

Accessible Rich Internet Applications (WAI-ARIA) 1.2

W3C Recommendation 06 June 2023



▼ More details about this document

This version:

<https://www.w3.org/TR/2023/REC-wai-aria-1.2-20230606/>

Latest published version:

<https://www.w3.org/TR/wai-aria-1.2/>

Latest editor's draft:

<https://w3c.github.io/aria/>

History:

<https://www.w3.org/standards/history/wai-aria-1.2>

[Commit history](#)

Implementation report:

<https://w3c.github.io/test-results/core-aam-1.2/>

Previous Recommendation:

<https://www.w3.org/TR/wai-aria-1.1/>

Editors:

Joanmarie Diggs ([Igalia, S.L.](#))

James Nurthen ([Adobe](#))

Michael Cooper ([W3C](#))

Carolyn MacLeod ([IBM](#))

Former editors:

Shane McCarron (Spec-Ops) (Editor until 2018)

Richard Schwerdtfeger ([Knowbility](#)) (Editor until October 2017)

James Craig ([Apple Inc.](#)) (Editor until May 2016)

Feedback:

[GitHub w3c/aria](#) ([pull requests](#), [new issue](#), [open issues](#))

Errata:

[Errata exists.](#)

See also [translations](#).

[Copyright](#) © 2013-2023 [World Wide Web Consortium](#). W3C[®] [liability](#), [trademark](#) and [permissive document license](#) rules apply.

Abstract

Accessibility of web content requires semantic information about widgets, structures, and behaviors, in order to allow assistive technologies to convey appropriate information to persons with disabilities. This specification provides an ontology of roles, states, and properties that define accessible user interface elements and can be used to improve the accessibility and interoperability of web content and applications. These semantics are designed to allow an author to properly convey user interface behaviors and structural information to assistive technologies in document-level markup. This version adds features new since [WAI-ARIA 1.1 \[wai-aria-1.1\]](#) to improve interoperability with assistive technologies to form a more consistent accessibility model for [\[HTML\]](#) and [\[SVG2\]](#). This specification complements both [\[HTML\]](#) and [\[SVG2\]](#).

This document is part of the [WAI-ARIA](#) suite described in the [WAI-ARIA Overview](#).

Status of This Document

This section describes the status of this document at the time of its publication. A list of current [W3C](#) publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <https://www.w3.org/TR/>.

[WAI-ARIA 1.2](#) is a [W3C](#) Recommendation. The Advisory Committee ([AC](#)) as well as the [W3C](#) Director have endorsed this specification to become a [W3C](#) Recommendation. For details about implementation experience, see the [WAI-ARIA 1.2 Implementation Report](#). A [history of changes to WAI-ARIA 1.2](#) is available in the appendix.

This document was published by the [Accessible Rich Internet Applications Working Group](#) as a Recommendation using the [Recommendation track](#).

[W3C](#) recommends the wide deployment of this specification as a standard for the Web.

A [W3C](#) Recommendation is a specification that, after extensive consensus-building, is endorsed by [W3C](#) and its Members, and has commitments from Working Group members to [royalty-free licensing](#) for implementations.

This document was produced by a group operating under the [W3C Patent Policy](#). [W3C](#) maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

This document is governed by the [2 November 2021 W3C Process Document](#).

Table of Contents

Abstract

Status of This Document

1. Introduction

- 1.1 Rich Internet Application Accessibility
- 1.2 Target Audience
- 1.3 User Agent Support
- 1.4 Co-Evolution of ~~WAI-ARIA~~ and Host Languages
- 1.5 Authoring Practices
 - 1.5.1 Authoring Tools
 - 1.5.2 Testing Practices and Tools
- 1.6 Assistive Technologies

2. Important Terms

3. Conformance

- 3.1 Non-interference with the Host Language
- 3.2 All ~~WAI-ARIA~~ in DOM
- 3.3 Assistive Technology Notifications Communicated to Web Applications
- 3.4 Conformance Checkers
- 3.5 Deprecated Requirements

4. Using ~~WAI-ARIA~~

- 4.1 ~~WAI-ARIA~~ Roles
- 4.2 ~~WAI-ARIA~~ States and Properties
- 4.3 Managing Focus and Supporting Keyboard Navigation
 - 4.3.1 Information for Authors
 - 4.3.2 Information for User Agents

5. The Roles Model

- 5.1 Relationships Between Concepts
 - 5.1.1 Superclass Role
 - 5.1.2 Subclass Roles
 - 5.1.3 Related Concepts
 - 5.1.4 Base Concept
- 5.2 Characteristics of Roles
 - 5.2.1 Abstract Roles
 - 5.2.2 Required States and Properties

- 5.2.3 Supported States and Properties
- 5.2.4 Inherited States and Properties
- 5.2.5 Prohibited States and Properties
- 5.2.6 Required Owned Elements
- 5.2.7 Required Context Role
- 5.2.8 Accessible Name Calculation
 - 5.2.8.1 Name Computation
 - 5.2.8.2 Description Computation
 - 5.2.8.3 Accessible Name and Description Computation
 - 5.2.8.4 Roles Supporting Name from Author
 - 5.2.8.5 Roles Supporting Name from Content
 - 5.2.8.6 Roles which cannot be named (Name prohibited)
- 5.2.9 Presentational Children
- 5.2.10 Implicit Value for Role
- 5.3 Categorization of Roles
 - 5.3.1 Abstract Roles
 - 5.3.2 Widget Roles
 - 5.3.3 Document Structure Roles
 - 5.3.4 Landmark Roles
 - 5.3.5 Live Region Roles
 - 5.3.6 Window Roles
- 5.4 Definition of Roles
- 6. Supported States and Properties**
 - 6.1 Clarification of States versus Properties
 - 6.2 Characteristics of States and Properties
 - 6.2.1 Related Concepts
 - 6.2.2 Used in Roles
 - 6.2.3 Inherits into Roles
 - 6.2.4 Value
 - 6.3 ARIA Attributes
 - 6.3.1 Multi-value Attribute Values
 - 6.3.2 IDL reflection of ARIA attributes
 - 6.3.3 Operating System Accessibility API mapping of multi-value ARIA attributes
 - 6.3.4 ARIA nullable DOMString Attributes
 - 6.3.4.1 Example Attribute Usage
 - 6.4 Translatable States and Properties
 - 6.5 Global States and Properties
 - 6.6 Taxonomy of WAI-ARIA States and Properties
 - 6.6.1 Widget Attributes

- 6.6.2 Live Region Attributes
- 6.6.3 Drag-and-Drop Attributes
- 6.6.4 Relationship Attributes
- 6.7 Definitions of States and Properties (all aria-* attributes)
- 7. Accessibility Tree**
 - 7.1 Excluding Elements from the Accessibility Tree
 - 7.2 Including Elements in the Accessibility Tree
- 8. Implementation in Host Languages**
 - 8.1 Role Attribute
 - 8.2 State and Property Attributes
 - 8.3 Focus Navigation
 - 8.4 Implicit WAI-ARIA Semantics
 - 8.5 Conflicts with Host Language Semantics
 - 8.6 State and Property Attribute Processing
 - 8.6.1 ID Reference Error Processing
 - 8.7 CSS Selectors
- 9. Handling Author Errors**
 - 9.1 Roles
 - 9.2 States and Properties
- 10. IDL Interface**
 - 10.1 Interface Mixin ARIAMixin
 - 10.2 ARIA Attribute Correspondence
 - 10.2.1 Disambiguation Pattern
 - 10.2.2 IDL Attribute Name Notes or Exceptions
 - 10.3 ARIAMixin Mixed in to Element
 - 10.4 Example IDL Attribute Usage
- 11. Privacy and Security Considerations**
- A. Mapping WAI-ARIA Value types to languages**
- B. Substantive changes since the WAI-ARIA 1.1 Recommendation**
- C. Acknowledgments**
 - C.1 Participants active in the ARIA WG at the time of publication
 - C.2 Other ARIA contributors, commenters, and previously active participants
 - C.3 Enabling funders

D. References

D.1 Normative references

D.2 Informative references

§ Dedication

This version of the ARIA specification is dedicated to the memory of Carolyn MacLeod whose contributions are found throughout this document. She graced our work with equanimity and sagacity, and her untimely passing will long be missed by our community.

§ 1. Introduction

This section is non-normative.

The goals of this specification include:

- expanding the accessibility information that may be supplied by the author;
- requiring that supporting host languages provide full keyboard support that may be implemented in a device-independent way, for example, by telephones, handheld devices, e-book readers, and televisions;
- improving the accessibility of dynamic content generated by scripts; and
- providing for interoperability with [assistive technologies](#).

~~WAI-ARIA~~ is a technical specification that provides a framework to improve the accessibility and interoperability of web content and applications. This document is primarily for developers creating custom widgets and other web application components. Please see the [WAI-ARIA Overview](#) for links to related documents for other audiences, such as [WAI-ARIA Authoring Practices](#) [WAI-ARIA-PRACTICES-1.2] that introduces developers to the accessibility problems that ~~WAI-ARIA~~ is intended to solve, the fundamental concepts, and the technical approach of ~~WAI-ARIA~~.

This document currently handles two aspects of [roles](#): user interface functionality and structural [relationships](#). For more information and use cases, see [WAI-ARIA Authoring Practices](#) [WAI-ARIA-PRACTICES-1.2] for the use of roles in making interactive content accessible.

Roles defined by this specification are designed to support the roles used by platform [accessibility APIs](#). Declaration of these roles on elements within dynamic web content is intended to support interoperability between the web content and assistive technologies that utilize [accessibility APIs](#).

The schema to support this standard has been designed to be extensible so that custom roles can be created by extending base roles. This allows [user agents](#) to support at least the base role, and user agents that support the custom role can provide enhanced access. Note that much of this could be formalized in [XMLSCHEMA11-2]. However, being able to define similarities between roles, such as [baseConcepts](#) and more descriptive definitions, would not be available in XSD.

WAI-ARIA 1.2 is a member of the [WAI-ARIA 1.2 suite](#) that defines how to expose semantics of WAI-ARIA and other web content languages to [accessibility APIs](#).

§ 1.1 Rich Internet Application Accessibility

The domain of web accessibility defines how to make web content usable by persons with disabilities. Persons with certain types of disabilities use [assistive technologies](#) (AT) to interact with content. Assistive technologies can transform the presentation of content into a format more suitable to the user, and can allow the user to interact in different ways. For example, the user may need to, or choose to, interact with a slider widget via arrow keys, instead of dragging and dropping with a mouse. In order to accomplish this effectively, the software needs to understand the [semantics](#) of the content. Semantics is the science of meaning; in this case, used to assign roles, states, and properties that apply to user interface and content elements as a human would understand. For instance, if a paragraph is semantically identified as such, assistive technologies can interact with it as a unit separable from the rest of the content, knowing the exact boundaries of that paragraph. An adjustable range slider or collapsible list (a.k.a. a tree [widget](#)) are more complex examples, in which various parts of the widget have semantics that need to be properly identified for assistive technologies to support effective interaction.

