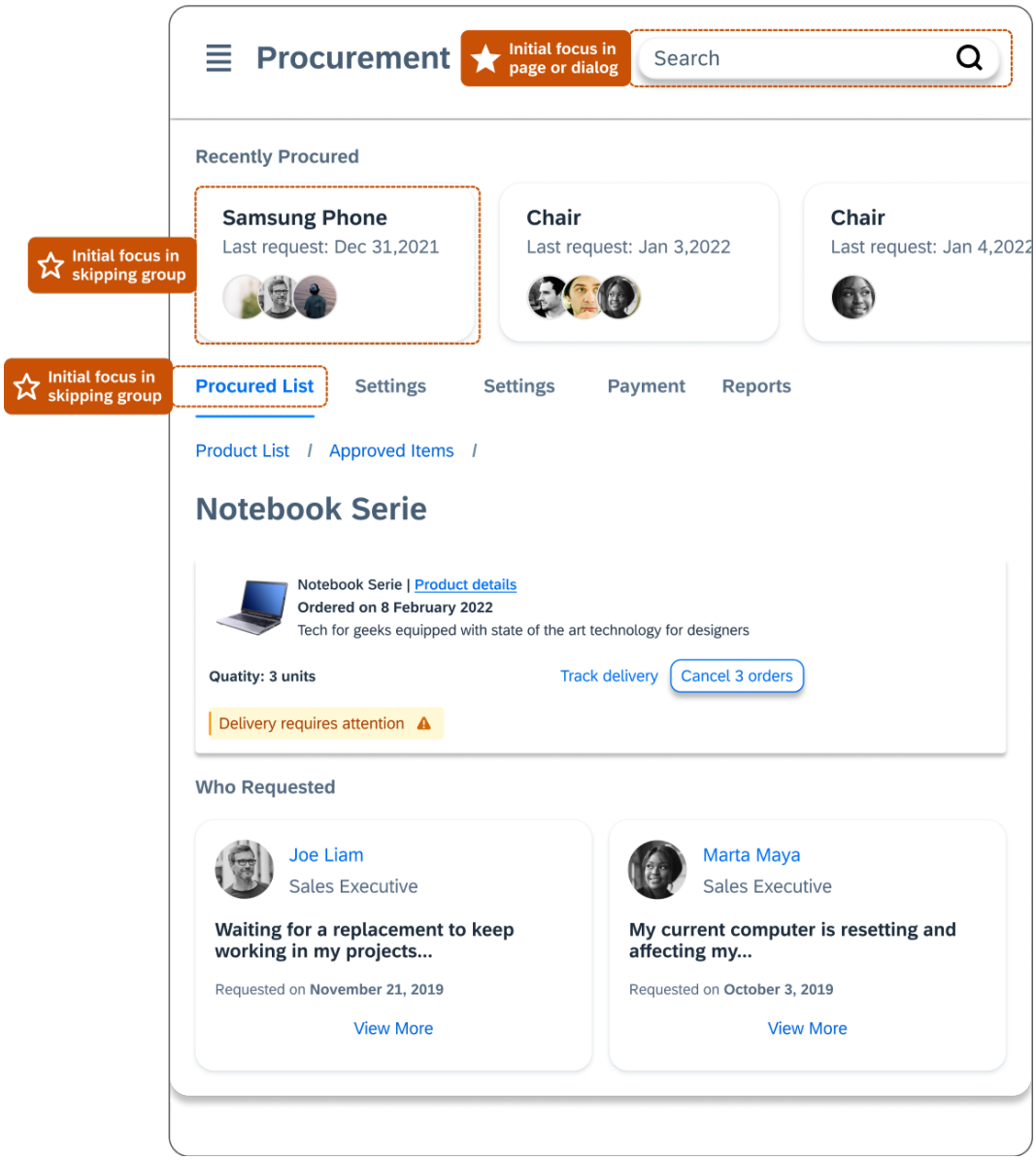


Figure 70: Initial focus position on page load and when navigating using skipping groups

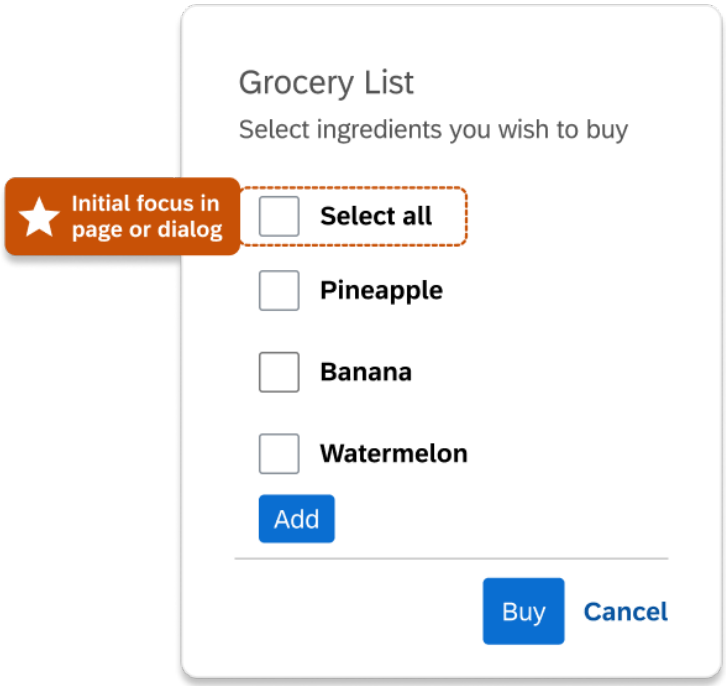


Figure 71: Initial focus position in a dialog

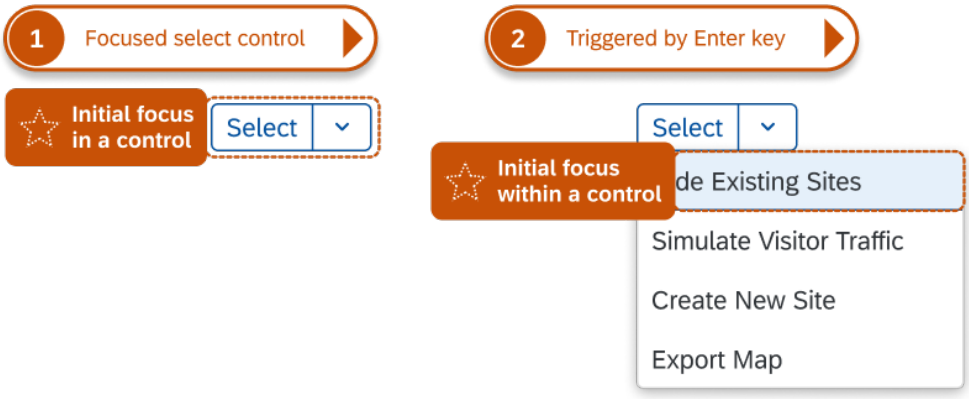


Figure 72: Initial focus in controls. Menu Button Split Mode - Menu Opening

Shared Benefits

Temporary Disabilities

All users

Thoughtful initial focus design helps users stay oriented, act efficiently, and remain confident while navigating complex interfaces. When focus starts in the appropriate location, such as the first actionable field in a form, or the close button of a modal, it not only saves time but reduces physical strain and cognitive effort for users, especially in enterprise or productivity environments where efficiency is key.

For Developers

Web Development: Developers should consult relevant documentation for setting focus on elements, such as:

- HTML autofocus global attribute - HTML | MDN
- HTML Element: focus() method - Web APIs | MDN

Native Mobile Development: Consult the relevant documentation for the SDKs of the given platforms to set the focus and check the following code snippets:

iOS: Use `@FocusState` to track or set a focus position.

```
enum Field: hashable {
case email
case password }

@FocusState private var focusedField: Field?
TextField("Email", text:$email).focused($focusedField,
equals: .email)
SecureField("Password", text:$pwd).focused($focusedField,
equals: .password)

.onSubmit {
  if !isEmailValid {
    focusedField = .email
  }
}
```

See also:

- https://developer.apple.com/videos/play/wwdc2021/10023/
- https://developer.apple.com/documentation/swiftui/focusstate

Android: Programmatically set focus on a view when an activity is opened.

```
private EditText mEdit;
mEdit = (EditText)findViewById(R.id.myEdit);
mEdit.requestFocus();
```

See also:

- https://developer.android.com/reference/android/view/View#requestFocus(int)
- Change focus behavior | Jetpack Compose | Android Developers

References

WCAG 2.2

2.4.7 Focus Visible (AA)

2.4.11 Focus Not Obscured (Minimum) (AA)

2.4.12 Focus Not Obscured (Enhanced) (AAA)

2.4.13 Focus Appearance (AAA)

Focus Restore

After the user completes a task, keyboard focus returns to the trigger element, or to the next logical UI element if the trigger is no longer present.

Component Variants

Focus Restore to Trigger

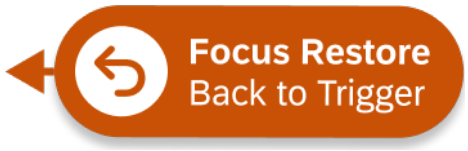


Figure 73: Indicates the UI element that triggered an action and shows where the focus should return.

Focus Restore to Next Logical Element

Figure 74: Indicates the next logical UI element should receive focus at the end of the flow.

About Focus Restore

Designing predictable and inclusive interaction flows is only complete when focus return is considered. Focus management is a frequent source of accessibility issues, particularly for keyboard and screen reader users. One critical, yet often overlooked, aspect is focus restore which is the practice of returning keyboard focus to a logical and expected location after a task or interaction is completed. Proper focus restore ensures that users do not lose their place in the interface, especially after dismissing dialogs, completing forms, or removing elements from a list. Designing and annotating where focus should land after a flow demonstrates careful attention to keyboard support and helps reinforce this requirement during implementation.

This annotation evolved from the concept of predictable context, previously featured in the Foundation chapter of the Accessibility Design Tools by SAP first edition. As the toolkit matured, the predictable context concept was split into two distinct annotations: focus restore, which

addresses the interactive experience, and wayfinding and orientation, which supports users in understanding transitions between pages and dialogs. Together, these annotations help ensure users maintain context and continuity throughout an interaction.

Designing for focus restore requires an understanding of the predictability of user flows. However, there are scenarios where simply restoring focus to the original trigger may not be possible or logical, for example, when the trigger element, such as a deleted list item, no longer exists, or when the item has been repositioned in a way that breaks continuity. In such cases, designers must plan alternative destinations for restored focus, such as the closest logical sibling element, the updated container, or a clearly labeled notification.

By planning for focus behavior early and documenting it through annotations, design teams reduce the risk of disorientation and accessibility issues. Focus restore is more than a technical fix; it is a user-centered practice that supports continuity, efficiency, and confidence for users who rely on keyboard navigation.

Focus Restore: Back to Trigger

When an interaction is completed, such as dismissing a dialog, popover, or overlay, focus should return to the element that originally triggered the interaction. This annotation is used to indicate a pattern where the trigger remains relevant and present in the UI, and the user journey expects a return to the point of origin. For example, closing a modal that was opened by an Edit button within a list item should return focus to that same button. This helps users stay oriented and maintain continuity in their workflow.

Design tip: Use when the trigger remains visible, unchanged, and still represents the next logical step in the user workflow. Doing so reinforces predictability and supports reactivation or re-entry.

Focus Restore: Back to Next Logical Element

In some cases, returning focus to the original trigger is not appropriate, especially if the trigger was deleted, moved, or is no longer relevant. In these situations, focus should move to the next logical element in the workflow of the user. This annotation applies when the action modifies the structure of the page, such as deleting a row in a table. Rather than returning focus to a missing or moved trigger, Designers should identify the most relevant next step in the task flow, such as the next row in a list, a message area, or an adjacent control. This approach prevents disorientation and helps users maintain a smooth workflow.

Design tip: Designers must understand the journey of the user and intended workflow. They should anticipate the next likely action of the user and annotate the interface accordingly to maintain momentum and minimize confusion.

Examples

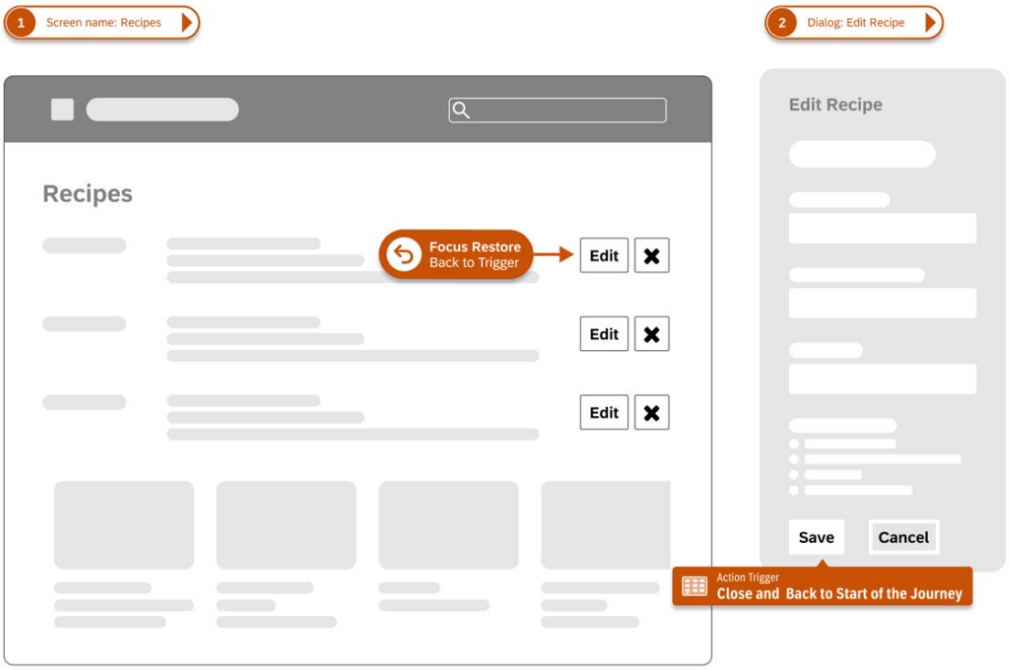


Figure 75: Focus restore and action trigger annotations in an Edit item use case

Figure 76: Focus restore annotation applied on the trigger UI element after the displayed dialog closes.

Shared Benefits

Temporary Disabilities

All users

Older users may experience a combination of physical, cognitive, or sensory limitations. Clear, predictable navigation, including focus restore, supports confidence and independence in digital environments.

Without Vision

Screen Reading

Blind users rely entirely on focus order to understand where they are in an interface. Without proper focus restore, they may be end up at the top of the page or in an unrelated area, causing confusion and disorientation. Predictable focus behavior helps maintain context and mental mapping of the UI.

Vision and Color Limitation

Visuals

Users with low vision often use screen magnifiers or high zoom levels. If focus is not restored predictably, they may lose their visual context, especially if the new focus is off-screen or unexpected. Proper focus placement avoids unnecessary panning or hunting for the next step.

Limited Cognition

Cognitive

Predictability and continuity are crucial for users who may struggle with memory, attention, or processing complex interactions. Sudden or illogical focus jumps can increase cognitive load, leading to task abandonment. Focus restore supports smoother interaction flows, reducing frustration.

For Developers

1. Review all information from the previous chapters as foundational knowledge.
2. Apply the focus restore concept according to the use case:
    a. Previously focused element is still visible: Return focus to that element. Typical case: a button opens a dialog; when the dialog closes, focus returns to the button.
    b. Previously focused element has completely disappeared: This can occur during page navigation. Record the last focused element and, when navigating back, restore focus to the element that was focused on before navigation, if it is available.

References

WCAG 2.2

3.2.1 On Focus (A)

3.2.3 Consistent Navigation (AA)

Skipping Group

Allow users to skip directly to the main content or bypass large blocks of repeated content across multiple pages, improving user efficiency.

Component Variants

Skipping Group

Figure 77: Identifies a group of UI elements that can be skipped in the tab order with a designated keyboard hotkey.

About Skipping Group

Skipping groups are a mechanism described by WCAG to bypass blocks of content that are repeated across multiple web pages. Guideline 2.4.1 Bypass Blocks (A) reminds designers to provide a way for users to skip repeated content, like navigation menus, so they can quickly reach the main content.

People with certain disabilities have difficulty reaching the main content of a page efficiently. For example, screen reader users visiting several pages on the same site can avoid listening to repeated headings and navigation links (imagine 200 words less to listen to) and reach the main content directly. This reduces the number of announced words, allowing users to navigate faster and feel more productive.

The same efficiency applies to keyboard users, who can reach the main content using fewer keystrokes, avoiding unnecessary repetition. Skipping to the main content can also prevent severe physical strain for some users. Users of screen magnifiers benefit as well, since they avoid long navigation paths through headings or repeated blocks of information when entering a new page. Quick access to the main content allows these users to locate the main body of the page without manually searching.

We propose skipping group annotation to define regions that can be bypassed. These groups improve navigation efficiency by allowing users to skip repeated content and access the main content more quickly.

Annotate a skip block to define a group of contextual information. Arrange the elements in a logical order to guide users through skipping navigation in the same way they would perceive the visual layout hierarchy. Ensure that users understand the hierarchy across groups during navigation. The

direction of skipping navigation is useful on complex layouts. Anticipate the initial focus when a user lands on a new group by using the focus placement annotation.

One mechanism to bypass blocks is by using the F6 key. Reverse navigation uses SHIFT + F6 keys to return the user to the previous skip block.

Examples

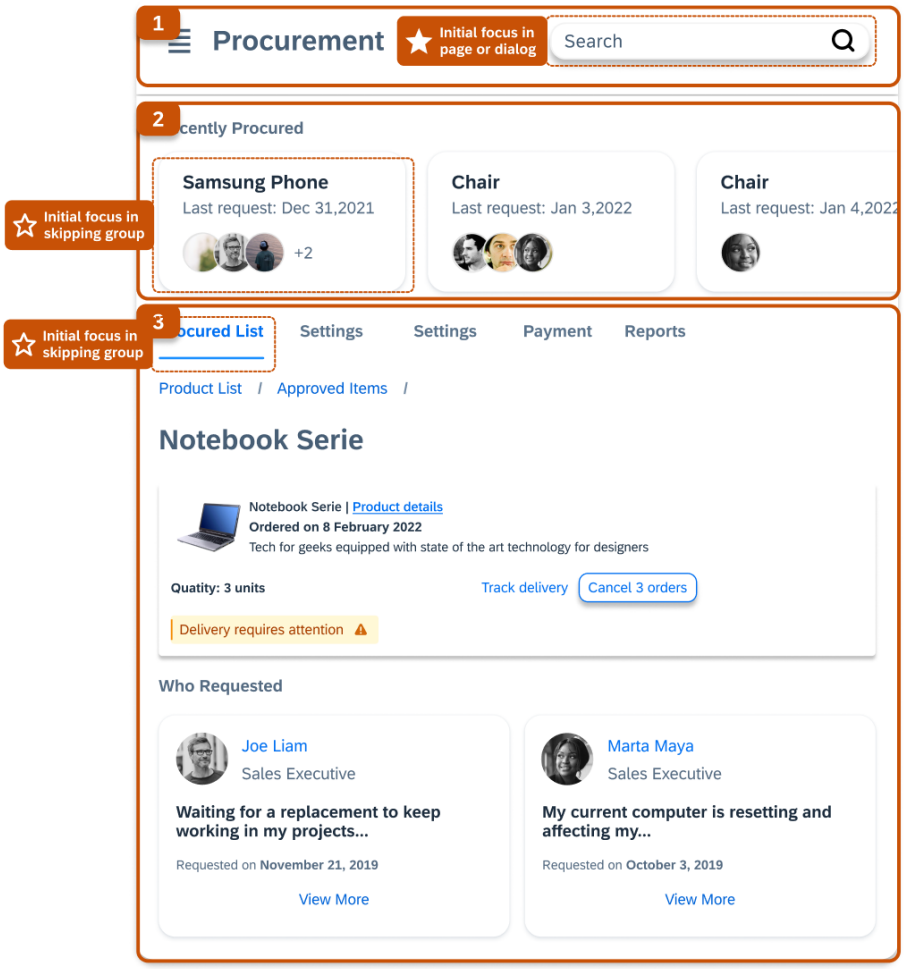


Figure 78: Skipping group sequence with initial focus positions on a procurement page.

Shared Benefits

Temporary Disabilities

All users

Expert users benefit from the skipping groups annotation because it supports faster navigation through large or repetitive content areas. By enabling shortcuts, such as utilizing the F6 key, to jump between major interface regions, such as toolbars, content areas, or side panels, expert users can quickly reach their target without unnecessary tabbing. This enhances workflow efficiency in complex applications.

Without Vision

Screen Reading

Blind users also rely heavily on group-skipping functionality to avoid navigating through every single item in a dense interface. When screen readers announce the ability to skip to specific regions, it helps blind users maintain orientation and reduce the time spent moving sequentially through unrelated elements. This improves usability, supports independence, and aligns with the structure they expect from well-designed, accessible interfaces.

For Developers

Web Development: Define a concept of skippable groups for framework components. In this context, groups, refer to large, complex controls such as tab strips and tables, or control containers such as panels and forms.

Technically, implement this by attaching a JavaScript `keyup` listener and calling `el.firstElementChild.focus()` while preventing event bubbling as shown below:

```
document.addEventListener('keyup', (e) => {
  // forward navigation  if (e.keyCode == 117 ) { // f6
keycode
    e.preventDefault();
    e.stopPropagation();
```

```
    var el = findNextSkipContainer();
    el.firstElementChild.focus(); // default, can be
overridden
  }
}, false);
```

Native Mobile Development: Consult the documentation of the SDKs of the given platforms to set focus, and review the following code snippets:

iOS:

Options to structure a screen:

- Use the heading trait to structure the screen, which improves voice over navigation.
- Use `focusSection` to define a group, which enhances keyboard and remote control navigation: https://developer.apple.com/documentation/swiftui/text/focussection()

```
import SwiftUI

struct ContentView: View {
    @State private var isEnabled = false

    var body: some View {
        VStack {
            Button("Print message") {
                print("Hello World!")
            }

            Button("Delete message") {
                print("Message deleted.")
            }
        }.keyboardShortcut("p")
         .focusSection()
    }
```

- Use app shortcuts to move focus on your own, which improves keyboard control

Android:

For entire app and its activities, declare a custom skipping key using `keyUp`:

```
public static final int KEYCODE_F6 = 136;
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
switch (keyCode) {
case KeyEvent.KEYCODE_F6:
skipToNextPanel();
return true;
default:
return super.onKeyUp(keyCode, event);
}
}
```

See Handling Keyboard Actions | Android Developers

References

WCAG 2.2

2.4.1 Bypass Blocks (A)

Shortcut

Give expert users shortcuts, hotkeys and access keys, to enhance productivity and increase work efficiency.

Component Variants

Hotkey

Figure 79: Associate a Hotkey to a button to enable users to trigger the action using a shortcut.

Access key

Figure 80: An access key assigned to a button, enabling users to navigate to it with a keyboard shortcut.

Transfer Focus - Navigation

Figure 81: Inform when a group control requires a switch navigation to continue exploring element within a container. Adding commonly used shortcut to switch navigation within groups speed annotations.

About Shortcuts

Shortcuts are keyboard combinations that enhance productivity by providing users a faster way to access functionality. They can be implemented as hotkeys or access keys.

Hotkeys

A hotkey allows users to trigger essential and frequently used functions from anywhere in the interface. Hotkeys typically use the CTRL + letter key combinations.

For example, users can run a search by pressing the Enter key while in the search field or save data using the combination CTRL+ S.

Inform all users about applied hotkeys in the UI. Show them in the tooltip of a button or as a part of a menu item that triggers a respective action.

Access keys

An access key improves navigation efficiency by making a control available at anytime, anywhere in the UI. Access keys typically consist of ALT + letter or ALT + number key combinations. Unlike hotkey, access keys do not trigger essential and frequently used controls; they just move the focus to the control bound to the key.

Available access keys are usually indicated by underlining the first character of the control label associated with the access key.

Transfer focus for inner navigation

When navigating a web interface with the Tab key, users typically move between interactive elements such as buttons, links, and form inputs. However, when they encounter group components such as lists, menus, or grids, the navigation behavior often changes. To avoid excessive tabbing and to improve efficiency, these groups commonly support arrow key navigation.

For example, in a list or menu, users can move vertically using the Up and Down arrow keys, allowing them to explore content quickly and logically without losing their place in the interface. This strategy preserves logical flow and reduces unnecessary keystrokes.

Complexity increased when list items or group elements contain nested interactive controls such as buttons, dropdowns, or editable fields. In these cases, users need a way to intentionally enter and exit the inner content.

A common solution is to use shortcuts such as F2, which transfers focus into the item, enabling interaction without disrupting the outer navigation flow. This ensures that users can browse lists or menus at a high level and dive deeper only when needed, improving efficiency for keyboard users and reducing cognitive load by minimizing unintentional focus jumps.

Additional shortcuts such as F4 to expand items or F6 to skip between regions or groups further support navigation in complex interfaces. These shortcuts allow users, especially those with limited mobility or who rely on assistive technologies, to navigate quickly and predictably across the UI.

This structured and layered navigation model aligns with accessibility best practices by maintaining logical focus flow, preserving user orientation, and providing greater control and flexibility when interacting with nested or grouped components.

Examples

Figure 82: Hotkey and transfer focus annotations applied to mark the respective controls in a UI

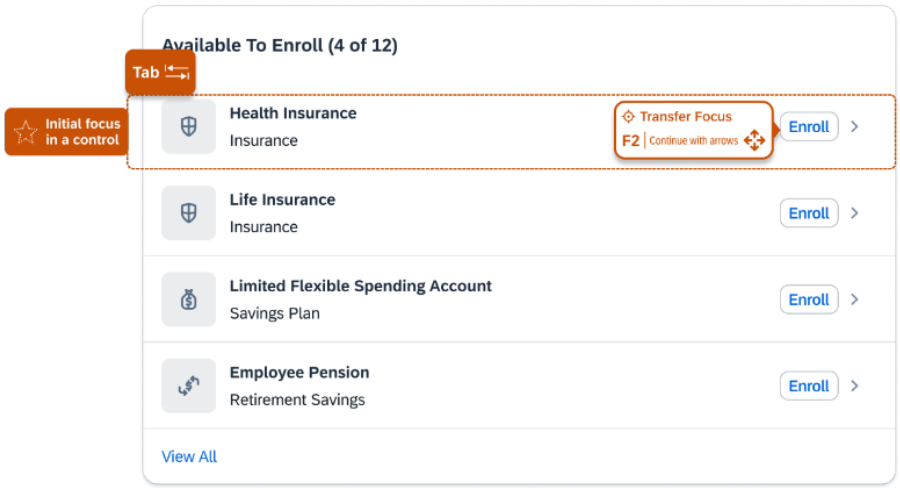


Figure 83: Example of transfer focus, initial focus, and tan annotations within a control

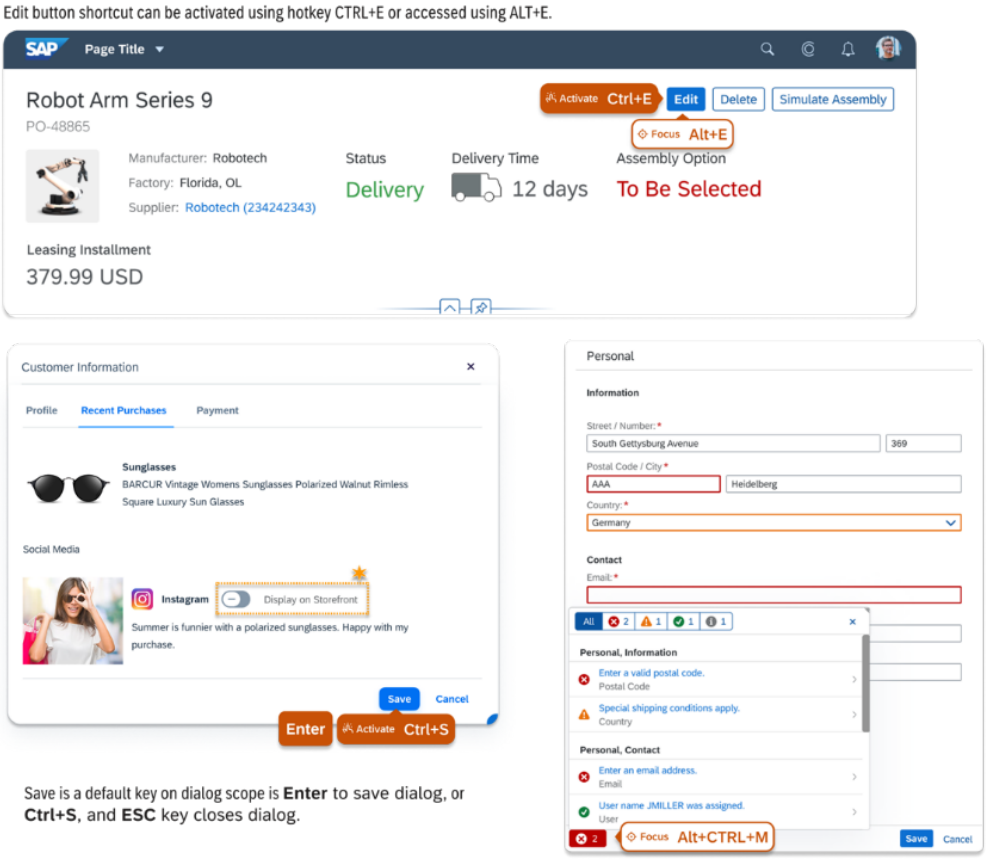


Figure 84: Examples of hotkey and access key annotations

Shared Benefits

Temporary Disabilities

All users

Expert users benefit from shortcuts because they enable faster, more efficient interaction with digital interfaces. By bypassing multiple clicks or tab stops, shortcuts streamline interaction and enhance productivity, especially in complex or repetitive workflows. This allows experienced users to stay in flow and complete tasks with greater speed and accuracy.

Without Vision

Screen Reading

Blind users rely on shortcuts to navigate interfaces without having to navigate through every element sequentially. Shortcuts provide direct access to key regions or actions, making the experience more efficient and predictable when using screen readers. This reduces cognitive load and improves orientation, especially in dense or dynamic interfaces.

For Developers

Web Development:

Hotkeys: Implement using `keyup` listener with event bubbling prevented

```
document.addEventListener('keyup', (e) => {
if (e.ctrlKey &&
String.fromCharCode(e.keyCode).toLowerCase() === 'e') {
e.preventDefault();
e.stopPropagation();
// Do edit stuff...
}
}, false);
```

Note: Register hotkeys and default keys for a view at top-level container. The container collects all defined keys using a subscription model.

Access Keys: Implement using the `accesskey` property.

```
<button accesskey="s"><span
class=”underline”>S</span>ave</button>
```

Note: `accesskey` declarations have a local scope and are managed by the browser.

Native Mobile Development:

iOS: For an example in defining shortcuts, refer to:
https://swiftwithmajid.com/2020/11/17/keyboard-shortcuts-in-swiftui/

```
var body: some View {
HStack {
VStack {
Button ("1") {}
.keyboardShortcut("1", modifiers: [.control])
Button ("2") {}
.keyboardShortcut("2", modifiers: [.control])
Button ("3") {}
.keyboardShortcut("3", modifiers: [.control])
Spacer()
}
.border(Color.white, width: 2)
....
}
}
```

Android Track Shortcuts:

```
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
if(Build.VERSION.SDK_INT>10 &&
event.getAction()==KeyEvent.ACTION_DOWN &&
event.isCtrlPressed()){
String actionType="NONE";
final int keyCode = event.getKeyCode();
switch(keyCode){
case KeyEvent.KEYCODE_C:
actionType = "COPY";
break;
case KeyEvent.KEYCODE_V:
actionType = "PASTE";
break;
```

```
...
}
return true;
}
return super.dispatchKeyEvent(event);
}
```

Populate Keyboard Shortcut Overview

```
override fun onProvideKeyboardShortcuts(
data: MutableList<KeyboardShortcutGroup>?,
menu: Menu?,
deviceId: Int
) {
super.onProvideKeyboardShortcuts(data, menu, deviceId)

// Requires API 24
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
val keyboardShortcutGroup = KeyboardShortcutGroup("Test
Group")
keyboardShortcutGroup.addItem(KeyboardShortcutInfo("Shortcut
One", KeyEvent.KEYCODE_Z, KeyEvent.META_ALT_ON))
data?.add(keyboardShortcutGroup)
}
}
```

See for instance Register hardware keyboard shortcuts in Android's help menu - Stack Overflow

References

WCAG 2.2

2.1.1 Keyboard (A)

2.1.2 No Keyboard Trap (A)

2.1.4 Character Key Shortcuts (A)

Trigger

Use a trigger to indicate the beginning of a journey. Clearly communicate what is expected after the button or link is activated, and respect semantics: links should look like and behave like links, and the same applies to buttons, although exceptions may exist. Clarity is essential, every interactive element should both its purpose and the action it will perform.

Component Variants

Action Trigger

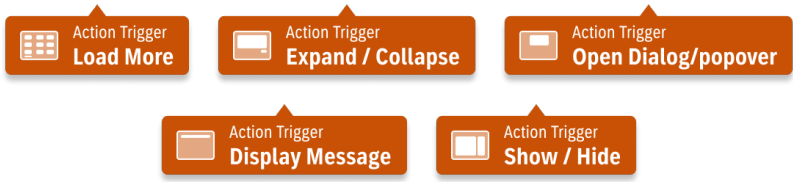


Figure 85: Indicates the expected outcome when a button is triggered. Common patterns keep the user in the same page.

Navigation Trigger

Figure 86: Indicates the expected outcome when a link (navigation) is triggered. In a common pattern, the link opens a new page.

About Triggers

Providing annotations that explain triggers and their outcomes such as buttons and links, is essential for meeting accessibility requirements and ensuring an inclusive user experience. Clear annotations help designers and developers define the purpose and expected behavior of interactive elements, reducing ambiguity and improving consistency in implementation. Therefore, designers should be explicit about the intended experience and the role of each UI element used in mock-ups.

For users relying on assistive technologies, such as screen readers, properly annotated triggers provide meaningful information about what an action will do before it is activated. This includes specifying whether a link

opens a new page in the same tab or a new tab, or whether a button triggers a dialog or popover.

Without these annotations, users may encounter unexpected changes in context, increased cognitive load, or navigation barriers. Documenting trigger behavior helps create predictable, accessible, and user-friendly interfaces.

Screen reader users rely on descriptive labels and ARIA attributes to understand the outcome of activating a button or link, preventing surprises such as a popover appearing or a new tab opening unexpectedly. Similarly, users with cognitive disabilities benefit from clear expectation, as sudden context changes can be disorienting or overwhelming. Missing or unclear annotations can lead to usability issues, such as Submit button unexpectedly opening a confirmation dialog or a link redirecting to a new tab without warning, which can cause frustration and precent task completion.

By proactively defining trigger behavior, Designers and Developers create more predictable and accessible experiences, reducing barriers for all users.

Examples

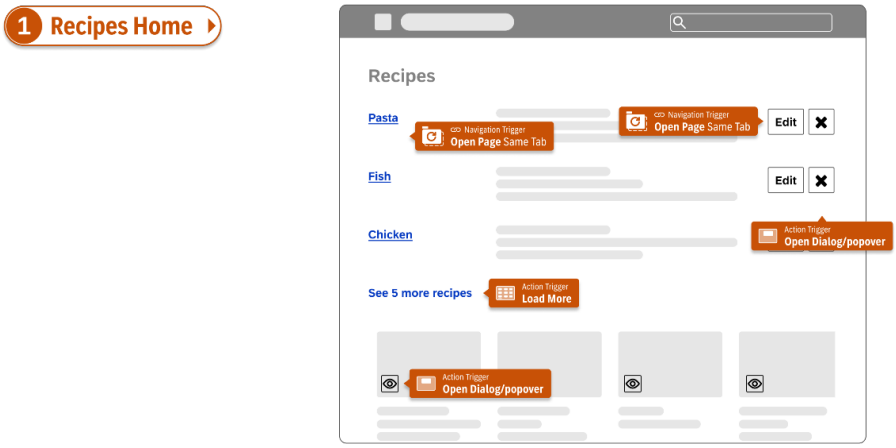


Figure 87: Example of indicating action and navigation triggers in a UI using the respective annotations

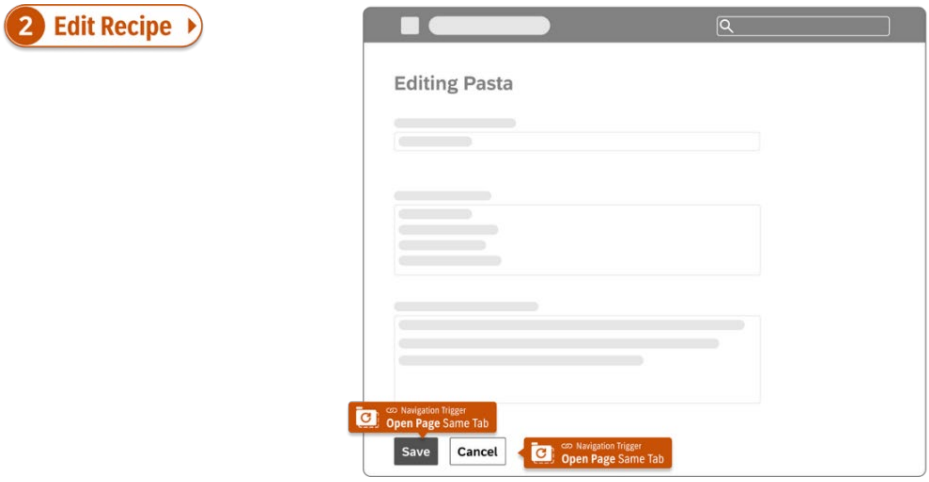


Figure 88: Example of navigation trigger annotations

Shared Benefits

Vision and Color Limitation

Visuals

Clear trigger annotations help ensure that the links and buttons are visually distinguishable through contrast, size, or spacing and consistently placed. This is especially important for users who rely on zoom or screen magnifiers to navigate interfaces.

Without Vision

Screen Reading

Trigger annotations help developers in correctly code programmatic labels, roles, and states, ensuring that screen reader users are informed of what the trigger is, what it will do, and whether it is currently active or disabled.

Limited Cognition

Cognitive

By promoting consistency and clear labeling, trigger annotations help users understand the outcome of interacting with an element. This reduces ambiguity and supports decision-making during complex tasks.

For Developers

Consult the API documentation for common trigger elements, such as links and buttons, in your framework.

References

WCAG 2.2

1.3.5 Identify Input Purpose (AA)

3.2.2 On Input (A)

4.1.2 Name, Role, Value (A)

Motion Alternative

If a feature is activated by moving the device, such as shaking, tilting, or gesturing, then the same function must be available through standard interface controls. The motion must be optional unless it is essential to the feature.

Component Variants

Motion Alternative

Figure 89: Indicates where in the interface an alternative trigger is available for a function that can also be activated by motion. Users should not rely solely on physical gestures.

About Motion Alternative

Designers, and Developers, can meet the WCAG 2.5.4 Motion Actuation (A) requirement by ensuring that any interaction based on motion has a visual element alternative in the UI.

In plain terms, if a feature is activated by moving the device, such as shaking, tilting, or gesturing, then the same function must be available through standard interface controls. The motion must be optional unless it is essential.

The goal is to ensure that users can still fully use the interface without relying on physical gestures. These applies to users who:

- Have limited motor abilities
- Cannot hold or move a device
- Use the interface in a fixed or mounted position
- Might accidentally trigger features via motion

Designers Guidelines:

1. Avoid motion-only interactions. Do not make motion gestures, such as shake or tilt, the only way to:
   - Undo actions (e.g., shake to undo)
   - Navigate
   - Trigger shortcuts or features

Always pair motion interactions with a standard UI control, such as a button or menu option.

2. Provide redundant, discoverable UI controls. For each motion-based trigger:
    - Include an on-screen button or an accessible keyboard or touch equivalent
    - Ensure it is visible, labeled, and reachable

3. Make motion features optional
    - Allow users to disable motion-based input in the app or via operational system settings
    - Detect when motion sensors are unavailable or turned off
4. Consider platform and context
    - On iOS and Android, users can limit or disable motion through system settings (e.g., Reduce Motion)
    - Apps should respect these preferences and not require motion

Exceptions

When Motion Can Be Required? Motion can be the only method if:

- It is essential for the function (e.g., augmented reality app requiring device movement)
- It is part of a physical simulation that loses meaning if replaced

Examples

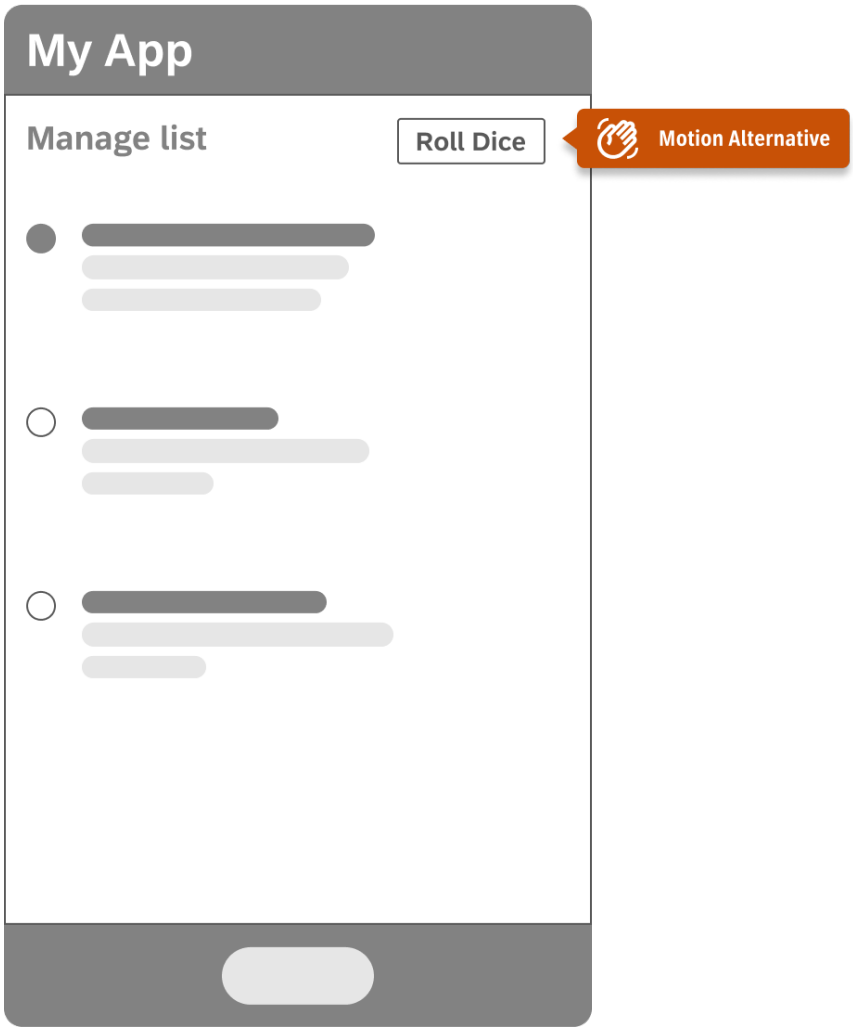


Figure 90: Example app mockup showing both a shake gesture to roll dice and a button labeled" Roll Dice" as an alternative trigger.

Do: Ensure a visible "Shake the Dice" button is always present on screen as an alternative.

Do Not: Make shaking the device the only way to roll the dice.

Shared Benefits

Benefits

Temporary Disabilities

All users

For people with temporary disabilities, (such as a broken arm, wrist sprain, eye patch, or concussion, relying on device motion can be difficult or even impossible. Having a visible alternative control means they do not have to shake, tilt, or gesture with precision while recovering. This makes the interface usable during periods of limited mobility, vision, or concentration, ensuring that access to features is not blocked to people with short-term conditions.

Without Vision

Screen Reading

Motion-based interactions, such as “shake to undo”, are not perceivable through screen readers. Providing a visible alternative control ensures they can activate the same functionality with standard navigation

Limited Cognition

Cognitive

Motion gestures can be hard to discover, remember, or perform consistently. A visible button or toggle makes the interaction clearer, more predictable, and less prone to errors.

For Developers

Provide typical alternatives to motion-based triggers, such as additional links and buttons, within the app.

References

WCAG 2.2

2.5.4 Motion Actuation (A)

Minimum Target Size

Make target sizes large enough for users to activate them using a pointing device, such as a mouse or stylus, or through touch. Aim for 24 pixels when minimum size is mandatory.

Component Variants

Minimum Target Size

Figure 91: The annotation is used on components to indicate that the target size fulfills the minimum of 24 pixels in both directions (x, y) and to assure interactive elements do not overlap.

About Minimum Target Size

In WCAG 2.1, a target size of at least 44 × 44 CSS pixels was an AAA requirement. WCAG 2.2 introduced a new AA requirement: targets must be at least 24 × 24 CSS pixels, with some exceptions. The 44 × 44 size remains a best practice. A large size of a target helps users activate actions more easily by using a mouse pointer, pen, or touch. The larger a clickable area is, the easier it is for users to reach an interactive element. The target size is considered the region where the pointer, also called a hit area, will reach the actionable element. The hit area of the interactive component can be larger than the component itself, including the surrounding whitespace, as in the case of an icon button or checkbox. Overlapping hit areas must be avoided.

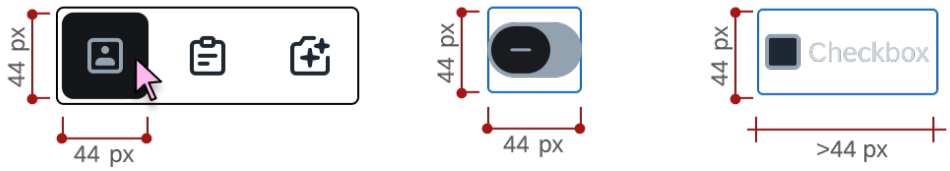


Figure 92: Examples of component sizes with hit areas larger than the minimum 24x 24 pixels

However, target size can vary depending on context, and some exceptions should be considered during the design:

- Text links in paragraphs or body text maintain the same size as the surrounding content.

- Help icons embedded in labels or sentences are considered part of the content and are not required to meet the minimum target size.
- Equivalent targets performing the same action on the same page: if one target meets the minimum size, the others are not required to do so.

User Benefits of Larger Targets:

- Users with limited dexterity: Users with fine motor control issues or hand tremors benefit from large targets, which allow them to interact comfortably and accurately using a mouse or touch device.
- Users of touch devices: Users with larger fingers, or who use toes, knuckles, or parts of fingers, benefit from larger targets. This also applies to users interacting with devices while in motion, as larger targets improve precision.
- Users with low vision: Larger targets make it easier to identify actionable elements among other content.

Mobile Guidelines:

- Native applications may follow platform-specific SDK requirements for minimum tappable areas. For example:
  - iOS: 44 by 44 points
  - Android: 48 by 48 pixels
- Following these guidelines helps ensure targets are easy to interact with across devices.

Examples

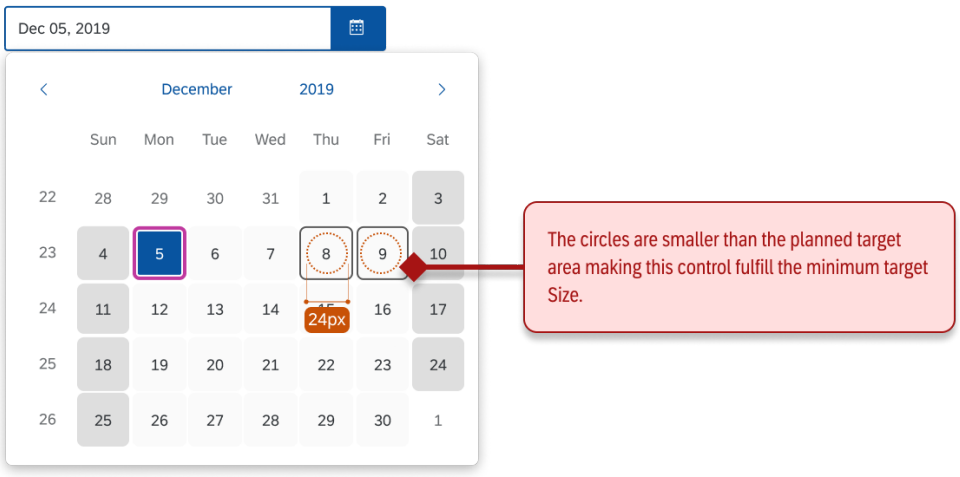


Figure 93: Calendar month view with two minimum target size annotations on day tiles, showing that the minimum target size is met

Do:

- Keep circles touching or separated.
- Adjust the size of the annotation to fit your components when placed side by side or stacked vertically.

Do not:

- Allow the circles representing the minimum target size to cross or overlap. This creates an accessibility violation.

Shared Benefits

Temporary Disabilities

All users

Situations like using a device with one hand, wearing gloves, or interacting in motion (e.g., walking) reduce precision. Users with temporary or situational disabilities also benefit from larger targets. They are easier to hit in challenging real-world scenarios, improving usability for everyone.

Limited Cognition

Cognitive

Small or densely packed targets can create confusion or require greater focus and precision, which may increase errors. Users with cognitive or learning disabilities also benefit from larger, well-spaced targets reducing cognitive load and help users to make clear choices more confident.

Vision and Color Limitation

Visuals

Larger targets are easier to locate visually, especially for users who rely on screen magnification tools that reduce visible context. Users with low vision also benefit because larger targets reduce the effort required to aim or locate interactive elements and help prevent accidental taps or clicks.

For Developers

Verify that the dimensions of all active framework elements fall within the recommended minimum sizes according to iOS and Android development guidelines. Additionally, ensure that the spacing between any two active elements meets the minimum distance requirements.

References

WCAG 2.2

2.5.8 Target Size (Minimum) (AA)

Journey

Plan an exploratory journey by outlining the steps required to interact with the component and navigate across pages to complete a task.

Component Variants

Journey Flow Step Name - Component

Figure 94: Indicates the name of a step in a component flow designed to be applied consistently in various design use cases.

Journey Flow Step Name – Page/Dialog

Figure 95: Indicates the name of a step in a business use case flow designed for a user to start, conduct, and conclude a task across pages and overlays.

Journey Direction Flow

Figure 96: Indicates the name of a step in a direction flow designed to be applied consistently in various design use cases.

About Journey

Journeys are an excellent way to describe navigation flows and interactions, at both the control and application levels. When an element is interactive and a trigger annotation is used, it is important to precisely document the flow of actions, capturing all steps of the journey.

In journey annotations, the trigger refers to the element or action that initiates a step, such as a button labeled “View Profile”, while the destination is the resulting screen, dialog, or component that follows, such as a profile details page.

Documenting both trigger and destination ensures consistent labeling, reinforces the user’ intent when the destination opens, and maintains clear wayfinding. This practice helps reduce confusion and supports inclusive experiences for users who rely on predictable flows, including those using screen readers and people with cognitive disabilities

Actions should be intuitively mapped to meet both visual requirements for sighted users, such as focus and states indicators and interactive requirements for all users such as vertical and horizontal navigation on pages or tables and announcements for screen reader users.

Journeys also clarify flows among screens. Before reviewing all the elements used to design a page, it is essential to understand the journey and associated triggers. This approach contributes to a more accurate accessibility review of screens and their content.

Examples

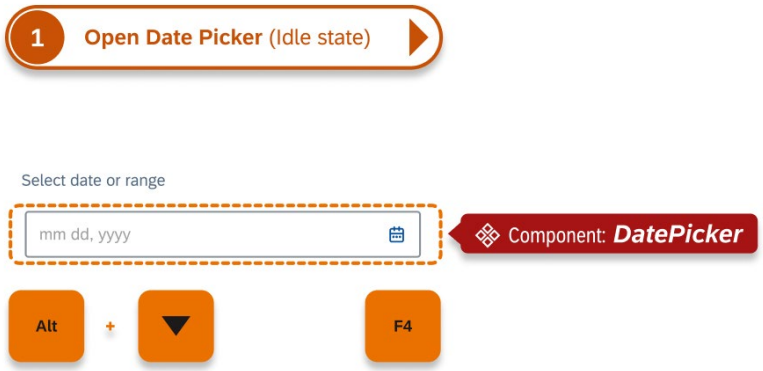


Figure 97: Step name annotation indicating that ALT + Arrow Down keys and the F4 key opens a date picker helper dialog

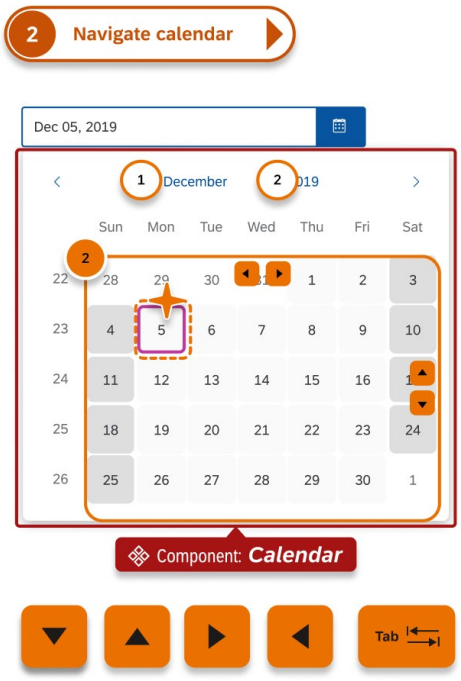


Figure 98: Step name annotation indicating that the Tab key navigates between functionalities, while Arrow keys navigate dates within the calendar

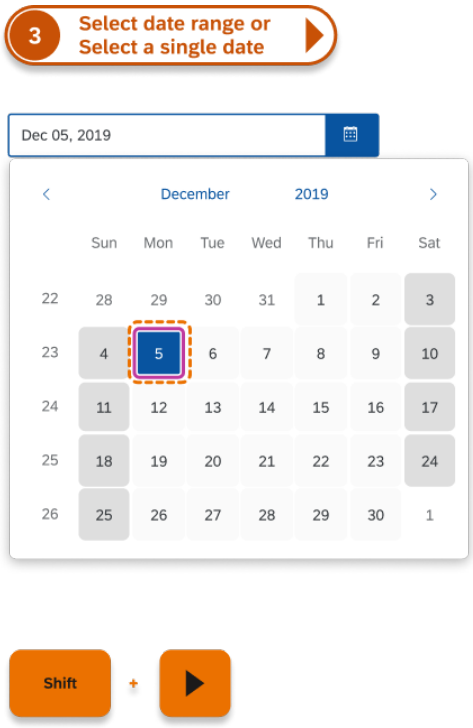


Figure 99: Step name annotation indicating that Shift + Arrow keys are used to select a range. Enter selects a single element. The initial focus position should be put to “today” by default.

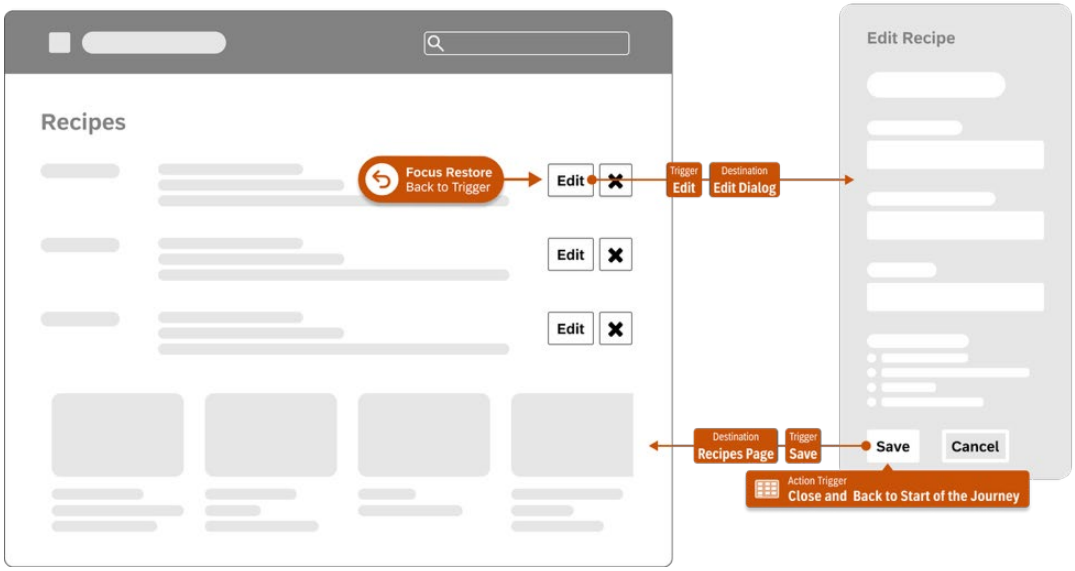


Figure 100: Journey direction flow annotation indicating the opening and closing of dialog screens

Shared Benefits

Temporary Disabilities

All users

For users with a broken arm, eye strain, fatigue, or even situational limitations, such as like using a device one-handed while carrying something, a predictable and minimal journey means less effort, fewer repeated attempts, and reduced frustration.

Vision and Color Limitation

Visuals

For low-vision users or screen magnifier users, knowing the trigger destination ensures they do not waste time scanning unrelated areas of the interface.

Without Vision

Screen Reading

For blind screen reader users, it ensures labels and announcements stay consistent, so the outcome of an action is clear and navigable without trial and error.

Limited Cognition

Cognitive

Journey annotations make navigation flows predictable and consistent, reducing cognitive load. When it is clear what will happen after pressing a button or link, users do not have to keep extra steps in working memory or deal with unexpected outcomes. This builds trust and confidence in completing tasks without confusion.

For Developers

Scrutinize workflows by discussing with the Designer which trigger elements navigate to each individual journey step.

References

WCAG 2.2

1.3.5 Identify Input Purpose (AA)

2.4.8 Location (AAA)

2.5.3 Label in Name (A)

4.1.2 Name, Role, Value (A)

4.1.3 Status Messages (AA)

“I love how everything is thought of from the various accessibility topics especially the keyboard and screen reader annotations.”

Aisling Noone – User Experience Designer

Screen Reading Experience

Imagine navigating a screen with only your hearing. What turns a simple list of words into a clear, navigable experience? The answer is a well-designed structure. This allows screen reader users to build a mental map, organizing page elements into an efficient experience.

This chapter shows designers how to implement screen reader support and create an inclusive experience for everyone, especially people who are blind.

The Inclusive Screen Reading Experience

Design your application and UI components to be fully understood and operated through a screen reader. This is essential for users who rely on screen readers, allowing them to perceive and interact with content that would otherwise be unavailable

Provide clear labels and text alternatives for all interactive and informational elements. For a screen reader user, an unlabeled button or a generic link like "Click Here" is ambiguous and confusing. Explicit labels for controls and descriptive text for images give every element a clear purpose, transforming a visual interface into a comprehensible auditory experience. This clarity benefits all users by reducing ambiguity and making the interface more predictable and easier to learn.

Structure content logically with headings, landmarks, and a consistent reading order. A well-organized page allows screen reader users to quickly grasp its layout and navigate efficiently. Instead of listening to everything sequentially, they can use headings to skim for content or use landmarks to jump directly to major sections like the "main" content or "navigation" bar. This semantic structure also creates a stronger visual hierarchy that makes content easier to find and understand for everyone.

Principles

Blindness

Persona: Bob

"I am legally blind, and I cannot see light or shapes. I use digital products with a screen reader to explore and operate digital products. My preferred interactive tools are the keyboard & touch in mobile devices."

This user group represents individuals who are legally blind and others with significant vision impairments. They rely on screen readers and assistive technologies that convert on-screen content into speech or braille readers and prefer to interact via keyboard and touch interactions to navigate and operate the digital product. Highly sequential, linear, and well-structured content is key to support usability and comply with accessibility. These users receive clues from the system by listening to announcements made by the screen reader. As a result, these users listen to many words per day to interact with digital products, which could be stressful.

Some users experience vision loss that can get worse over time. This may happen because of medical conditions such as glaucoma or diabetic retinopathy.[2, 4]

There are two main types of blindness:

- Total Blindness: People with total blindness cannot see anything and rely fully on assistive technologies like screen readers, braille, or audio and tactile feedback to navigate the world.[12]
- Legal Blindness: Defined as 20/200 vision or worse in the better eye (with correction) or a visual field of 20 degrees or less, people who are legally blind can detect shapes and brightness but have trouble seeing details. They often use assistive tools like screen readers to help with daily tasks.[12]

Some statistics:

- Global: 2.2 billion people have a near or distance vision impairment, where 36 million people are blind, and 217 million have moderate to severe vision impairment (WHO Vision Fact Sheet, 2023).

Situational & Temporary Disabilities

By including situational and temporary disabilities into this disability group, we help design teams address challenges that might not be permanent but still impact user experience under certain conditions.

Situational and temporary:

- Dark or bright environments: Difficulty seeing due to temporary lighting conditions, such as extreme glare or darkness.
- Temporary blindness: Due to events like a sudden injury or temporary eye condition (e.g., eye infection or trauma).

General Design Tips

1. **Accurate and Complete Announcements:**
   Screen readers depend on accurate labels and programmatic associations to convey content. Missing, unclear, or duplicate labels can leave users confused or unaware of available options.

   Design Tip: Ensure every interactive or informative element (e.g., buttons, links, form fields, icons, images) has a clear, unique accessible name using appropriate attributes like aria-label, aria-labelled by, alt, or semantic HTML elements (e.g., <label>).

2. **Clear Page Structure:**
   Without a well-defined heading hierarchy or landmarks, screen reader users may struggle to understand the layout or jump to relevant sections.

   Design Tip: Use semantic headings (<h1>–<h6>) to establish a clear content structure and add ARIA or native landmarks (<main>, <nav>, <header>, <footer>) to support orientation and quick navigation.

3. **Logical Reading Order:**
   If content is visually arranged in one way but coded in another, the screen reader may read it out of sequence, leading to confusion. Some screen reader users can still perceive a certain number of visual elements in the interface.

Design Tip: Ensure the DOM (document order) reflects the visual and logical order of content. Avoid using CSS tricks that misalign reading and focus order.

4. **Hybrid Navigation (Focus + Reading Order):**
   Challenge: Blind users often combine keyboard navigation (Tab/Shift+Tab) with arrow-key reading. Inconsistent focus management or skipped elements can break this experience.

   Design Tip: Maintain a predictable focus order and supplement it with reading-friendly design. Announce state changes, use ARIA roles (role="alert", aria-live) to communicate updates without requiring focus shifts.

5. **Unique and Contextual Labels for Repeating Elements:**
   Repeating components (e.g., icon buttons, links, or lists) with identical labels, confuses screen reader users.

   Design Tip: Provide context to each instance, e.g., "Edit profile" vs. "Edit address", using visible or invisible text tied to neighboring content using the aria-labelledby property.

6. **Supplementary Descriptions for Complex Elements:**
   Visual-only cues like color or icons are not announced by screen readers unless explicitly labeled to indicate the color semantic.

   Design Tip: Use aria-describedby or <span hidden> to associate additional info with elements like buttons or charts, ensuring screen reader users receive the same guidance as sighted users.

Annotations

Screen Reading Annotations

This checklist helps designers to remember the important aspects of accessibility that should be addressed in the design phase.

Screen Reading Experience Checklist

- [ ] Reading Order
- [ ] Label
- [ ] Description
- [ ] Live Message
- [ ] Heading
- [ ] Landmark
- [ ] Page Title
- [ ] Role and Properties
- [ ] Speech Output
- [ ] Audio Control
- [ ] Audio Description
- [ ] Language

Reading Order

Indicate and number the sequence of UI elements on the layout, including hidden elements when applicable, that must be logically perceived during screen reader announcements.

Component Variants

Reading Order

Figure 101: Indicate order of UI elements to be announced by the screen reader in reading navigation by using this annotation with numbers that correspond to the reading order.

Reading Order (with add. inner order)

Figure 102: UI elements that require an unknown number of further inner navigation/swipes, such as toolbars, breadcrumbs and tables, should be annotated with the Reading Order with additional inner order variant.

Reading Scope

Figure 103: When appropriate, enclose multiple UI elements to be read together by the screen reader in reading navigation, which reduces reading navigation effort.

Reading Flow

Figure 104: Indicate a general reading navigation flow direction, left to right or top to bottom (not depicted here), when individual annotation may not be required.

About Reading Order

The reading order is the sequence of UI elements announced when navigating between elements with the screen reader (in opposite to that, the *Focus Order* is the sequence of focusable UI elements when using TAB key, or Arrow keys in interactive lists).

On desktops, this will be done in a special reading mode of assistive technologies (called “virtual mode”, “browse mode” etc.). The user will use an assistive tool to navigate the UI elements using the up and down arrow keys. On mobile, the reading order is known as swipe order and navigation is done using swipe gestures.

The designer needs to identify the logical Reading Order of elements including texts, images and interactive controls. A logical order of elements enhances the understanding of the content and its context. Use the appropriate annotation variation to identify a single UI element, a group with inner navigation, or a group of content that should be announced contiguously by the screen reader. This helps to form logical contexts (what belongs together is spoken together) and reduces reading navigation effort.

The Reading Scope annotation should be used to represent UI elements that require further inner navigation (like toolbars, breadcrumbs and tables) where the total number of steps inside is unknown or not important in the current context.

The two variants of Reading Flow represent the default navigation directions. Use these when annotating individual annotation sequences is not required in your design case.

SAP Figma Plugin

The SAP Figma Plugin “Reading Order” helps designers annotate their screens identifying relevant focus order sequence.

Examples

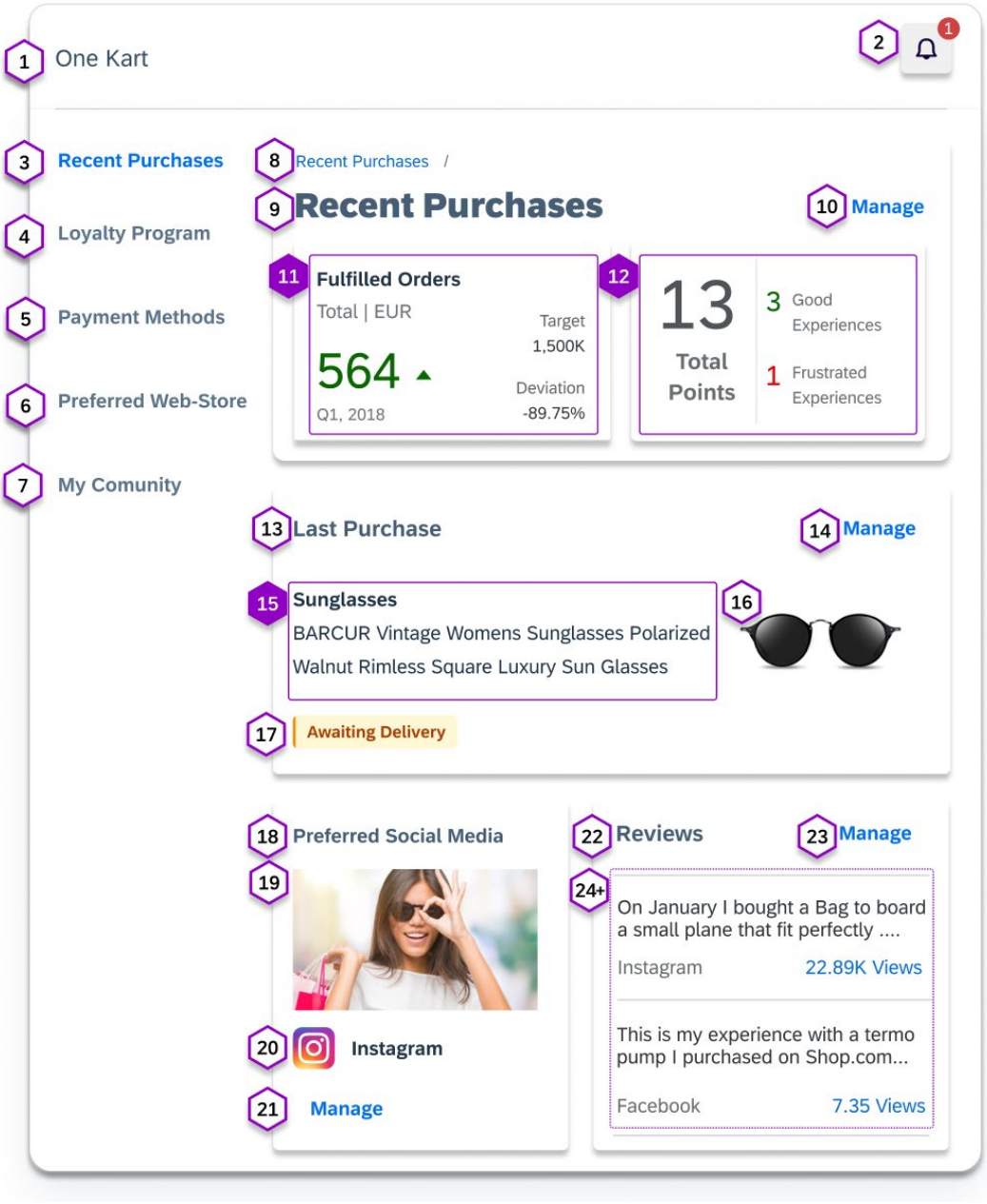


Figure 105: Reading sequence using reader keys or swipe gestures should follow a logical order and should respect the logical nesting of grouping containers in UI. Text belonging to a common context should be contiguously (11, 12, 15). Components with predefined inner reading orders but of unknown number are indicated with the Reading Order + annotation (24).

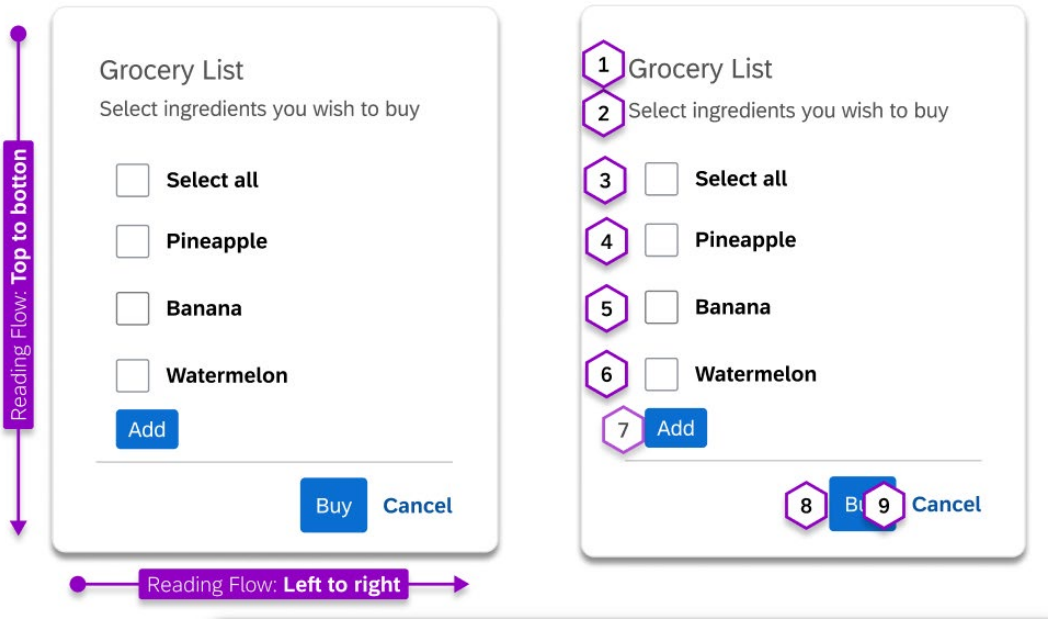


Figure 106: Default Reading order flow indication for a simple dialog. In this case, using Reading Flow annotation is like annotating individual Reading Order steps and saves considerable work, therefore it is preferred and recommended that the content follows a natural flow.

Shared Benefits

Temporary Disabilities

All users

A clearly structured, easily perceivable and easy to read UI benefits all users. Audio feedback is generally helpful in situations where users are not looking at the user interface since they are occupied with other tasks.

For Developers

Reading order, during arrowing with screen readers in HTML or swipe next operations on mobile devices, is established in all technologies by layout and content node element order and nesting.