New technologies often overlook semantics required for accessibility, and new authoring practices often misuse the intended semantics of those technologies. [Elements](#) that have one defined meaning in the language are used with a different meaning intended to be understood by the user.

For example, web application developers create collapsible tree widgets in HTML using CSS and JavaScript even though HTML has no semantic tree element. To a non-disabled user, it may look and act like a collapsible tree widget, but without appropriate semantics, the tree widget may not be [perceivable](#) to, or [operable](#) by, a person with a disability because assistive technologies may not recognize the role. Similarly, web application developers create interactive button widgets in SVG using JavaScript even though SVG has no semantic button element. To a non-disabled user, it may look and act like a button widget, but without appropriate semantics, the button widget may not be [perceivable](#) to, or [operable](#) by, a person with a disability because assistive technologies may not recognize the role.

The incorporation of WAI-ARIA is a way for an author to provide proper semantics for custom widgets to make these widgets accessible, usable, and interoperable with assistive technologies. This specification identifies the types of widgets and structures that are commonly recognized by accessibility products, by

providing an [ontology](#) of corresponding [roles](#) that can be attached to content. This allows elements with a given role to be understood as a particular widget or structural type regardless of any semantics inherited from the implementing host language. Roles are a common property of platform [accessibility APIs](#) which assistive technologies use to provide the user with effective presentation and interaction.

The Roles Model includes interaction [widgets](#) and elements denoting document structure. The Roles Model describes inheritance and details the [attributes](#) each role supports. Information about mapping of roles to accessibility APIs is provided by the [Core Accessibility API Mappings](#) [CORE-AAM-1.2].

Roles are element types and will not change with time or user actions. Role information is used by assistive technologies, through interaction with the user agent, to provide normal processing of the specified element type.

States and properties are used to declare important attributes of an element that affect and describe interaction. They enable the [user agent](#) and operating system to properly handle the element even when the attributes are dynamically changed by client-side scripts. For example, alternative input and output technology, such as screen readers and speech dictation software, need to be able to recognize and effectively manipulate and communicate various interaction states (e.g., disabled, checked) to the user.

While it is possible for assistive technologies to access these properties directly through the [Document Object Model](#) [DOM], the preferred mechanism is for the user agent to map the states and properties to the accessibility API of the operating system. See the [Core Accessibility API Mappings](#) [CORE-AAM-1.2] and the [Accessible Name and Description Computation](#) [ACCNAME-1.2] for details.

Figure 1.0 illustrates the relationship between user agents (e.g., browsers), accessibility APIs, and assistive technologies. It describes the "contract" provided by the user agent to assistive technologies, which includes typical accessibility information found in the accessibility API for many of our accessible platforms for GUIs (role, state, selection, [event](#) notification, [relationship](#) information, and descriptions). The DOM, usually HTML, acts as the data model and view in a typical model-view-controller relationship, and JavaScript acts as the controller by manipulating the style and content of the displayed data. The user agent conveys relevant information to the operating system's accessibility API, which can be used by any assistive technologies, such as screen readers.

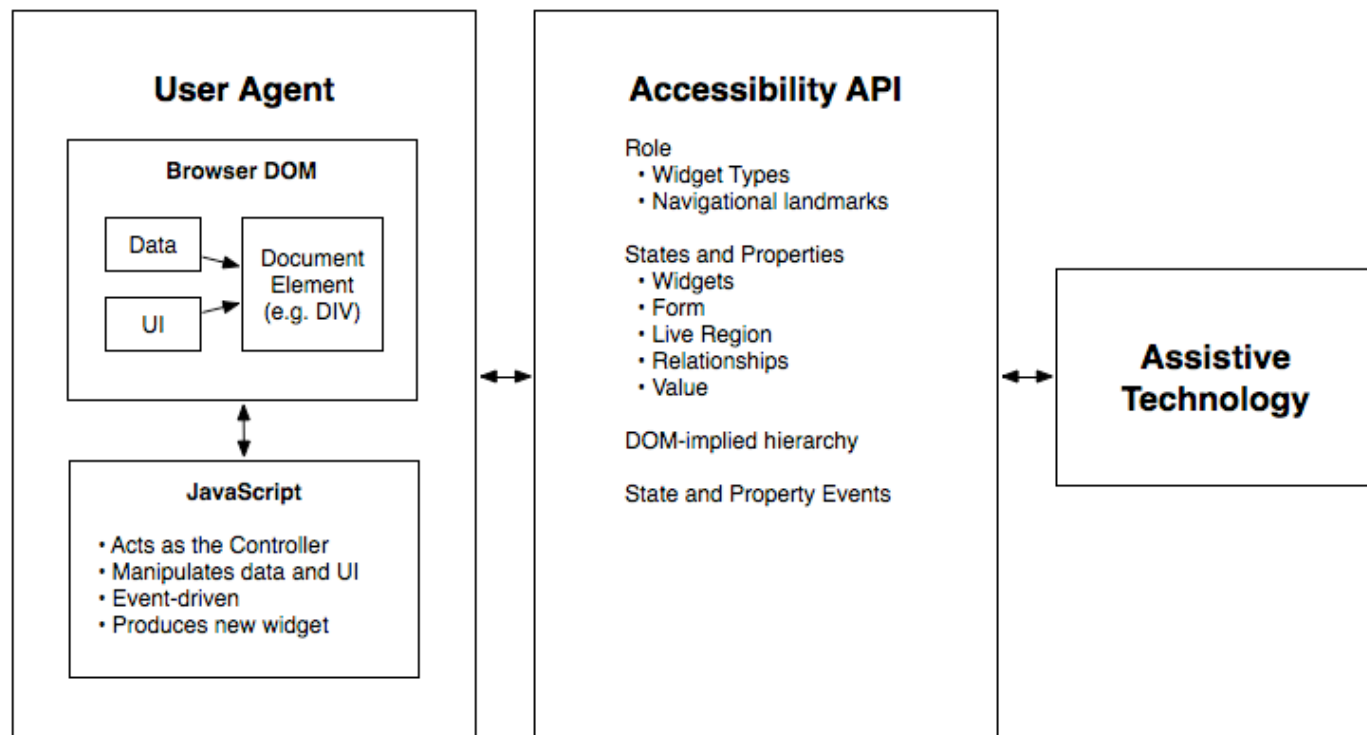


Figure 1: The contract model with accessibility APIs

For more information see [WAI-ARIA Authoring Practices](#) for the use of roles in making interactive content accessible.

Users of alternate input devices need [keyboard accessible](#) content. The new semantics, when combined with the recommended keyboard interactions provided in [WAI-ARIA Authoring Practices](#), will allow alternate input solutions to facilitate command and control via an alternate input solution.

~~WAI-ARIA~~ introduces navigational [landmarks](#) through its Roles Model and the ~~XHTML~~ role landmarks, which can help persons with dexterity and vision impairments by providing for improved keyboard navigation. ~~WAI-ARIA~~ may also be used to assist persons with cognitive learning disabilities. The additional semantics allow authors to restructure and substitute alternative content as needed.

[Assistive technologies](#) need the ability to support alternative inputs by getting and setting the current value of [widget](#) states and properties. Assistive technologies also need to determine what [objects](#) are selected and manage widgets that allow multiple selections, such as list boxes and grids.

Speech-based command and control systems can benefit from ~~WAI-ARIA~~ semantics like the `role` attribute to assist in conveying audio information to the user. For example, upon encountering an element with a role of `menu` with child elements of role `menuitem` each containing text content representing a different flavor, a speech system might state to the user, "Select one of three choices: chocolate, strawberry, or vanilla."

~~WAI-ARIA~~ is intended to be used as a supplement for native language semantics, not a replacement. When the host language provides a feature that provides equivalent accessibility to the ~~WAI-ARIA~~ feature, use the host language feature. ~~WAI-ARIA~~ should only be used in cases where the host language lacks the needed

[role](#), [state](#), and [property](#) indicators. Use a host language feature that is as similar as possible to the [WAI-ARIA](#) feature, then refine the meaning by adding [WAI-ARIA](#). For instance, a multi-selectable grid could be implemented as a table, and then [WAI-ARIA](#) used to clarify that it is an interactive grid, not just a static data table. This allows for the best possible fallback for user agents that do not support [WAI-ARIA](#) and preserves the integrity of the host language semantics.

§ 1.2 Target Audience

This specification defines the basic model for [WAI-ARIA](#), including roles, states, properties, and values. It impacts several audiences:

- [User agents](#) that process content containing [WAI-ARIA](#) features;
- [Assistive technologies](#) that present content in special ways to user with disabilities;
- Authors who create content;
- Authoring tools that help authors create conforming content; and
- Conformance checkers that verify appropriate use of [WAI-ARIA](#).

Each conformance requirement indicates the audience to which it applies.

Although this specification is applicable to the above audiences, it is not specifically targeted to, nor is it intended to be the sole source of information for, any of these audiences. The following documents provide important supporting information:

- [\[WAI-ARIA-PRACTICES-1.2\]](#) addresses authoring recommendations for [HTML](#), and is also of interest to developers of authoring tools and conformance checkers.
- [\[CORE-AAM-1.2\]](#) addresses developers of [user agents](#) and [assistive technologies](#).
- [\[ACCNAME-1.2\]](#) also addresses developers of [user agents](#) and [assistive technologies](#).

§ 1.3 User Agent Support

[WAI-ARIA](#) relies on user agent support for its features in two ways:

- Mainstream [user agents](#) use [WAI-ARIA](#) to alter how host language features are exposed to [accessibility APIs](#) in order to improve accessibility. The mechanism for this is defined in the [Core Accessibility API Mappings](#).

- [Assistive technologies](#) use the enhanced information available in an accessibility API, or uses the WAI-ARIA markup directly via the DOM, to convey semantic and interaction information to the user.

Aside from using WAI-ARIA markup to improve what is exposed to accessibility APIs, user agents behave as they would natively. Assistive technologies react to the extra information in the accessibility API as they already do for the same information on non-web content. User agents that are not assistive technologies, however, need do nothing beyond providing appropriate updates to the accessibility API.

The WAI-ARIA specification neither requires nor forbids user agents from enhancing native presentation and interaction behaviors on the basis of WAI-ARIA markup. Mainstream user agents might expose WAI-ARIA navigational landmarks (for example, as a dialog box or through a keyboard command) with the intention to facilitate navigation for all users. User agents are encouraged to maximize their usefulness to users, including users without disabilities.

WAI-ARIA is intended to provide missing semantics so that the intent of the author may be conveyed to assistive technologies. Generally, authors using WAI-ARIA will provide the appropriate presentation and interaction features. Over time, host languages may add WAI-ARIA equivalents, such as new form controls, that are implemented as standard accessible user interface controls by the user agent. This allows authors to use them instead of custom WAI-ARIA enabled user interface components. In this case the user agent would support the native host language feature. Developers of host languages that implement WAI-ARIA are advised to continue supporting WAI-ARIA semantics when they do not adversely conflict with implicit host language semantics, as WAI-ARIA semantics more clearly reflect the intent of the author if the host language features are inadequate to meet the author's needs.