Always make sure that elements that form a common context are adjacent in layout and not distributed between different containers, except if presented in an extra control context for this purpose (e.g., row/columns in tables).

Web Development:

By combining text that belongs together in one larger DOM Node

```
<p>
<span>Text 1</span>
<span>Text 2</span>
<span>Text 3</span>
</p>
```

Note: A swipe or arrowing in reading mode to the div will read multiple items at once

```
<div role=”group” aria-label=”Text 1 Text 2 Text 3”>
<span>Text 1</span>
<span>Text 2</span>
<span>Text 3</span>
</div>
```

Fallback: Concatenating all inner info in an aria-label of the container

Mobile Framework Developer:

iOS

Reading Order depends on view construction

```
VStack(alignment: .leading) {
Text(landmark.description)
.font(.subheadline)
HStack(alignment: .top) {
HStack(alignment: .top) {
Button(action: {}) {
Text("Route")
}
.accessibility(label: Text( "Plan a route"))
Spacer(minLength: 10)
Button(action: {}) {
Text("Website")
}
.accessibility(label: Text("Visit website"))
}
}
}
.accessibilityElement(children: .combine)
```

See Building layouts with stack views | Apple Developer Documentation

Android

Group content to be read at once in ConstrainedLayout or LinearLayout

```
<ConstraintLayout
android:id="@+id/song_data_container" ...
android:screenReaderFocusable="true">
<TextView
android:id="@+id/song_title" ...
android:focusable="false"
android:text="@string/my_song_title" />
<TextView
android:id="@+id/song_artist"
android:focusable="false"
android:text="@string/my_songwriter" /> ...
</ConstraintLayout>
```

Use AccessibilityTraversal methods to change default left to right swipe order

```
volupButton.setAccessibilityTraversalAfter(myView.findViewBy
Id(R.id.remote0).getId());
voldownButton.setAccessibilityTraversalAfter(myView.findView
ById(R.id.volup).getId());
chaineplusButton.setAccessibilityTraversalAfter(myView.findV
iewById(R.id.voldown).getId());
chainemoinsButton.setAccessibilityTraversalAfter(myView.find
ViewById(R.id.chaineplus).getId());
```

See View | API reference | Android Developers

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

1.3.2 Meaningful Sequence (A)

Label

Provide labels, visible or invisible, to identify and describe the purpose of UI elements for all users and screen readers.

Component Variants

Visible Label (Relationship)

Figure 107: Use this variant to reinforce the presence of a visible label that is tied to another UI element, like input field and label

Invisible Label

Figure 108: When a visual text cannot be used in the design, the Invisible Label variant is used to define an element purpose providing context to blind users

About Label

A label is the text or visual indicator that names or describes the purpose of a user interface element, such as a form field, button, or checkbox. It helps all users, including those using assistive technologies, understand what the element does or what information is expected.

Labels can be visible or invisible. Visible labels are easily recognized in forms as they are positioned typically above or adjacent to its interactive component. If you have a choice, use a visible label. If this is not possible, use an invisible label associated with a control to ensure that screen readers will announce the purpose of the element. An invisible label is used in minimalist designs to reduce visual clutter, such as when icon-only buttons are shown to sighted users, but it is essential to include an invisible label so that screen reader users, who are blind or have low vision, can perceive the purpose of the control. It ensures that the functionality remains understandable and operable for all users, even when visual text is hidden.

For sighted users, visible labels should be straightforward and aligned with common expectations, while for blind users, invisible labels should supplement visual context without redundancy. This approach ensures an inclusive experience, enabling all users to navigate interfaces efficiently and confidently.

Use cases where invisible labels are accepted are:

- Combined input fields like Address (number and street)
- List of checkboxes
- Icon button
- Semantic buttons

The content used in an invisible label should also be provided as a tooltip to ensure it is accessible to sighted users, so that essential context is not exclusive to blind users but available to everyone.

Good labels improve usability, accessibility, and reduce user errors. In design, always ensure labels are:

- Visible and placed close to the element they describe (e.g., above or beside a text field).
- Consistent in wording and placement.
- Programmatically associated with the element for screen readers, via label for in HTML or aria-label when appropriate.

Do not use placeholders as labels. Once the user enters a value, the “label-placeholder” will be gone, leaving users to wonder what information was requested in the input field.

When crafting labels, UX writers should prioritize clarity, ensuring that labels are immediately understandable without requiring users to interpret vague or ambiguous wording. Avoid abbreviations unless they are widely recognized, as unfamiliar shorthand can create confusion and increase cognitive load. Every label should be concise yet descriptive enough to provide clear meaning, reducing the mental effort needed to understand its purpose.

Here are some tips for UX writers to deliver a clean screen reader speech output:

- Use plain text on labels.
- Do not use a colon (":") as this is a decorative style - If needed, use it separated from the label.
- Use caps only on the first letter of the first word to secure a smooth tone of voice.

Labels are also useful to provide contextualization, for instance, on repeating Landmarks. The Landmark Region is often repeated in a page, which requires a contextualization to help blind users distinguish them.

Examples

Single Reference Annotations

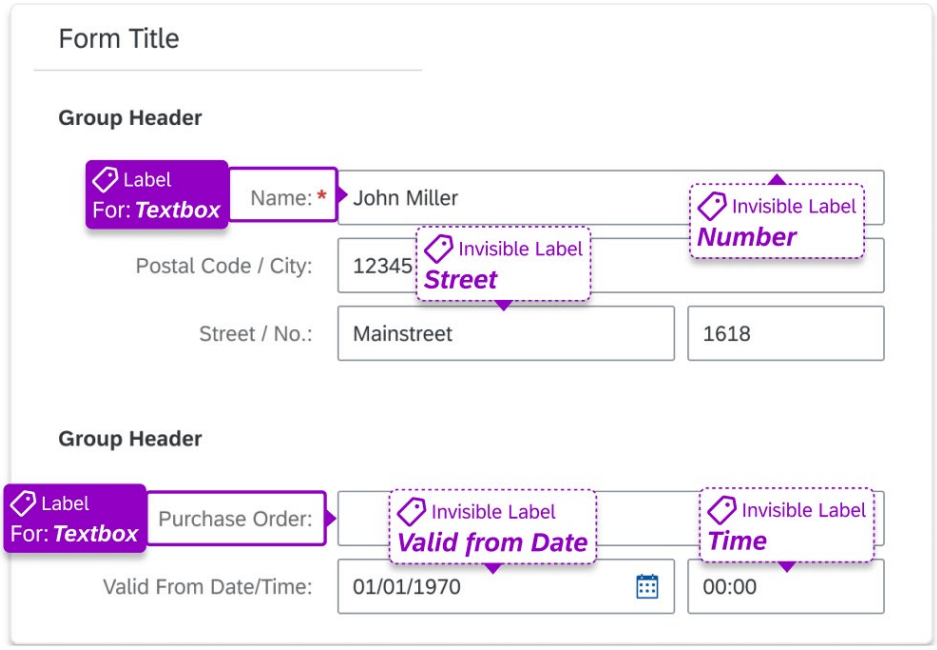


Figure 109: Different fields of a combined input field are presented with one unique visible label. Invisible labels help to clarify the purpose of the single fields.

Label Scope Annotations

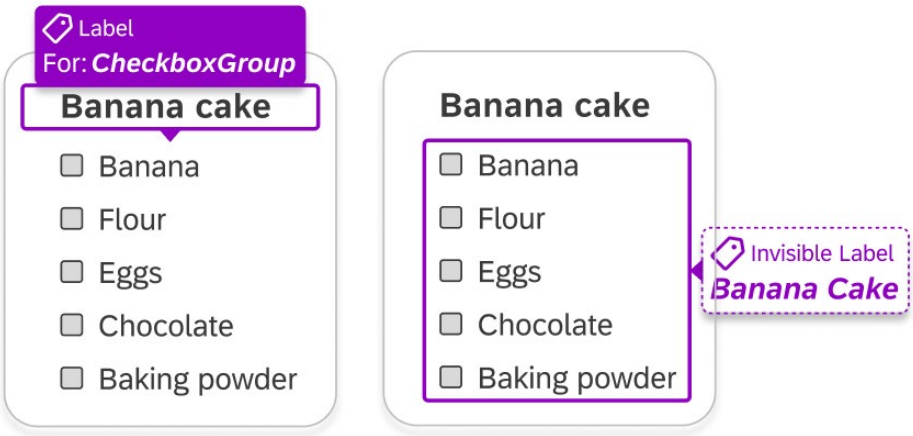


Figure 110: Two ways to annotate group labelling: Group of checkboxes identified by visible label relation or invisible label scope annotations assigning in both cases group header text.

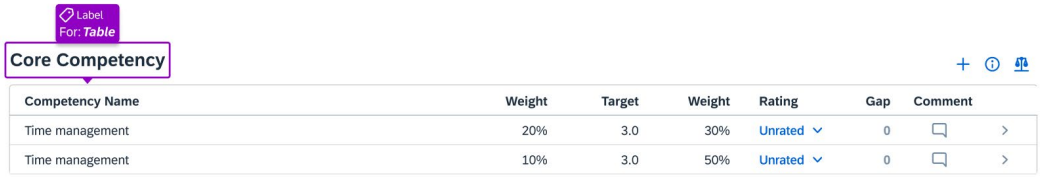


Figure 111: Visible table titles must be associated as label with corresponding table

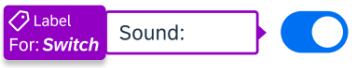


Figure 112: Label-control association for a custom Android switch component

Shared Benefits

Temporary Disabilities

All users

Labels provide clarity and efficiency by making controls easy to understand at a glance. They reduce mistakes, especially in forms, and they stay visible when placeholder text disappears.

Limited Mobility

Interactions

Labels support voice input and dictation by giving commands a clear target. By reducing repeated actions and guesswork, labels make interactions less tiring for users with limited movement.

Limited Cognition

Cognitive

Labels reduce cognitive load by keeping instructions visible, so users don't have to remember what to type once they start.

For Developers

Labels support the Info and Relationships principle by clearly linking UI controls to their purpose. They serve as an accessible name or description, whether visible or hidden (e.g., using aria-label), ensuring that form controls and inputs are understandable to all users.

Ensure an invisible label is informed where the design uses a limited number of words to provide context.

Accessible Name

Accessible Name is the Primary Identifier used by assistive technologies to identify an element. It is given to UI elements as a supplement to create

meaningful labels associated to meaningful visual cues where needed (like icons or section areas). Examples of Accessible Names are “aria-label” for custom names when there is no good visible label, “aria-labelledby” to reference repeating instances with an existing text to make them unique in the UI - instead of duplicating content.

Key Differences Between ‘aria-label’ and ‘aria-labelledby’

Feature	aria-label	aria-labelledby
When to use	When there is no visible text to describe the element	When you can reuse an existing visible element as the label
How it works	Provides a custom, standalone name	References the text of another element
Example usage	<nav aria-label="Main Navigation">	<nav aria-labelledby="menu-heading">
Downside	Can create inconsistencies if not aligned with visible UI	Requires maintaining correct ID references

UI technologies have different labeling techniques but there are only three basic principles to denote the relationship between a label and a control:

1. Inclusion: Label and control form a common context, which is mostly represented by control having a label text attribute or markup that encloses label and control together
2. Forward labelling: Separate label control has an id pointer to id of a different control (“label for=” approach in HTML)
3. Backward labelling: Control has an id pointer to id of a control used as a label (WAI-ARIA approach)

To apply correct labelling techniques for some base technologies, read the respective info in the links given below:

Web Development

HTML:
H44: Using label elements to associate text labels with form controls | WAI | W3C

WAI-ARIA:
ARIA16: Using aria-labelledby to provide a name for user interface controls | WAI | W3C

Mobile Development

iOS:
accessibilityLabel | Apple Developer Documentation

Android:
Principles for improving app accessibility | App quality | Android Developers

If you are using an UI framework that possibly encapsulates one or more of the methods above, look for respective framework API property documentation instead.

References

WCAG 2.2 Reference

1.1.1 Non-Text Content (A)

1.3.1 Info and Relationships (A)

1.3.5 Identify Input Purpose (AA)

2.4.6 Headings and Labels (AA)

2.5.3 Label in Name (A)

3.3.2 Labels or Instructions (A)

Description

Describe images, add invisible descriptions for screen reader users and relate visible descriptions to other UI elements.

Component Variants

Visible Description (Relation)

Figure 113: Use this variant to describe the invisible description and its relationship to the element

Invisible Description

Figure 114: Use this variant to describe the invisible description for an element

Decorative Element

Figure 115: Use this variant on visual UI elements to indicate that no screen reader output should be generated for those

About Description

Use Description to explain images. These descriptions are very popular and widely used to fulfill accessibility for a long time. They are known as Alternative Text (ALT Text).

Most of the time the description will be an Invisible Description to be announced by the screen reader.

But a description can also be a Visual Description, which gives all users access to the same content, avoiding privileged information shared only with blind users.

Besides images, visible descriptions can also be used as complementary information next to form fields. This description appears as a static text part

of the interactive UI control. For example: Label <Quantity>, Value <3>, Description <kg>.

Decorative images do not need any description and are to be hidden from screen readers.

When photos, illustrations or similar non-text content have an important part in the context, they must be described to allow users of screen readers to understand the complete information.

Image of text must be described containing the visible text to allow users of screen readers to have equal access to information seen by sighted users.

Examples

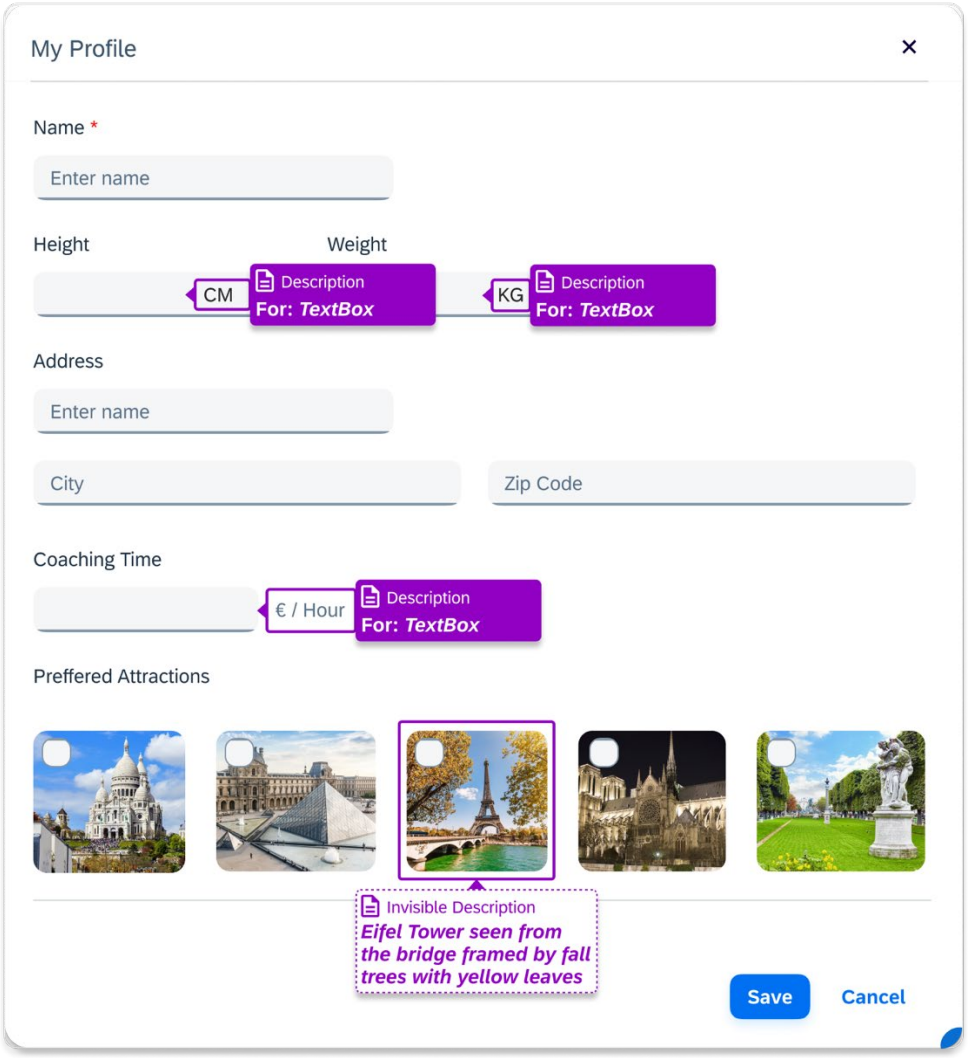


Figure 116: Description relations and image descriptions

Custom object status descriptions can be added as invisible description.

Figure 117: Invisible Description Annotation used for the positive value of a status.

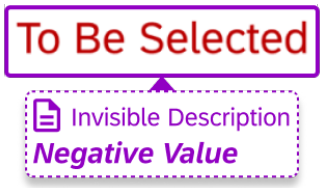


Figure 118: Invisible Description Annotation used for the negative value of a status

Figure 119: Decorative images must not have any descriptions and are to be hidden from screen readers by technical means

Figure 120: Image of text must be described including the visible text

Shared Benefits

Temporary Disabilities

All users

Visible descriptions provide context that benefits everyone, ensuring transparency and avoiding situations where important information is only available to some. For example, visual descriptions next to form fields clarify values, units, or instructions, supporting better usability and reducing errors.

For Developers

UI technologies have different techniques for assigning descriptions, but there are only two basic principles to denote the relationship between a description and a control:

1. Inclusion: Description and control form a common context, which is mostly represented by control having a description text attribute or markup that encloses description and control together
2. Backward describing: Control has an id pointer to id of a control used as a description (WAI-ARIA approach)

If you are using an UI framework that possibly encapsulates one or more of the methods below, look also for respective framework API property documentation.

Web Development

Visible/Invisible descriptions:

By using the “aria-describedby” reference attribute (“backward” describing)

```
<label for=”i1”>Weight</label><input id=”i1” aria-
describedby=”d1”/>
<span id=”d1”>kg</span>
```

Invisible descriptions: By using the HTML alt attribute OR the “aria-description” attribute OR by additional “hidden” DOM text (“direct” describing)

```
<img src=”x5ayk.gif” alt=”Warning”/><span
style=”background-image: eiffel.png”
aria-description=”Eiffel tower in autumn on a sunny day”>
<div>
<span>Assembly Option</span>
<span class=”urgent”>To be selected <span
class=”hiddentext”>Negative Status</span> </span>
</div>
```

Mobile Development

iOS:
Descriptions via accessibility hint

```
VStack(alignment: .leading) {
Text(landmark.description)
```

```
.font(.subheadline)
HStack(alignment: .top) {
HStack(alignment: .top) {
Button(action: {}) {
Text("Route")
}
.accessibility(hint: Text( "Navigates to next screen to plan
a tracking route"))
Spacer(minLength: 10)
Button(action: {}) {
Text("Website")
}
.accessibility(hint: Text("Open the website of the park's
visitor information"))
}
}
}
.accessibilityElement(children: .combine)
```

See [accessibilityHint | Apple Developer Documentation]

Android:

Describe images

```
<ImageView
...
android:src="@drawable/my_image"
android:contentDescription="@string/my_image_description"
/>
```

Give additional object information

```
<TextView
...
android:contentDescription="To be selected - Negative Value"
/>
```

Identify decorative images

```
<ImageView
...
android:importantForAccessibility="no"
android:contentDescription="null"
/>
```

Describe a complex context

```
containerView.setContentDescription("3 mails not read,
button");
containerView.setImportantForAccessibility(View.IMPORTANT_FO
R_ACCESSIBILITY_YES);
mailImageView.setImportantForAccessibility(View.IMPORTANT_FO
R_ACCESSIBILITY_NO);
infobulleTextView.setImportantForAccessibility(View.IMPORTAN
T_FOR_ACCESSIBILITY_NO;
```

See: View | API reference | Android Developers

References

WCAG 2.2 Reference

1.1.1 Non-Text Content (A)

1.4.5 Images of Text (AA)

Live Message

Provide clear and concise updates for dynamic content changes, such as form errors, status updates, or confirmations, using live regions. Ensure the message is announced without moving focus or disrupting the current task of the user.

Component Variants

Visible Message

Figure 121: Use this variant to indicate that this section on the UI represents a visible live updated message

Invisible Live Message

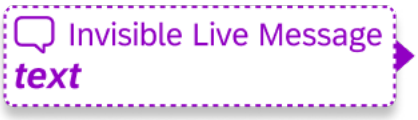


Figure 122: Use this annotation to indicate an invisible message and its relationship

About Live Message

Visible messages are also perceived by non-blind people and contribute to a usable experience. Messages are triggered and presented after user interaction or system updates. They are usually visible but often fade away in a few seconds, like toast messages.

An essential aspect of toast messages is that, whenever they appear, the focus remains where it was before the message was triggered. Screen readers should be prepared to announce visible messages (like in message areas of business applications) while keeping the focus on the current UI element; focus does not move to the message. This is important, especially when a message disappears in a few seconds.

Dialog messages, on the other hand, will change focus positioning the user into the message.

Invisible Messages happen on a few occasions. Some examples are messages that should announce search results, filter results, feed list additions or auto-save. These use cases may not provide one specific visual feedback. Instead, the state of existing UI elements should react and

be announced in a way that is easily understood by both: blind and visually abled users.

Remember:

- When messages pop up, focus does not change. This applies to toast messages. Focus remains on the element the user was working on before the message appears. This rule does not apply to dialog messages.
- Messages that fade away in a few seconds is not recommended. Neurodivergent users need extra time to read the message. It is best to give control to the user to dismiss the message when the user ends the reading.

Alert sounds alone create a barrier for blind users because the information is not conveyed in a way they can perceive. If a sound, like a beep, occurs without a visible or programmatically exposed message, screen reader users will not know what happened. WCAG indirectly addresses this in Success Criterion 4.1.3 Status Messages (AA), which requires that status updates (e.g., “message sent,” “connection lost,” “new message received”) be exposed to assistive technologies, typically through mechanisms like ARIA live.

ARIA live regions are critical for accessibility, but can be overlooked during the design phase, can cause confusion or loss of important updates for screen reader users. Application and component designers should specify ARIA live Regions for these three reasons:

1. **Screen readers do not “see” visual changes.** If a success message, error, or dynamic update appears visually (e.g., a toast, notification bar, or inline message), it will not be announced unless it is placed inside a properly configured ARIA live region.

2. **Developers cannot guess your intent.** If your design does not explicitly indicate which dynamic updates are important for screen reader users, they may not implement aria-live, or may use the wrong priority (e.g., polite vs assertive).

3. **It ensures parity in user experience.** By planning for ARIA live regions in design, you ensure that blind and screen reader users receive timely feedback about system responses, errors, confirmations, or live updates—just like sighted users do.

Interruptions such as system alerts, popups, notifications, banners, or auto-refreshing content should be avoidable or user-controlled, except when urgent. This requirement is aligned with WCAG 2.2.4 Interruptions (AAA)

which ensures users can stay focused, not lose progress, and control or postpone unexpected interruptions. This is also important for users with cognitive disabilities, attention-related challenges, or limited motor control, who may find it difficult to return to their task or regain focus after an unplanned disruption.

Interruptions can break the flow of interaction, causing users to lose their place or input. To prevent this, designers should use non-intrusive alerts that do not steal focus and provide settings to postpone or manage notifications. This will help users stay oriented and maintain control without unnecessary distraction.

Examples

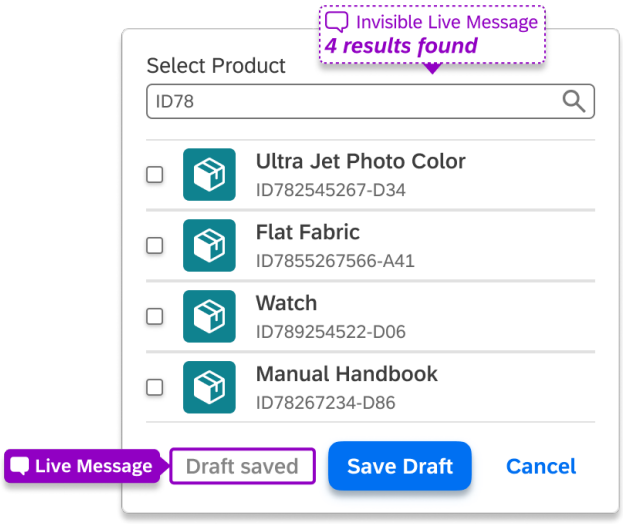


Figure 123:
1. Invisible Message announced after triggering search,
2. Message “Draft saved” added to a toolbar because of a save operation.

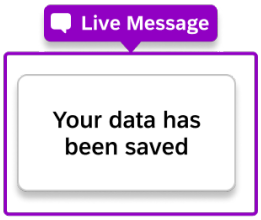


Figure 124: Message announced when toast message appears temporarily in the screen

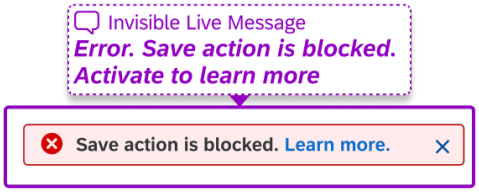


Figure 125: Sometimes an invisible message must be announced differently from a visual message including extended information, such as semantic description associated to visual cues (error sign)

Shared Benefits

Temporary Disabilities

All users

Live messages benefit everyone by providing immediate feedback on actions and system updates, improving situational awareness and interaction confidence, even when the visual change is subtle or transient.

Limited Cognition

Cognitive

Clear, concise live messages reduce confusion by explicitly communicating system state changes and feedback from interactions, supporting better understanding and task completion.

For Developers

If you are using an UI framework that possibly encapsulates one or more of the methods below, look also for respective framework API property documentation.

Web Development

By declaring live regions (self-announcing on DOM update)

```
<div role=”group” aria-live=”assertive”>
<span>Error: Cannot perform save. Reason: Object locked by
other user</span>
</div>
```

Mobile Development

iOS

- Notifying about the exceptional case
- Notify user about errors and allow user to move focus to the point where the context of the notification is described.
- Create a notification area (e.g. a banner) which contains/displays the context
- Create a notification and notify user
- Allow user to set focus on notification area and perceive the context information.

```
.onChange(of: notification) { notification in
if notification?.priority == .high {
isNotificationFocused = true
} else {
postAccessibilityNotification() // notification.text
}
}
```

SwiftUI Accessibility: Beyond the basics
https://developer.apple.com/videos/play/wwdc2021/10119/

(see from minute 24)

Android

Announce error messages

```
<TextView
[…]
android:accessibilityLiveRegion="polite" />
```

Give instructions

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
myView.announceForAccessibility(R.string.SearchResultInfo);
}

See Android Live Regions | Mobile A11y, View.AccessibilityLiveRegion Property (Android.Views) | Microsoft Learn, View | API reference | Android Developers

References

WCAG 2.2 Reference

1.3.5 Identify Input Purpose (AA)

2.5.3 Label in Name (A)

2.2.4 Interruptions (AAA)

4.1.2 Name, Role, Value (A)

4.1.3 Status Messages (AA)

Heading

Prepare a meaningful headings structure to enable all users and screen readers to find relevant blocks of content quickly.

Component Variants

Visible Heading

Figure 126: Use this variant to identify a visible heading pointing to text that appears on the screen

Invisible Heading

Figure 127: Use this variant to identify an invisible heading and informing the content of the hidden heading

About Headings

The Headings navigation strategy is used with screen readers to skim the page as they communicate the organization of the content.

Visible headings are directly tied to best practices on usability. The layout and headings must respect the logical and consistent structure of the page to provide predictability to the application exploration.

Use invisible headings when a page layout has a clear visual structure but no visible heading to represent it. In your annotation, indicate which section the invisible heading belongs to. This ensures that screen reader users receive a meaningful, contextual announcement when navigating by reading order.

Screen reader users often rely on heading navigation to quickly move through content. In JAWS and NVDA, pressing the “H” key jumps from one heading to the next, allowing users to scan and orient themselves efficiently without reading everything line by line. This makes proper use of heading hierarchy (H1, H2, H3, etc.) essential—not just for visual structure, but also for creating a logical, navigable outline of the page. Poor or inconsistent

heading structure can make navigation confusing and force users to work harder to find information.

Dedicate the **H1** to convey the main purpose of the page, since assistive technologies often announce it as the page title. Make sure the rest of the visible heading hierarchy matches the code structure (<h1>, <h2>, <h3>, and so on).

Lower-level headings (<h2>, <h3>, etc.) should create subsections within the higher-level section above them.

Note: Heading hierarchy is not consistently supported or recognized in mobile screen readers. Users rely on swipe through headings without any reference to the content structure. Even though many mobile screen readers recognize only a single heading level, it is still important to define all headings in the reading order. Proper headings ensure content remains logically organized and understandable, supporting navigation and context for all users.

SAP Figma Plugin

The SAP Figma Plugin “Headings” helps designers annotate their screens identifying relevant focus order sequence.

Examples

Figure 128: Complex Main Page with different heading levels. Additionally, invisible headings have been defined for enhanced navigation within page structure.

Shared Benefits

Temporary Disabilities

All users

A concise and clear heading structure helps all users to orient in any user interface. Audio feedback is generally helpful in all situations when users are not looking at the user interface since they are occupied with other tasks.

Vision and Color Limitation

Visuals

Large or clearly styled headings help users identify sections visually and orient themselves within the content, reducing cognitive load and enhancing readability.

Limited Mobility

Interactions

For users relying on keyboard navigation, headings provide logical landmarks to jump between sections without needing fine motor control for scrolling or pointing.

Limited Cognition

Cognitive

Well-structured headings break content into meaningful, digestible sections, supporting comprehension, focus, and memory retention.

For Developers

If you are using an UI framework that possibly encapsulates one or more of the methods below, look for respective framework API property documentation instead.

Web Development

By adding heading role with level info to HTML element

```
<div role="heading" aria-level="2">
My Heading
</div>
```

By using respective HTML element directly

```
<h2>My Heading </h2>
```

Hidden headings that can be read by screen readers can be created using respective CSS on the heading like so:

```
.pseudoInvisibleText {
   position:absolute!important;
   clip:rect(1px,1px,1px,1px);
}
```

Do not use display:none since this will remove the heading from the DOM and it will no longer be detected by screen readers.

Mobile Framework Development

iOS

Declare headings as .isHeader trait

Screen reader VoiceOver allows Rotor-based navigation to headings.

```
VStack(alignment: .leading) {
Text(currentTile.name)
.font(.title)
.accessibility(addTraits: .isHeader)

VStack(alignment: .left) {
Text(listItem.label)
.font(.subheadline)
Spacer()
Text(listitem.value)
.font(.subheadline)
}
```

See accessibilityAddTraits(_:) | Apple Developer Documentation ; isHeader | Apple Developer Documentation

Android

Use accessibilityHeading on TextViews

```
<TextView
[...]
android:id="@+id/myTextLevel1viewId"
android:accessibilityHeading="true"
/>
```

See View | API reference | Android Developers

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

1.3.2 Meaningful Sequence (A)

2.4.2 Page Title (A)

2.4.6 Headings and Labels (AA)

2.4.10 Section Headings (AAA)

Landmark

Define semantic groupings in your layout and assign appropriate labelled landmarks around them to enhance screen reader navigation.

Component Variants

Landmark

Figure 129: Use the annotation variants to define and label groups of related information in your layout

About Landmark

Navigation by landmarks is used by users of assistive technology as an alternative to “skip to main content.” They are used to speed navigation, similar to skipping groups. Use them to indicate a region or a group within a page to enhance screen reader navigation strategies.

Sighted users find it helpful to understand the structure of a page by identifying groups of information visually laid out. Blind users understand the structure of a page using landmarks embedded in screen reader tools.

Landmarks are only available for screen readers and are NOT a substitute to skip blocks (F6). Landmark navigation on JAWS uses the “R” Key. In NVDA, landmark navigation uses the “D” Key; the user uses shift + Key to navigate backward. Both technologies (or tools) offer an interface listing the landmarks, something not available on skip groups.

Please note: Currently the mobile operating systems iOS and Android do not support Landmarks in views. This may change in future releases.

Although landmarks may overlap with skipping groups, the keyboard navigation strategies differ. The same general structure of landmark blocks (L & R keys in Jaws and NVDA) can match skipping groups (F6).

Recommendations:

(1) Use the “Main” landmark only once on the page.

(2) Avoid repeating the same landmark as it decreases the ability to outline unique regions of a page.

(3) Region, form, content info, complementary, navigation, and search can be repeated. But you must add a label to make them distinguishable.

(4) Use them sparingly as too many landmark roles create "noise" in screen readers, making it difficult to understand the overall layout of the page.

(5) The Complementary landmark is used to group tertiary information that will help users understand the context of the content.

This list helps UX designers and developers to map HTML elements to their ARIA roles correctly while minimizing unnecessary ARIA attributes.

- Banner: Represents introductory content or site-wide branding. It is typically at the top of a page but should not be inside <article> or <section> unless unique. ARIA landmark role <header>
- Navigation: Contains primary or secondary navigation links. It needs aria-labelledby when there are multiple navigation regions. ARIA landmark role <nav>
- Search: Identifies a section dedicated to search functionality. It acts as an implicit landmark without needing role="search". ARIA landmark role <search>
- Main: The primary content of the page, one per page. It helps screen reader users skip repetitive elements like headers and sidebars. ARIA landmark role <main>
- Complementary: Holds related, but non-essential content (e.g., sidebars, ads, or widgets). It needs aria-labelledby if context is needed. ARIA landmark role <aside>
- Region: Defines a section of content that is meaningful on its own. It needs aria-labelledby if it needs a name for navigation. ARIA landmark role <section>
- Contentinfo: Contains site-wide or section-wide footer information. It should not be used inside <article> unless unique to that article. ARIA landmark role <footer>
- Form: Becomes a landmark only when it has an accessible name (e.g., aria-labelledby or <legend> in fieldsets). ARIA landmark role <form>
- Dialog: Represents a self-contained interaction, such as a pop-up or modal. It becomes a landmark when it has role="dialog" or aria-labelledby. ARIA landmark role <dialog>

- Article: Represents self-contained content (e.g., news articles, blog posts). It typically used in feeds or lists of independent content pieces. ARIA landmark role <article>

The notation <> represents the HTML element that naturally map to each ARIA landmark role. These elements are semantic by default, meaning they automatically act as landmarks without needing extra ARIA roles (unless there are multiple, and they need differentiation).

For example:

- <header> automatically acts as a banner landmark (unless inside <article> or <section>).
- <nav> automatically acts as a navigation landmark (but might need aria-label if multiple exist).
- <main> automatically acts as a main landmark (only one per page).

Figure 130: Screen readers users may use these keys to navigate landmarks.

SAP Figma Plugin

The SAP Figma Plugin “Landmarks” helps designers annotate their screens identifying relevant focus order sequence.

Examples

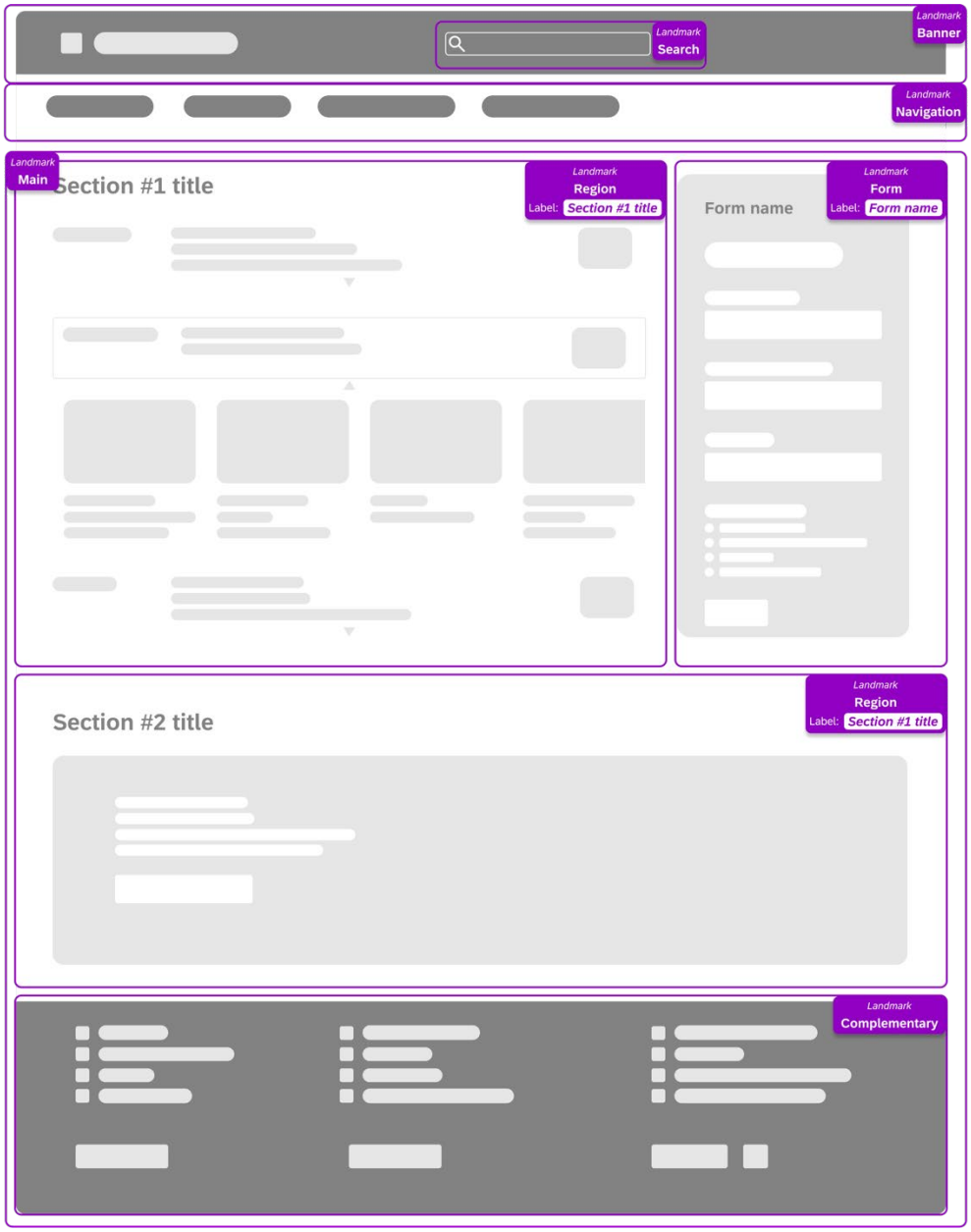


Figure 131: Scheme for applying different types of landmarks for a complex page. Regions and Forms require extra contextualization by labels to help blind users distinguish between them.

Shared Benefits

Temporary Disabilities

All users

When landmarks align with the visual structure, the interface becomes more usable for everyone. Screen reader users and sighted users can refer to the same sections, making it easier to locate and access key areas while supporting consistency and efficient interaction across the interface.

Limited Mobility

Interaction

Users navigating via keyboard or alternative input devices can move between landmark regions quickly without relying on precise pointing or scrolling.

For Developers

UI technologies have different message raising techniques:

If you are using an UI framework that possibly encapsulates one or more of the methods below, look for respective framework API property documentation instead.

Web Development

By adding landmark role to HTML element

```
<div id="content" role="main">
<!-- ... Main Content goes here ... -->
</div>

<div id="nav" role="navigation">
<!-- ... navigation entries ... -->
</div>

<form role="search" ...>
<input type="search" name="searchinput" />
...
```

```
</form>
```

By using respective HTML element directly

```
<main id="content">
<!-- ... Main Content goes here ... -->
</main>
```

Mobile Development

Currently there is no API for landmarks in iOS and Android.

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

2.4.1 Bypass Blocks (A)

Page Title

All users must be able to identify unique app contexts. Blind users with screen readers must recognize the page they landed on.

Component Variants

Web Browser Title

Figure 132: Indicate the page title on user agents (browser tab text) using a standard pattern all users can perceive to identify unique app contexts and screen readers announce correct title when a page loads.

Mobile App Title

Figure 133: Indicate the app title using a standard pattern all users can perceive to identify apps in mobile app switchers

About Page Title

Page Title is a unique identification text that helps users orient themselves when page titles are presented on browser tabs or in OS app/task switchers. The page title is the first announcement from the screen reader when a page or a view is loaded.

A visible page title helps all users locate where they are amongst the many pages of a product. Page titles in mobile devices are identical to the current view titles.

A Page Title is a combination of various identifiers, for instance one or more visible section titles, product name, and application name.

Sighted users easily recognize all these elements, but the information may be scattered or ungrouped on the page on desktops. Also, mobile apps save space and usually hide some of this information. Use the Page Title

annotation to combine the visually scattered UI elements into one meaningful information to be announced by the screen reader when user opens the page. This information will also be available to sighted people because it will be presented in the browser tab.

Page Titles must be planned and informed during the design of a screen to allow the developer to code the title of the page correctly for all users.

Sometimes a page title is composed of invisible text. The Invisible Page Title variant should be used in all cases where some titles or names could be invisible. This will create a clear context for users of screen readers, especially in mobile scenarios for pages without visible titles.

A variety of combinations are possible to present a page title pattern. The sequence and presentation format of the individual parts has to be defined by company guidelines.

Examples for page titles:

Page Title = Section Title(s) | App Name | Company Name

OR

Page Title = Product Name | Section Title(s)

Note: The OS or the browser may append to that the OS process name automatically, such as “Google Chrome” for Chrome browser or the app name such as “Microsoft Word”.

Examples

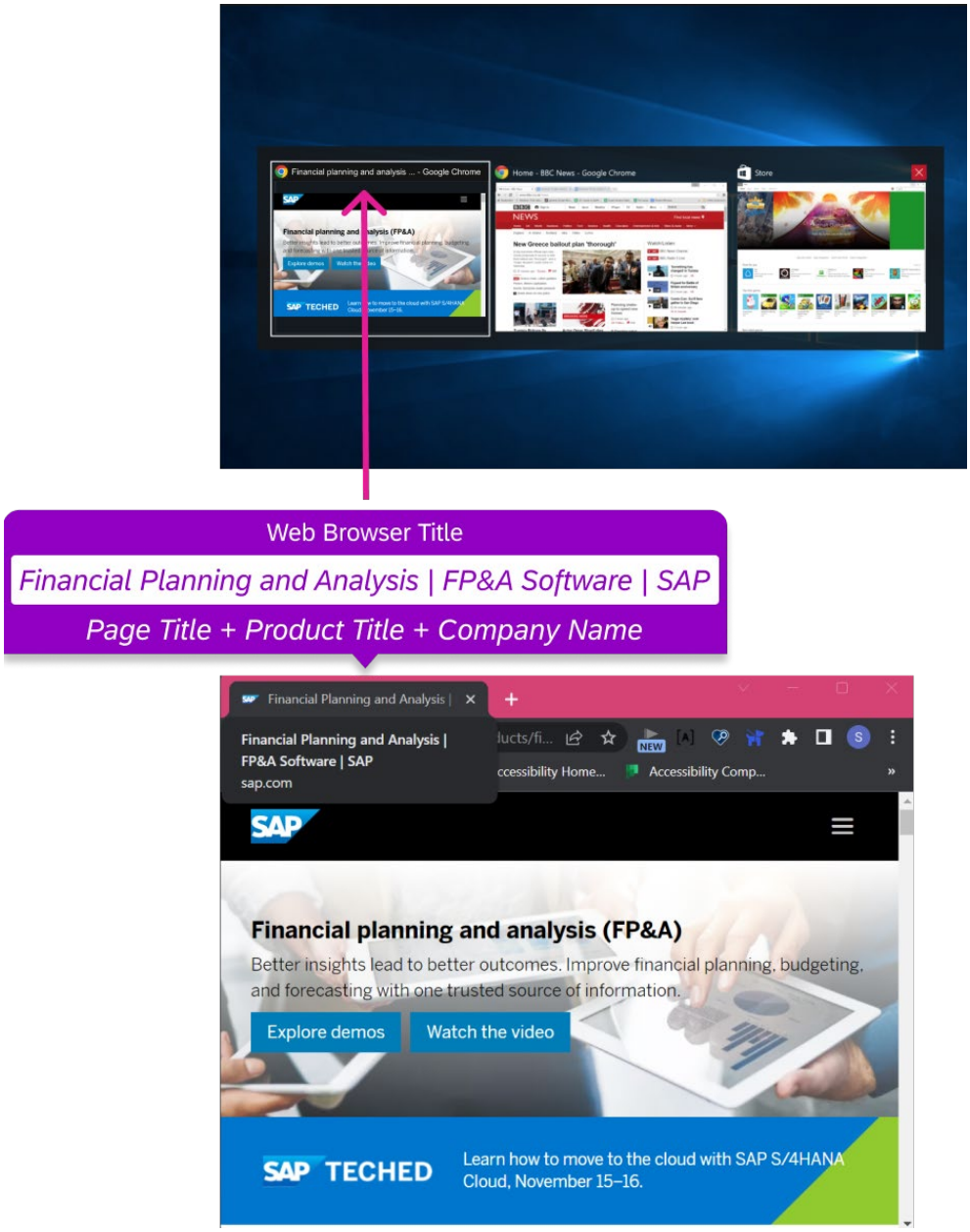


Figure 134: Web Browser window as part of open windows in OS Task Switcher overview. The title of the browser tab is labelling the respective task switcher item.

Browser Tab Text as Page Title

The browser Tab text is considered a page title. The example combines scattered elements in the UI and recommends the following composition:

- *Page Title + Product Title + Company Name*

It is important to be consistent across pages. Create a Page Title formula that follows your product or corporate guidelines, combining the same visible text elements across pages, and invisible text if applicable, to deliver a consistent experience across the product that helps users easily recognize page titles.

Once defined, this text is also used to identify the Window in the OS App switcher.

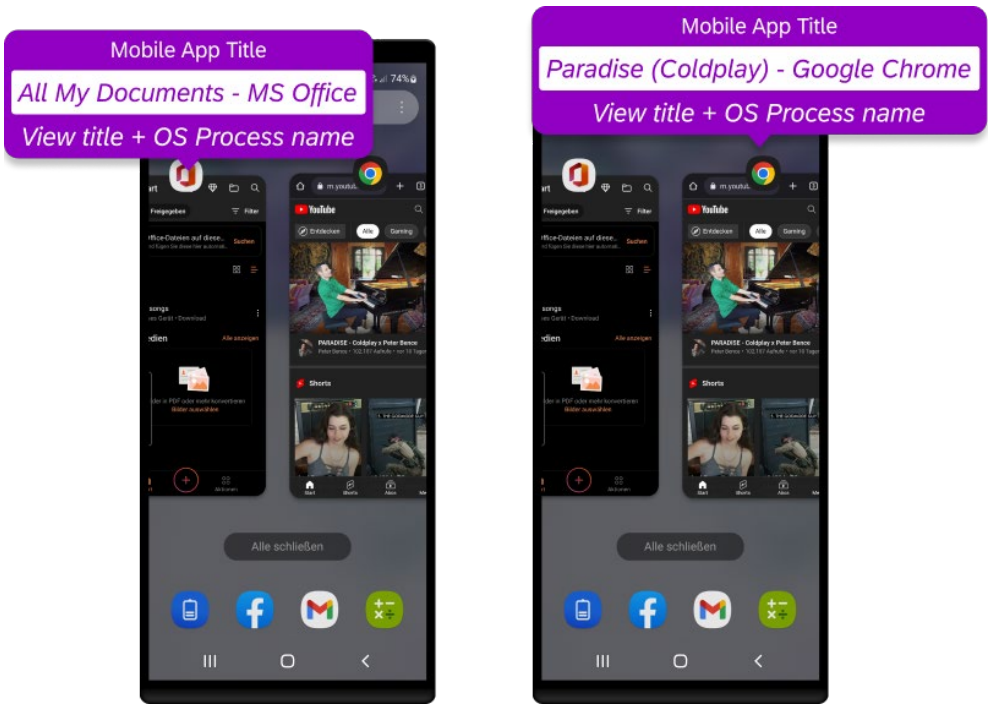


Figure 135: App window as part of open windows in OS Task Switcher overview. The title of the App window is labelling the respective task switcher item.

Mobile App View Titles

Main view titles should be used to indicate the app in the task switcher.

This page title is composed of:

- *View Title + OS Process name*

(“OS Process name” content may be appended by the OS automatically)

Shared Benefits

Temporary Disabilities

All users

Page titles improve usability for everyone by providing a clear, consistent reference for navigation, bookmarking, and multi-tab browsing, allowing all users to quickly understand and return to relevant content. A visible page or view title helps all users locate where they are among the many pages of a product.

For Developers

UI technologies have different message raising techniques:

If you are using an UI framework that possibly encapsulates one or more of the methods below, look for respective framework API property documentation instead.

Web Development

By adding title element to head of page

```
<!DOCTYPE html>
<html>
<head>
<title>Customer Sales Orders - SAP CRM</title>
</head>
<body> ...
```

By modifying the title element content dynamically via JavaScript:

```
top.document.title = "Customer Sales Orders - SAP CRM";
```

Mobile Development

iOS:

For entire app and single screen views

```
ContentView()
    .navigationTitle("My Title")
```

```
    .navigationDocument(
        myDocument,
        preview: SharePreview(
            "My Preview Title", image: myDocument.image))
```

See Configure your apps navigation titles | Apple Developer Documentation

Android:

For entire app and single screen activities/views

```
<application
android:label="@string/turns_up_in_manage_apps" >
<activity
android:name=".MainActivity"
android:label="@string/mainViewTitle" >
...
</activity>
</application>
```

See <activity> | App architecture | Android Developers

You can also use binding.rootView.announceForAccessibility("My fragment title") in onViewCreated() function of your fragment.

If you are not using viewBinding then just use the root view of the fragment's layout to call the function announceForAccessibility()

See AccessibilityInfo · React Native

References

WCAG 2.2 Reference

2.4.2 Page Title (A)

Role and Properties

Every design element must have a role name that follows common terminology and, when applicable, clear information about its properties.

Define the meaning and purpose of each UI element when designing the component or when using the UI element to design a screen.

Observe and inform clearly the UI component properties, including their states and attributes.

Component Variants

WAI-ARIA Widget Role

Figure 136: These interactive roles act as standalone user interface widgets/elements or as larger composite widgets/elements

WAI-ARIA Structure Role

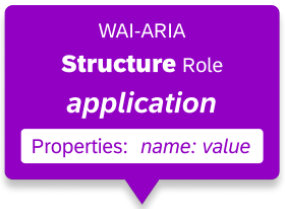


Figure 137: These roles describe structures that organize content in the user interface and are usually not interactive

WAI-ARIA Landmark Role

Figure 138: These roles are regions of the page intended as navigational landmarks

WAI-ARIA Live Region Role

Figure 139: These roles indicate live (=self-updating) regions in the user interface

WAI-ARIA Window Role

Figure 140: These roles act as windows within the browser or application

iOS UIAccessibilityTraits

Figure 141: Set these traits in native iOS apps to tell assistive apps how an UI element behaves or how to treat it

Android Roles

Figure 142: Set these roles in native Android apps to tell assistive apps how an UI element behaves or how to treat it

About Role and Properties

Connecting Roles and Properties Across Platforms

In both web and mobile interfaces, accessibility begins with clarity. This includes clarity of purpose, structure, and interaction. Roles and properties are foundational tools that help assistive technologies interpret and communicate the purpose of each element to users.

Whether it is a button, a status update, or a live region, assigning the right role ensures that the element behaves as expected across platforms. Properties, such as labels or state indicators, enrich that experience by adding meaningful context.

The annotations proposed in this section explores how roles and properties work together to provide accessible, predictable, and inclusive user experiences, on both the web and native mobile applications. While the syntax may vary between platforms, the principles of semantics, clarity, and user support remain constant.

We cover common patterns, key differences, and how to annotate these elements effectively for diverse user needs so that they can be implemented for accessibility.

Combined annotation

The Role & Properties annotation is used to specify the semantic role name assigned to an implemented control, along with relevant ARIA properties or mobile traits developers should apply. This ensures clarity in the expected functionality, behavior, and accessibility support of each component. The terms “role” and “properties” are explained in detail in next chapter.

Component designers are encouraged to reference commonly used WAI-ARIA roles for web and mobile platform traits and roles (iOS and Android) to guide their annotation process. This alignment ensures accessibility is planned early and correctly implemented across platforms.

For example, consider a navigation control that appears as a menu bar on desktop in full-screen view. On mobile devices, you might switch to a toggleable menu pattern. In such cases, the entire markup structure, and thus the role, changes. Rather than simply replacing a menu bar role with a menu role, you would likely restructure the component entirely to reflect the new interaction pattern. Annotating this shift early ensures the appropriate roles and properties are applied for both layouts, supporting accessibility without confusion.

1. Desktop – Menu Bar

Role: menubar - A menu bar is a container for menu items, typically used in desktop-style navigation.

Associated ARIA properties and states:

- aria-haspopup="true" (if a menu item opens a submenu)
- aria-expanded="false" or "true" (used dynamically on items that can be toggled open/closed)
- aria-controls (references the ID of the submenu controlled by the menu item)
- aria-label or aria-labelledby (gives an accessible name to the menu bar)
- aria-orientation="horizontal" (optional but clarifies layout)

Child roles: menuitem, menuitemcheckbox, menuitemradio

2. Mobile Device – Toggleable Menu

Role: menu (or simplified navigation role inside a toggleable container) – On small screen (like on mobile devices), navigation is often hidden behind a "hamburger" icon and expanded via user interaction.

Associated ARIA properties and states:

- aria-expanded="false" (on the toggle button, changes to true when menu is open)
- aria-controls="menuID" (to associate the toggle button with the menu container)
- aria-label="Main menu" (on the toggle button, describing its function) aria-hidden="true" (used on the menu itself when it is visually and semantically hidden)
- aria-labelledby (used to associate the menu with a visible label, if applicable)

Mobile platform traits:

- iOS: accessibilityTraits, for example, button, selected, header
- Android: button role, android:contentDescription with structural hierarchy hints

Role

A well-designed component communicates its function through both form and semantics.

One of the most critical elements in this process is the role, a standardized term. Role is a key part of accessibility. For blind users, the role of an element is understood through a screen reader in the same way sighted

users rely on visual cues. Just as a button's look tells a sighted person that it can be clicked, the announced role 'button' gives a blind user the meaning they need to decide the next best action that identifies the type of user interface element.

Component designers should rely on familiar, commonly understood role names to reinforce expected behaviors. Roles serve as the primary indicator of the purpose of an element. This semantic association enables assistive technologies to present and interact with components in ways that align with user expectations, enhancing both usability and accessibility.

When designing accessible components, assigning the correct ARIA role is not just a developer concern. It is a critical part of the design specification. Roles define the semantic purpose of UI elements, enabling assistive technologies to communicate what an object is and how users can interact with it.

For component designers, observing roles involves more than naming:

- Match roles to expected behaviors. Roles must always align with user expectations and keyboard interaction patterns. If a component looks and behaves like a button, it must have the button role, regardless of its visual design.
- Specify roles explicitly in design specs. Annotate custom components (e.g., tab lists, accordions, sliders) with their intended ARIA roles. This helps ensure consistent implementation across development teams and avoids accessibility regressions.
- Mind role restrictions. Some roles can only be used with specific parent or child roles (e.g., tab must be inside a tab list). Improper use can break accessibility.
- Use native elements first. Whenever possible, leverage semantic HTML elements (<button>, <table>, <nav>, etc.) that come with implicit roles. This simplifies development and ensures robust support across platforms and assistive technologies.

  Note: To indicate the corresponding native HTML element for developers, also use our WAI-ARIA annotations since there is role parity with direct 1:1 mapping between ARIA roles and HTML elements which can be studied in the following W3C specification: ARIA in HTML.

- Do not override native semantics unless necessary. Avoid removing or altering native roles (e.g., turning a <button> into a div with role="button") unless you have a specific accessibility enhancement in mind, and even then, ensure all necessary keyboard and ARIA support is added.

- Think cross-platform. Some ARIA roles are interpreted differently across web and native mobile screen readers. Be cautious when designing components that will be implemented across multiple platforms.

By proactively addressing roles during the design phase, you support predictable, meaningful, and accessible user interactions. This reduces ambiguity for developers and ensures inclusivity for all users.

Properties

In accessibility design, properties are vital in defining key characteristics of an object that assistive technologies rely on to interpret and communicate meaning to users. Properties are another essential part of accessibility. For sighted users, properties such as color, size, or state (like checked or disabled) are understood visually.

For blind users, those same properties are communicated through assistive technology—for example, announcing whether a checkbox is checked, or a button is disabled—so they can interpret the element's condition and decide the next best action.

Properties are another essential part of accessibility. For sighted users, properties such as color, size, or state (like checked or disabled) are understood visually. For blind users, those same properties are communicated through assistive technology—for example, announcing whether a checkbox is checked, or a button is disabled—so they can interpret the element's condition and decide the next best action.

A designer will inform when a control offers various combinations. For example, a button may include properties such as 'disabled,' 'expanded,' or 'pressed,' while a form field may include 'required' or 'invalid.' These can be implemented in different ways, such as using visual cues like color and icons for sighted users, and semantic properties such as aria-disabled or aria-required for screen reader users.

A component designer is responsible for accurately assigning all possible properties when specifying a component to be implemented, ensuring that both visual and non-visual users can understand its full range of behaviors.

WAI-ARIA properties such as aria-labelledby, aria-describedby, aria-controls, aria-colspan, and aria-autocomplete help describe relationships, structure, or behaviors that are not visually evident but crucial for screen reader users and other assistive tools.

While many properties tend to remain static (e.g., aria-labelledby typically points to a persistent label), others may change depending on user interaction. For instance:

- aria-activedescendant, aria-valuenow, and aria-valuetext are dynamic properties, often updated as the user interacts with a widget (e.g., sliders, combo boxes).
- Some, such as aria-multiline, are configuration-like properties, which are set at the time of design and are unlikely to change during use.

It is important to understand that while states and properties both start with the aria- prefix, they serve different purposes:

- States (e.g., aria-checked, aria-expanded, aria-hidden) reflect a condition that changes during interaction or automation.
- Properties describe attributes that often remain consistent, though exceptions exist (such as aria-expanded, which can also act as a state in interactive contexts).

Understanding and applying properties correctly ensures that user agents and assistive technologies can communicate clear, consistent meaning and relationships, which is critical to creating accessible, intuitive experiences, on both web and mobile platforms.

Let's now go through the examples and identify ARIA properties (and mobile traits and roles, where applicable) that would typically be associated with a menu bar on desktop and a menu or toggleable navigation on mobile.

Examples

WAI-ARIA Examples for annotating roles and properties of user interface elements in a web component inventory:

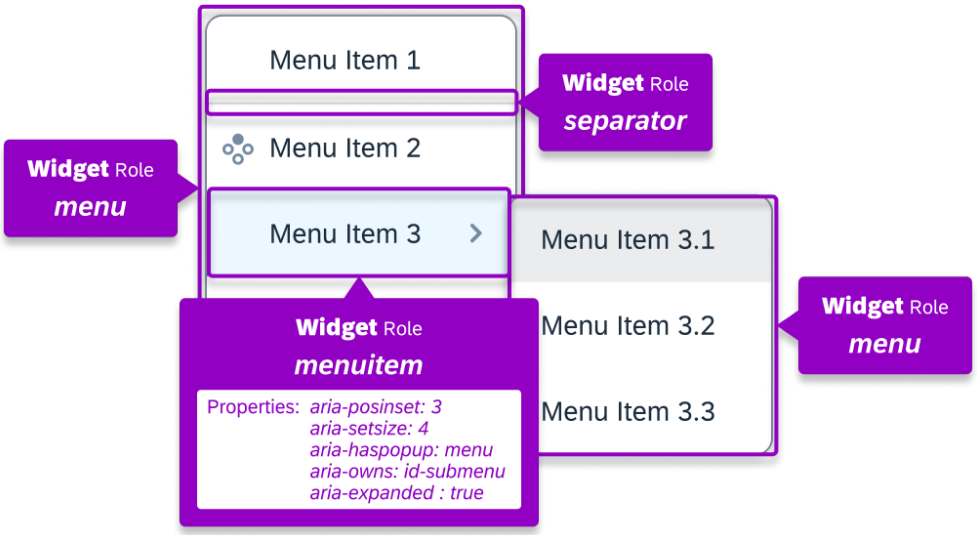


Figure 143: WAI-ARIA roles and properties for a menu with a submenu

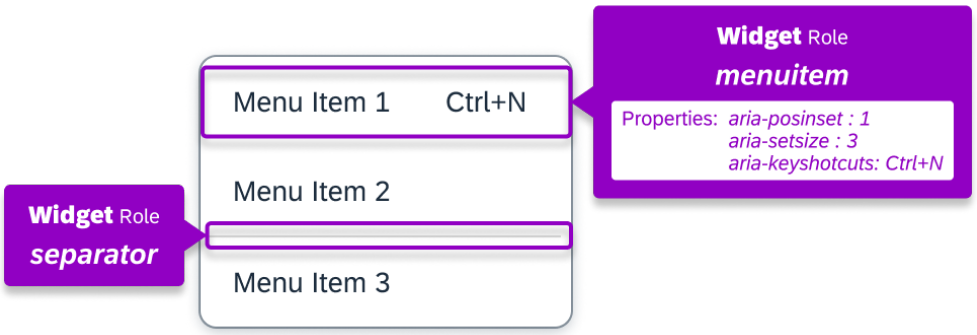


Figure 144: WAI-ARIA roles and properties for menu items

Figure 145: Non-Decorative interactive icon (acts on click)

Figure 146: Icon declared as decorative

Figure 147: Non-Interactive Icon (but needed for dragging operations as focal point)

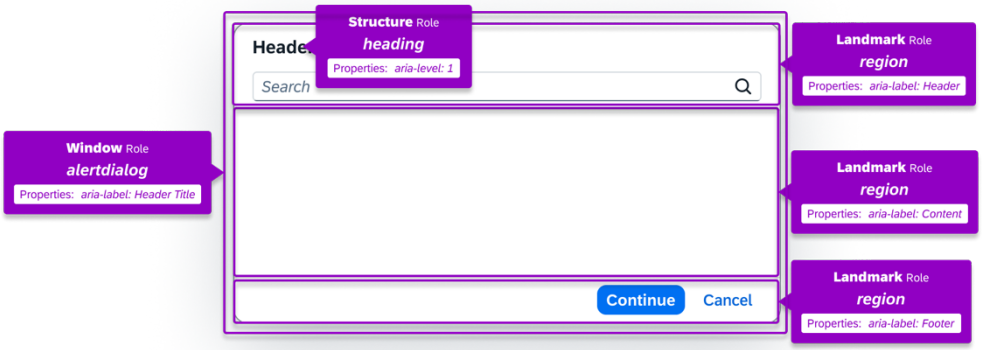


Figure 148: Section subdivisions in a floorplan for a large dialog

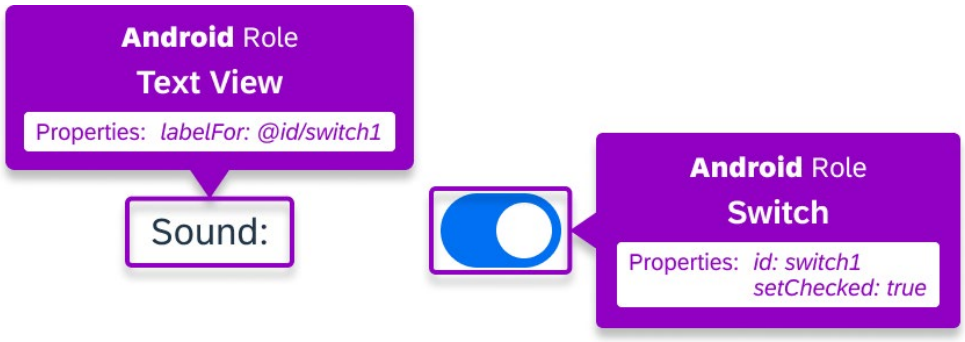


Figure 149: Role declaration and Label-control association for a custom Android switch component

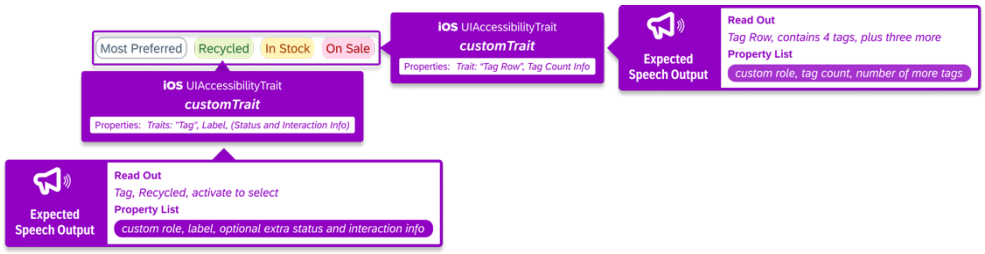


Figure 150: Accessibility Traits declarations for a custom iOS “Tag Row” component, using additionally the Expected Speech Output annotation for clarification

Shared Benefits

Temporary Disabilities

All users

Clear roles and properties help users understand what an element does and how it behaves. Even for sighted users, predictable behavior improves confidence, reduces errors, and supports efficient interaction, making the interface feel more intuitive and reliable.

For Developers

Please use the roles the designer assigned. When discussing your choices for control role assignments, please draw on your experience.

UI Frameworks

If you're using a UI framework that includes these roles, please refer to the framework's control documentation. You can then check if the control aligns with the intended use in the design.

For New or Unfamiliar Roles

If you don't have experience with a particular role, please take some time to review the platform documentation to get familiar with it.

Web Development

WAI-ARIA roles for Web Development

Accessible Rich Internet Applications (WAI-ARIA) 1.3

Table 10: **WAI-ARIA roles**

Widget Roles	Structure Roles	Landmark Roles	Live Region Roles	Window Roles
button	application	banner	alert	alertdialog
checkbox	article	complementary	log	dialog
gridcell	blockquote	contentinfo	marquee	
link	caption	form	status	
menuitem	cell	main	timer	
menuitemcheckbox	code	navigation		
menuitemradio	columnheader	region		
option	comment	search		
progressbar	definition			
radio	deletion			
scrollbar	directory			
searchbox	document			
separator (when focusable)	emphasis			
slider	feed			
spinbutton	figure			
switch	generic			
tab	group			
tabpanel	heading			
textbox	img			
treeitem	insertion			
combobox	list			
grid	listitem			
listbox	mark			
menu	math			
menubar	meter			
radiogroup	none			
tablist	note			
tree	paragraph			
treegrid	presentation			
	row			
	rowgroup			
	rowheader			
	separator (when not focusable)			
	strong			
	subscript			
	suggestion			
	superscript			
	table			
	term			
	time			
	toolbar			
	tooltip			

Mobile Development

In iOS, control roles are called traits, while Android uses the term “role” instead. For correct usage of these roles, please consult the respective platform documentation.

iOS Traits

- UIAccessibilityTraits at Apple Developer Documentation

Android Roles

- Android accessibility: roles and TalkBack - TetraLogical
- Talkback accessibility roles

Table 11: iOS Traits and Android roles

iOS Traits	Android Roles
adjustable	Action Bar Tab
allowsDirectInteraction	Alert Dialog
button	Button
causesPageTurn	Check Box
header	Checked Text Preview
image	Date Picker
keyboardKey	Date Picker Dialog
link	Drawer Layout
none	Drop Down List
notEnabled	Edit Text
playsSound	Horizontal Scroll View
searchField	Icon Menu
selected	Image
startsMediaSession	Image Button
staticText	Keyboard Key
summaryElement	List
supportsZoom	Number Picker
tabBar	None
toggleButton	Pager
updatesFrequently	ProgressBar
	Radio Button
	Role Grid
	Scroll View
	Seek Control
	Sliding Drawer
	Staggered Grid
	Switch
	Tab Bar
	Talkback Edit Text Overlay
	Text Entry Key
	Time Picker
	Time Picker Dialog
	Toast
	Toggle Button
	View Group
	Web View

Custom Components

Sometimes it is necessary to create custom components, instead of using the available controls, and assign roles to them.

iOS:

Make your component to be an AccessibilityContainer then give it a label, a group (if needed), child elements, etc.

Custom components require much effort, see videos and documentation:

- https://developer.apple.com/videos/play/wwdc2021/10119/
- https://developer.apple.com/videos/play/wwdc2021/10122

Android:

How to make accessible components

1. Extend existing view or subclass with own class
2. Override the methods from the superclass
3. Handle directional controller clicks
4. Implement accessibility API methods
5. Send AccessibilityEvent objects specific to your custom view
6. Populate AccessibilityEvent and AccessibilityNodeInfo for your view
7. Add custom actions for components

```
ViewCompat.setAccessibilityDelegate(set_actions_button
, object : AccessibilityDelegateCompat() {
override fun onInitializeAccessibilityNodeInfo(v:
View, info: AccessibilityNodeInfoCompat) {
super.onInitializeAccessibilityNodeInfo(v, info)
info.addAction(AccessibilityActionCompat(
AccessibilityNodeInfoCompat.ACTION_CLICK, "Edit
note"))
info.addAction(AccessibilityActionCompat(
AccessibilityNodeInfoCompat.ACTION_LONG_CLICK, "Copy
note"))
}
})
```

8. Add custom role names for components

```
ViewCompat.setAccessibilityDelegate(login_text_view_as
_button_with_role,
object : AccessibilityDelegateCompat() {
override fun onInitializeAccessibilityNodeInfo(v:
View, info: AccessibilityNodeInfoCompat) {
super.onInitializeAccessibilityNodeInfo(v, info)
info.roleDescription = "Button"
}
})
```

See also:

https://developer.android.com/develop/ui/views/layout/custom-views/custom-components

https://developer.android.com/guide/topics/ui/accessibility/custom-views

https://developer.android.com/codelabs/advanced-android-kotlin-training-custom-views#0

https://tetralogical.com/blog/2022/07/07/android-accessibility-roles-and-talkback/

https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary#Slider

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

4.1.2 Name, Role, Value (A)

Speech Output

Plan the Speech output by describing what a screen reader would read out during keyboard focus.

Component Variants

Speech Output

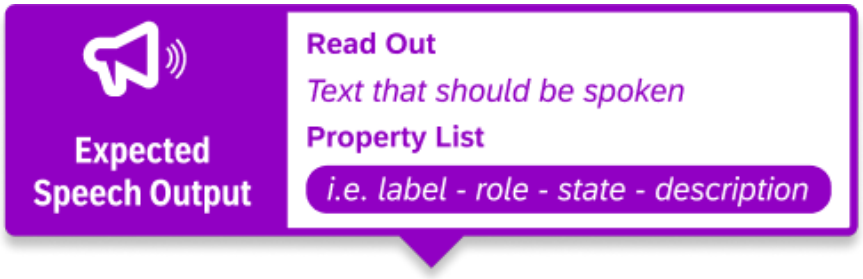


Figure 151: Describe what a screen reader should announce on element focus

About Speech Output

Accessibility needs to be addressed at every level of a webpage, from the big picture down to the smallest details.

Think of it this way: a whole page needs to be accessible so users can explore and understand the content on a macro level, like navigating between major sections. Similarly, individual components, such as a button, form field, or dropdown menu, must also be accessible to ensure users can perform micro-interactions successfully.

In short: a page is for exploring, and a component is for interacting. Both must be accessible for the user to have a smooth experience.

Simulating screen reader announcements is a powerful way to verify the usability of a digital interface for users who are blind or have low vision. By translating the visual design into a text-based script, we can test whether the announcements are logical, coherent, and helpful for task completion.

The screen reader does not just read the words on the screen; it announces a combination of elements that provide a complete picture of the interface. This includes **landmarks** that act as navigational signposts, such as "main" or "footer," helping users quickly move between major sections of a page. **Headings**, announced as H1, H2, and so on, create an outline of the content, allowing users to scan and jump to specific topics. The **roles** of elements, such as "button," "link," or "checkbox," are announced to tell the user what they are interacting with. Additionally, **states and properties**

provide crucial information about the current condition of an element (e.g., "checked" or "collapsed"), while **labels and names** are the actual text that the screen reader reads, like "submit" for a button. This combination of announced elements is what truly defines the user experience for someone relying on a screen reader.

To create an effective screen reader announcement script, you must step through the design as a screen reader user would. For each interactive element or section, write down exactly what the screen reader would announce, combining all the relevant information such as the role, state, and name. For example, a **"Sign in**" button would be annotated as "*Sign in, button*," (where the first is the visible label and the second is the role) and a "**Remember me**" checkbox might be "R*emember me, checkbox, not checked*" (where the first is the visible label, the second is the role and third is the property).