§ 1.4 Co-Evolution of WAI-ARIA and Host Languages

WAI-ARIA is intended to augment semantics in supporting languages like [HTML] and [SVG2], or to be used as an accessibility enhancement technology in other markup-based languages that do not explicitly include support for ARIA. It clarifies semantics to assistive technologies when authors create new types of objects, via style and script, that are not yet directly supported by the language of the page, because the invention of new types of objects is faster than standardized support for them appears in web languages.

It is not appropriate to create objects with style and script when the host language provides a semantic element for that type of object. While WAI-ARIA can improve the accessibility of these objects, accessibility is best provided by allowing the user agent to handle the object natively. For example, it's better to use an h1 element in HTML than to use the [heading](#) role on a div element.

It is expected that, over time, host languages will evolve to provide semantics for objects that currently can only be declared with WAI-ARIA. This is natural and desirable, as one goal of WAI-ARIA is to help stimulate the emergence of more semantic and accessible markup. When native semantics for a given feature

become available, it is appropriate for authors to use the native feature and stop using `WAI-ARIA` for that feature. Legacy content may continue to use `WAI-ARIA`, however, so the need for user agents to support `WAI-ARIA` remains.

While specific features of `WAI-ARIA` may lose importance over time, the general possibility of `WAI-ARIA` to add semantics to web pages is expected to be a persistent need. Host languages may not implement all the semantics `WAI-ARIA` provides, and various host languages may implement different subsets of the features. New types of objects are continually being developed, and one goal of `WAI-ARIA` is to provide a way to make such objects accessible, because web authoring practices often advance faster than host language standards. In this way, `WAI-ARIA` and host languages both evolve together but at different rates.

Some host languages exist to create semantics for features other than the user interface. For example, `SVG` expresses the semantics behind production of graphical objects, not of user interface components that those objects may represent. Host languages might, by design, not provide native semantics that map to `WAI-ARIA` features. In these cases, `WAI-ARIA` could be adopted as a long-term approach to add semantic information to user interface components.

§ 1.5 Authoring Practices

§ 1.5.1 Authoring Tools

Many of the requirements in the definitions of `WAI-ARIA` [roles](#), [states](#), and [properties](#) can be checked automatically during the development process, similar to other quality control processes used for validating code. To assist authors who are creating custom widgets, authoring tools may compare widget roles, states, and properties to those supported in `WAI-ARIA` as well as those supported in related and cross-referenced roles, states, and properties. Authoring tools may notify authors of errors in widget design patterns, and may also prompt developers for information that cannot be determined from context alone. For example, a scripting library can determine the labels for the tree items in a tree view, but would need to prompt the author to label the entire tree. To help authors visualize a logical accessibility structure, an authoring environment might provide an outline view of a web resource based on the `WAI-ARIA` markup.

In both `HTML` and `SVG`, `tabindex` is an important way browsers support keyboard [focus navigation](#) for implementations of `WAI-ARIA`; authoring and debugging tools may check to make sure `tabindex` values are properly set. For example, error conditions may include cases where more than one `treeitem` in a tree has a `tabindex` value greater than or equal to 0, where `tabindex` is not set on any `treeitem`, or where [aria-activedescendant](#) is not defined when the element with the role `tree` has a `tabindex` value of greater than or equal to 0.

§ 1.5.2 Testing Practices and Tools

The accessibility of interactive content cannot be confirmed by static checks alone. Developers of interactive content should test for device-independent access to [widgets](#) and applications, and should verify accessibility [API](#) access to all content and changes during user interaction.

§ 1.6 Assistive Technologies

Programmatic access to accessibility semantics is essential for assistive technologies. Most assistive technologies interact with user agents, like other applications, through a recognized accessibility [API](#). Perceivable objects in the user interface are exposed to assistive technologies as accessible objects, defined by the accessibility [API](#) interfaces. To do this properly, accessibility information – role, states, properties as well as contextual information – needs to be accurately conveyed to the assistive technologies through the accessibility [API](#). When a state change occurs, the user agent provides the appropriate event notification to the accessibility [API](#). Contextual information, in many host languages like [HTML](#), can be determined from the [DOM](#) itself as it provides a contextual tree hierarchy.

While some assistive technologies interact with these accessibility [APIs](#), others may access the content directly from the [DOM](#). These technologies can restructure, simplify, style, or reflow the content to help a different set of users. Common use cases for these types of adaptations may be the aging population, persons with cognitive impairments, or persons in environments that interfere with use of their tools. For example, the availability of regional navigational landmarks may allow for a mobile device adaptation that shows only portions of the content at any one time based on its semantics. This could reduce the amount of information the user needs to process at any one time. In other situations it may be appropriate to replace a custom user interface control with something that is easier to navigate with a keyboard, or touch screen device.

§ 2. Important Terms

This section is non-normative.

While some terms are defined in place, the following definitions are used throughout this document.

Accessibility [API](#)

Operating systems and other platforms provide a set of interfaces that expose information about [objects](#) and [events](#) to [assistive technologies](#). Assistive technologies use these interfaces to get information about and interact with those [widgets](#). Examples of accessibility [APIs](#) are [Microsoft Active Accessibility](#)

[MSAA], [Microsoft User Interface Automation](#) [UI-AUTOMATION], MSAA with [UIA Express](#) [UIA-EXPRESS], the [Mac OS X Accessibility Protocol](#) [AXAPI], the [Linux/Unix Accessibility Toolkit](#) [ATK] and [Assistive Technology Service Provider Interface](#) [AT-SPI], and [IAccessible2](#) [IAccessible2].

Accessibility Subtree

An [accessible object](#) in the [accessibility tree](#) and its descendants in that tree. It does not include objects which have relationships other than parent-child in that tree. For example, it does not include objects linked via [aria-flowto](#) unless those objects are also descendants in the [accessibility tree](#).

Accessibility Tree

Tree of [accessible objects](#) that represents the structure of the user interface (UI). Each node in the accessibility tree represents an element in the UI as exposed through the [accessibility API](#); for example, a push button, a check box, or container.

Accessible Description

An accessible description provides additional information, related to an interface element, that complements the [accessible name](#). The accessible description might or might not be visually perceivable.

Accessible Name

The accessible name is the name of a user interface element. Each platform [accessibility API](#) provides the accessible name property. The value of the accessible name may be derived from a visible (e.g., the visible text on a button) or invisible (e.g., the text alternative that describes an icon) property of the user interface element. See related [accessible description](#).

A simple use for the accessible name property may be illustrated by an "OK" button. The text "OK" is the accessible name. When the button receives focus, assistive technologies may concatenate the platform's role description with the accessible name. For example, a screen reader may speak "push-button OK" or "OK button". The order of concatenation and specifics of the role description (e.g., "button", "push-button", "clickable button") are determined by platform [accessibility APIs](#) or [assistive technologies](#).

Accessible object

A [node](#) in the [accessibility tree](#) of a platform [accessibility API](#). Accessible objects expose various [states](#), [properties](#), and [events](#) for use by [assistive technologies](#). In the context of markup languages (e.g., HTML and SVG) in general, and of WAI-ARIA in particular, markup [elements](#) and their [attributes](#) are represented as accessible objects.

Activation behavior

The action taken when an [event](#), typically initiated by users through an input device, causes an element to fulfill a defined role. The role may be defined for that element by the host language, or by author-defined variables, or both. The role for any given element may be a generic action, or may be unique to that element. For example, the activation behavior of an HTML or SVG <a> element shall be to cause the user agent to traverse the link specified in the href attribute, with the further optional parameter of

specifying the browsing context for the traversal (such as the current window or tab, a named window, or a new window); the activation behavior of an `HTML <input>` element with the `type` attribute value `submit` shall be to send the values of the form elements to an author-defined `IRI` by the author-defined `HTTP` method.

Assistive Technologies

Hardware and/or software that:

- relies on services provided by a [user agent](#) to retrieve and render Web content
- works with a user agent or web content itself through the use of `APIs`, and
- provides services beyond those offered by the user agent to facilitate user interaction with web content by people with disabilities

This definition may differ from that used in other documents.

Examples of assistive technologies that are important in the context of this document include the following:

- screen magnifiers, which are used to enlarge and improve the visual readability of rendered text and images;
- screen readers, which are most-often used to convey information through synthesized speech or a refreshable Braille display;
- text-to-speech software, which is used to convert text into synthetic speech;
- speech recognition software, which is used to allow spoken control and dictation;
- alternate input technologies (including head pointers, on-screen keyboards, single switches, and sip/puff devices), which are used to simulate the keyboard;
- alternate pointing devices, which are used to simulate mouse pointing and clicking.

Attribute

In this specification, attribute is used as it is in markup languages. Attributes are structural features added to *elements* to provide information about the *states* and *properties* of the *object* represented by the element.

Class

A set of instance *objects* that share similar characteristics.

Deprecated

A deprecated *role*, *state*, or *property* is one which has been outdated by newer constructs or changed circumstances, and which may be removed in future versions of the `WAI-ARIA` specification. [User agents](#) are encouraged to continue to support items identified as deprecated for backward compatibility. For more information, see [Deprecated Requirements](#) in the Conformance section.

Desktop focus event

Event from/to the host operating system via the accessibility [API](#), notifying of a change of input focus.

DOMString

Sequence of 16-bit unsigned integers, typically interpreted as UTF-16 code units. This corresponds to the JavaScript primitive String type.

Element

In this specification, element is used as it is in markup languages. Elements are the structural elements in markup language that contains the data profile for [objects](#).

Event

A programmatic message used to communicate discrete changes in the [state](#) of an [object](#) to other objects in a computational system. User input to a web page is commonly mediated through abstract events that describe the interaction and can provide notice of changes to the state of a document object. In some programming languages, events are more commonly known as notifications.

Expose

Translated to platform-specific [accessibility APIs](#) as defined in the [Core Accessibility API Mappings](#).

Graphical Document

A document containing graphic representations with user-navigable parts. Charts, maps, diagrams, blueprints, and dashboards are examples of graphical documents. A graphical document is composed using any combination of symbols, images, text, and graphic primitives (shapes such as circles, points, lines, paths, rectangles, etc).

Hidden

Indicates that the [element](#) is not visible, [perceivable](#), or interactive to *any* user. An element is considered *hidden* if it or any one of its ancestor elements is not rendered or is explicitly hidden.

Informative

Content provided for information purposes and not required for conformance. Content required for conformance is referred to as [normative](#).

Keyboard Accessible

Accessible to the user using a keyboard or [assistive technologies](#) that mimic keyboard input, such as a sip and puff tube. References in this document relate to [WCAG 2.1 Guideline 2.1: Make all functionality available from a keyboard](#) [WCAG21].