Using the speech output annotation enable designers and developers to test the flow and logic of the announcements, ensuring that the language is clear, the order is correct, and the announcements make sense in the context of completing a task. This process helps identify potential accessibility issues before the code is even written, ensuring a more inclusive and usable experience for everyone.

The speech output simulation will help designers answer:

- Is the order correct?
- Is the language clear?
- Do the announcements make sense in the context of completing a task, like logging in?

This process helps identify potential accessibility issues before the code is written, ensuring a more inclusive and usable experience for everyone.

Speech Output is an example of a screen reader reading out according to the component attributes. However, a simulated speech output is just an expectation expressed by component and application designers. There is no guarantee that implementation will correctly map the text to how a screen reader will read it out.

A sequence of properties recommended consists of the following:

- LABEL(S): visible or invisible associated label(s)
- ROLE: component type/role (such as “button”)
- VALUE: contained value(s) (such as numbers or text)
- STATE(S): state(s) (such as “disabled”)
- OTHER PROPERTIES: other properties (such as “required” or the placeholder)

- DESCRIPTION(S): visible or invisible description(s) suggesting some helpful information to the user such as units (“EUR”)
- HOT KEY: assigned hot keys (such as “Ctrl+S” for saving operation)
- ACCESS KEY: assigned access keys (such as “Alt+S”)
- TUTOR INFO: additional info for blind users (other keys to use)

When crafting content for screen readers, remember that brevity is key. Blind and low-vision users listen to every word, and excessive detail can quickly become tedious.

Your goal is to provide just enough information for them to understand and complete their tasks without making the application overly "chatty." **Less is more**. Each word should be intentional, concise, and purposeful to ensure a smooth and efficient experience.

Examples

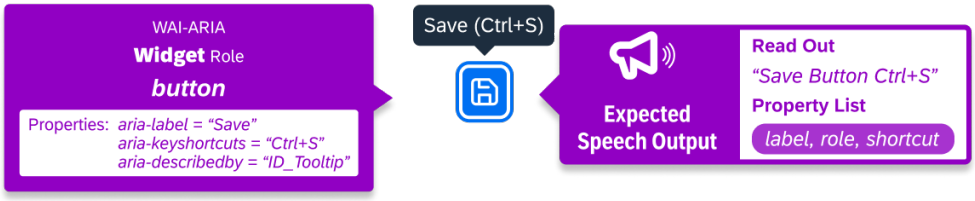


Figure 152: Expected speech output on keyboard focus for a custom tooltip of an icon button as combination of hidden button label and keyboard shortcut

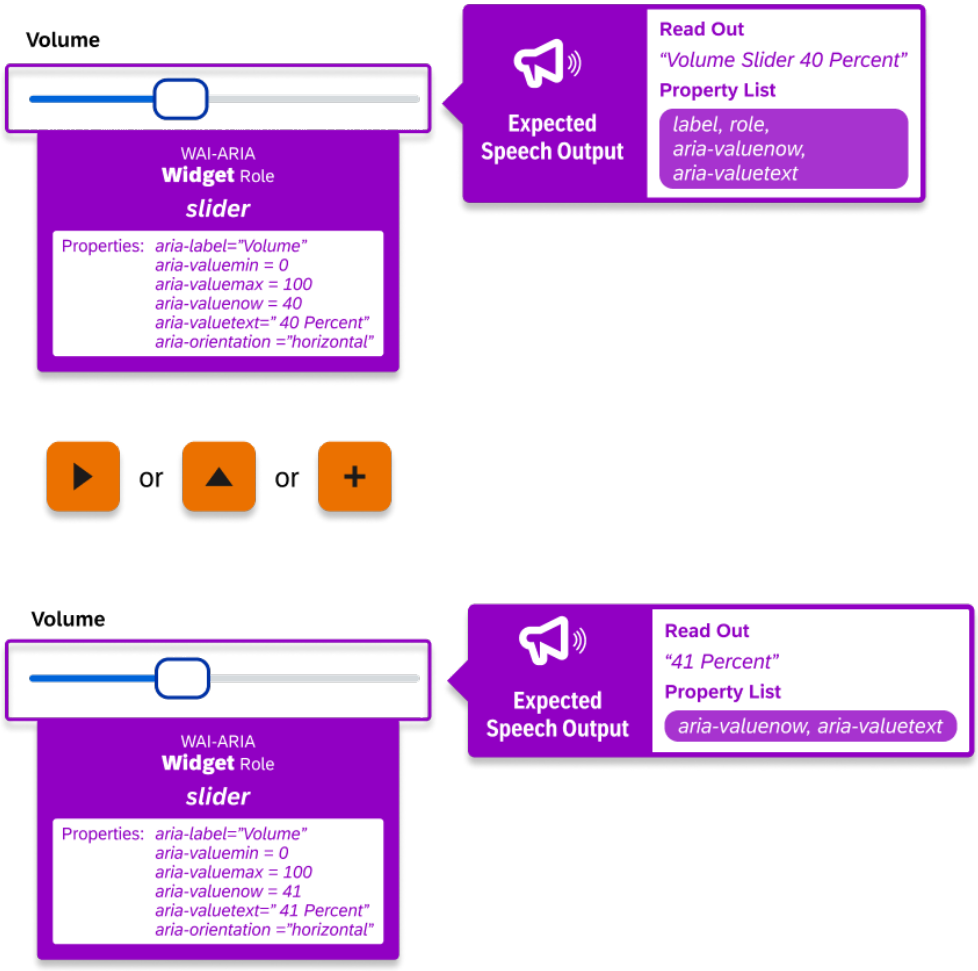


Figure 153: Expected speech output for a change in a value of a focused volume slider control (combined with Role and Property annotations, arrow right/up plus keys increment value by one)

Shared Benefits

Temporary Disabilities

All users

The Speech Output annotation simulates the announcements a screen reader would make, allowing designers and users to preview how content will be conveyed audibly. Simulated speech output helps designers verify content clarity, labels, and interaction feedback, indirectly benefiting everyone by ensuring interfaces are intuitive, consistent, and accessible.

Vision and Color Limitation

Visuals

Some users with partial sight use screen readers to complement visual information. Speech output reinforces comprehension, helps confirm actions, and supports interaction with complex interfaces.

For Developers

What you get from the annotation is a rough expectation of the individual parts of the speech string to be spoken, it is NOT a binding command with respect to the sequence of the different parts.

Screen readers on the different platforms have built-in heuristics in which they speak the roles and attributes. Do not try to change that by artistically juggling with properties and over-denote property meaning and content.

For instance, it is not appropriate to put descriptions or even object values in object labels. Instead, use description properties for these cases.

Follow the rules for correct role assignment and attributing given in the API documentation for web and mobile development, test with a screen reader and user agent combination of choice on the platform and evaluate if the result fits to the output the designers have given you as speech expectation.

References

WCAG 2.2 Reference

1.3.5 Identify Input Purpose (AA)

2.5.3 Label in Name (A)

3.1.5 Reading Level (AAA)

4.1.2 Name, Role, Value (A)

4.1.3 Status Messages (AA)

Audio Control

Keep sound auto play under 3 seconds and provide mechanisms to pause or stop, or to control volume.

Component Variants

Audio Control Pause – Stop

Figure 154: Stop and Pause helps users prevent screen reader announcement overlap with background sound.

Audio Control - Volume

Figure 155: Volume controls mainly benefit blind and cognitive users, not those with hearing loss.

About Audio Control

If any audio on a Web page plays automatically for more than 3 seconds, either a mechanism is available to pause or stop the audio, or a mechanism is available to control audio volume independently from the overall system volume. Any user, including those who are hard of hearing, neurodivergent, or easily distracted must be able to identify, control, stop, or muting these sounds.

The WCAG Success Criteria 1.4.2 Audio Control (A) is meant to provide mechanisms to stop or pause sound or to make available an audio volume control independently from the overall system volume level. Blind users need control over audio playback and descriptions of visuals because they find unexpected audio disorienting due to sound overlap, which blocks screen reader from accessing and announcing content and context.

People who use screen readers rely heavily on audio for navigation. Auto-playing media with sound can be a major barrier for screen reader users. The unexpected audio overlapping sounds from auto-playing media and screen reader announcements can create confusion, cognitive overload, or even make it impossible to understand either stream.

Sound control helps prevent interference, allowing users to manage the timing and presence of audio.

If any audio plays automatically for more than 3 seconds, the sound can overlap with screen reader announcement, making it hard or impossible for blind users to hear what the screen reader is saying. To avoid this chaotic scenario, provide a mechanism to pause or stop the audio, or warn the user given them an opportunity to control audio volume independently from the overall system volume.

Why does this matter? Screen reader users rely on audio output. Overlapping sounds (from autoplay videos, music, or alerts) can interrupt or obscure the screen reader.

When documenting audio in the design make controls accessible to provide an accessible experience to all users. Avoid auto playing audio. If autoplay is necessary, ensure it lasts less than 3 seconds, or provide a visible and programmatically accessible control to pause or stop the audio. This ensures blind users can use their screen readers to access content confidently without interference.

Compliance with accessibility guidelines requires sounds to be available to a broader range of accessibility needs:

- controls are accessible and navigable via keyboard.
- labels are clear for screen reader users.
- UI element from control is focusable.
- sound is paired with captions or transcripts.

Controls:

- Provide visible, easy-to-use controls to Play and Stop media content.
- Controls should be clearly labeled (e.g., "Play" and "Stop") and
- accessible via keyboard and screen reader.
- Ensure the controls are prominent in the UI, so users can quickly pause or stop content as needed.
- Allow for easy muting of audio or adjusting volume when content includes background sound, music, or noise that might interfere with captions or reading.

Note: In addition to voice announcements, screen readers use a variety of sounds to provide non-verbal feedback. These audio cues are an essential part of the user experience, signaling everything from entering a new page to the successful submission of a form. They complement the spoken words to give users a complete understanding of what is happening on the screen. Sounds, like the Windows notification chime or a pop-up alert can often be disruptive and confusing because they are not part of the

consistent language of the screen reader, making it difficult to distinguish between helpful assistive technology cues and random system alerts. For users who depend on these technologies, a clear and consistent sound design is as critical as the spoken words themselves.

Examples

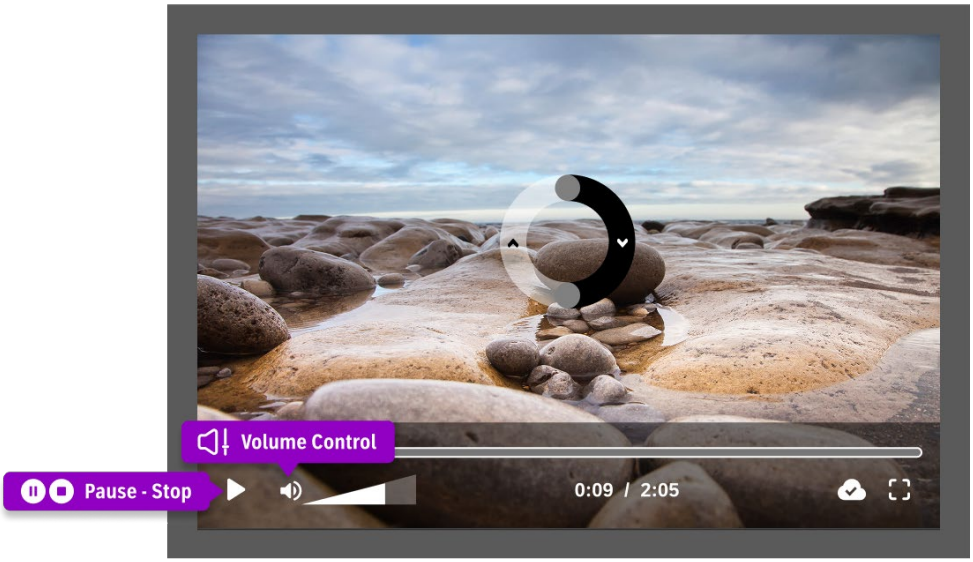


Figure 156: Volume of automatically played audio content can be changed independently of the system volume. Videos can be paused or stopped.

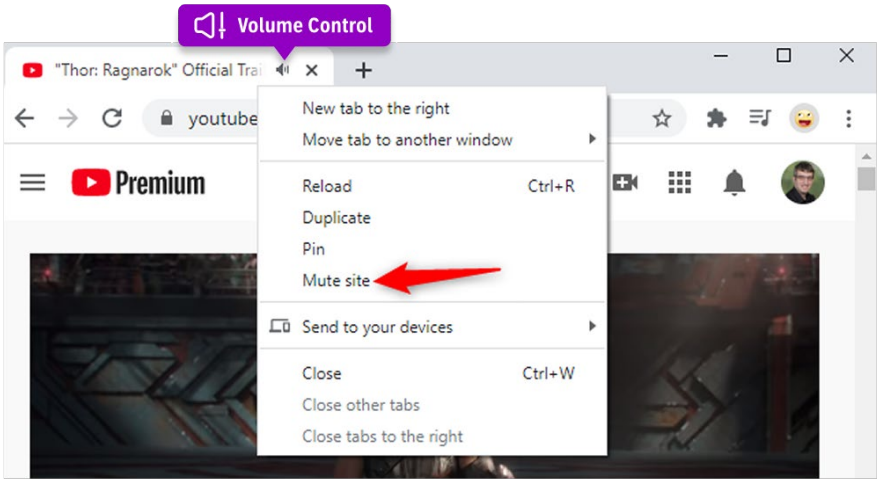


Figure 157: Automatically played audio content can be muted

Shared Benefits

Vision and Color Limitation

Visuals

Some low vision users rely on screen magnifiers and audio cues. Being able to pause or control audio allows them to consume information at their own pace, without being rushed by background media they can't quickly identify or stop.

Limited Cognition

Cognitive

For users who are sensitive to unexpected sounds, such as people with autism, ADHD, or anxiety, unanticipated audio can be distressing or disruptive. Sound controls give them the option to avoid sensory overload, helping them stay focused on the task. Reducing unwanted audio by muting or stopping the audio entirely helps reading captions or reviewing a transcript. Clear Play/Stop or Mute controls ensures these users are not forced to listen to audio content they may not need, enhancing their experience.

Without or Limited Hearing

Sound

Deaf and hard-of-hearing users need text or sign-based alternatives to sounds - audio alone is not accessible. These users rely on visual alternatives such as captions and transcripts. But accessibility is not just about providing those alternatives. It is also about giving control. Include clear Play, Pause, Stop, and Mute controls for audio and media content so that users read at their own pace while reviewing captions or transcripts. Without these controls, users may miss key information. Let them control playback timing, mute or skip audio they do not need, reducing distraction or overload. Sound control supports autonomy, improves comprehension,

and respects user preferences, especially for those navigating media content visually rather than aurally.

For Developers

Use the audio controls for the platform or framework inventory components triggering respective sound output API properties on the platform.

References

WCAG 2.2 Reference

1.4.2 Audio Control (A)

Audio Description

Use the “Audio Description” annotation to indicate that key visual content is described in audio or a separate track, ensuring users who are blind or have low vision can fully understand the media.

Component Variants

Audio Description

Figure 158: Available audio describes key visual content inside or in a separate track

About Audio Description

Audio Description is a required requirement to describe important visual details not spoken in the main audio. It is meant for technical writers to provide description to non-spoken dialogs. Screen readers cannot handle this task yet but maybe AI will soon help screen readers provide complementary description to important visual details from videos.

To make videos accessible to people who are blind or cannot understand visual content, provide synchronized or extended audio descriptions, spoken explanations of key visual elements. These descriptions ensure users can follow what is happening on screen even without seeing it.

An audio description must be provided for any visual content in the video that is not conveyed through the existing audio (e.g., silent actions, facial expressions, on-screen text). This description is typically added as a secondary audio track that fits into pauses in the main narration.

When standard pauses in the video are not enough to convey essential visuals, extended audio descriptions must be added to meet accessibility requirements. This supports compliance with WCAG Success Criteria 1.2.5 Audio Description (Prerecorded) (A) and 1.2.7 Extended Audio Description (Prerecorded) (AAA), which require audio descriptions for all pre-recorded synchronized media. Extended Audio Description highlights temporal accommodation for more detailed descriptions and requires pausing video to insert longer descriptions of visual content. WCAG 1.3.3 Sensory Characteristics ensures that information that is not conveyed by visuals alone is also accessible in non-visual formats.

If your design provides an audio description and a text description, there is no need to fulfill the 1.2.8 Media Alternative (Prerecorded) (AAA) requirement.

Examples

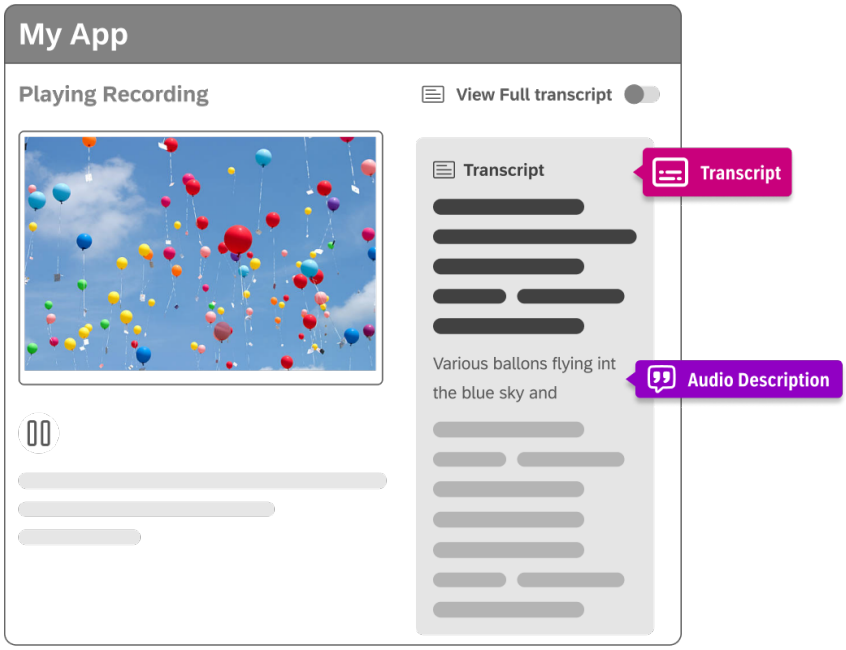


Figure 159: Synchronized Audio Descriptions as text alternative for non-text content (video) as part of a transcript

Shared Benefits

Temporary Disabilities

All users

Providing audio description benefits everyone. It improves comprehension in situations where users cannot look at the screen (multitasking, hands-free contexts, or all-screen devices). For second-language learners, audio description reinforces meaning by explicitly naming actions, settings, or expressions that might otherwise be missed. Audio description enhances usability and supports inclusive learning.

Limited Cognition

Cognitive

Audio description supports people with cognitive disabilities who may rely on additional narration to follow complex visual cues.

For Developers

Use framework inventory components containing or linking to respective transcript and audio description resources on the platform.

References

WCAG 2.2 Reference

1.2.5 Audio Description (Prerecorded) (AA)

1.2.7 Extended Audio Description (Prerecorded) (AAA)

1.3.3 Sensory Characteristics (A)

Language

Specify the default language for the user interface and identify where language changes on the user interface are present.

Component Variants

Page Language

Figure 160: Specify the default language of a page

Part Language

Figure 161: Specify where language changes are expected

About Language

Success Criteria 3.1.1 Language of Page (A) and 3.1.2 Language of Parts (AA) ensure correct assignment of primary language of a webpage or any change in language within a page. The goal is to consider multilingual content by ensuring the primary language is clearly defined for screen readers and other assistive technologies. Planning language of page and language of parts prevents miscommunication across globalized interfaces.

Examples where language may be considered in the page layout are:

- quote or sentence in French within an English page (e.g., Bonjour)
- greetings in a banner that updates the language upon page refresh (e.g. Bienvenido, Bienvenue, Willkommen, Benvenuto)

Why It Matters: Screen readers rely on this to read content with correct pronunciation and intonation. It also important for screen readers to pronounce foreign phrases correctly, while supporting language learners and global users. For public websites it improves SEO and machine translation quality.

As a designer you should observe and specify the following:

- Default Page Language: Define the main language for the entire page or screen. Indicate the default language in the design documentation to guide developers to set the lang attribute in the HTML tag. Use consistent language throughout the page to avoid confusion for users relying on assistive technologies. If designing for localization, specify which version of the design corresponds to which language (e.g., English, French, Spanish).
- Part Language: Indicates the human language of each passage or phrase. Designers must identify any text that is not in the main language of the page and indicate the content within the page that contains the non-prevalent language.
- Flag it as a different-language element that needs to be marked with the correct language attribute. Avoid mixing languages without clear separation in UI.

Users relying on Assistive Technologies or Smart Agents benefit from voice assistants (like Siri or Google Assistant) and screen readers use lang tags to determine pronunciation, pitch, and grammar parsing. This also affects speech input systems (for example, dictation in a bilingual context), which may misinterpret words without a clear language definition.

Examples

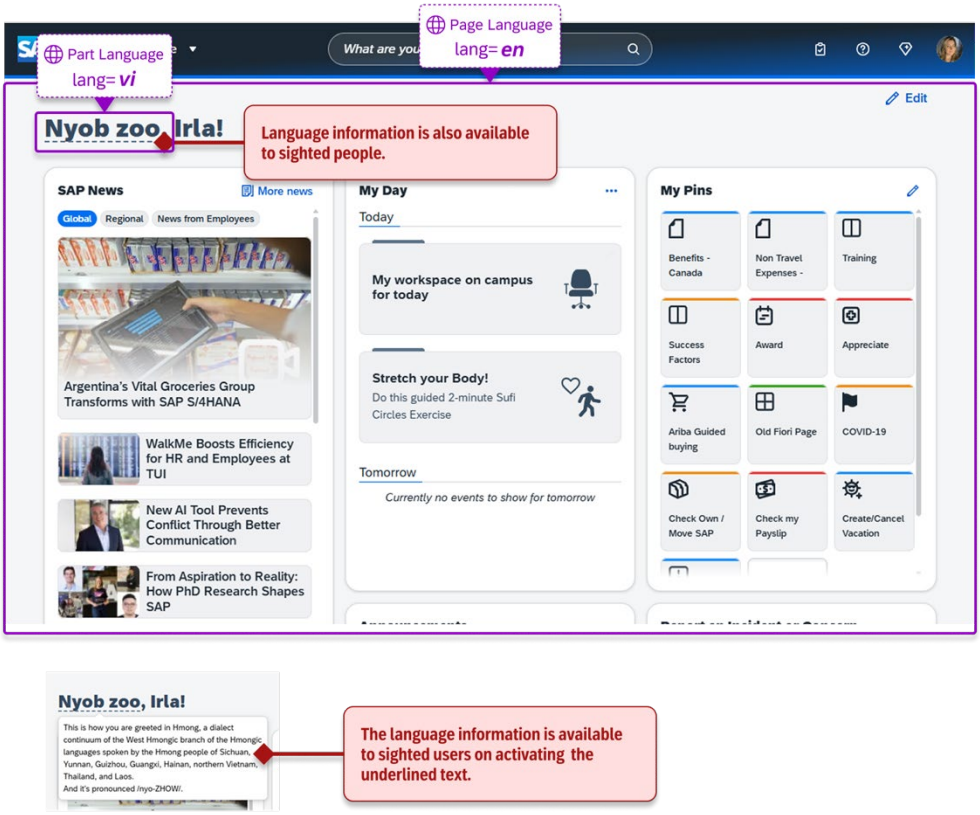


Figure 162: Page is in English but contains quotes in other languages

Shared Benefits

Limited Cognition

Cognitive

Those with cognitive and learning disabilities may use text-to-speech (TTS) tools that rely on correct language metadata to pronounce words correctly. For example, someone with dyslexia might use TTS in English, and without lang="es", a phrase like “¿Cómo estás?” might be read in a garbled way. Proper language cues help comprehension and reduce confusion.

Temporary Disabilities

All users

Multilingual or Non-Native Speakers may rely on machine translation tools (like Google Translate), which perform better when the language is explicitly defined. Clear lang attributes improve translation accuracy and grammar context, making information more accessible. Helps language learners navigate hybrid-language pages (e.g., English sites with French quotes).

Sighted Users using read-aloud tools such as Read Aloud, Natural Reader, or native browser TTS (e.g., Safari's Reader Mode) rely on language tags to deliver natural-sounding speech. Incorrect language attribution can create frustration due to robotic or incorrect pronunciation. This also applies for sighted users listening instead of reading.

Organizations and Legal/Compliance Teams: while not a “user” group, this requirement serves legal departments trying to mitigate risk and demonstrate inclusive practice. Mispronounced phrases due to missing language declarations can be used in legal cases to show lack of due diligence, especially in public sector or multilingual countries.

For Developers

Web Development

Developers should ensure proper markup to use the `lang` attribute for specific sections.

On body tag (for page language, all UI text) and in page (for parts of UI text):

```
<body lang=”en”>
Search results:
<a href=”oldage_german.doc” lang=”de”>
Die Geschichte des Mittelalters</a>
<a href=”us_history.doc” >
American History</a>
</body>
```

Mobile Development

As of today, there is no concept for part language identification in mobile platforms.

References

WCAG 2.2 Reference

3.1.1 Language of Page (A)

3.1.2 Language of Parts (AA)

Cognitive Experience

Imagine looking at a screen with lots of information and feeling confident about where to start. What turns a confusing interface into an empowering experience? The answer is a predictable interface that is easy to use and navigate. This allows users with cognitive disabilities and neurodivergent users to build mental models and accomplish their goals successfully.

This chapter shows designers how to create a clear and predictable experience that reduces cognitive load for everyone, especially people with cognitive or learning disabilities.

The Inclusive Cognitive Experience

A cognitive experience is the way our mind takes in information, pays attention, remembers, learns, uses language, and solves problems. It is how we process information and make sense of the world.[19]

Here are a few examples:

- **Perception**: noticing a red button on the screen.
- **Memory**: recalling where you find vegetables in the online store.
- **Attention**: focusing on a lecture despite background noise.
- **Reasoning**: figuring out the solution to distribute the compensation budget.
- **Decision-making**: choosing between 5 training courses in a carousel.

Design your application to be clear, predictable, and simple for everyone to understand, especially users with cognitive disabilities and neurodivergent users.

Users with cognitive disabilities and neurodivergent users benefit from interfaces that are clear, predictable, and easy to navigate. When interfaces support different ways of thinking and processing information, users can accomplish their tasks confidently through predictable, well-structured design.

Consistent navigation and layout enable all users to build mental models and navigate with confidence. This predictable structure is especially crucial for users with cognitive disabilities and neurodivergent users, allowing them to focus fully on accomplishing their intended tasks.

Users with cognitive disabilities and neurodivergent users particularly benefit from clear, step-by-step guidance that helps them move through complex processes without feeling overwhelmed. Plain language, paired with visual cues such as icons or tooltips, makes content easy to understand. Similarly, keeping critical information visible eliminates the need to remember previous steps or context, enabling successful task completion.

Customizable interfaces enable all users to tailor the interface to meet their specific needs, whether that involves reducing motion, adjusting text size, or simplifying layouts. This ability to customize helps ensure successful, empowering experiences for everyone, especially for neurodivergent users and those with cognitive disabilities.

Principles

Neurodiversity

Persona: Chloe

"I experience memory loss, difficulty focusing, trouble solving complex tasks, and occasional confusion or disorientation. My dyslexia, dyscalculia, and ADHD affect my concentration, attention, and ability to learn. I prefer using different digital helpers such as tooltips, Agentic AI, in-application help, and product documentation."

Cognitive disabilities encompass a range of permanent conditions such as **ADHD, dyslexia, and learning or developmental disabilities**, which can impact how individuals think, process information, and learn.
These conditions often affect memory, problem-solving, attention, and the ability to understand complex concepts.

This user group represents individuals with dyslexia, dyscalculia, and ADHD, who rely on helpers and documentation in digital products to support their focus, attention, and learning. We use the term Neurodivergent to talk about cognitive challenges.

The term "neurodivergent" is often used as an umbrella description for people with a variety of neurological or cognitive differences, which may include conditions like ADHD, autism, dyslexia, or dyspraxia. It is the opposite of the term "neurotypical", which is commonly used to describe individuals whose thinking or behavior differs from what society often considers typical. These individuals may have unique cognitive strengths and perspectives that contribute to diverse ways of thinking, learning, problem-solving and socializing. Neurodiversity includes all human brains; after all, everyone experiences the world in a slightly different way.

Some statistics:

- 1 in 6 people worldwide live with a neurological condition that can affect cognitive functioning.[21]

- Around 13.9% of adults in the US report having a cognitive disability, defined as serious difficulty concentrating, remembering, or making decisions.[5]
- Dyslexia is estimated to affect up to 7% of the population.[18]
  ADHD affects approximately 8% of children and adolescents and 3% of adults worldwide.[14, 16]

Situational & Temporary Disabilities

By integrating situational and temporary disabilities into this disability group, we help design teams address challenges that might not be permanent but still impact user experience under certain conditions.

Situational and temporary:

- Stress-induced cognitive overload: Situations of high emotional stress (e.g., work deadline or personal crisis) could impair cognitive processing, making it hard to focus on tasks.
- Fatigue: A lack of sleep, burnout, or exhaustion could impair concentration or memory, like cognitive disabilities.
- Multitasking demands: In busy environments, such as open offices or crowded spaces, cognitive overload can temporarily impact the ability to process information effectively.

General Design Tips

1. **Clear and Predictable Interfaces:**
   Sudden layout shifts, ambiguous flows, or inconsistent patterns increase anxiety and disrupt focus.

   Design Tip: Keep navigation, layout, and terminology consistent. Use progressive disclosure and prepare users for change through tooltips or banners.

2. **Step-by-Step Guidance:**
   Difficulty solving problems, remembering previous steps, or interpreting system feedback.

   Design Tip: Break tasks into smaller steps. Use clear instructions, plain language, familiar icons, and specific feedback messages.

3. **Sensory-friendly spaces:**
   Sensory sensitivity and difficulties with focusing or maintaining attention. Visual clutter, motion, or loud sounds can trigger anxiety or distraction.

Design Tip: Minimize anxiety triggers like visual clutter and motion, while also recognizing that some users may feel anxious in overly calm environments. Offer customizable options to create a sensory-friendly space allowing each user to tailor the experience to their specific needs: include distraction-free modes, use soft, muted color palettes, and control luminance.

4. **Reduced Reliance on Memory:**
   Users may forget context, instructions, or prior steps. Working memory limitations can lead to errors or abandonment.

   Design Tip: Keep critical info visible (e.g., task summaries, form labels). Use persistent headers or indicators to anchor users in the flow.

5. **Visual Structure and Organization:**
   Disorganized interfaces make it harder to track progress or complete tasks.

   Design Tip: Use progress bars, checklists, or visual groupings to clarify structure. Minimize decisions per screen.

6. **Plain Language:**
   Complex language or vague content increases cognitive load and misunderstanding.

   Design Tip: Use short sentences, familiar terms, and bullet points. Highlight key messages and avoid jargon.

7. **Routine and Familiarity:**
   Unexpected design changes disrupt mental models and create stress

   Design Tip: Maintain predictable behavior. If change is necessary, announce it clearly and give support.

8. **Customizable Experience:**
   Cognitive needs vary widely; therefore, a one-size-fits-all design approach may not meet everyone's needs effectively.

   Design Tip: Let users adjust layout density, motion, font size, and colors. Whenever possible, save preferences across sessions.

9. **Visual Support for Comprehension:**
   Trouble understanding abstract UI elements or instructions.

   Design Tip: Use clear labels, icons with text, and simple metaphors. Reinforce meaning visually and verbally.

10. **Focus-Friendly Layouts:**
    Difficulty sustaining attention due to cognitive overload or distractions.

Design Tip: Design clean interfaces with a clear hierarchy. Group related items, reduce noise, and avoid multitasking traps.

Annotations

Cognitive Annotations

The checklist assists designers in identifying and addressing key accessibility considerations during the design phase.

Cognitive Experience Checklist

- [ ] Wayfinding and Orientation
- [ ] Semantic Strategies
- [ ] Error Handling
- [ ] Motion Content
- [ ] Time Limit
- [ ] Multiple Ways
- [ ] Redundant Entry

Wayfinding and Orientation

Provide descriptive labels to allow users to understand transfers between pages. This allows users to locate and orient themselves within a set of pages. Offer more than one method to find individual screens and content, like navigation menus and search.

Component Variants

Navigation Identifier

Figure 163: Navigation identifiers promote safe transfers and predictability. Links transfer users to new pages or sections within a page. Buttons open dialogs, pop-ups, panels, or expand areas in the screen to show more information. Menus open pages or sections and are part of a common navigation structure organized vertically or horizontally, like tab lines or breadcrumbs.

Search

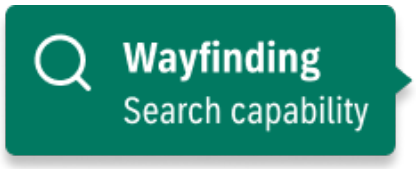


Figure 164: Search Capability annotation Search capability offers more ways for users to find what they are looking for. A list of possible destinations complements the navigation of complex sites or products with several layers of information and dense content.

Label and Title Match

Figure 165: The Label and Title Match annotation helps build expectations for planning and executing routes with labels on buttons, links, and menus that are carried over to the title on the destination.

Change in Context

Figure 166: Change in Context annotation helps eliminate potential confusion caused by a shift in focus, a change in perspective, or a new setting situation: like opening a new window, moving the user's focus to a different component, or a form automatically submitting when a value is changed. Significant changes like these should only happen when a user has clearly initiated them.

About Wayfinding and Orientation

The WCAG requirements associated to Wayfinding and Orientation are directly associated to two usability principles. **Predictability** helps users understand their current position within a set of pages and ensures users know where a link or a button will take them. **Consistency** relates to consistent navigation and helps requirements which focus on how users systematically and coherently navigate and access content.

Wayfinding strategies, including visual cues and descriptive texts, ensure a clear and consistent exploration. It allows users to identify and plan routes or navigation flows, maintaining their path and helping them stay on track to reach desired contents and pages. Wayfinding is commonly associated with transfers, also known as navigation. It is also supported by key UI elements, such as links, buttons, menus, search and breadcrumb.

Navigation ID

Users orient themselves looking for elements strategically located on the screen. They will have a successful exploration if these elements are placed consistently across different pages, accounting for primary, secondary, and tertiary navigation patterns. Wayfinding allows them to ascertain their position and plan new routes.

Navigation identifiers help users understand where they are within a digital product, making it easier to navigate, avoid getting lost, and stay oriented, especially in multi-step processes or large content structures.

The WCAG 2.4.8 Location (AAA) ensures that users can determine where they are within a website or application. Besides helping those with cognitive disabilities, this strategy also helps screen reader users, or anyone exploring unfamiliar content. This means that users should have access to information about their current location in the interface by identifying navigation identifiers, such as:

- **Breadcrumb** navigation and trails (e.g., Home > Products > Electronics > Laptops).
- Clear and unique **page titles** that reflects the content or section.
- Visible **section headers** in complex flows.
- **Menu lists** or navigation tabs. Highlight the current navigation state in menus and tabs showing which item is currently active.
- ARIA landmarks or headings for screen reader users. Use semantic structure and ARIA landmarks to support assistive technology.

Navigation identifiers are common UI elements that clarify information architecture and help users identify their current location. For example, titles and highlighted navigation controls, such as breadcrumbs, tabs, or menus, signal where a user is. Long navigation sequences can confuse people with short attention spans, so clear indicators benefit users with memory, attention, or learning challenges. Screen reader users also rely on page titles, headings, and landmarks to understand their position in the overall structure.

Labels for links, buttons, and menus also contribute to wayfinding and predictability.

Search

Search capabilities offer another way for users to find what they are looking for. WCAG success criterion 2.4.5 Multiple Ways (AA) recommends providing users with multiple alternatives to locate a page or content within a product, unless the page is a result of, or a step in, a process.

Still, in this context, a consistent placement of a help mechanism should be available to users, same place across different pages. According to WCAG 3.2.6 Consistent Help (A), this feature should be offered in the same order among the screens or pages.

Label name matches title on destination

Meaningful text, on labels and titles, is not only critical to people with cognitive disabilities. It also helps blind users manage expectations based on what they hear when using navigation strategies available in screen reader tools: headings, links, buttons, or lists. Preserving the context after selecting an option, button or link is paramount to orient blind users. Match the title or header of the destination with the label description announced in the trigger (button or link). The text on both ends should closely match each other to reassure the user about a navigation choice.

Differentiating links from buttons visually and semantically makes it clear to users what to expect from the interaction. This clarity helps everyone understand what to expect when interacting with links and buttons. By differentiating these elements, designers can ensure both clarity and predictability.

Change in Context with Links and Buttons

A change of context is a major change to the content or layout of a page that, if unexpected, can disorient a user. This includes things like opening a new page, moving the focus of the user to a different component, opening a dialog or a popover, or an automatic form submission when a value is changed. The core principle is that these significant shifts should only happen when a user has clearly initiated them.

Links and buttons are common triggers found in all digital products, but they have different transportation expectations. Links opens new pages. Buttons execute tasks within the page: open dialog, open popover, expand and collapse sections. While a link transfers the user to another page, a button keeps the user in the same page.

To ensure users clearly understand where a link will take them, the following WCAG guidelines apply:

- 2.4.4 Link Purpose (A): The purpose of each link must be clear, either from its text or surrounding context.
- 2.4.9 Link Purpose (AAA): The link text alone must express its purpose, without relying on adjacent content.
- This helps users, including those with cognitive impairments or using screen readers, navigate efficiently and independently.

Any change in context initiated by buttons or links should be understandable, clear, and provide a predictable navigation.

Sometimes links direct users to an external application. This pattern is usually indicated with an icon next to the link that informs the user in advance where that user will be redirected.

Examples

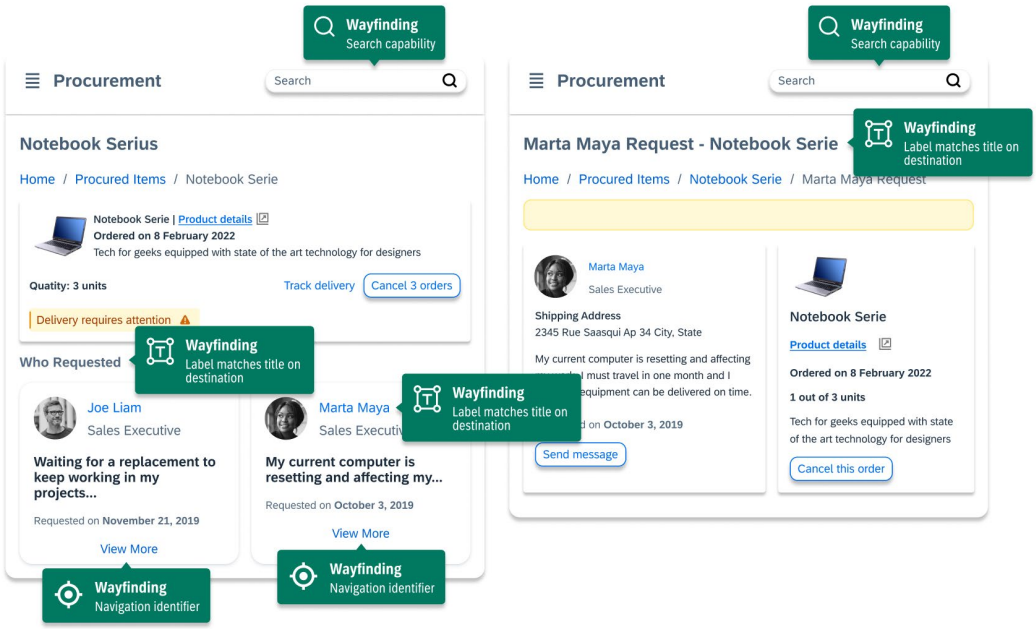


Figure 167: Wayfinding and Orientation annotations used on procurement pages. The page on the right is opened by activating “View More” buttons. The Title/Label/Breadcrumb Step of the right page is composed of respective label/name elements of the previous page, meeting expectations for orientation. In both pages, search capability is kept.

Shared Benefits

Temporary Disabilities

All users

People with temporary cognitive issues caused by stress, fatigue, illness, medication, distractions, or multitasking, also benefit from clear wayfinding and orientation. Providing descriptive labels and titles, consistent navigation, and more than one method to locate content reduces the mental effort required to stay on track, helps users quickly reorient after interruptions, and prevents mistakes or repetition. The wayfinding and orientation annotation ensures a more resilient experience including those facing temporary challenges in focus and memory.

For Developers

Discuss with the designer the workflows and the controls to be used for transportation. Double-check in review cycles if navigation targets are named appropriately according to the labels of the navigation triggers.

Make sure that backward navigation really leads you to the last position in the workflow or navigation step.

References

WCAG 2.2 Reference

2.4.4 Link Purpose (In Context) (A)

2.4.5 Multiple Ways (AA)

2.4.9 Link Purpose (Link Only) (AAA)

2.4.8 Location (AAA)

3.2.3 Consistent Navigation (AA)

3.2.6 Consistent Help (A)

Semantic Strategies

Semantics are expressed both visually and programmatically, guiding users to comprehend, navigate, and orient themselves. These are essential cognitive processes. This term refers to strategies that clearly represent their intended purpose. That meaning refers to the significance, purpose, or definition of something that is conveyed through language, symbols, or actions.

Both visual and programmatic elements support comprehension and orientation, which are core cognitive tasks. Semantics refers to the meaning of a UI element, ensuring that it represents what is intended, with meaning conveyed through language, symbols, or actions. This overlap between perception, cognition, and semantics in accessibility covers visual semantic cues, such as color, icons and spacing. Also, it adds up to clarity and reduces reading complexity and helps screen reader users by adding roles, properties and relationships.

Component Variants

Semantic Strategies

Figure 168: Use this annotation to provide a reassurance that semantics are respected using different methods: Text, Icon, Color, Role, Description, Clear Content, Pattern, Look and behave

About Semantic Strategies

This annotation was called ‘Sensory’ and located in the visual experience in the first edition of these Design Tools Guidelines. The change to Semantic Strategies aims to create a direct relationship of information and perception not only visually, but also consistent to various assistive tools. Although semantic concept for design can be based on relevant visual cues like color-coding status icons or conveying information via shape, the rationale to move the annotation to the Cognitive Experience checklist is grounded in the underlying meaning, structure, and interpretability that converge. Semantics is the meaning of an element, and meaning defines its purpose. In web development, this relationship is essential for accessibility. The purpose of the content is important for sighted and blind people.

Those who are color blind and those who are neurodivergent rely on visual cues to perceive semantics (color, icon, shape, text, position). Screen reader users rely on roles and properties cues, and the comprehension of these technical terms (roles and properties) goes beyond visual perception and gets into cognitive territory.

Consistency is what all of them expect to find: consistency of meaning, purpose, and interactions.

We simplified the annotation on the design to include declarations of how semantics are achieved in that design. This broadens the reach of WCAG standards by including the Success Criterion 1.3.1 Info and Relationships from the WCAG 2.2 to ensure roles and properties match the visual outcome. The proposal is to avoid changing aria roles to maintain the intention consistently when communicating with a blind user (using roles) or a sighted user (using visual elements). All items that look like links should act like links; the same goes for buttons.

The semantic concept relies on the ability to explore two or more strategies to trigger different senses. This is especially helpful to differentiate similar elements such as statuses, items on a wizard, progress trackers, graphs, and data visualization. Add a second signifier, or more if needed, to convey messages unequivocally. Semantics are applicable to Text, Icon, Color, Role, Description, Headings structure, Clear Content, Pattern, Look, interactions, and behaviors.

Visual Semantics

Visually safe color semantics are still perceived in greyscale. When converting a colorful screen to grayscale, the meaning of all semantic colors used in the design should be preserved.

Use semantic annotation to validate the usage of two or more channels to present information, including alternatives to the colors, such as text, shape, position, decoration, and orientation. These alternative sensory channels complement color since color alone should not be used to convey a message. If a message is not also communicated through other alternative sensory channels, a user who is color-blind, blind, or has low vision may not be able to understand it.

Designing for at least one more sensory channel using alternatives to stimulate visual senses: color and text or color and shape. If all the information maintains its meaning and purpose on grayscale, you have successfully achieved a sensory concept for elements represented by color. Color, shape, size, or location without text may not be understood by a color-blind person, blind users, or users with low vision.

Do not use color alone to convey a message. If shape, size, or location, or instructions are not provided with text, a color-blind person, blind users, or users with low vision may not be able to understand them.

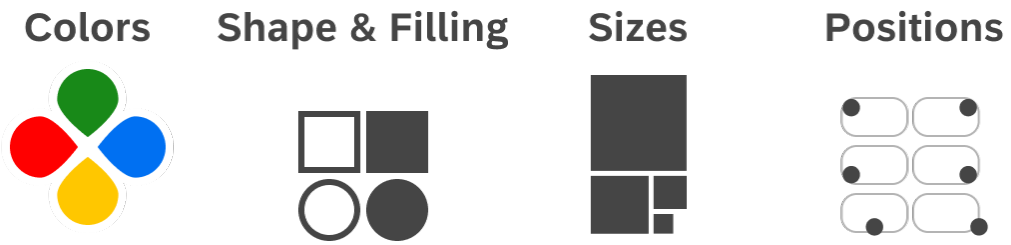


Figure 169: Multiple rectangular areas with a dot inside. every area has the dot at a different position

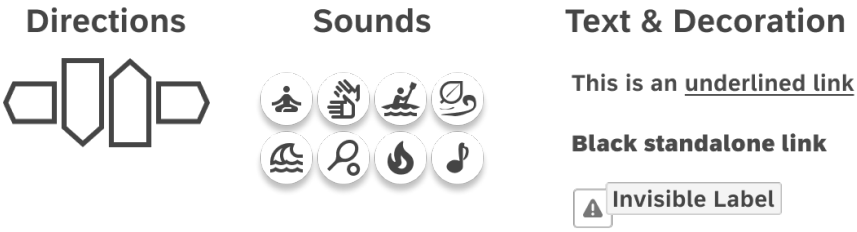


Figure 170: Different semantics - Directions, Sounds and Text Styles or Decorations

Hidden Semantics

Screen readers are expected to announce semantic information in addition to the visual information presented. Every semantic content communicated visually, such as color, shape, position, must be accessible to screen readers. If visible text is provided, such as a status name, it should be able to communicate semantic information by itself. If this is not sufficient, provide invisible labels, text alternatives, invisible headings, and descriptions.

But sometimes the visual semantics are associated with names like roles and properties used on UI controls. The expectation in this case is to maintain consistent semantics between visual and screen reader terminology, avoiding alterations. For example, a link that appears as a link should behave like a link. A link always directs the user to a new URL page, except when it transports the user to another section within the same page, such as with anchor links. In contrast, a button should both look and behave like a button. However, there are times when a button performs the role of a link, and this is perfectly acceptable. Typical button labels, such as Submit, Save, Add, and Create, often guide the user to a new page.

Address how different users will give meaning and consistently understand information, instructions and interactions, visually or through screen readers. Observe how properties and roles can provide the same experience to both sighted and blind users.

Semantics for Screen Readers

Using semantic annotations helps ensure that ARIA roles and properties are properly defined, allowing screen readers to accurately interpret and announce content for users who rely on assistive technologies.

A rule of thumb for Links vs. Buttons is that buttons and links are visually distinctive. Buttons should look like buttons, and links should look like links.

The button keeps the user within the page. Links transport the user to another page (same or new browser tab).

The types of actions performed by buttons are distinctly different from the function of a link:

- Use a **button** when the action causes a change on the same page, like opening a popover or a dialog.
- Use a **link** when navigating to another page (URL).

Links are used to direct users away from the current page, but they can also be used to jump to a different section within the same page (anchor bar, tabs).

Elements on a webpage convey meaning through their appearance, such as a button clearly looking like a button or a link presenting itself as a link. Screen reader accessibility features rely on assigning appropriate roles that align with how each element is visually designed. This ensures that individuals using assistive technology can easily recognize and interact with every part of the page.

Sighted users form expectations based on clear visual style and thoughtful content choices. A link should appear clickable even before a mouse hover over it, which means it needs to be styled to signal interactivity. Using color and underline are two common ways to indicate that something is a link.

Screen readers announce the role, which should match the visual elements that sighted users view on the screen. When a blind user hears the announced role, they form expectations about what happens next. For example, if they hear "link," they know it will take them to another page; if they hear "button," they know it will perform an action and keep them on the same page.

Semantic Structure

Semantic text principles ensure that content is meaningful, well-structured, and accessible to all users. Providing clear, meaningful, structured and accessible content also contributes to semantics. Some examples are proper use of headings, lists, emphasis, statuses, and labels (for links and buttons). The goal is to enhance readability and comprehension while providing clear context and information.

For example, headings should follow a logical hierarchy (H1 for the main topic, followed by H2s and H3s for subsections) to convey structure, rather than relying on visual styling alone.

Lists should be correctly formatted using <ul>, <ol>, or <dl> elements to group related information meaningfully.

Additionally, links should have descriptive text that conveys their purpose rather than vague phrases like "Click here." By applying semantic text principles, content becomes more inclusive, reduces cognitive effort, and improves the overall user experience.

We propose five key principles to guide UX writers in formatting and structuring text to improve readability, comprehension, and navigation, particularly for assistive technologies.

1. **Proper Heading Hierarchy**

Use headings (<h1> to <h6>) to create a clear structure, ensuring logical flow. The <h1> should represent the main topic, followed by <h2> for sections, <h3> for subsections, and so on. Avoid skipping heading levels (e.g., jumping from <h2> to <h4>), as it disrupts assistive technology navigation.

2. **Meaningful Link Text**

Links should describe their purpose, avoid “Click here”. Repeating links should be prepared for screen readers, using a reference for association (ariaLabelledBy). Avoid generic phrases that require users to rely on surrounding context to understand the function of the link. Use an appropriate aria element to provide a clear label or description, like title attributes if extra clarification is needed.

3. **Emphasis and Meaning, Not Just Style**

Use for important content and for emphasis, rather than relying solely on bold or italic styling. Avoid using all caps for emphasis, as it can be harder to read and may be misinterpreted by screen readers. Use meaningful status names (labels). If color cannot be perceived or

translated to statuses categories, the label alone will be able to convey the semantic.

4. **Avoiding Ambiguity and Redundant Text**

Keep text concise and to the point, avoiding unnecessary jargon. Ensure that error messages and instructions are direct and actionable (e.g., “Enter a valid email address” instead of “Invalid input”).

Identify Purpose

Lastly, the WCAG 1.3.6 Identify Purpose (AAA) bridges the gap between what something looks like and what it is meant to do, helping designers and developers build interfaces that can adapt meaningfully to diverse user needs. This requirement suggests that users adapt the interface to their needs, making the purpose of elements clear to assistive technologies so that common user interface components and icons are understood semantically, whether through symbol systems, simplified language, or personalized UI tools. This matters because it:

- Helps users who rely on symbol-based assistive technologies, cognitive support tools, or custom UI adaptations.
- Improves predictability, especially for users with cognitive disabilities or low digital literacy.
- Enables assistive technologies to personalize the interface, such as replacing text with symbols or grouping functions by type.

Examples

Visual Semantic

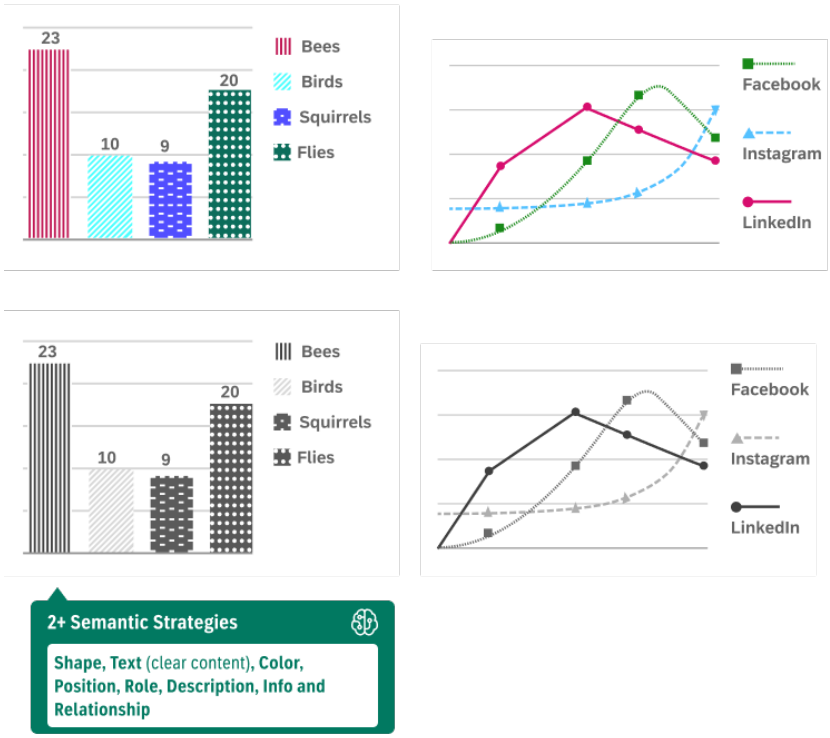


Figure 171: Data visualization with treatment to differentiate data not only with colors, but line types and patterns to meet the needs of blind color users

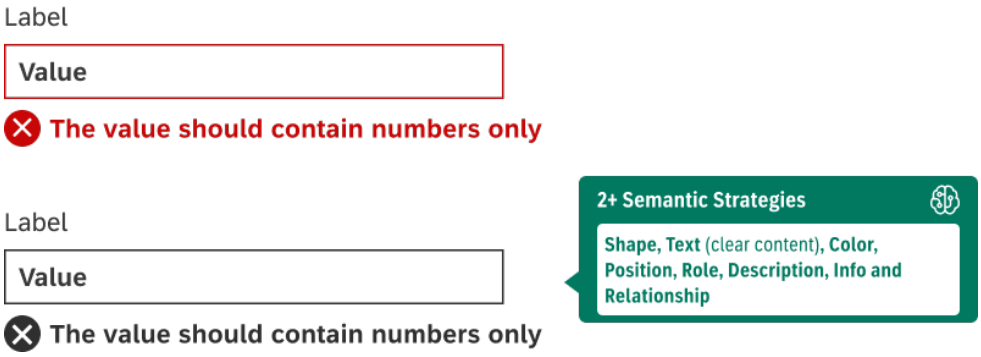


Figure 172: Error message treatment to indicate an error has occurred by showing an icon, beside the red color

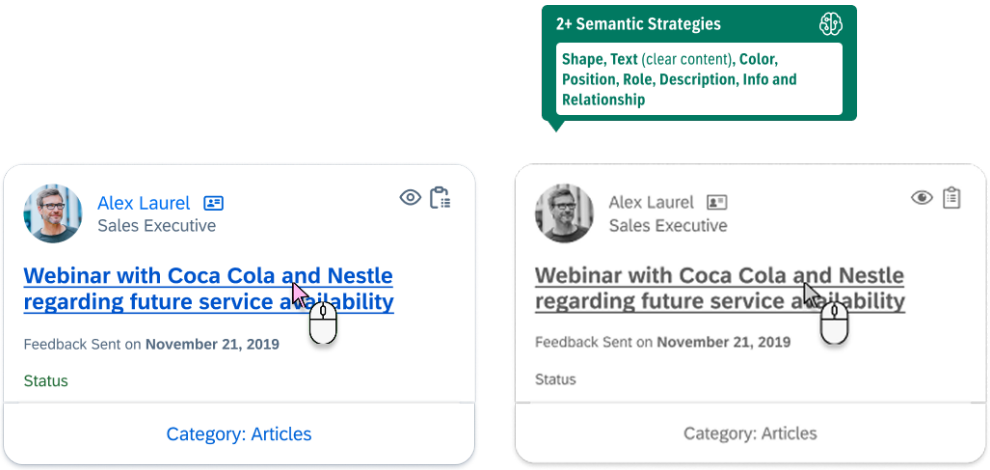


Figure 173: Link is still perceived because the text is not only blue, but also underlined

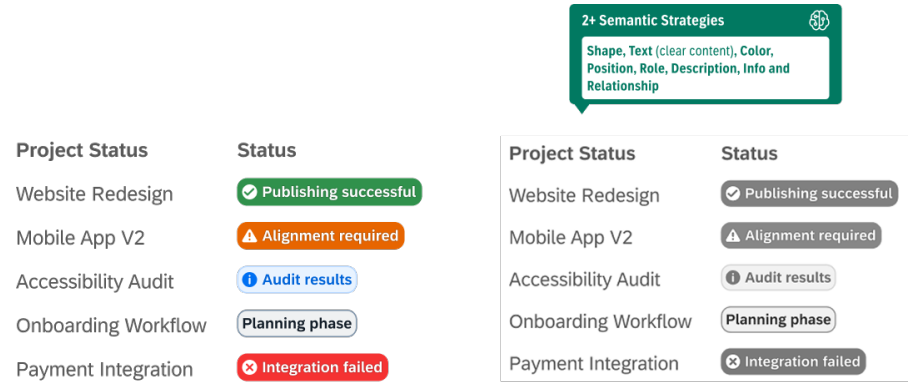


Figure 174: Statuses are recognized if color is not available by their labels or icons

Semantics are used to inform about:

- Colors + Labels + Icons + message (status of the system or element)
- Components (links vs buttons, tags and object status, strip messages, dialog message)
- Font size on Heading to indicate hierarchy

Color combinations to avoid:

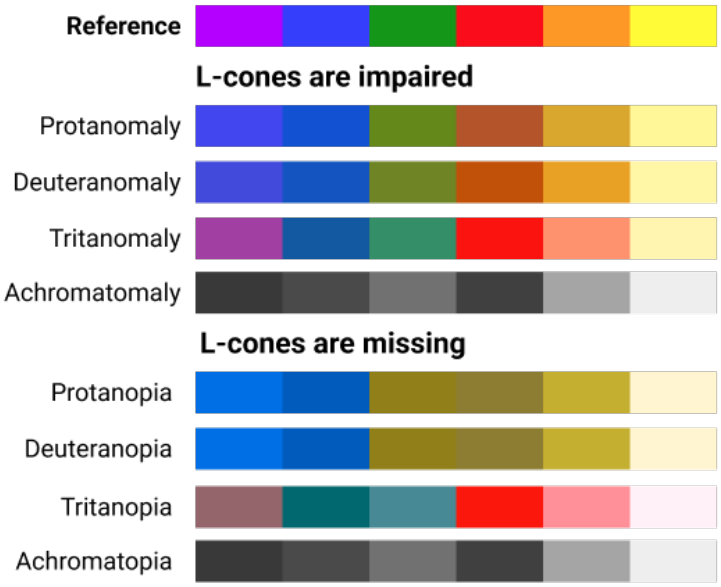


Figure 175: The sensory concept supports design decisions where colors are used to represent something such as a state. Make informed decisions about combining colors. Investigate how a person with protanopia or deuteranopia sees green and red.

Avoid problematic color combinations when possible:

- Green & Black
- Green & Red
- Green & Grey
- Green & Brown
- Green & Blue
- Blue & Gray
- Blue & Purple

Shared Benefits

Temporary Disabilities

All users

Stress, fatigue, illness, medication effects, or environmental distractions may reduce the mental load required to interpret and navigate an interface. Clear semantic design will make it easier for these people to stay focused and avoid errors when attention or memory is impaired. Visual cues like icons and spacing, straightforward text, and consistent programmatic semantics help users quickly understand where they are, what actions are possible, and how to move forward without confusion. Clear semantics support comprehension and orientation, easing task completion during temporary limitations.

Vision and Color Limitation

Visuals

People who are color blind benefit from clear semantics because meaning is conveyed through multiple cues, like labels, icons, or patterns. Rather than relying on color alone, alternative or complementary information reduce ambiguity and support accurate comprehension.

For Developers

Check with the designer in the review cycle that the semantics for a respective UI part are expressed visually and non-visually in multiple ways using different technical attributes for color, text, relationships, etc.

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

1.3.3 Sensory Characteristics (A)

1.3.6 Identify Purpose (AAA)

Error Handling

Prevent errors from happening. When errors occur, clearly identify them. Allow users to review, correct, or cancel inputs so that they can avoid or fix errors. Associate an error with its location. To handle errors, use sufficient descriptions so the user can understand what happened and make suggestions to fix it.

Component Variants

Error Prevention

Figure 176: Error prevention helps users with information to get things right from the start.

Error Identification and Error Recovery

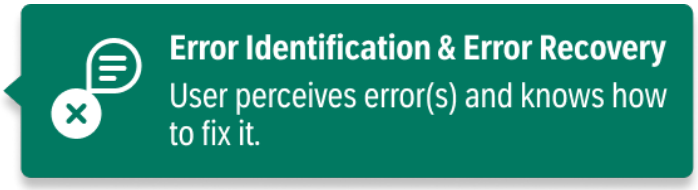


Figure 177: Error Identification and Error Recovery help users to recover from mistakes.

About Error Handling

Suppose the input of a user creates a legal commitment, involves a financial transaction, or results in data being lost or modified in data repository systems. In such cases, the application must ensure that the user can check, correct, or cancel their actions.

Error Prevention

Error prevention helps users avoid errors, while error identification and recovery allow the user to fix errors if they occur.

Preventing errors from happening should be a primary objective of the designer when creating forms. The interface must describe how a form field should be filled out, and input fields should refuse incorrect values and redirect users to enter correct values.

Efficient error handling saves vast amounts of effort for:

- Finding errors
- Navigating to the element causing the error
- Resolving the error by using pre-selected values or hints.

Error Identification and Error Recovery

When error prevention is not possible, the application must allow the user to identify the error, review, correct, or cancel an input. The designer should anticipate such scenarios and address the following:

- When and how the error will appear.
- How the error message will address the problem to enable user to handle the error.
- Along with a detailed description of the error, what resources will be offered to the user to fix it.

Efficient error handling should clearly identify where an error has occurred, describe the nature of the issue, and offer practical solutions for correction. This approach greatly streamlines troubleshooting and reduces unnecessary effort.

Context-specific suggestions presented to the user next to the form field enhance error identification awareness and enables the user to recover faster from the error. This strategy helps users of screen readers, as it provides quick access to the specific information related to the input error.

When error validation cannot be implemented while the form is being filled out, error messages are presented in the form fields. These messages inform and invite the user to fix issues. When errors are resolved, the error messages disappear visually and screen reader updates accordingly. A positive confirmation message is commonly used to notify users that the form was successfully completed or successfully submitted.

Examples

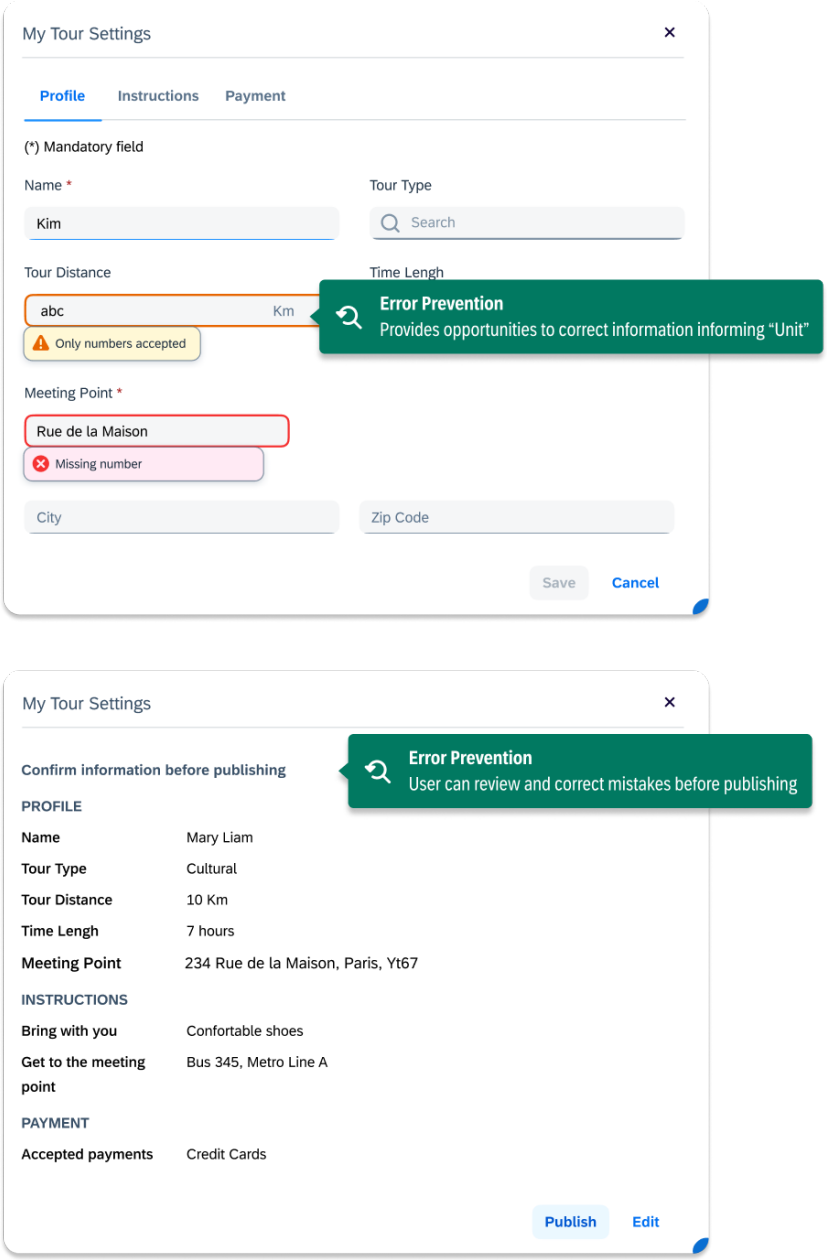


Figure 178: Error Prevention annotation used to indicate respective areas or functionality in the application design

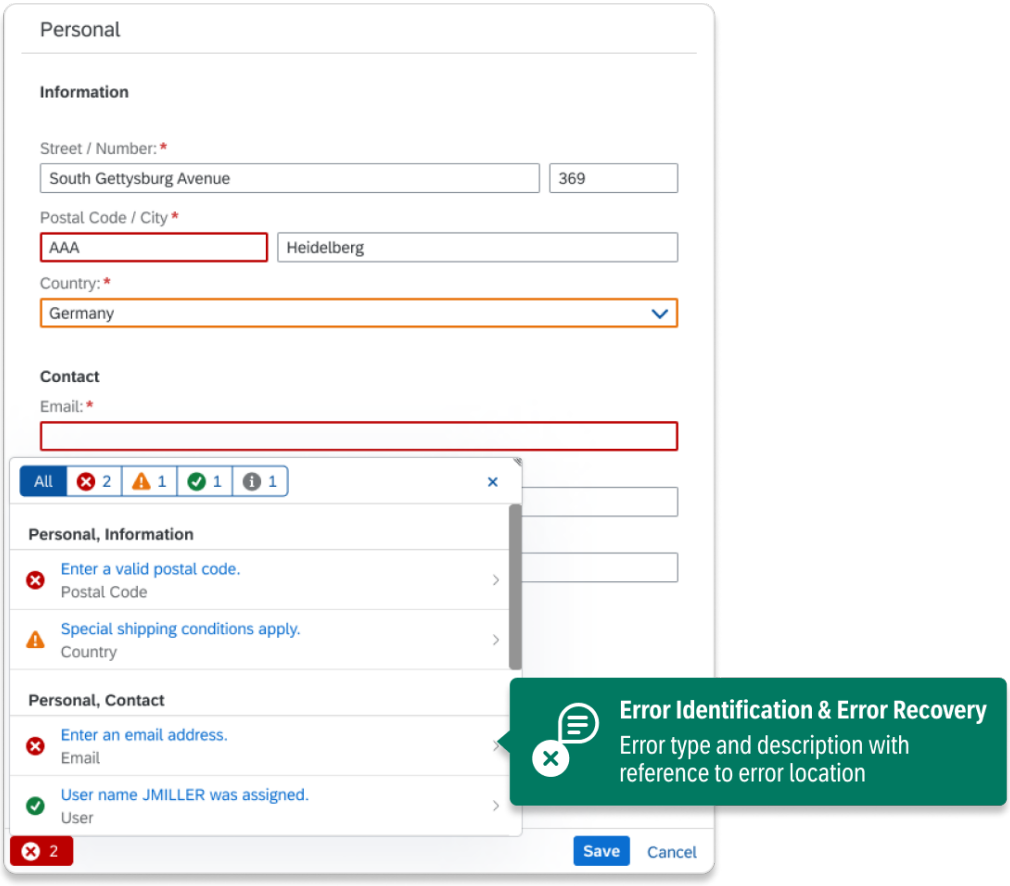


Figure 179: Error Identification & Error Recovery annotation used to indicate respective applied patterns in the application design

Shared Benefits

Temporary Disabilities

All users

Users with temporary limitations due to fatigue or distractions, expect the application to support them during error handling by reducing the mental and physical effort needed to recognize and to recover from errors, preventing task abandonment.

Without Vision

Screen Reading

Blind users also benefit from clear error handling. Error messages and programmatic feedback ensure they can detect and correct mistakes without relying on visuals.

For Developers

All developers should provide mechanisms to indicate and to correct errors if they occur by using respective controls or patterns from their framework inventories.

References

WCAG 2.2 Reference

1.3.1 Info and Relationships (A)

3.1.3 Unusual Words (AAA)

3.1.4 Abbreviations (AAA)

3.1.5 Reading Level (AAA)

3.3.1 Error Identification (A)

3.3.3 Error Suggestion (AA)

3.3.4 Error Prevention (Legal, Financial, Data) (AA)

Motion Content

Animations can be stopped or disabled. Video can be paused or stopped. Automatically played audio content can be muted. The volume of the automatically played content can be changed independently of the system volume. Generally, avoid flickering and flashing content.

Component Variants

Autoplay & Motion-Based Content

Figure 180: Use to indicate content affected by autoplay or motion based (moving or updating content). Indicates also where users can find controls or settings to control motion.

About Motion Content

Animation, video, and audio streams attract the attention of the user. But content that moves or auto-updates can be a barrier to anyone who has trouble reading static text. It also affects users who have difficulty tracking moving objects and users who use screen readers. Screen reader users find it difficult to listen to speech output if other audio files are played automatically at the same time. Users with hearing loss are also distracted by content played automatically.

All users must be able to pause, stop, or hide videos, animations, and audio. Any automatic audio or video content must allow users to control and change the volume independently of the system volume.

Common examples include motion pictures, synchronized media presentations, animations, real-time games, and scrolling stock tickers. Use the annotation ‘Multimedia’ to point to where users can find the controls to pause, stop, and hide moving content.

Certain groups, particularly those with attention deficit disorders, find blinking or flashing content distracting, making it difficult for them to concentrate on other parts of the page. Therefore, avoid flickering and flashing content. Ensure that the application has no flickering content that flashes more than three times per second. If your component or screen uses a content in motion, identify it as “Flicker safe” using the annotation to indicate that content is secure for sensitive users with epilepsy or seizures.

Refined terminology for UX and Accessibility Guidelines related to motion includes three groups: Autoplay & Motion-Based Content, Moving & Updating Content, and High-Risk Motion Content.

Autoplay & Motion-Based Content:

- "Auto playing Media" – Covers videos, GIFs, and animations that start playing without user interaction.
- "Auto-advancing Content" – For carousels, slideshows, or other elements that change automatically.
- "Motion-Based Content" – Includes parallax effects, animated scrolling, and any UI elements that respond to movement.

Moving & Updating Content:

- "Moving Content" – Covers auto-scrolling text, ticker messages, and elements that shift without user interaction.
- "Auto-Updating Content" – Refers to changing dashboards, live sports scores, or stock tickers that refresh dynamically.

High-Risk Motion Content (Accessibility Considerations):

- "Flickering Content" – Any flashing, blinking, or strobing elements (e.g., fast animations, certain video effects). We recommend following WCAG guidelines 2.3 Seizures and Physical Reactions on the three flashes or below threshold to avoid seizures or discomfort.
- "Looping Motion Content" – Covers endlessly repeating animations or videos that might be distracting or overwhelming.