Landmark

A type of region on a page to which the user may want quick access. Content in such a region is different from that of other regions on the page and relevant to a specific user purpose, such as navigating, searching, perusing the primary content, etc.

Live Region

Live regions are perceivable regions of a web page that are typically updated as a result of an external event when user focus may be elsewhere. These regions are not always updated as a result of a user interaction. Examples of live regions include a chat log, stock ticker, or a sport scoring section that updates periodically to reflect game statistics. Since these asynchronous areas are expected to update outside the user's area of focus, assistive technologies such as screen readers have either been unaware of their existence or unable to process them for the user. WAI-ARIA has provided a collection of properties that allow the author to identify these live regions and process them: `aria-live`, `aria-relevant`, `aria-atomic`, and `aria-busy`.

Primary Content Element

An implementing host language's primary content element, such as the `body` element in `HTML`.

Managed State

[Accessibility API state](#) that is controlled by the user agent, such as focus and selection. These are contrasted with "unmanaged states" that are typically controlled by the author. Nevertheless, authors can override some managed states, such as `aria-posinset` and `aria-setsize`. Many managed states have corresponding `CSS` pseudo-classes, such as `:focus`, and pseudo-elements, such as `::selection`, that are also updated by the user agent.

Nemeth Braille

The Nemeth Braille Code for Mathematics is a braille code for encoding mathematical and scientific notation. See [Nemeth Braille on Wikipedia](#).

Node

Basic type of [object](#) in the `DOM` tree or [accessibility tree](#). `DOM` nodes are further specified as [Element](#) or [Text nodes](#), among other types. The nodes of an [accessibility tree](#) are [accessible objects](#).

Normative

Required for conformance. By contrast, content identified as [informative](#) or "non-normative" is not required for conformance.

Object

In the context of user interfaces, an item in the perceptual user experience, represented in markup languages by one or more [elements](#), and rendered by [user agents](#).

In the context of programming, the instantiation of one or more [classes](#) and interfaces which define the general characteristics of similar objects. An object in an [accessibility API](#) may represent one or more `DOM` objects. [Accessibility APIs](#) have defined interfaces that are distinct from `DOM` interfaces.

Ontology

A description of the characteristics of [classes](#) and how they relate to each other.

Operable

Usable by users in ways they can control. References in this document relate to [WCAG 2.1 Principle 2: Content must be operable](#) [WCAG21]. See [Keyboard Accessible](#).

Owned Element

An 'owned element' is any **DOM** descendant of the [element](#), any element specified as a child via [aria-owns](#), or any **DOM** descendant of the owned child.

Owning Element

An 'owning element' is any **DOM** ancestor of the [element](#), or any element with an [aria-owns](#) attribute which references the ID of the element.

Perceivable

Presentable to users in ways they can sense. References in this document relate to [WCAG 2.1 Principle 1: Content must be perceivable](#) [WCAG21].

Property

[Attributes](#) that are essential to the nature of a given [object](#), or that represent a data value associated with the object. A change of a property may significantly impact the meaning or presentation of an object. Certain properties (for example, [aria-multiline](#)) are less likely to change than [states](#), but note that the frequency of change difference is not a rule. A few properties, such as [aria-activedescendant](#), [aria-valuenow](#), and [aria-valuetext](#) are expected to change often. See [clarification of states versus properties](#).

Relationship

A connection between two distinct things. Relationships may be of various types to indicate which [object](#) labels another, controls another, etc.

Role

Main indicator of type. This [semantic](#) association allows tools to present and support interaction with the object in a manner that is consistent with user expectations about other objects of that type.

Root WAI-ARIA node

The primary element containing non-metadata content. In many languages, this is the document element but in **HTML**, it is the <body>.

Semantics

The meaning of something as understood by a human, defined in a way that computers can process a representation of an [object](#), such as [elements](#) and [attributes](#), and reliably represent the object in a way that various humans will achieve a mutually consistent understanding of the object.

State

A state is a dynamic [property](#) expressing characteristics of an [object](#) that may change in response to user action or automated processes. States do not affect the essential nature of the object, but represent data associated with the object or user interaction possibilities. See [clarification of states versus properties](#).

Sub-document

Any document created from a <frame>, <iframe> or similar mechanism. A sub-document may contain a document, an application or any widget such as a calendar pulled in from another server. In

the [accessibility tree](#) there are two [accessible objects](#) for this situation—one represents the `<frame>/<iframe>` element in the parent document, which parents a single [accessible object](#) child representing the spawned document contents.

Target Element

An element specified in a WAI-ARIA relation. For example, in `<div aria-controls="elem1">`, where “elem1” is the ID for the target element.

Taxonomy

A hierarchical definition of how the characteristics of various [classes](#) relate to each other, in which classes inherit the properties of superclasses in the hierarchy. A taxonomy can comprise part of the formal definition of an [ontology](#).

Text node

Type of DOM [node](#) that represents the textual content of an [attribute](#) or an [element](#). A Text node has no child nodes.

Tooltip attribute

Any host language attribute that would result in a user agent generating a tooltip such as in response to a mouse hover in desktop user agents.

Understandable

Presentable to users in ways they can construct an appropriate meaning. References in this document relate to [WCAG 2.1 Principle 3: Information and the operation of user interface must be understandable](#) [WCAG21].

Unicode Braille Patterns

In Unicode, braille is represented in a block called Braille Patterns (U+2800..U+28FF). The block contains all 256 possible patterns of an 8-dot braille cell; this includes the complete 6-dot cell range which is represented by U+2800..U+283F. In all braille systems, the braille pattern dots-0 (U+2800) is used to represent a space or the lack of content; it is also called a blank Braille pattern. See [Braille Patterns on Wikipedia](#).

User Agent

Any software that retrieves, renders and facilitates end user interaction with Web content. This definition may differ from that used in other documents.

Valid IDREF

A reference to a target element in the same document that has a matching ID

Widget

Discrete user interface [object](#) with which the user can interact. Widgets range from simple objects that have one value or operation (e.g., check boxes and menu items), to complex objects that contain many managed sub-objects (e.g., trees and grids).

§ 3. Conformance

The main content of Accessible Rich Internet Applications is [normative](#) and defines requirements that impact conformance claims. Introductory material, appendices, sections marked as "non-normative" and their subsections, diagrams, examples, and notes are [informative](#) (non-normative). Non-normative material provides advisory information to help interpret the guidelines but does not create requirements that impact a conformance claim.

Normative sections provide requirements that [user agents](#) must follow for an implementation to conform to this specification. The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in [Keywords for use in RFCs to indicate requirement levels](#) [RFC2119]. RFC-2119 keywords are formatted in uppercase and contained in an element with `class="rfc2119"`. When the keywords shown above are used, but do not share this format, they do not convey formal information in the RFC 2119 sense, and are merely explanatory, i.e., informative. As much as possible, such usages are avoided in this specification.

Normative sections provide requirements that authors, user agents and assistive technologies **MUST** follow for an implementation to conform to this specification.

Non-normative (informative) sections provide information useful to understanding the specification. Such sections may contain examples of recommended practice, but it is not required to follow such recommendations in order to conform to this specification.

§ 3.1 Non-interference with the Host Language

~~WAI-ARIA~~ processing by the [user agent](#) **MUST NOT** interfere with the normal operation of the built-in features of the host language.

If a ~~CSS~~ selector includes a ~~WAI-ARIA~~ attribute (e.g.,

```
input[aria-invalid="true"]
```

), user agents **MUST** update the visual display of any elements matching (or no longer matching) the selector any time the attribute is added/changed/removed in the ~~DOM~~. The user agent **MAY** alter the mapping of the host language features into an [accessibility API](#), but the user agent **MUST NOT** alter the ~~DOM~~ in order to remap ~~WAI-ARIA~~ markup into host language features.

§ 3.2 All WAI-ARIA in DOM

A conforming [user agent](#) which implements a document object model that does not conform to the [W3C DOM](#) specification **MUST** include the content attribute for role and its [WAI-ARIA role values](#), as well as the [WAI-ARIA States and Properties](#) in the [DOM](#) as specified by the author, even though processing may affect how the elements are exposed to accessibility [APIs](#). Doing so ensures that each role attribute and all [WAI-ARIA](#) states and properties, including their values, are in the document in an unmodified form so other tools, such as assistive technologies, can access them. A conforming [W3C DOM](#) meets this criterion.

§ 3.3 Assistive Technology Notifications Communicated to Web Applications

[Assistive technologies](#), such as speech recognition systems and alternate input devices for users with mobility impairments, require the ability to control a web application in a device-independent way. [WAI-ARIA states](#) and [properties](#) reflect the current state of rich internet application components. The ability for assistive technologies to notify web applications of necessary changes is essential because it allows these alternative input solutions to control an application without being dependent on the standard input device which the user is unable to effectively control directly.

User agents **MUST** provide a method to notify the web application when a change occurs to states or properties in the system accessibility [API](#). Likewise, web application authors **SHOULD** update the web application accordingly when notified of a change request from the user agent or assistive technology.

NOTE

Many state and properties can be changed by assistive technologies through existing accessibility [APIs](#) by responding to a default action event. For example, the [aria-selected](#) state of a [tab](#) in a [tabpanel](#) can be changed by triggering the default action on the element.

§ 3.4 Conformance Checkers

Any application or script verifying document conformance or validity **SHOULD** include a test for all of the [normative](#) author requirements in this specification. If testing for a given requirement, conformance checkers **MUST** issue an error if an author "**MUST**" requirement isn't met, and **MUST** issue a warning if an author "**SHOULD**" requirement isn't met.

§ 3.5 Deprecated Requirements

As the technology evolves, sometimes new ways to meet a use case become available, that work better than a feature that was previously defined. But because of existing implementation of the older feature, that feature cannot be removed from the conformance model without rendering formerly conforming content non-conforming. In this case, the older feature is marked as "deprecated". This indicates that the feature is allowed in the conformance model and expected to be supported by user agents, but it is recommended that authors do not use it for new content. In future versions of the specification, if the feature is no longer widely used, the feature could be removed and no longer expected to be supported by user agents.

§ 4. Using WAI-ARIA

Complex web applications become inaccessible when [assistive technologies](#) cannot determine the [semantics](#) behind portions of a document or when the user is unable to effectively navigate to all parts of it in a usable way (see [WAI-ARIA Authoring Practices](#)). WAI-ARIA divides the semantics into [roles](#) (the type defining a user interface element) and [states](#) and [properties](#) supported by the roles.

Authors need to associate [elements](#) in the document to a WAI-ARIA role and the appropriate states and properties (aria-* [attributes](#)) during its life-cycle, unless the elements already have the appropriate [implicit WAI-ARIA semantics](#) for states and properties. In these instances the equivalent host language states and properties take precedence to avoid a conflict while the role attribute will take precedence over the implicit role of the host language element.

§ 4.1 WAI-ARIA Roles

A WAI-ARIA [role](#) is set on an [element](#) using a [role attribute](#), similar to the role attribute defined in [Role Attribute](#) [ROLE-ATTRIBUTE].