Examples

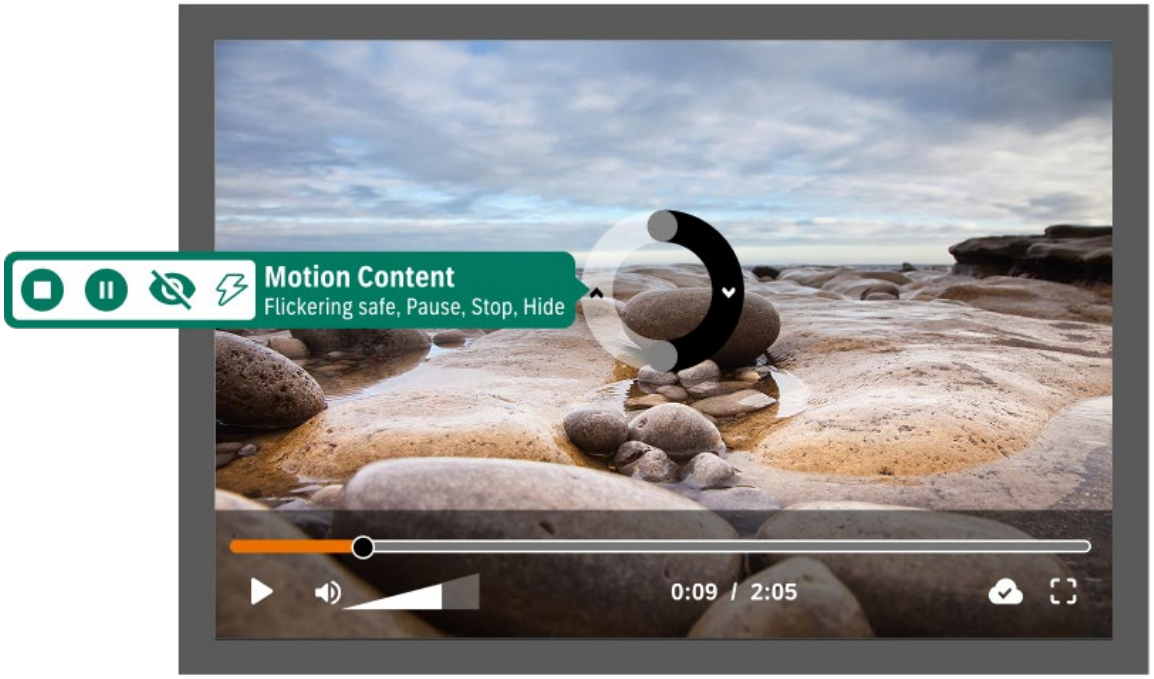


Figure 181: Volume of automatically played audio content can be changed independently of the system volume. Videos can be paused or stopped. Animated controls must be created to be flickering safe, like busy indicators.

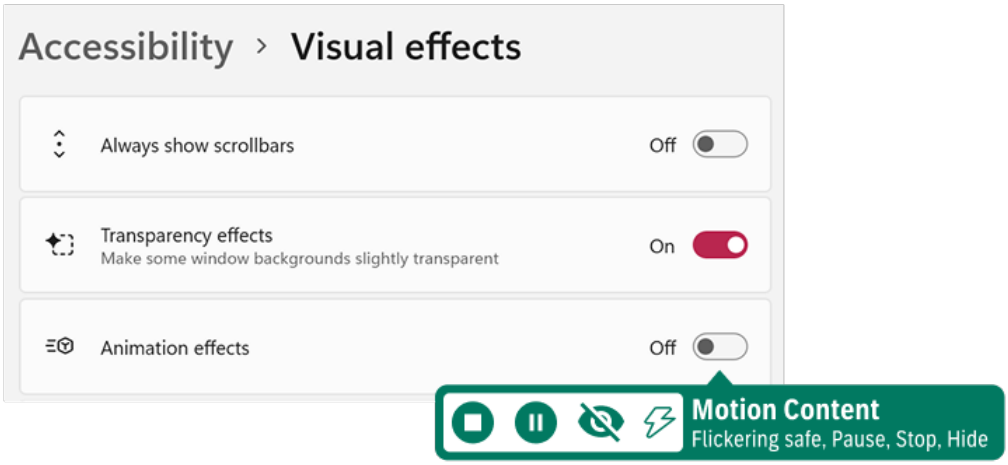


Figure 182: Animations can be hidden or disabled

Shared Benefits

Without Vision

Screen Reading

Observing autoplay, motion, and dynamic content during design helps screen reader users by preventing unexpected changes that can disrupt navigation, shift focus, or overload announcements, as well as ensure a stable, predictable experience. It also allows designers to provide proper pause controls, ARIA live regions, or alternatives, making content updates accessible without overwhelming or disorienting users relying on assistive technology.

For Developers

Ensure by providing respective options that:

- Animations can be stopped or disabled
- Video can be paused or stopped
- Automatically played audio content can be muted
- The volume of the automatically played content can be changed independently of the system volume

Generally, avoid flickering and flashing content

References

WCAG 2.2 Reference

2.2.2 Pause, Stop, Hide (A)

2.3.1 Three Flashes or Below Threshold (A)

2.3.2 Three Flashes (AAA)

Time Limit

Provide options so that users can disable, customize, or extend time limits before they expire. This should indicate areas on the screen that are refreshed automatically or due to user interaction.

Component Variants

Time Limit - Timeout Progress

Figure 183: Indicates that the UI prepares users for system events tied to timing (e.g., session expiration or auto-logouts). A message confirms data and progress will be saved.

Time Limit - Timeout Info

Figure 184: Indicates that the UI prepares users for system events tied to timing. A message reinforces the risk of data loss or progress reset due to inactivity or predefined time limits.

Time Limit - Timeout Interruption Expected

Figure 185: Indicates that the UI prepares users for system events tied to timing (e.g., session expiration or auto-logouts). A message reinforces the risk of data loss or progress reset due to inactivity or predefined time limits.

Time Limit – Configuration

Figure 186: Use this annotation to indicate where users can adjust and manage time to be turned off, adjusted, or extended.

Time Limit – Content Refresh Area

Figure 187: Indicates the area in the UI that will be affected by a refresh upon time limit

Time Limit – Visual Status Update

Figure 188: System Status Awareness for Background Processes ensure users are aware of ongoing background processes and can access or perceive system status at any time.

About Time Limit

Time-based interaction, such as automatic updates, periodic processes, and session expirations, can interrupt tasks. These actions may erase work and confuse users.

This separation highlights the relationship between time-based interactions and their specific effects, making the information easier to digest. The Time Limits annotation helps designers identify these risks early, giving users predictability, control, and protection.

According to WCAG 2.2.3 No Timing (AAA), content must not depend on time limits for its functionality, unless timing is essential, such as with live events. Users should be able to complete tasks at their own pace. However, sometimes this is not possible, and this chapter explores the proper handling.

Automatic updates, session expirations, auto-logout, form timeout, dashboard refresh, expiring access to a task, and timeouts can introduce unexpected barriers, disrupt user tasks, or even result in data loss. Any process that fires after a set period counts as a time limit. Such triggers can be major barriers for people who need extra time to read, think, navigate, or act.

A poorly designed time constraint may cause confusion, frustration, lost work or loss of data. Thoughtful and well-considered time limit design shields users from disruption and protects users from disruption supporting a smoother, more accessible experience.

Designers should aim to avoid time-dependent features whenever possible or minimize the use of time-dependent features if possible. When time constraints are necessary, interfaces must offer users clear control and predictability. If unavoidable, give users control in this order of preference:

a. Disable the time limit entirely (most inclusive).
b. Customize the duration of the time limit. Configure the length to suit individual needs.
c. Extend it on demand via a clear prompt before it expires. Request more time before the limit expires.

The Time Limits annotation supports inclusive design by helping teams plan for four critical aspects:

- **Timeouts**
- **Content Refresh**
- **System Status Updates**
- **Configuration Options**

These dimensions ensure users are informed, protected from unexpected behaviors, and empowered to complete tasks at their own pace. Disabling time limits is always preferred. If that is not possible, customizing them or offering time extensions is essential to support diverse user needs and reduce cognitive load.

When users understand and control the timing of system behaviors, they avoid unwelcomed surprises and interact with greater confidence. This leads to more inclusive, frustration-free experiences.

Timeouts

Timeouts include three scenarios that may or may not notify users about a timeout:

1. Data entered or progress tracked: when progress and input data is saved and tracked when time is out.
2. Data loss due to inactivity: when progress or input might be lost due to inactivity.
3. Interruption Expected: anticipate and inform about automatic refreshes or system updates.

WCAG 2.2.6: Timeouts (AAA) state that users must be warned of any inactivity that could result in data loss unless the data is preserved for more than 20 hours when the user does not take any action. The Timeout annotation reminds designers to notify users about possible data loss due to inactivity, alert them of upcoming or ongoing interruptions, and provide reassurance that their data and progress have been saved.

Content Refresh

Informing users about content that will refresh after a time limit is a requirement covered by WCAG 2.1. The relevant WCAG 2.2.1: Timing Adjustable defines that content updates after a time limit, such as auto-refresh, and users must be able to turn off, adjust, or extend that time unless it is essential. This definition is relevant to inform users which area will refresh and prepare them to act within the time limit or allow support for adjusting timing.

The WCAG 2.2.2: Pause, Stop, Hide (level A) also supports this requirement. For moving, blinking, scrolling, or auto-updating information, users must be able to pause, stop, or hide the content unless it is essential. If the refreshed content is auto-updating, such as a dashboard or notification panel, controls must be offered to manage the update behavior, see Time Limits Configuration.

System Status Updates

Communicate clearly when system behaviors are time-triggered, helping users anticipate outcomes and maintain focus. When users are informed and in control of timing, they avoid surprises and complete tasks more confidently. Use messages with precise information about timeout outcomes.

Communicating system status clearly becomes essential when time-based behaviors are in play, examples include session timeouts, auto-logout, or delayed content refresh. Users need to know what is happening, how much time they have left, and what action (if any) is required.

But the Status Update also includes visual and verbal cues that show system status in real time, using consistent indicators such as countdowns, progress indicators, alert messages, spinners, progress bars, and badges. Status Updates represent the start and end of the process and stop when complete. The design should focus on clarity and give control to the user with plain, concise messages (“Uploading 45 %...”) and, where possible, buttons to pause, resume, or cancel. This helps reduce anxiety, supports informed decisions, and gives users a sense of control.

Assistive Tech requires ARIA live region or role="status" / aria-busy="true" so screen readers announce updates without shifting focus. Keyboard Access should be provided to reach the status info and dismiss controls reachable with the keyboard.

This requirement is in alignment with WCAG Success Criteria:

- 4.1.3 Status Messages to announce updates programmatically,

- 3.2.2 On Input to warn the user before auto-refreshing on user action, and
- 2.2.2 Pause, Stop, Hide to allow users to stop auto-updates, which may distract.

Clear status feedback, combined with timeout and refresh annotations, ensures a smoother, more accessible experience across the board.

Configuration Options

Including configuration options in the application provides users with flexibility regarding time limits. While information is good to keep users aware, it is preferred when the time limit can be extended and controlled. Designers address time outs by providing options to the user to disable time limits (switch off), customize the length of time limits by requesting more time before a time limit occurs, or extend the time before the time limit expires.

Time limit configuration helps with:

- Refresh content after a certain amount of time
- Signal the time to log off from an application
- Keep a network connection to a server online (keep it alive)
- Define time limit to work on a task

The WACG 2.2.1: Timing Adjustable determines that for each time limit that is set by the content, at least one of the following is true: the user can turn off, adjust, or extend it. The Time Limit Configuration annotation reminds designers to provide users with the ability to control session timeouts, task limits, or inactivity-based events

Examples

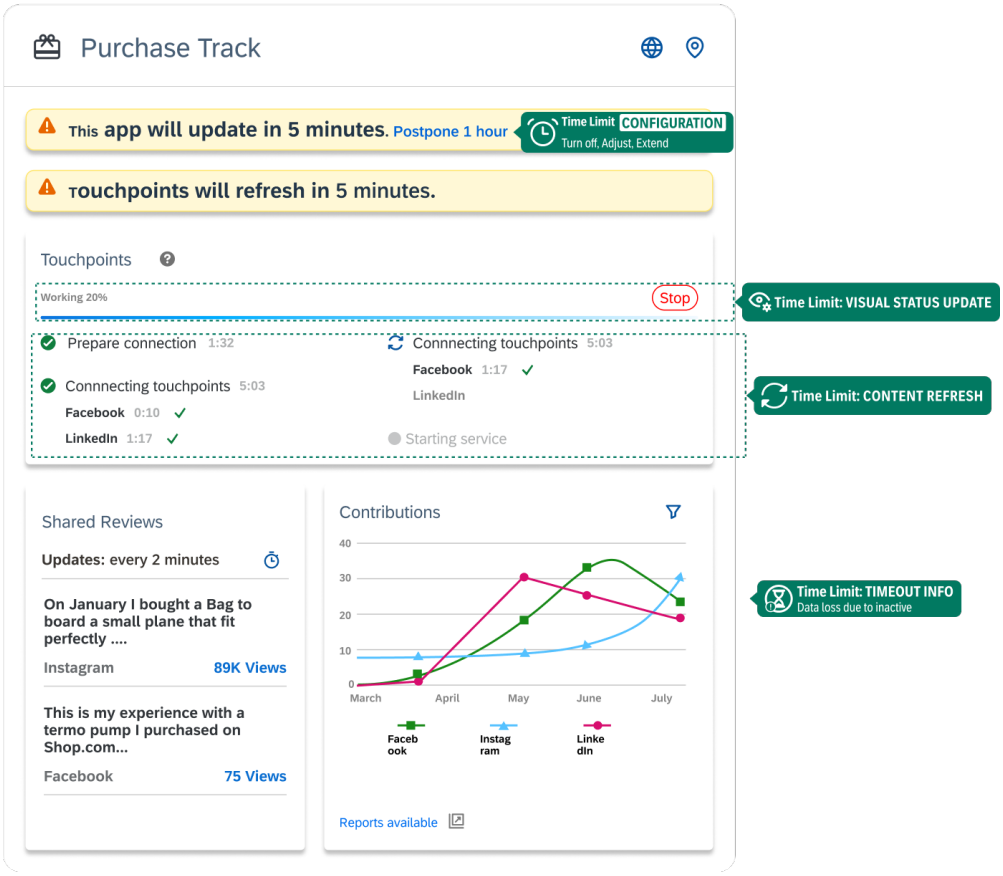


Figure 189: Multiple UI elements that update, refresh or change based on time are annotated with time limit annotations.

Figure 190: A progress bar annotated with the Timeout - Data and Progress tracked annotation

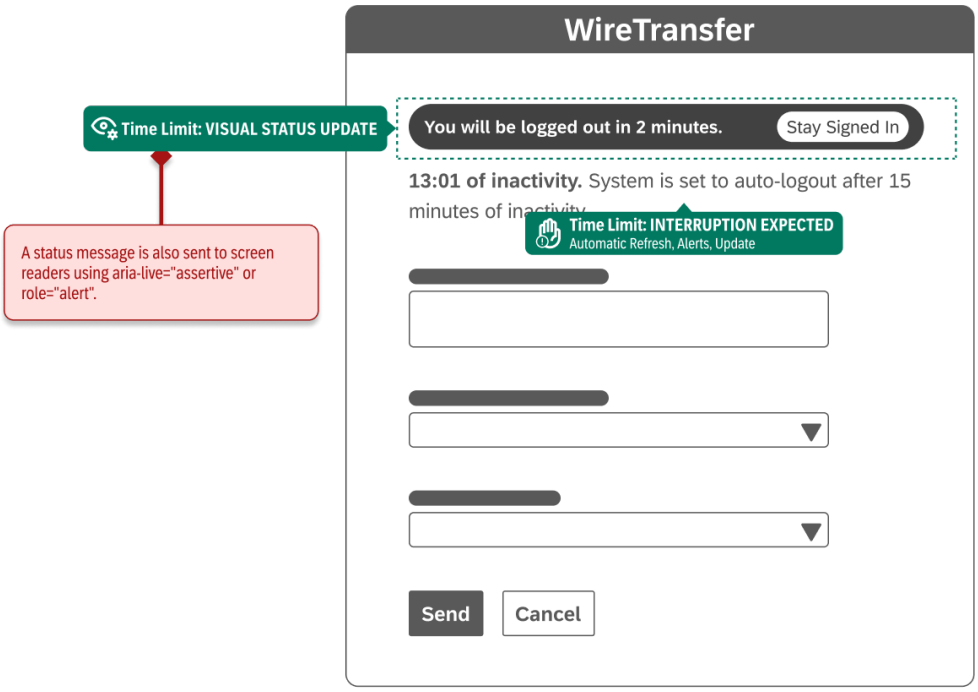


Figure 191: A dialog for a wire transfer that will automatically log the user out in 2 minutes annotated with the Timeout - Interruption expected annotation and the Time Limit: Visual Status Update

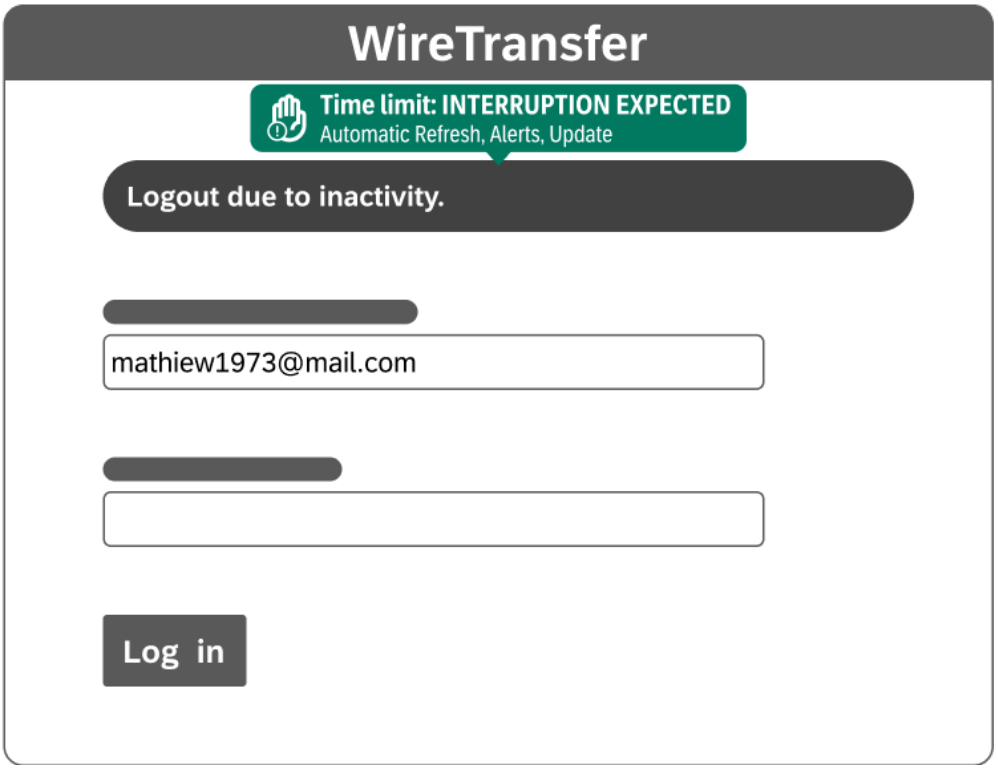


Figure 192: A dialog for a wire transfer that logs the user out due to inactivity annotated with the Timeout - Interruption expected annotation

Shared Benefits

Without Vision

Screen Reading

For screen reader users, updating regions should use ARIA live regions (e.g., aria-live="polite" or assertive") to notify without disrupting interaction. This relates to WCAG 4.1.3: Status Messages.

Limited Mobility

Interactions

Time limits can exclude users with limited mobility who need more time to navigate or complete tasks due to slower or assistive input methods. Giving options to extend or disable time limits reduces stress, prevents errors, and supports equal access.

For Developers

Ensure, by providing respective framework controls, that there are options to:

- Disable time limits
- Customize the length of time limits
- Request to extend the time limit before it expires

Indicate clearly areas on the screen that are refreshed automatically or because of user interaction.

References

WCAG 2.2 Reference

2.2.1 Timing Adjustable (A)

2.2.2 Pause, Stop, Hide (A)

2.2.3 No Timing (AAA)

2.2.6 Timeouts (AAA)

3.2.2 On Input (A)

4.1.3 Status Messages (AA)

Multiple Ways

Provide users with multiple clear ways to find and return to content, such as search, navigation menus, breadcrumbs, or contextual links, so they can move forward with confidence and easily backtrack without losing their place.

Component Variants

Multiple Ways

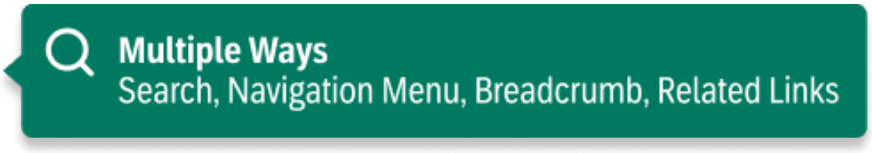


Figure 193: Use this to indicate multiple ways for accessing content by navigation

About Multiple Ways

Not everyone interacts with content in the same way. To fulfill the requirement for offering multiple ways to access information, you should provide at least two distinct options for users to locate and reach any given page or piece of content in a digital product. Alternative navigation methods ensure users are not stuck in one path to find content and help them discover and recover information distributed across multiple pages. This requirement does not apply if the page or content is a mandatory part of a process or a specific step that must be completed in order.

Providing users Multiple Ways to access the pages in a product comes in alignment with WCAG 3.2.3 Consistent Navigation requirement. Repeating UI elements and features like search, navigation menus, site maps, and breadcrumbs provides a predictable and consistent strategy for users to get to the content they are looking for using the ways that most fit their needs. Users may find one technique easier or more comprehensible to use than another

The design specification is likely to live at the product level so that such options are found consistently applied across pages of the same product. Here is what designers should observe and implement:

1. Include a search function where relevant: If the product has a large set of content, such as help documentation, product listings, FAQs, is search available? Designers are responsible for designing a prominent and accessible search field as an alternate method to locate specific

content. Category filters are also helpful as an alternate way to find content.

2. Provide navigation menus or site maps: Is there a consistent global or section-level navigation system? Designers should ensure users can browse through menus and that these menus provide access to all major areas of the interface.
3. Use lists of related content: Are related or recently viewed items presented? Designers should find patterns that suggest additional paths to content that users may find helpful.
4. Provide clear hierarchical navigation (Breadcrumbs): Are users able to understand where they are and jump to previous levels? Designers should include breadcrumbs in your design for multi-level page structures.
5. Facilitate filtering and sorting: Can users control how they access content, such as filters for job listings? Designers should provide flexible browsing tools that support diverse information-seeking behaviors.

Observe flags that could cause accessibility violations:

- Design lacks Search or secondary navigation.
- Content is deeply nested and inaccessible.
- Content that leads to more content on a non-linear exploration.

Examples

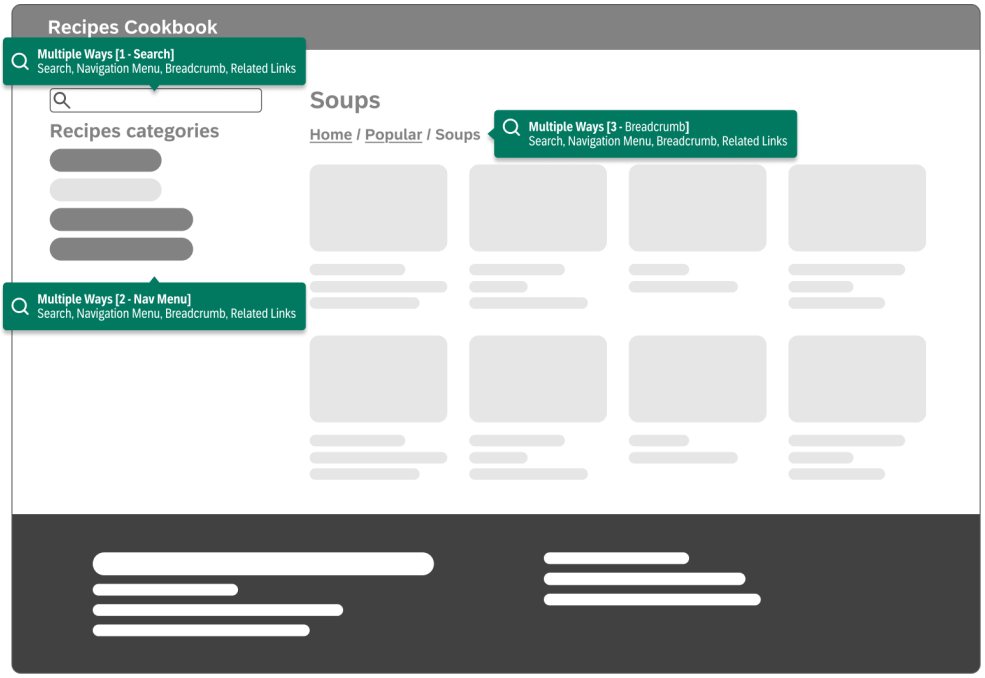


Figure 194: A cooking webpage with three different ways to find recipes annotated with the multiple ways annotation

Shared Benefits

Limited Mobility

Interactions

Users with limited dexterity, switch devices, or speech navigation may struggle to navigate through nested menus or perform precise movements. Multiple Ways helps them because it enables shortcuts to content (e.g., through search or quick links), reducing the need for repeated or fine motor interactions. It also reduces fatigue by avoiding long sequences of navigation or scrolling. And finally, it allows users to choose the most accessible navigation method for their setup (e.g., skipping menus in favor of filtered tables or voice-based navigation).

Vision and Color Limitation

Visuals

Users who rely on screen readers or magnification tools may not perceive visual menus or might get lost in long pages. This requirement helps users because search or landmark navigation allows users to jump directly to content without navigating visual menus.

- Consistent heading structure and landmark roles help screen reader users skim and locate content more efficiently.
- For magnification users, avoiding excess scrolling through alternative paths (e.g., filters, in-page links) reduces cognitive and physical strain.

For Developers

Provide users with established UI patterns to find and return to content such as search, navigation menus, breadcrumbs, or contextual links, so they can move forward with confidence and easily backtrack without losing their place.

References

WCAG 2.2 Reference

2.4.5 Multiple Ways (AA)

3.2.3 Consistent Navigation (AA)

Redundant Entry

Reuse previously entered information by prefill or auto-populate input

Component Variants

Redundant Entry

Figure 195: Reuse previously entered information by prefill or auto-populate input.

About Redundant Entries

If a user has already entered information earlier in a process, they should not have to re-enter it unless it is essential. This requirement fulfills WCAG 3.3.7 (A) Redundant Entry, which reminds designers to observe how to reduce the cognitive workload of the user and physical burden in multi-step forms, processes, or flows (e.g., checkout, onboarding, applications).

Follow these practical questions to reduce redundant entries:

Is the process multi-step?

- If yes, track what data is being collected at each step.
- Avoid asking for the same piece of info more than once (e.g., name, address).

Can earlier info be prefilled or referenced?

- Design forms to carry user input forward to later steps.
- Provide clear confirmation. Example confirmation message: “Is this still your shipping address?”.

Design for Memory Relief. You should not expect users to recall information they have already entered. Instead, show previously entered values clearly and accessibly throughout the process. This practice reduces the cognitive load on the user, prevents mistakes, and makes the overall experience smoother and more reliable.

Lastly, when users need to fix something, give them the ability to edit previously entered data. This can be done by providing a link back to a specific step or by making fields editable directly within a summary screen.

Examples

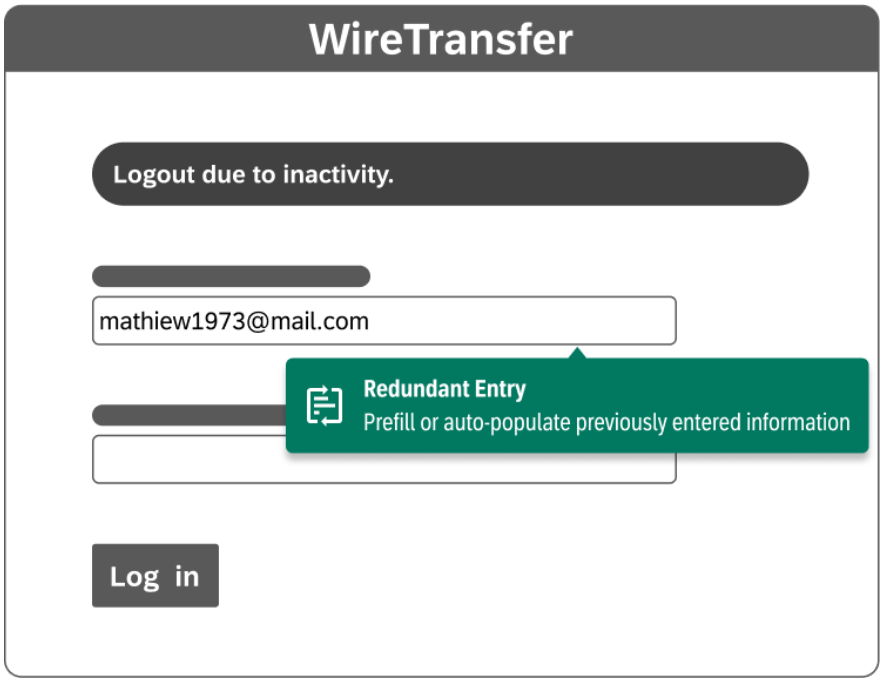


Figure 196: Login Screen with the E-Mail input field auto populated. This field is annotated with the redundant entry annotation

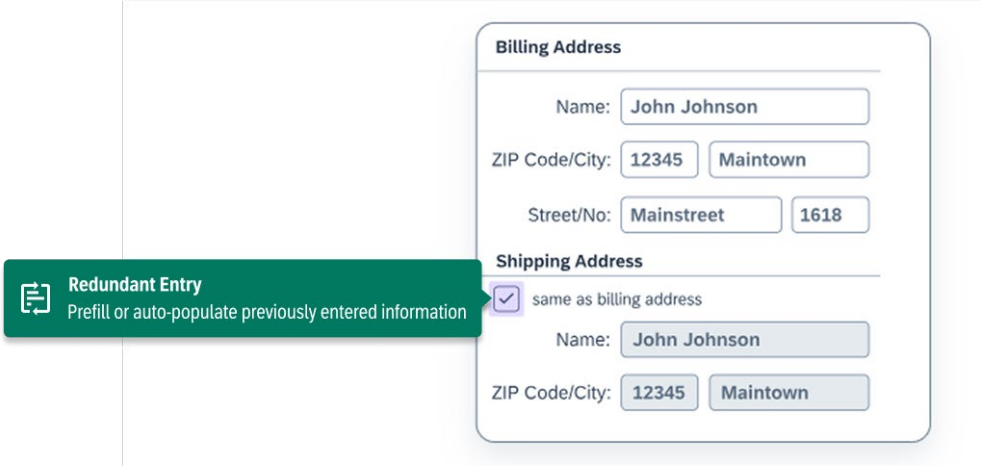


Figure 197: Dialog showing how the shipping address in a checkout process is auto populated, when the option “same as billing address” is selected. This is annotated with the redundant entry annotation

Shared Benefits

Limited Mobility

Interactions

Minimize physical effort for users with limited dexterity by avoiding the need to retype lengthy entries. Instead, incorporate features such as autofill and straightforward selection options, such as checkboxes. For instance, include a choice to set the billing address as the shipping address and implement single sign-on (SSO) or account prefill options whenever feasible.

For Developers

Prefill information as often as possible.

Use caching mechanisms of the platform to reuse previously entered information to auto-populate respective input on new pages.

References

WCAG 2.2 Reference

3.3.7 Redundant Entry (A)

“A convenient file with amazing ready to use assets available.”

Zhanara Baisalova – UX Designer

Auditory Experience

Imagine watching a video presentation and never missing important information. What turns audio-only content into inclusive communication? The answer is visual notifications, transcripts, and captions that allow users who are deaf or hard of hearing to stay fully informed and engaged.

This chapter helps designers to address audio content to meet the needs of users who are deaf and hard of hearing.

The Inclusive Auditory Experience

Design your application with visual and text-based communication so everyone can access information, especially users who are deaf and hard of hearing.

Users who are deaf and hard of hearing benefit from interfaces that provide multiple ways to access and communicate information. When interfaces support different communication methods, users can navigate and interact with confidence.

Interfaces with vibration alerts and visual notifications help users who are deaf or hard of hearing get essential updates. These combined approaches also benefit users who have their audio turned off, users with cognitive disabilities who find audio alerts distracting, and anyone in environments where audio isn't practical.

Captions enable users who are deaf or hard of hearing to follow audio and video content as it plays. These synchronized text alternatives also help people with cognitive disabilities who process text better than audio, have sound sensitivities, or are easily distracted by background noise. Captions can benefit all users who want to access audio content in noisy or quiet settings.

Transcripts benefit users who are deaf or hard of hearing. Unlike captions, they can access content without relying on video playback, allowing them the freedom to read at their own pace. Transcripts also benefit anyone who wants to skim, review, or search content for specific information.

Principles

Hard of Hearing

Persona: Harold

"I have a severe hearing loss. I struggle to acquire knowledge or follow meetings when audio alternatives are not available: captions, transcripts, or sign language."

Hearing differences can affect people in many ways. According to the World Health Organization, disabling hearing loss is defined as hearing loss greater than 35 decibels in the ear that hears better, meaning the ear with the least hearing loss or the one that can detect sounds more clearly. Individuals who identify as Deaf or hard of hearing may experience a broad spectrum of hearing levels.[22] These terms describe experiences rather than limitations.

The prevalence of hearing loss increases with age. Among people older than 60 years, over 25 percent are affected by disabling hearing loss (WHO, 2025). These individuals may encounter various challenges when using digital products due to their communication needs and preferences.

Common causes of hearing loss are:

- Age, Medication, Infection
- Foreign body, Fluid in the middle ear
- Perforated tympanic membrane
- Otosclerosis
- Neurologic conditions

Some statistics:

- Over 430 million people worldwide have disabling hearing loss.[22]
- About 15% of US adults report some hearing difficulty.[17]
- In the U.S., hearing loss affects about one-third of people aged 65-74 and nearly half of those over 75.[13]

- 12.3% of interviewed US adults had some difficulties hearing even when using a hearing aid in 2020.[3]

Permanent disabilities include users with chronic hearing loss or impairment that may be associated to genetic factors. It may also include age-related hearing loss (presbycusis), or due to noise exposure, infections and illnesses, injury or trauma and medication.

Situational & Temporary Disabilities

By integrating situational and temporary disabilities into this disability group, we help design teams address challenges that might not be permanent but still impact user experience under certain conditions.

Temporary hearing challenges, such as those caused by noisy environments or infections, also affect many people. These are some examples of situational and temporary disability for hearing.

- Background noise: A noisy environment, such as a crowded restaurant or public transportation, could temporarily impair hearing, making it hard to hear conversations or instructions.
- Temporary hearing loss: Conditions like ear infections, colds, or exposure to loud noises could lead to short-term hearing loss.
- Fatigue or stress: Mental fatigue or emotional stress could cause temporary difficulty processing sounds, making communication harder.

General Design Tips

1. **Captions, Transcripts, and Sign Language Interpretation:**
   Audio cues in videos, podcasts, or voice instructions can often be inaccessible, causing important information to be missed. Video conferencing tools may lack real-time captions or transcription, and inaccessible social media features can limit participation.

   Design Tips: Provide closed captions, subtitles, and transcripts for all audio and video content. Use clear visual alerts and error indicators. Ensure video platforms support live transcription and interpretation. Promote accessible social media and community features.

2. **Real-Time Transcription for Conversations and Meetings:**
   Users may feel excluded without live transcription during one-on-one or small group conversations.

   Design Tips: Support live transcription and captioning features during meetings to ensure inclusive communication.

3. **Adjustable Volume and Vibration Notifications:**
   Audio-only alerts can be missed by users with hearing impairments.

   Design Tips: Incorporate vibration alerts and visible notifications alongside audio cues to ensure that all users notice important updates.

4. **Visual Alerts and Clear Error Signifiers:**
   Relying only on sound for warnings or errors excludes users with hearing loss.

   Design Tips: Use visual cues such as icons, color changes, or message banners to communicate errors and warnings clearly.