EXAMPLE 1

```
<li role="menuitem">Open file...</li>
```

The definition of each role in the model provides the following information :

- an informative description of the role;
- hierarchical information about related roles (e.g., a [searchbox](#) is a type of [textbox](#));
- context of the role (e.g., a [listitem](#) is contained inside a [list](#));
- references to related concepts in other specifications;
- supported [states](#) and [properties](#) for each role (e.g., a [checkbox](#) supports being checked via [aria-checked](#)).

Attaching a role gives [assistive technologies](#) information about how to handle each element. When [WAI-ARIA](#) roles override host language semantics, there are no changes in the [DOM](#), only in the [accessibility tree](#).

User agents **MUST** use the first token in the sequence of tokens in the role [attribute](#) value that matches the name of any non-abstract [WAI-ARIA role](#). The following steps will correctly identify the applicable [WAI-ARIA](#) role:

1. Use the rules of the host language to detect that an element has a role attribute and to identify the attribute value string for it.
2. Separate the attribute value string for that attribute into a sequence of whitespace-free substrings by separating on whitespace.
3. Compare the substrings to all the names of the non-abstract [WAI-ARIA](#) roles. Case-sensitivity of the comparison inherits from the case-sensitivity of the host language.
4. Use the first such substring in textual order that matches the name of a non-abstract [WAI-ARIA](#) role.

§ 4.2 [WAI-ARIA States and Properties](#)

[WAI-ARIA](#) provides a collection of accessibility [states](#) and [properties](#) which are used to support platform [accessibility APIs](#) on various operating system platforms. [Assistive technologies](#) may access this information through an exposed [user agent DOM](#) or through a mapping to the platform accessibility [API](#). When combined with [roles](#), the user agent can supply the assistive technologies with user interface information to convey to the user at any time. Changes in states or properties will result in a notification to assistive technologies, which could alert the user that a change has occurred.

In the following example, a list item (`html:li`) has been used to create a checkable menu item, and JavaScript [events](#) will capture mouse and keyboard events to toggle the value of [aria-checked](#). A role is used to make the behavior of this simple [widget](#) known to the user agent. [Attributes](#) that change with user actions (such as [aria-checked](#)) are defined in the [states and properties](#) section.

EXAMPLE 2

```
<li role="menuitemcheckbox" aria-checked="true">Sort by Last Modified</li>
```

Some accessibility states, called *managed states*, are controlled by the user agent. Examples of managed state include keyboard focus and selection. Managed states often have corresponding CSS pseudo-classes (such as `:focus` and `::selection`) to define style changes. In contrast, the states in this specification are typically controlled by the author and are called *unmanaged states*. Some states are managed by the user agent, such as [aria-posinset](#) and [aria-setsize](#), but the author can override them if the DOM is incomplete and would cause the user agent calculation to be incorrect. User agents map both managed and unmanaged states to the platform accessibility APIs.

Most modern user agents support [CSS attribute selectors](#) ([CSS3-SELECTORS]), and can allow the author to create UI changes based on WAI-ARIA attribute information, reducing the amount of scripts necessary to achieve equivalent functionality. In the following example, a CSS selector is used to determine whether or not the text is bold and an image of a check mark is shown, based on the value of the [aria-checked](#) attribute.

EXAMPLE 3

```
[aria-checked="true"] { font-weight: bold; }  
[aria-checked="true"]::before { background-image: url(checked.gif); }
```

If CSS is not used to toggle the visual representation of the check mark, the author could include additional markup and scripts to manage an image that represents whether or not the [menuitemcheckbox](#) is checked.

EXAMPLE 4

```
<li role="menuitemcheckbox" aria-checked="true">  
    
  <!-- note: additional scripts required to toggle image source -->  
  Sort by Last Modified  
</li>
```

§ 4.3 Managing Focus and Supporting Keyboard Navigation

When using standard [HTML](#) interactive elements and simple [WAI-ARIA widgets](#), application developers can manipulate the tab order or associate keyboard shortcuts with elements in the document.

[WAI-ARIA](#) includes a number of "managing container" widgets, also known as "composite" widgets. When appropriate, the container is responsible for tracking the last descendant that was active (the default is usually the first item in the container). It is essential that a container maintain a usable and consistent strategy when focus leaves a container and is then later refocused. While there may be exceptions, it is recommended that when a previously focused container is refocused, the active descendant be the same element as the active descendant when the container was last focused. Exceptions include cases where the contents of a container widget have changed, and widgets like a menubar where the user expects to always return to the first item when focus leaves the menu bar. For example, if the second item of a tree group was the active descendant when the user tabbed out of the tree group, then the second item of the tree group remains the active descendant when the tree group gets focus again. The user may also activate the container by clicking on one of the descendants within it. When the container or its active descendant has focus, the user may navigate through the container by pressing additional keys, such as the arrow keys, to change the currently active descendant. Any additional press of the main navigation key (generally the TAB key) will move out of the container to the next widget.

Usable keyboard navigation in a rich internet application is different from the tabbing paradigm among interactive elements, such as links and form controls, in a static document. In rich internet applications, the user tabs to significantly complex [widgets](#), such as a menu or spreadsheet, and uses the arrow keys to navigate within the widget. The changes that [WAI-ARIA](#) introduces to keyboard navigation make this enhanced accessibility possible. In [WAI-ARIA](#), any element can be keyboard focusable. In addition to host language mechanisms such as `tabindex`, [aria-activedescendant](#) provides another mechanism for keyboard operation. Most other aspects of [WAI-ARIA](#) widget development depend on keyboard navigation functioning properly.

When implementing [aria-activedescendant](#) as described below, the user agent keeps the [DOM](#) focus on the container element or on an input element that controls the container element. However, the user agent communicates [desktop focus events](#) and states to the assistive technology as if the element referenced by [aria-activedescendant](#) has focus. User agents are not expected to validate that the active descendant is a descendant of the container element. It is the responsibility of the user agent to ensure that keyboard events are processed at the [element](#) that has [DOM](#) focus. Any keyboard events directed at the active descendant bubble up to the [DOM](#) element with focus for processing.

§ 4.3.1 Information for Authors

If the author removes the element with focus, the author **SHOULD** move focus to a logical element. Similarly, authors **SHOULD** not scroll the element with focus off screen unless the user performed a scrolling action.

Authors **SHOULD** ensure that all interactive [elements](#) are focusable and that all parts of composite widgets are either focusable or have a documented alternative method to achieve their function.

Authors **MUST** manage focus on the following container roles:

- [grid](#)
- [listbox](#)
- [menu](#)
- [menubar](#)
- [radiogroup](#)
- [tree](#)
- [treegrid](#)
- [tablist](#)

User agents that support WAI-ARIA expand the usage of host language mechanisms such as `tabindex`, `focus`, and `blur` to allow them on all [elements](#). Where the host language supports it, authors **MAY** add any element such as a `div`, `span`, or `img` to the default tab order by setting `tabindex="0"`. In addition, any item with `tabindex` equal to a negative integer is focusable via script or a mouse click, but is not part of the default tab order. This is supported in both [\[HTML\]](#) and [\[SVG2\]](#).

Authors **MAY** use [aria-activedescendant](#) to inform [assistive technologies](#) which descendant of a [widget](#) element is treated as having keyboard focus in the user interface if the role of the widget element supports `aria-activedescendant`. This is often a more convenient way of providing keyboard navigation within widgets, such as a [listbox](#), where the widget occupies only one stop in the page Tab sequence and other keys, typically arrow keys, are used to focus elements inside the widget.

Typically, the author will use host language [semantics](#) to put the widget in the Tab sequence (e.g., `tabindex="0"` in [HTML](#).) and `aria-activedescendant` to point to the ID of the currently active descendant. The author, not the user agent, is responsible for styling the currently active descendant to show it has keyboard focus. The author cannot use `:focus` to style the currently active descendant since the actual focus is on the container.

More information on managing focus can be found in the [Developing a Keyboard Interface](#) section of the *WAI-ARIA Authoring Practices*.

§ 4.3.2 Information for User Agents

The user agent **MUST** do the following to implement [aria-activedescendant](#):

1. Implement the host language method for keyboard navigation so that widgets that support [aria-activedescendant](#) may be included in the tab order.
2. For platforms that expose [desktop focus](#) or [accessibility API](#) focus separately from [DOM](#) focus, do not expose the focused state in the accessibility [API](#) for any element when it has [DOM](#) focus and also has [aria-activedescendant](#) which points to a valid [ID reference](#).
3. When the [aria-activedescendant](#) attribute changes on an element that currently has [DOM](#) focus, remove the focused state from the previously focused object and fire an accessibility [API](#) [desktop focus event](#) on the new element referenced by [aria-activedescendant](#). If [aria-activedescendant](#) is cleared or does not point to an element in the current document, fire a desktop focus event for the [object](#) that had the attribute change.
4. Apply the following accessibility [API](#) states to any element with an ID attribute that can be referenced by an element with both an [aria-activedescendant](#) attribute and has [DOM](#) focus. There are two ways an element can be referenced by [aria-activedescendant](#). One way is when it is [owned](#) by an element with [aria-activedescendant](#) and the other is when it is [owned](#) by an element that is controlled by an element with role of [combobox](#), [textbox](#) or [searchbox](#) with an [aria-activedescendant](#) attribute:
 - A. Focusable, if the element also has a [WAI-ARIA role](#). The element needs to be focusable because it could be referenced by the [aria-activedescendant](#) attribute. Native elements that have no [role](#) attribute do not need to be checked; their native semantics determine the focusable state.
 - B. Focused, whenever the element is the target of the [aria-activedescendant](#) attribute and the element with the [aria-activedescendant](#) attribute has [DOM](#) focus.

When an assistive technology uses its platform's accessibility [API](#) to request a change of focus, user agents **MUST** do the following:

1. Remove the platform's focused state from the previously focused object.
2. Set the [DOM](#) focus:
 - A. If the [element](#) can take [DOM](#) focus, the [user agent](#) **MUST** set the [DOM](#) focus to it.
 - B. Otherwise, if the current element has an ID and the ID is referenced by the [aria-activedescendant](#) attribute of an element that is focusable, the user agent **MUST** set [DOM](#)

focus to the element that has the [aria-activedescendant](#) attribute.

NOTE

An element with an ID can be referenced when it is [owned](#) by a container element that has the [aria-activedescendant](#) attribute or by a container element that is controlled by an element that has the [aria-activedescendant](#) attribute (e.g. see [combobox](#)). Otherwise the [aria-activedescendant](#) attribute reference indicates an author error.

NOTE

The inability to set [DOM](#) focus to the containing element indicates an author error.

C. Otherwise, the user agent **MAY** attempt to set [DOM](#) focus to the child element itself.

3. If the current element has an ID and is [owned](#) by either a container element with both an [aria-activedescendant](#) attribute and has [DOM](#) focus, or by a container element that is controlled by an element with both an [aria-activedescendant](#) attribute and has [DOM](#) focus, the user agent **MUST** set the accessibility [API](#) focused state and fire an accessibility [API](#) focus [event](#) on the element identified by the value of [aria-activedescendant](#).

§ 5. The Roles Model

This section defines [WAI-ARIA roles](#) and describes their characteristics and properties.

The roles, their characteristics, the states and properties they support, and specification of how they may be used in markup, shall be considered normative.

In order to reflect the content in the [DOM](#), user agents **SHOULD** map the role attribute to the appropriate value in the implemented accessibility [API](#), and user agents **SHOULD** update the mapping when the role attribute changes.

§ 5.1 Relationships Between Concepts

The Roles Model uses the following relationships to relate [WAI-ARIA](#) roles to each other and to concepts from other specifications, such as [HTML](#).

§ 5.1.1 Superclass Role

The [role](#) that the current subclassed role extends in the Roles Model. This extension causes all the properties and constraints of the superclass role to propagate to the subclass role. Other than well known stable specifications, inheritance may be restricted to items defined inside this specification, so that external items cannot be changed and affect inherited [classes](#).

§ 5.1.2 Subclass Roles

Informative list of [roles](#) for which this role is the superclass. This is provided to facilitate reading of the specification but adds no new information.

§ 5.1.3 Related Concepts

Informative data about a similar or related idea from other specifications. Concepts that are related are not necessarily identical. Related concepts do not inherit properties from each other. Hence if the definition of one concept changes, the properties, behavior, and definition of its related concept is not affected.

For example, a progress bar is like a status indicator. Therefore, the [progressbar widget](#) has a related concept which includes [status](#). However, if the definition of [status](#) is modified, the definition of a [progressbar](#) is not affected.

§ 5.1.4 Base Concept

Informative data about [objects](#) that are considered prototypes for the [role](#). Base concept is similar to type, but without inheritance of limitations and properties. Base concepts are designed as a substitute for inheritance for external concepts. A base concept is like a [related concept](#) except that the base concept is almost identical to the role definition.

For example, the [checkbox](#) defined in this document has similar functionality and anticipated behavior to a `<input [ type="checkbox" ] >` defined in [HTML]. Therefore, a [checkbox](#) has an [HTML] checkbox as a baseConcept. However, if the original [HTML] checkbox baseConcept definition is modified, the definition of a [checkbox](#) in this document will not be affected, because there is no actual inheritance of the respective type.

§ 5.2 Characteristics of Roles

Roles are defined and described by their characteristics. Characteristics define the structural function of a role, such as what a role is, concepts behind it, and what instances the role can or must contain. In the case of [widgets](#) this also includes how it interacts with the [user agent](#) based on mapping to [HTML](#) forms. States and properties from [WAI-ARIA](#) that are supported by the role are also indicated.

Roles define the following characteristics.

§ 5.2.1 Abstract Roles

Values

Boolean

Abstract [roles](#) are the foundation upon which all other [WAI-ARIA](#) roles are built. Content authors **MUST NOT** use abstract roles because they are not implemented in the [API](#) binding. User agents **MUST NOT** map abstract roles to the standard role mechanism of the accessibility [API](#). Abstract roles are provided to help with the following:

1. Organize the Roles Model and provide roles with a meaning in the context of known concepts.
2. Streamline the addition of roles that include necessary features.

§ 5.2.2 Required States and Properties

[States](#) and [properties](#) specifically required for the [role](#) and subclass roles. Content authors **MUST** provide a non-empty value for required states and properties. Content authors **MUST NOT** use the value `undefined` for required states and properties, unless `undefined` is an explicitly-supported value of that state or property.

When an [object](#) inherits from multiple ancestors and one ancestor indicates that property is supported while another ancestor indicates that it is required, the property is required in the inheriting object.

NOTE

A host language attribute with the appropriate [implicit WAI-ARIA semantic](#) fulfills this requirement.

§ 5.2.3 Supported States and Properties

[States](#) and [properties](#) specifically applicable to the [role](#) and child roles. Content authors **MAY** provide values for supported states and properties, but need not in cases where default values are sufficient. [User agents](#) **MUST** map all supported states and properties for the role to an accessibility API. If the state or property is undefined and it has a default value for the role, [user agents](#) **SHOULD** expose the default value.

NOTE

A host language attribute with the appropriate [implicit WAI-ARIA semantic](#) fulfills this requirement.

§ 5.2.4 Inherited States and Properties

Informative list of properties that are inherited by a [role](#) from superclass roles. [States](#) and [properties](#) are inherited from superclass roles in the Roles Model, not from ancestor [elements](#) in the DOM tree. These properties are not explicitly defined on the role, as the inheritance of properties is automatic. This information is provided to facilitate reading of the specification. The set of supported states and properties combined with inherited states and properties forms the full set of states and properties supported by the role.

§ 5.2.5 Prohibited States and Properties

List of states and properties that are prohibited on a [role](#). Authors **MUST NOT** specify a prohibited state or property.

NOTE

A host language attribute with the appropriate [implicit WAI-ARIA semantic](#) would also prohibit a state or property in this section.

§ 5.2.6 Required Owned Elements

Any [element](#) that will be [owned](#) by the element with this [role](#). For example, an element with the role [list](#) will own at least one element with the role [listitem](#).

When multiple roles are specified as *required owned elements* for a role, at least one instance of one required [owned](#) element is expected. This specification does *not* require an instance of each of the listed owned roles. For example, a menu should have at least one instance of a [menuitem](#), [menuitemcheckbox](#), or [menuitemradio](#). The menu role does not require one instance of each.

There may be times that required [owned](#) elements are missing, for example, while editing or while loading a data set. When a widget is missing *required owned elements* due to script execution or loading, authors **MUST** mark a containing element with [aria-busy](#) equal to `true`. For example, until a page is fully initialized and complete, an author could mark the document element as busy.

NOTE

A role that has 'required owned elements' does not imply the reverse relationship. While processing of a role may be incomplete without elements of given roles present as descendants, elements with roles in this list do not always have to be found within elements of the given role. See [required context role](#) for requirements about the context where elements of a given role will be contained.

NOTE

An element with a [subclass role](#) of the 'required owned element' does not fulfill this requirement. For example, the [listbox](#) role requires ownership of an element using the [option](#) or [group](#) role. Although the [group](#) role is the superclass of [row](#), adding an [owned](#) element with a role of [row](#) will not fulfill the requirement that [listbox](#) owns an [option](#) or a [group](#).

NOTE

An element with the appropriate [implicit WAI-ARIA semantic](#) fulfills this requirement.

§ 5.2.7 Required Context Role

The required context role defines the owning container where this [role](#) is allowed. If a role has a required context, authors **MUST** ensure that an element with the role is contained inside (or [owned](#) by) an element

with the required context role. For example, an element with role `listitem` is only meaningful when contained inside (or [owned](#) by) an element with role `list`.

NOTE

A role that has 'required context role' does not imply the reverse relationship. While an element with the given role needs to appear within an element of the listed role(s) in order to be meaningful, elements of the listed roles do not always need descendant elements of the given role in order to be meaningful. See [required owned elements](#) for requirements about elements that require presence of a given descendant to be processed properly.

NOTE

An element with the appropriate [implicit WAI-ARIA semantic](#) fulfills this requirement.

§ 5.2.8 Accessible Name Calculation

Values

One of the following values:

1. author: name comes from values provided by the author in explicit markup features such as the [aria-label](#) attribute, the [aria-labelledby](#) attribute, or the host language labeling mechanism, such as the `alt` or `title` attributes in HTML, with HTML `title` attribute having the lowest precedence for specifying a text alternative.
2. contents: name comes from the text value of the [element node](#). Although this may be allowed in addition to "author" in some [roles](#), this is used in content only if higher priority "author" features are not provided. Priority is defined by the [accessible name and description computation](#) algorithm [ACCNAME-1.2].
3. prohibited: the element does not support name from author. Authors **MUST NOT** use the [aria-label](#) or [aria-labelledby](#) attributes to name the element.

§ 5.2.8.1 Name Computation

[Name Computation](#) is defined in the Accessible Name and Description specification.

§ 5.2.8.2 *Description Computation*

[Description Computation](#) is defined in the Accessible Name and Description specification.

§ 5.2.8.3 *Accessible Name and Description Computation*

[Accessible Name and Description Computation](#) is defined in the Accessible Name and Description specification.

§ 5.2.8.4 *Roles Supporting Name from Author*

- [alert](#)
- [alertdialog](#) (name required)
- [application](#) (name required)
- [article](#)
- [banner](#)
- [blockquote](#)
- [button](#) (name required)
- [cell](#)
- [checkbox](#) (name required)
- [columnheader](#) (name required)
- [combobox](#) (name required)
- [command](#)
- [complementary](#)
- [composite](#)
- [contentinfo](#)
- [definition](#)
- [dialog](#) (name required)
- [directory](#)

- [document](#)
- [feed](#)
- [figure](#)
- [form](#)
- [grid](#) (name required)
- [gridcell](#)
- [group](#)
- [heading](#) (name required)
- [img](#) (name required)
- [input](#)
- [landmark](#)
- [link](#) (name required)
- [list](#)
- [listbox](#) (name required)
- [listitem](#)
- [log](#)
- [main](#)
- [marquee](#) (name required)
- [math](#)
- [meter](#) (name required)
- [menu](#)
- [menubar](#)
- [menuitem](#) (name required)
- [menuitemcheckbox](#) (name required)
- [menuitemradio](#) (name required)
- [navigation](#)
- [note](#)
- [option](#) (name required)
- [progressbar](#) (name required)
- [radio](#) (name required)

- [radiogroup](#) (name required)
- [range](#)
- [region](#) (name required)
- [row](#)
- [rowgroup](#)
- [rowheader](#) (name required)
- [scrollbar](#)
- [search](#)
- [searchbox](#) (name required)
- [sectionhead](#)
- [select](#)
- [separator](#)
- [slider](#) (name required)
- [spinbutton](#) (name required)
- [status](#)
- [switch](#) (name required)
- [tab](#)
- [table](#) (name required)
- [tablist](#)
- [tabpanel](#) (name required)
- [term](#)
- [textbox](#) (name required)
- [time](#)
- [timer](#)
- [toolbar](#)
- [tooltip](#) (name required)
- [tree](#) (name required)
- [treegrid](#) (name required)
- [treeitem](#) (name required)
- [window](#)

§ 5.2.8.5 Roles Supporting Name from Content

- [button](#) (name required)
- [cell](#)
- [checkbox](#) (name required)
- [columnheader](#) (name required)
- [gridcell](#)
- [heading](#) (name required)
- [link](#) (name required)
- [menuitem](#) (name required)
- [menuitemcheckbox](#) (name required)
- [menuitemradio](#) (name required)
- [option](#) (name required)
- [radio](#) (name required)
- [row](#)
- [rowheader](#) (name required)
- [sectionhead](#)
- [switch](#) (name required)
- [tab](#)
- [tooltip](#) (name required)
- [treeitem](#) (name required)

§ 5.2.8.6 Roles which cannot be named (Name prohibited)

- [caption](#)
- [code](#)
- [deletion](#)
- [emphasis](#)
- [generic](#)

- [insertion](#)
- [paragraph](#)
- [presentation](#)
- [strong](#)
- [subscript](#)
- [superscript](#)

§ 5.2.9 Presentational Children

Values

Boolean (true | false)

The DOM descendants are presentational. [User agents](#) **SHOULD NOT** expose descendants of this [element](#) through the platform [accessibility API](#). If [user agents](#) do not hide the descendant nodes, some information may be read twice.