5. **Text-Based Communication (Chat and Messaging):**
   Many users may not be aware of assistive technologies such as speech-to-text apps, limiting their communication options.

   Design Tips: Integrate robust chat and messaging features. Support speech-to-text tools that convert spoken words into text and vice versa. Provide user guidance and education on available assistive technologies.

Annotations

Hearing Annotations

This checklist helps designers remember the key accessibility aspects that should be addressed during the design phase.

Auditory Experience Checklist

- [ ] Caption
- [ ] Transcript

Caption

Provide synchronized captions for all pre-recorded and live media containing speech or other meaningful sound.

Component Variants

Caption as an audio media alternative

Figure 198: Indicates written text alternatives for audio content (speech, sounds, etc.)

Live Caption as an audio media alternative

Figure 199: Indicates caption for live audio events

About Captions

People who are Deaf or Hard of Hearing rely on visual alternatives to understand pre-recorded or live audio-based content, such as videos and live events. Captions are visual text alternatives and must be synchronized with the media timeline for spoken dialogue, narration, and important sounds within videos, ensuring that users can fully engage with media and audio content. Pre-recorded tutorials, promotional videos, training materials, and other recorded content are expected to include captions to ensure accessibility.

Live Captioning as per WCAG 2.2 1.2.2 Captions (Prerecorded) (A) all pre-recorded audio in video (synchronized media) should include captions, unless the video is already a text alternative. While captions make speech, sound effects, and audio cues visible, it can also include non-spoken words, for example, meaningful sounds like phone rings and cheering complement the context of the audio. WCAG 2.2 1.2.4 Captions (Live) (AA) recommends providing live captions for live audio content, also for live events ensuring real-time delivery.

Live captions are intended to help Deaf and Hard of Hearing users' access to real-time presentations. They are used to support broadcast of synchronized media but are not required on two-way multimedia calls. Live captions should identify who is speaking and indicate relevant sound effects or other significant audio. Responsibility for providing captions falls to the content providers (the callers) or the "host" caller, and not the application itself.

Automated live captions may contain errors if the audio quality is poor, or the content is complex. In these cases, captions created by a human captioner may need manual editing. assuming an audio transcript was created. When an audio transcript is created, a synchronized and searchable transcript will then appear alongside the recorded video, providing more accurate content than the original live captions.

WCAG 2.2 1.2.9 Audio-only (Live) (AAA) recommends live caption as real-time audio alternative synchronized with speech and sounds. This service is typically provided by trained human operators. Captions are preferred over transcripts for live events because they can adapt to unscripted content and follow the pace of the audio. While transcripts may be acceptable for scripted events, they do not meet accessibility needs when audio deviates from the script.

Design Tips

- The annotations allow designers to guide the placement of the caption button and determine where the caption text should appear within the video.
- The design should ensure that captions are clearly visible by considering caption placement, font size, and contrast. Captions should be synchronized with the video and must not block key visuals.
- Where possible, give users control to turn captions on or off. Captions must remain within the video.

User Assistants (UA)

When creating captions include speaker identification and relevant non-speech sounds, such as laughter and door slams. High-quality captions that cover both speech and environmental context help improve comprehension for all users.

As a User Assistant, make sure captions include:

- All spoken dialogue
- Speaker identification, especially in videos with multiple speakers

- Relevant non-speech sounds, such as [Applause], [Dramatic music], [Phone ringing], etc.

Captions vs Transcripts

Use captions when designing media players, video components, or live streaming tools. Include a toggle in your UI and provide space for captions near the video.

Use transcripts in supporting documentation, below or next to media content, especially for podcasts, training materials, and legal content.

For more details consult the comparison table below.

Table 12: Comparison table with key features for caption and transcript.

Feature	Caption	Transcript
Definition	Text displayed on screen that represents audio content in real-time, synchronized with video.	Text Document that contains the full dialogue and audio content. It does not have to be time-synchronized with a video.
Purpose	To provide real-time access to spoken dialogue, sound effects, and music cues during a video.	To allow users to read or search through the content at their own pace, often for reference or review.
Media Type	Video with audio (transcript is optional)	Audio-only
Includes speaker identification	Yes, often	Yes
Supports real-time access	Yes	Yes (but less immediate)
Time-synced to media	Yes	Sometimes, often not precisely
Includes non-speech sounds	Yes (e.g., [laughter], [door slams])	Rarely
Displayed with media	Yes. On screen, during playback	Separate from the media (can be downloaded and viewable as text)
Who benefits	Deaf and hard of hearing users, non-native speakers, anyone in noisy or silent environments	Users who are Deafblind (screen-reader users), people who prefer reading, researchers, or those seeking quick reference

Examples

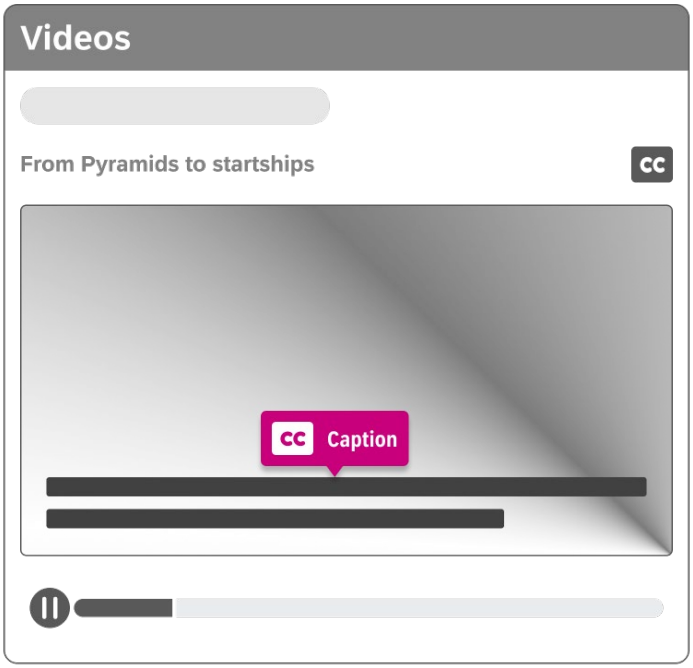


Figure 200: Annotate where and how the captions within the video will appear. Observe placement, text size and contrast.

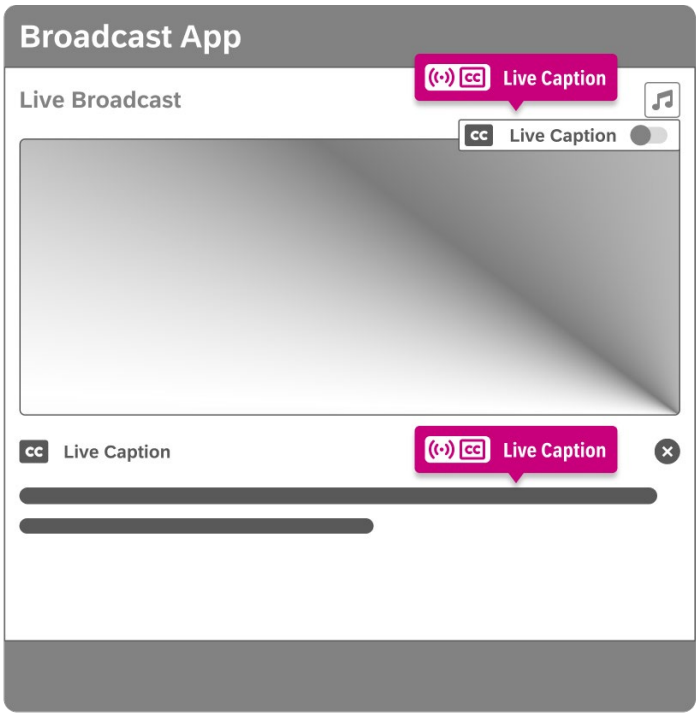


Figure 201: Extra services or functionalities that could be made available are move caption window on the screen, pin or dock the caption window, and translation that could be delivered with live caption.

Shared Benefits

Temporary Disabilities

All users

Captions are not just for Deaf or Hard of Hearing users. They benefit everyone. In noisy settings, such as conferences or public transportation, captions make it possible to follow videos without missing important information. In quiet environments, such as libraries or shared workspaces, they allow users engage with content discreetly.

During meetings, events, or webinars, live captions help non-native speakers keep up with fast speech, unfamiliar accents, or technical terminology, making communication more inclusive. They are especially useful in conferences, live events, and broadcasts, where live captions enable everyone to follow the conversation clearly.

Limited Cognition

Cognitive

Captions can help users with cognitive disabilities, including ADHD and other forms of neurodivergence, as well as anyone who processes text more easily than audio. Reading along with spoken content can reinforce memory, clarify meaning, and reduce cognitive load.

Captions also provide a visual buffer for users who are sensitive to sound or easily distracted by background noise, allowing them to engage with media without being overwhelmed by sudden or repetitive audio. When combined with system preferences like “Reduce Motion” or “Do Not Disturb”, captions support a more comfortable and focused experience.

For Developers

Always use semantic `<track>` elements for captions and provide accessible transcript links in code. Make sure that they load correctly across browsers and support multiple languages when needed.

Add `<track>` elements in your `<video>` tag:

```
<video controls> <source src="example.mp4" type="video/mp4"
/> <track src="captions-en.vtt" kind="captions" srclang="en"
label="English" default /> </video>
```

- Provide captions for all spoken content and meaningful sounds.
- Include multiple languages when required (using `srclang` + `label`).
- Test captions on major browsers and devices, including both desktop and mobile.

iOS and Android

Please refer to the respective platform documentation and guidelines for video controls and captions.

References

WCAG 2.2 Reference

1.2.2 Captions (Prerecorded) (A)

1.2.4 Captions (Live) (AA)

1.2.9 Audio-only (Live) (AAA)

Transcript

Provide text transcripts for all audio-only content to support users who prefer reading or rely on screen readers. Transcripts should include spoken dialogue and relevant sound cues when appropriate.

Component Variants

Transcript as an audio media alternative

Figure 202: Indicates written text alternatives for audio content (speech, sounds, etc.)

Live Transcript as an audio media alternative

Figure 203: Indicates written text alternatives for audio content (speech, sounds, etc.)

About Transcripts

People who are Deaf or Hard of Hearing rely on visual alternatives to understand audio-based content. Transcripts are text-based documents that provided as an alternative version of audio content, including dialog, speech, and meaningful sounds. They can be used for pre-recorded or live media such as interviews, podcasts, and webinars, allowing users to read, search, and reference the information at their own pace. This makes the experience more inclusive for everyone.

Pre-recorded transcripts are especially helpful for users who need complete control over timing, such as those who read transcripts at their own pace. They are useful when users need to skim, search, or copy text from a media source.

There are two main types of transcripts:

- Plain Transcript: Covers all spoken dialogue or narration ensuring that users who cannot hear can access the full meaning of the content. It does not include visual descriptions or audio cues, and it is typically

used for podcasts, interviews, and audio messages. Reference for audio only: WCAG 2.2 SC 1.2.1 Audio-only and Video-only (Prerecorded) (A).

- Full Description Transcript: Includes spoken dialogue plus detailed visual descriptions, providing a comprehensive alternative to video media. This type benefits not only deaf users, but also those who cannot see the video, such as blind or low vision users, describing important visual elements. Example: “The presenter picks up a pen and pretends to draw a circle over the diagram.” Reference for video with visuals: WCAG 2.2 SC 1.2.3 Audio Description or Media Alternative (Prerecorded) (A) and SC 1.2.8 Audio Description or Media Alternative (Prerecorded) (AAA).

Live Transcripts (for Audio-only or Video Content)

Live transcripts provide a real-time, running text version of live audio content. They support Deaf, Hard of Hearing users, non-native speakers, and people in noisy environments follow the content as it unfolds. Unlike live captions, live captions are not strictly time-synced, may include speaker names, and can omit sound effects. They are especially useful when live captioning is not available or practical.

Live transcripts can:

- Be displayed alongside the live media
- Include dialogue and sometimes speaker names
- Not be perfectly time-synced like live captions
- Usually do not include sound effects or visual cues unless added manually

Live transcripts are useful when full captioning is not feasible. They provide a valuable way to follow content in real time, review information, or skim for key points. Because they are more flexible in format and delivery, they may sometimes lag slightly more than captions.

Design tips for annotating transcripts:

- Provide clear buttons or links labeled “Transcript” that are easy to find.
- Position transcripts below the video or player in a collapsible section with a download option, such as a download link.
- Clearly label all transcript links or buttons (e.g., “Transcript”)
- Allow users to control playback speed and transcript scrolling when possible.
- For live content, offer real-time text through a visible transcript panel or chat-style display

When audio-only live content is presented, provide a real-time transcript, live notes, or chat-based summaries. If possible, offer a text alternative

channel for participation, such as a Q&A in chat. Allow users to control the pace by pausing or replaying the audio when streaming platforms support it. Most importantly, provide post-event captioned recordings or full transcripts.

User Assistant Roles

Provide clear transcripts so users can read content at their own pace without relying on sound. Note whether the transcript includes speaker names or non-verbal cues. Make sure it fully covers spoken content, audio descriptions, file format information, and download options. Also include descriptions of any visual elements shown in the media.

Examples

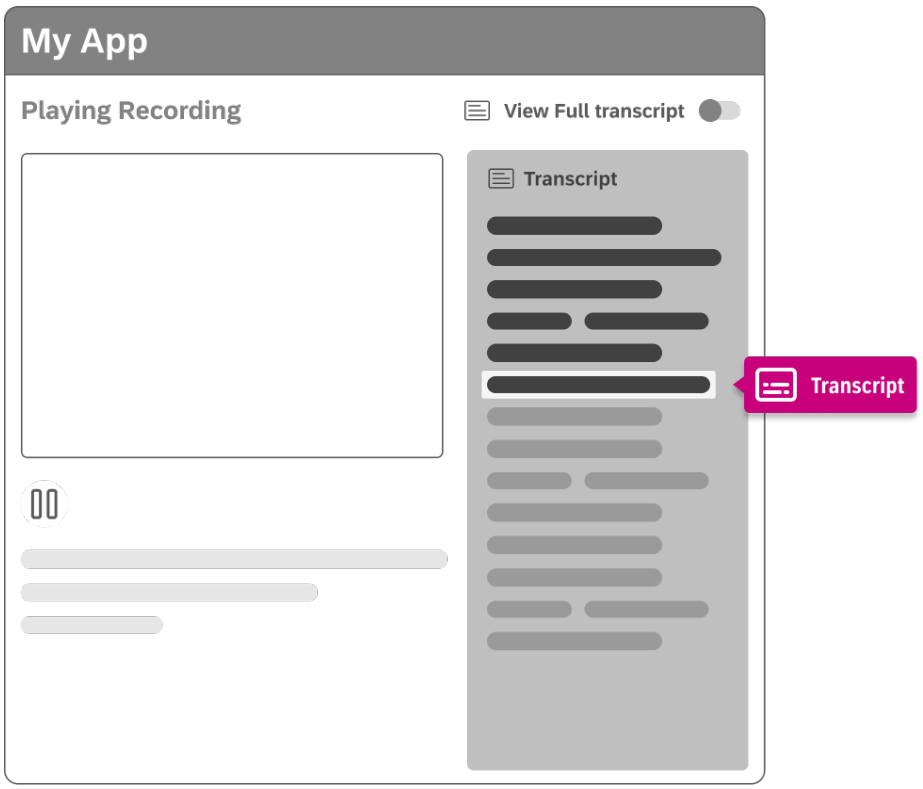


Figure 204: Pre-recorded audio with an option to show transcript

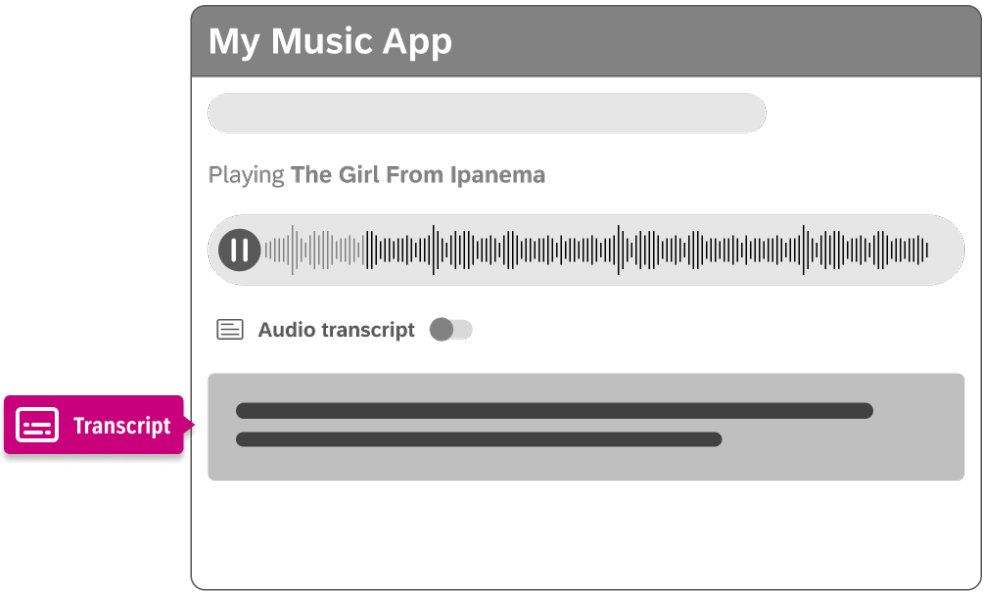


Figure 205: Pre-recorded audio with an option to show transcript

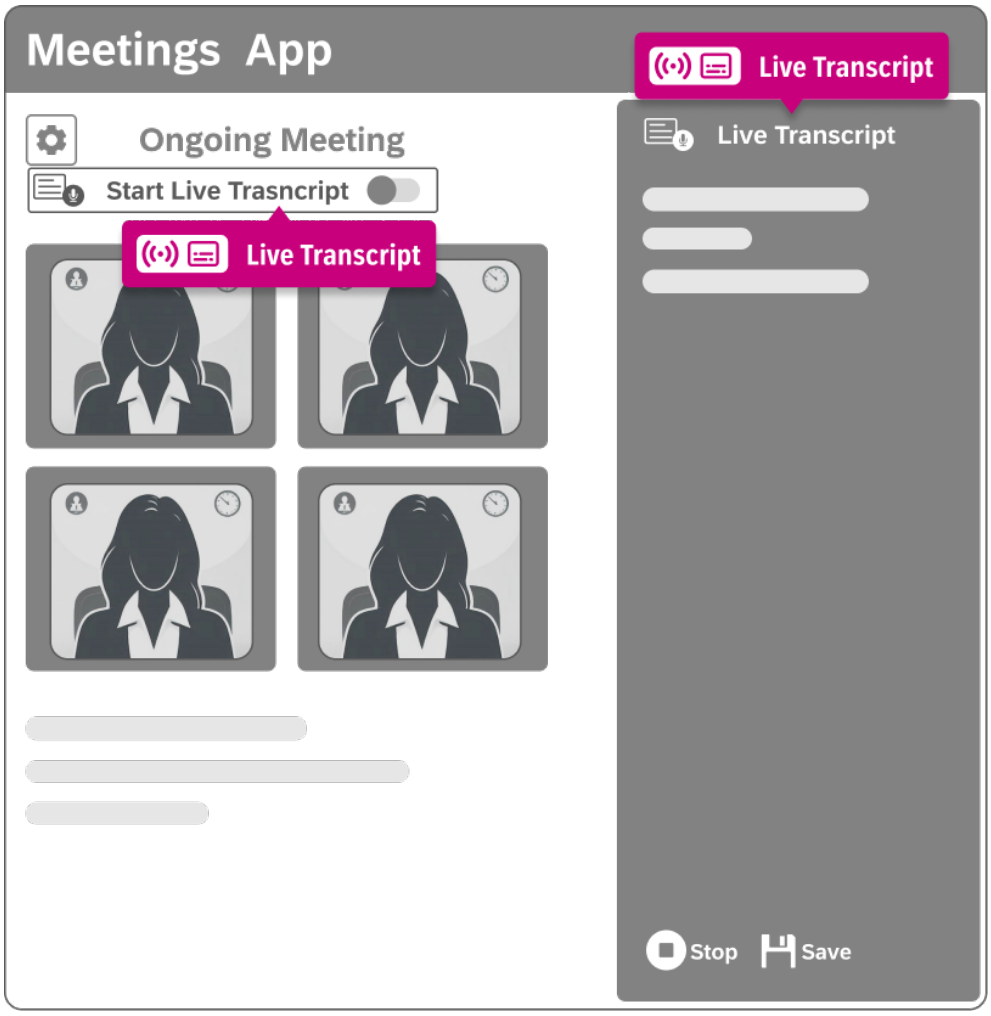


Figure 206: Live transcript window: functionalities such as stop and save are available to the user.

Shared Benefits

Temporary Disabilities

All users

Transcripts are a valuable resource for everyone, and not just for accessibility. They make it easy to skim or search through audio or video content to find specific information quickly, which is especially useful for busy professionals, students, or researchers.

In meetings or webinars, transcripts offer a convenient way to review what was said, catch up if something was missed, or reference quotes accurately.

They are also helpful for people in quiet environments or those who prefer reading over listening. Live transcripts help people who are non-native speakers in conferences, events, and broadcasts follow real-time conversation, especially if the speaker talks fast or uses unfamiliar terms.

Without Vision

Screen Reading

While Blind users may not benefit from captions, transcripts offer a scannable, screen-reader-friendly alternative to audio content. A well-structured transcript, especially one enhanced with headings, timecodes, or ARIA landmarks, allows users to navigate, search, or review the material without needing to listen from start to finish.

Transcripts also minimize reliance on complex sound-based navigation, which can conflict with screen reader output. When audio cues, such as chimes or voice prompts, are used, they should be clearly documented, optional, and timed to avoid overlap with assistive technology feedback. In summary, transcripts empower Blind users with greater control, efficiency, and independence when accessing media content.

Limited Cognition

Cognitive

Transcripts support users with cognitive or learning disabilities by allowing them to process content at their own pace. The ability to scan, reread, highlight, or translate written information helps users retain key ideas and manage attention.

For individuals who experience anxiety, language processing challenges, or information overload, transcripts offer a calm, predictable alternative to listening. They also promote task continuity, allowing users to return to key information when needed, and without the pressure of real-time comprehension.

For Developers

Always use semantic <track> elements (for captions) and accessible transcript links in code. Verify they load correctly across browsers, support multiple languages when needed.

Provide a transcript file (HTML or text) linked near the media:

`<a href="t`ranscript.html">Read Transcript

Ensure transcript includes:

- Spoken dialogue
- Relevant sound effects
- Speaker identification

Make it accessible: plain HTML text (no images of text).

iOS and Android

Please refer to the respective platform documentation and guidelines for adding transcripts to media content.

References

WCAG 2.2 Reference

1.2.1 Audio-only and video-only (Prerecorded) (A)

1.2.3 Audio Description or Media Alternative (Prerecorded) (A)

1.2.8 Media Alternative (Prerecorded) (AAA)

Closing Remarks

Knowledge transforms into impact when practiced.

We hope you found this updated guideline helpful. If you have made it this far, congratulations! Your journey toward accessible design is well underway. Now it is your turn to lead the way.

Crawl. Walk. Run. Iterate.

We know that building internal alignment is not always easy. Like any complex practice, accessibility in design demands intention, planning, collaboration, and persistence. Do not let obstacles slow you down, start small, stay focused, and keep learning.

Always remember accessibility is amplified usability. If you are feeling like a team of one, start with what is in your control.

Choose a few meaningful annotations and embed them into your current design process. Every element you document, label, or clarify brings real value to the user experience.

This could be the spark that changes how your team thinks about inclusion. This could be the shift that inspires a broader conversation across your organization. Be patient, be persistent, and above all, stay curious.

Let this edition be your toolkit. We are excited to see what you do next.

Irla, Nina and Stefan

Authors

Irla Bocianoski Rebelo

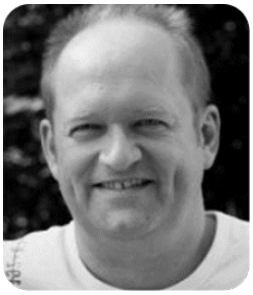
User Experience Design Expert and Accessibility Lead
SAP Success Factors

Montreal, Canada

Nina Krauss

Accessibility Specialist
SAP Accessibility and Inclusive Design

Walldorf, Germany



Stefan Schnabel

User Experience Design Expert
SAP Accessibility and Inclusive Design

Walldorf, Germany

Contributors

Handbook Layout

Anton Steckert
Accessibility Specialist

Cover Design

Alexandra Yaneva
Accessibility Specialist

Olga Kordowski
Senior UX Designer

Collaborators

Anton Fischer
UX Designer SuccessFactors

Alvaro Laura Garcia
Senior User Experience Design Specialist Signavio

Liu Joy
UX Designer SuccessFactors

Sanket Kundu
UX Designer SuccessFactors

Ryan Lum
Senior User Experience Design Specialist Concur

Gundula Niemann
Accessibility Expert

Tanisha Rastogi
UX Designer SuccessFactors

Reviewers

Roldon Brown
Accessibility Specialist

Roland Buss
User Interface Design Expert

Tananda Darling
Accessibility Compliance Specialist

Carrie Hall
Senior Product Inclusion Specialist

Kimberly McGee
Senior Equity Strategist

Testimonials

Zhanara Baisalova
UX Designer CX
CX Product Design

Marvin Gille
UX Designer & Accessibility Advocate
Web Design System

Hanna Klymenko
Product Design Expert SuccessFactors
People PM Learning

Aisling Noone
UX Designer
UX Design Concepts

Sup Suh
Senior Product Designer
Mobile Design System

Adelina Todorova
UX Designer
Web Design System

Dominika Zamojska
UX Designer and Accessibility Lead CX UX
CX Product Design

Appendix

WCAG 2.2 Coverage

SAP created this set of annotations in alignment with the Web Content Accessibility Guidelines (WCAG) version 2.2 from W3C.

The WCAG 2.2 guideline is divided into 4 groups of Success Criterion and 78 success criteria used by many organizations to create accessibility reports.

This documentation covers all A and AA success criteria and some AAA.

Annotation Symbols

 Foundation

 Visual Experience

 Interaction Experience

 Screen Reader Experience

 Cognitive Experience

 Auditory Experience

Perceivable

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

1.1 Text Alternative

1.1.1 Non-Text Content (A) 

1.2 Time-Based Media

1.2.1 Audio-only and video-only (Prerecorded) (A) 

1.2.2 Captions (Prerecorded) (A) 

1.2.3 Audio Description or Media Alternative (Prerecorded) (A) 

1.2.4 Captions (Live) (AA) 

1.2.5 Audio Description (Prerecorded) (AA) 

1.2.6 Sign Language (Prerecorded) (AAA) no annotation

1.2.7 Extended Audio Description (Prerecorded) (AAA) 

1.2.8 Media Alternative (Prerecorded) (AAA) 

1.2.9 Audio-only (Live) (AAA) 

1.3 Adaptable

1.3.1 Info and Relationships (A) 

1.3.2 Meaningful Sequence (A) 

1.3.3 Sensory Characteristics (A) 

1.3.4 Orientation (AA) 

1.3.5 Identify Input Purpose (AA) 

1.3.6 Identify Purpose (AAA) 

1.4 Distinguishable

1.4.1 Use of Color (A) 

1.4.2 Audio Control (A) 

1.4.3 Contrast (Minimum) (AA) 

1.4.4 Resize text (AA) 

1.4.5 Images of Text (AA) 

1.4.6 Contrast (Enhanced) (AAA) no annotation

1.4.7 Low or No Background Audio (AAA) no annotation

1.4.8 Visual Presentation (AAA) 

1.4.9 Images of Text (No Exception) (AAA) no annotation

1.4.10 Reflow (AA) 

1.4.11 Non-text Contrast (AA) 

1.4.12 Text Spacing (AA) 

1.4.13 Content on Hover or Focus (AA) 

Operable

User interface components and navigation must be operable.

2.1 Keyboard Accessible

2.1.1 Keyboard (A) 

2.1.2 No Keyboard Trap (A) 

2.1.3 Keyboard (No Exception) (AAA) 

2.1.4 Character Key Shortcuts (A) 

2.2 Enough Time

2.2.1 Timing Adjustable (A) 

2.2.2 Pause, Stop, Hide (A) 

2.2.3 No Timing (AAA) 

2.2.4 Interruptions (AAA) 

2.2.5 Re-Authenticating (AAA) no annotation

2.2.6 Timeouts (AAA)

2.3 Seizures and Physical Reactions

2.3.1 Three Flashes or Below Threshold (A)

2.3.2 Three Flashes (AAA)

2.3.3 Animation from Interactions (AAA) no annotation

2.4 Navigable

2.4.1 Bypass Blocks (A)

2.4.2 Page Title (A)

2.4.3 Focus Order (A)

2.4.4 Link Purpose (In Context) (A)

2.4.5 Multiple Ways (AA)

2.4.6 Headings and Labels (AA)

2.4.7 Focus Visible (AA)

2.4.8 Location (AAA)

2.4.9 Link Purpose (Link Only) (AAA)

2.4.10 Section Headings (AAA)

2.4.11 Focus Not Obscured (Minimum) (AA)

2.4.12 Focus Not Obscured (Enhanced) (AAA)

2.4.13 Focus Appearance (AAA)

2.5 Input Modalities

2.5.1 Pointer Gestures (A)

2.5.2 Pointer Cancellation (A)

2.5.3 Label in Name (A)

2.5.4 Motion Actuation (A)

2.5.5 Target Size (AAA)

2.5.6 Concurrent Input Mechanisms (AAA)

2.5.7 Dragging Movements (AA)

2.5.8 Target Size (Minimum) (AA)

Understandable

Information and the operation of user interface must be understandable.

3.1 Readable

3.1.1 Language of Page (A)

3.1.2 Language of Parts (AA)

3.1.3 Unusual Words (AAA)

3.1.4 Abbreviations (AAA)

3.1.5 Reading Level (AAA)

3.1.6 Pronunciation (AAA) no annotation

3.2 Predictable

3.2.1 On Focus (A)

3.2.2 On Input (A)

3.2.3 Consistent Navigation (AA)

3.2.4 Consistent Identification (AA)

3.2.5 Change on Request (AAA) no annotation

3.2.6 Consistent Help (A)

3.3 Input Assistance

3.3.1 Error Identification (A)

3.3.2 Labels or Instructions (A)

3.3.3 Error Suggestion (AA)

3.3.4 Error Prevention (Legal, Financial, Data) (AA)

3.3.5 Help (AAA)	no annotation
3.3.6 Error Prevention (all) (AAA)	no annotation
3.3.7 Redundant Entry (A)	
3.3.8 Accessible Authentication (Minimum) (AA)	no annotation
3.3.9 Accessible Authentications (Enhanced) (AAA)	no annotation

Robust

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

4.1 Compatible

4.1.1 Parsing (A)	(obsolete in WCAG 2.2 and removed)
4.1.2 Name, Role, Value (A)	
4.1.3 Status Messages (AA)	

Conformance

For a web page to conform to WCAG 2.2, all of the following conformance requirements must be satisfied:

5.1 Interpreting Normative Requirements

5.1 Interpreting Normative Requirements	no annotation

5.2 Conformance Requirements Conformance Claims (Optional)

5.2.1 Conformance Level	no annotation
5.2.2 Full Pages	no annotation
5.2.3 Complete Process	no annotation
5.2.4 Only Accessibility Supported Ways of Using Technologies	no annotation
5.2.5 Non-Interference	no annotation

5.3 Conformance Claims (Optional)

5.3.1 Required Components of a Conformance Claim	no annotation
5.3.2 Optional Components of a Conformance Claim	no annotation

5.4 Statement of Partial Conformance - Third Party Content

5.4 Statement of Partial Conformance - Third Party Content 

5.5 Statement of Partial Conformance – Language

5.5 Statement of Partial Conformance - Language no annotation

5.6 Privacy Considerations

5.6 Privacy Considerations no annotation

5.7 Security Considerations

5.7 Security Considerations no annotation

References

1. ADA Site Compliance. (2023). 2023 ADA web accessibility lawsuit statistics – Full report. https://adasitecompliance.com/2023-ada-web-accessibility-lawsuit-statistics-full-report/

2. American Academy of Ophthalmology. (2022). Glaucoma. https://www.aao.org/eye-health/diseases/what-is-glaucoma

3. Centers for Disease Control and Prevention. (2022). QuickStats: Percentage of adults aged ≥18 years who have difficulty hearing even when using a hearing aid, by age group — National Health Interview Survey, United States, 2020. *MMWR Morbidity and Mortality Weekly Report, 71*(12). https://www.cdc.gov/mmwr/volumes/71/wr/mm7112a5.htm

4. Centers for Disease Control and Prevention. (2023). Diabetic retinopathy. https://www.cdc.gov/diabetes/diabetes-complications/diabetic-retinopathy.html

5. Centers for Disease Control and Prevention. (2025). Disability impacts all of us infographic. https://www.cdc.gov/disability-and-health/articles-documents/disability-impacts-all-of-us-infographic.html

6. Centers for Disease Control and Prevention. (2025). Related conditions. https://www.cdc.gov/disability-and-health/conditions/index.html

7. Community Eye Health Journal. (2018). World blindness and visual impairment: Despite many successes, the problem is growing. *Community Eye Health Journal*. https://cehjournal.org/articles/10.56920/cehj.342

8. Frontiers in Physiology. (2020). Mobility in older community-dwelling persons: A narrative review. *Frontiers in Physiology, 11*, 881. https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2020.00881/full

9. Global Burden of Disease Study 2021 Collaborators. (2025). A systematic analysis for the Global Burden of Disease Study 2021. *The Lancet, 403*(10440), 2133–2162. https://doi.org/10.1016/S0140-6736(24)00812-2

10. IBM Systems Sciences Institute. (2023). The cost of finding bugs later in the software development lifecycle. https://www.functionize.com/blog/the-cost-of-finding-bugs-later-in-the-sdlc

11. Journal of the Optical Society of America. (2012). Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America A*, *29*(3), 313–320. https://opg.optica.org/josaa/abstract.cfm?URI=josaa-29-3-313

12. National Eye Institute. (2023). Facts about blindness.

13. National Institute on Deafness and Other Communication Disorders. (2023, August 23). Age-related hearing loss. National Institutes of Health. https://www.ncbi.nlm.nih.gov/books/NBK559220

14. Psychiatry Research. (2023). Prevalence of attention deficit hyperactivity disorder in adults: A systematic review and meta-analysis. *Psychiatry Research, 328*, 115042. https://pubmed.ncbi.nlm.nih.gov/37708807/

15. Straits Research. (2024). Accessibility testing market size, share & growth forecast by 2033. https://straitsresearch.com/report/accessibility-testing-market

16. The Lancet Psychiatry. (2023). The global prevalence of ADHD in children and adolescents: A systematic review and meta-analysis. *The Lancet Psychiatry, 10*(8), 567–578. https://pubmed.ncbi.nlm.nih.gov/37495084/

17. Vital Health Stat 10. (2014). Summary health statistics for U.S. adults: National Health Interview Survey, 2012. *National Center for Health Statistics*. https://pubmed.ncbi.nlm.nih.gov/24819891/

18. Wagner, R. K., Zirps, F. A., Edwards, A. A., Wood, S. G., Joyner, R. E., Becker, B. J., Liu, G., & Beal, B. (2020). The prevalence of dyslexia: A new approach to its estimation. *PLOS ONE*, *15*(6), e0233877. https://pmc.ncbi.nlm.nih.gov/articles/PMC8183124/

19. World Health Organization. (2019). International Classification of Diseases 11th Revision (ICD-11) – Neurocognitive disorders. https://icd.who.int/dev11/f/en

20. World Health Organization. (2023). Fact sheet on blindness and vision impairment. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

21. World Health Organization. (2024, March 14). Over 1 in 3 people affected by neurological conditions, the leading cause of illness and disability worldwide. https://www.who.int/news/item/14-03-2024-over-1-in-3-people-affected-by-neurological-conditions--the-leading-cause-of-illness-and-disability-worldwide

22. World Health Organization. (2025). Deafness and hearing loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

23. World Wide Web Consortium. (2018). W3C issues improved accessibility guidance for websites and applications. https://www.w3.org/press-releases/2018/wcag21/