§ 5.2.10 Implicit Value for Role

Many states and properties have default values. Occasionally, the default value when used on a given role should be different from the usual default. Roles that require a state or property to have a non-standard default value indicate this in the "Implicit Value for Role". This is expressed in the form "Default for state or property name is new default value". Roles that define this have the new default value for the state or property if the author does not provide an explicit value.

§ 5.3 Categorization of Roles

To support the current user scenario, this specification categorizes [roles](#) that define user interface [widgets](#) (sliders, tree controls, etc.) and those that define page structure (sections, navigation, etc.). Note that some assistive technologies provide special modes of interaction for regions marked with role application or document.

A visual description of the relationships among roles is available in the [ARIA 1.2 Class Diagram](#).

Roles are categorized as follows:

1. [Abstract Roles](#)
2. [Widget Roles](#)
3. [Document Structure Roles](#)
4. [Landmark Roles](#)
5. [Live Region Roles](#)
6. [Window Roles](#)

§ 5.3.1 Abstract Roles

The following [roles](#) are used to support the WAI-ARIA Roles Model for the purpose of defining general role concepts.

Abstract roles are used for the ontology. Authors ***MUST NOT*** use abstract roles in content.

- [command](#)
- [composite](#)
- [input](#)
- [landmark](#)
- [range](#)
- [roletype](#)
- [section](#)
- [sectionhead](#)
- [select](#)
- [structure](#)
- [widget](#)
- [window](#)

§ 5.3.2 Widget Roles

The following roles act as standalone user interface widgets or as part of larger, composite widgets.

- [button](#)

- [checkbox](#)
- [gridcell](#)
- [link](#)
- [menuitem](#)
- [menuitemcheckbox](#)
- [menuitemradio](#)
- [option](#)
- [progressbar](#)
- [radio](#)
- [scrollbar](#)
- [searchbox](#)
- [separator](#) (when focusable)
- [slider](#)
- [spinbutton](#)
- [switch](#)
- [tab](#)
- [tabpanel](#)
- [textbox](#)
- [treeitem](#)

The following roles act as composite user interface widgets. These roles typically act as containers that manage other, contained widgets.

- [combobox](#)
- [grid](#)
- [listbox](#)
- [menu](#)
- [menubar](#)
- [radiogroup](#)
- [tablist](#)
- [tree](#)
- [treegrid](#)

§ 5.3.3 Document Structure Roles

The following [roles](#) describe structures that organize content in a page. Document structures are not usually interactive.

- [application](#)
- [article](#)
- [blockquote](#)
- [caption](#)
- [cell](#)
- [columnheader](#)
- [definition](#)
- [deletion](#)
- [directory](#)
- [document](#)
- [emphasis](#)
- [feed](#)
- [figure](#)
- [generic](#)
- [group](#)
- [heading](#)
- [img](#)
- [insertion](#)
- [list](#)
- [listitem](#)
- [math](#)
- [meter](#)
- [none](#)
- [note](#)
- [paragraph](#)

- [presentation](#)
- [row](#)
- [rowgroup](#)
- [rowheader](#)
- [separator](#) (when not focusable)
- [strong](#)
- [subscript](#)
- [superscript](#)
- [table](#)
- [term](#)
- [time](#)
- [toolbar](#)
- [tooltip](#)

§ 5.3.4 Landmark Roles

The following [roles](#) are regions of the page intended as navigational [landmarks](#). All of these roles inherit from the landmark base type and all are imported from the [Role Attribute](#) [ROLE-ATTRIBUTE]. The roles are included here in order to make them clearly part of the WAI-ARIA Roles Model.

- [banner](#)
- [complementary](#)
- [contentinfo](#)
- [form](#)
- [main](#)
- [navigation](#)
- [region](#)
- [search](#)

§ 5.3.5 Live Region Roles

The following [roles](#) are [live regions](#) and may be modified by [live region attributes](#).

- [alert](#)
- [log](#)
- [marquee](#)
- [status](#)
- [timer](#)

§ 5.3.6 Window Roles

The following [roles](#) act as windows within the browser or application.

- [alertdialog](#)
- [dialog](#)

§ 5.4 Definition of Roles

Below is an alphabetical list of [WAI-ARIA roles](#) to be used by authors.

Abstract roles are used for the ontology. Authors **MUST NOT** use abstract roles in content.

[alert](#)

A type of [live region](#) with important, and usually time-sensitive, information. See related [alertdialog](#) and [status](#).

[alertdialog](#)

A type of dialog that contains an alert message, where initial focus goes to an [element](#) within the dialog. See related [alert](#) and [dialog](#).

[application](#)

A [structure](#) containing one or more focusable elements requiring user input, such as keyboard or gesture events, that do not follow a standard interaction pattern supported by a [widget](#) role.

[article](#)

A section of a page that consists of a composition that forms an independent part of a document, page, or site.

[banner](#)

A [landmark](#) that contains mostly site-oriented content, rather than page-specific content.

[blockquote](#)

A section of content that is quoted from another source.

[button](#)

An input that allows for user-triggered actions when clicked or pressed. See related [link](#).

[caption](#)

Visible content that names, and may also describe, a [figure](#), [table](#), [grid](#), or [treegrid](#).

[cell](#)

A cell in a tabular container. See related [gridcell](#).

[checkbox](#)

A checkable input that has three possible values: true, false, or mixed.

[code](#)

A section whose content represents a fragment of computer code.

[columnheader](#)

A cell containing header information for a column.

[combobox](#)

An [input](#) that controls another element, such as a [listbox](#) or [grid](#), that can dynamically pop up to help the user set the value of the [input](#).

[command](#)

A form of widget that performs an action but does not receive input data.

[complementary](#)

A [landmark](#) that is designed to be complementary to the main content at a similar level in the DOM hierarchy, but remaining meaningful when separated from the main content.

[composite](#)

A [widget](#) that may contain navigable descendants or [owned](#) children.

[contentinfo](#)

A [landmark](#) that contains information about the parent document.

[definition](#)

A definition of a term or concept. See related [term](#).

[deletion](#)

A deletion contains content that is marked as removed or content that is being suggested for removal. See related [insertion](#).

[dialog](#)

A dialog is a descendant window of the primary window of a web application. For [HTML](#) pages, the primary application window is the entire web document, i.e., the body element.

[directory](#)

[Deprecated in ARIA 1.2] A list of references to members of a group, such as a static table of contents.

[document](#)

An [element](#) containing content that [assistive technology](#) users may want to browse in a reading mode.

[emphasis](#)

One or more emphasized characters. See related [strong](#).

[feed](#)

A scrollable [list](#) of [articles](#) where scrolling may cause [articles](#) to be added to or removed from either end of the list.

[figure](#)

A perceivable [section](#) of content that typically contains a [graphical document](#), images, code snippets, or example text. The parts of a figure **MAY** be user-navigable.

[form](#)

A [landmark](#) region that contains a collection of items and objects that, as a whole, combine to create a form. See related [search](#).

[generic](#)

A nameless container [element](#) that has no semantic meaning on its own.

[grid](#)

A composite [widget](#) containing a collection of one or more rows with one or more cells where some or all cells in the grid are focusable by using methods of two-dimensional navigation, such as directional arrow keys.

[gridcell](#)

A [cell](#) in a [grid](#) or [treegrid](#).

[group](#)

A set of user interface [objects](#) that is not intended to be included in a page summary or table of contents by [assistive technologies](#).

[heading](#)

A heading for a section of the page.

[img](#)

A container for a collection of [elements](#) that form an image.

[input](#)

A generic type of [widget](#) that allows user input.

[insertion](#)

An insertion contains content that is marked as added or content that is being suggested for addition. See related [deletion](#).

[landmark](#)

A perceivable [section](#) containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.

[link](#)

An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource. See related [button](#).

[list](#)

A [section](#) containing [listitem](#) elements. See related [listbox](#).

[listbox](#)

A [widget](#) that allows the user to select one or more items from a list of choices. See related [combobox](#) and [list](#).

[listitem](#)

A single item in a list or directory.

[log](#)

A type of [live region](#) where new information is added in meaningful order and old information may disappear. See related [marquee](#).

[main](#)

A [landmark](#) containing the main content of a document.

[marquee](#)

A type of [live region](#) where non-essential information changes frequently. See related [log](#).

[math](#)

Content that represents a mathematical expression.

[meter](#)

An [element](#) that represents a scalar measurement within a known range, or a fractional value. See related [progressbar](#).

[menu](#)

A type of [widget](#) that offers a list of choices to the user.

[menubar](#)

A presentation of [menu](#) that usually remains visible and is usually presented horizontally.

[menuitem](#)

An option in a set of choices contained by a [menu](#) or [menubar](#).

[menuitemcheckbox](#)

A [menuitem](#) with a checkable state whose possible values are true, false, or mixed.

[menuitemradio](#)

A checkable [menuitem](#) in a set of elements with the same role, only one of which can be checked at a time.

[navigation](#)

A [landmark](#) containing a collection of navigational [elements](#) (usually links) for navigating the document or related documents.

[none](#)

An [element](#) whose implicit native role semantics will not be mapped to the [accessibility API](#). See synonym [presentation](#).

[note](#)

A section whose content is parenthetical or ancillary to the main content of the resource.

[option](#)

A selectable item in a [listbox](#).

[paragraph](#)

A paragraph of content.

[presentation](#)

An [element](#) whose implicit native role semantics will not be mapped to the [accessibility API](#). See synonym [none](#).

[progressbar](#)

An [element](#) that displays the progress status for tasks that take a long time.

[radio](#)

A checkable input in a group of elements with the same role, only one of which can be checked at a time.

[radiogroup](#)

A group of [radio](#) buttons.

[range](#)

An element representing a range of values.

[region](#)

A [landmark](#) containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.

[roletype](#)

The base [role](#) from which all other roles inherit.

[row](#)

A row of cells in a tabular container.

[rowgroup](#)

A structure containing one or more row elements in a tabular container.

[rowheader](#)

A cell containing header information for a row.

[scrollbar](#)

A graphical object that controls the scrolling of content within a viewing area, regardless of whether the content is fully displayed within the viewing area.

[search](#)

A [landmark](#) region that contains a collection of items and objects that, as a whole, combine to create a search facility. See related [form](#) and [searchbox](#).

[searchbox](#)

A type of textbox intended for specifying search criteria. See related [textbox](#) and [search](#).

[section](#)

A renderable structural containment unit in a document or application.

[sectionhead](#)

A structure that labels or summarizes the topic of its related section.

[select](#)

A form widget that allows the user to make selections from a set of choices.

[separator](#)

A divider that separates and distinguishes sections of content or groups of menuitems.

[slider](#)

An input where the user selects a value from within a given range.

[spinbutton](#)

A form of [range](#) that expects the user to select from among discrete choices.

[status](#)

A type of [live region](#) whose content is advisory information for the user but is not important enough to justify an [alert](#), often but not necessarily presented as a status bar.

[strong](#)

Content that is important, serious, or urgent. See related [emphasis](#).

[structure](#)

A document structural [element](#).

[subscript](#)

One or more subscripted characters. See related [superscript](#).

[superscript](#)

One or more superscripted characters. See related [superscript](#).

[switch](#)

A type of checkbox that represents on/off values, as opposed to checked/unchecked values. See related [checkbox](#).

[tab](#)

A grouping label providing a mechanism for selecting the tab content that is to be rendered to the user.

[table](#)

A [section](#) containing data arranged in rows and columns. See related [grid](#).

[tablist](#)

A list of [tab elements](#), which are references to [tabpanel](#) elements.

[tabpanel](#)

A container for the resources associated with a [tab](#), where each [tab](#) is contained in a [tablist](#).

[term](#)

A word or phrase with a corresponding definition. See related [definition](#).

[textbox](#)

A type of input that allows free-form text as its value.

[time](#)

An element that represents a specific point in time.

[timer](#)

A type of [live region](#) containing a numerical counter which indicates an amount of elapsed time from a start point, or the time remaining until an end point.

[toolbar](#)

A collection of commonly used function buttons or controls represented in compact visual form.

[tooltip](#)

A contextual popup that displays a description for an element.

[tree](#)

A [widget](#) that allows the user to select one or more items from a hierarchically organized collection.

[treegrid](#)

A [grid](#) whose rows can be expanded and collapsed in the same manner as for a [tree](#).

[treeitem](#)

An option item of a [tree](#). This is an [element](#) within a tree that may be expanded or collapsed if it contains a sub-level group of tree item elements.

[widget](#)

An interactive component of a graphical user interface (GUI).

[window](#)

A browser or application window.

alert role

A type of [live region](#) with important, and usually time-sensitive, information. See related [alertdialog](#) and [status](#).

Alerts are used to convey messages that may be immediately important to users. In the case of audio warnings, alerts provide an accessible alternative for hearing-impaired users. The [alert role](#) is applied to the element containing the alert message. An alert is a specialized form of the [status](#) role, which is processed as an atomic [live region](#).

Alerts are assertive live regions, which means they cause immediate notification for assistive technology users. If the operating system allows, the [user agent](#) **SHOULD** fire a system alert [event](#) through the accessibility API when the ~~WAI-ARIA~~ alert is created.

Neither authors nor user agents are required to set or manage focus to an alert in order for it to be processed. Since alerts are not required to receive focus, authors **SHOULD NOT** require users to close an alert. If an author desires focus to move to a message when it is conveyed, the author **SHOULD** use [alertdialog](#) instead of alert.

Elements with the role alert have an implicit [aria-live](#) value of assertive, and an implicit [aria-atomic](#) value of true.

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	alertdialog
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts

Characteristic	Value
	aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-live is assertive. Default for aria-atomic is true.

alertdialog role

A type of dialog that contains an alert message, where initial focus goes to an [element](#) within the dialog. See related [alert](#) and [dialog](#).

Alert dialogs are used to convey messages to alert the user. The `alertdialog` [role](#) goes on the node containing both the alert message and the rest of the dialog. Content authors **SHOULD** make alert dialogs modal by ensuring that, while the `alertdialog` is shown, keyboard and mouse interactions only operate within the dialog. See [aria-modal](#).

Unlike [alert](#), `alertdialog` can receive a response from the user. For example, to confirm that the user understands the alert being generated. When the alert dialog is displayed, authors **SHOULD** set focus to an active element within the alert dialog, such as a form control or confirmation button. The [user agent](#) **SHOULD** fire a system alert [event](#) through the accessibility [API](#) when the alert is created, provided one is specified by the intended [accessibility API](#).

Authors **SHOULD** use [aria-describedby](#) on an `alertdialog` to reference the alert message element in the dialog. If they do not, an [assistive technology](#) can resort to its internal recovery mechanism to determine the contents of the alert message.

Characteristics:

Characteristic	Value
Superclass Role:	alert dialog
Inherited States and Properties:	aria-atomic

Characteristic	Value
	aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-modal aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

application role

A [structure](#) containing one or more focusable elements requiring user input, such as keyboard or gesture events, that do not follow a standard interaction pattern supported by a [widget](#) role.

Some [user agents](#) and [assistive technologies](#) have a browse mode where standard input events, such as up and down arrow key events, are intercepted and used to control a reading cursor. This browse mode behavior prevents elements that do not have a [widget](#) role from receiving and using such keyboard and gesture events to provide interactive functionality.

When there is a need to create an element with an interaction model that is not supported by any of the [WAI-ARIA widget](#) roles, authors **MAY** give that element role application. And, when a user navigates into an element with role application, [assistive technologies](#) that intercept standard input events **SHOULD** switch to a mode that passes most or all standard input events through to the web application.

For example, a presentation slide editor uses arrow keys to change the positions of textbox and image elements on the slide. There are not any [WAI-ARIA widget](#) roles that correspond to such an interaction model so an author could give the slide container role application, an [aria-roledescription](#) of "Slide Editor", and use [aria-describedby](#) to provide instructions.

Because only the focusable elements contained in an application element are accessible to users of some assistive technologies, authors **MUST** use one of the following techniques to ensure all non-decorative static text or image content inside an application is accessible:

- 1. Associate the content with a focusable element using [aria-labelledby](#) or [aria-describedby](#).
- 2. Place the content in a focusable element that has role [document](#) or [article](#).
- 3. Manage focus of [owned](#) elements as described in [Managing Focus](#), updating the value of [aria-activedescendant](#) to reference the [element](#) containing the focused content.

Characteristics:

Characteristic	Value
Superclass Role:	structure
Supported States and Properties:	aria-activedescendant aria-disabled aria-errormessage aria-expanded aria-haspopup aria-invalid

Characteristic	Value
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-flowto aria-grabbed (state) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

article role

A section of a page that consists of a composition that forms an independent part of a document, page, or site.

An article is not a navigational [landmark](#), but may be nested to form a discussion where assistive technologies could pay attention to article nesting to assist the user in following the discussion. An article could be a forum post, a magazine or newspaper article, a web log entry, a user-submitted comment, or any other independent item of content. It is *independent* in that its contents could stand alone, for example in syndication. However, the [element](#) is still associated with its ancestors; for instance, contact information that applies to a parent body element still covers the article as well. When nesting articles, the child articles represent content that is related to the content of the parent article. For instance, a web log entry on a site that

accepts user-submitted comments could represent the comments as articles nested within the article for the web log entry. Author, heading, date, or other information associated with an article does not apply to nested articles.

When the user navigates to an element assigned the role of `article`, [assistive technologies](#) that typically intercept standard keyboard events **SHOULD** switch to document browsing mode, as opposed to passing keyboard events through to the web application. Assistive technologies **MAY** provide a feature allowing the user to navigate the hierarchy of any nested `article` elements.

When an `article` is in the context of a [feed](#), the author **MAY** specify values for [aria-posinset](#) and [aria-setsize](#).

Characteristics:

Characteristic	Value
Superclass Role:	document
Related Concepts:	<article> in [HTML]
Supported States and Properties:	aria-posinset aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state)

Characteristic	Value
	aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

banner role

A [landmark](#) that contains mostly site-oriented content, rather than page-specific content.

Site-oriented content typically includes things such as the logo or identity of the site sponsor, and a site-specific search tool. A banner usually appears at the top of the page and typically spans the full width.

User agents **SHOULD** treat elements with the role of banner as navigational [landmarks](#).

Within any [document](#) or [application](#), the author **SHOULD** mark no more than one [element](#) with the banner [role](#).

NOTE

Because document and application elements can be nested in the DOM, they may have multiple banner elements as DOM descendants, assuming each of those is associated with different document nodes, either by a DOM nesting (e.g., [document](#) within [document](#)) or by use of the [aria-owns attribute](#).

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Inherited States and Properties:	aria-atomic aria-busy (state)

Characteristic	Value
	aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

blockquote role

A section of content that is quoted from another source.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	section
Related Concepts:	<blockquote> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

button role

An input that allows for user-triggered actions when clicked or pressed. See related [link](#).

Buttons are mostly used for discrete actions. Standardizing the appearance of buttons enhances the user's recognition of the [widgets](#) as buttons and allows for a more compact display in toolbars.

Buttons support the optional [attribute](#) [aria-pressed](#). Buttons with a non-empty [aria-pressed](#) attribute are toggle buttons. When [aria-pressed](#) is true the button is in a "pressed" [state](#), when [aria-pressed](#) is false it is not pressed. If the attribute is not present, the button is a simple command button.

Characteristics:

Characteristic	Value
Superclass Role:	command
Base Concept:	<button> in [HTML]
Related Concepts:	link
Supported States and Properties:	aria-disabled aria-haspopup aria-expanded aria-pressed
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

caption role

Visible content that names, and may also describe, a [figure](#), [table](#), [grid](#), or [treegrid](#).

When using caption authors **SHOULD** ensure:

- The caption is a direct child of a [figure](#), [table](#), [grid](#), or [treegrid](#).
- The caption is the first child of a [table](#), [grid](#), or [treegrid](#).
- The caption is the first or last child of a [figure](#).

Authors **SHOULD** set [aria-labelledby](#) on the parent figure, table, grid, or [treegrid](#) to reference the element with role caption. However, if a caption contains content that serves as both a name and description for its parent, authors **MAY** instead set [aria-labelledby](#) to reference an element within the caption that contains a concise name, and set [aria-describedby](#) to reference an element within the caption that contains the descriptive content.

EXAMPLE 5

```
<div role="table" aria-labelledby="name" aria-describedby="desc">
  <div role="caption">
    <div id="name">Contest Entrants</div>
    <div id="desc">
      This table shows the total number of entrants (500) the
      contest accepted over the past four weeks.
    </div>
  </div>
</div>
<!-- ... -->
```

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<caption> in [HTML] <figcaption> in [HTML]
Required Context Role:	figure grid table treegrid
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state)

Characteristic	Value
	aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

cell role

A cell in a tabular container. See related [gridcell](#).

Authors **MUST** ensure [elements](#) with [role](#) cell are contained in, or owned by, an element with the [role](#) row.

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	columnheader gridcell rowheader
Base Concept:	<td> in [HTML]
Required Context Role:	row
Supported States and Properties:	aria-colindex aria-colspan aria-rowindex

Characteristic	Value
	aria-rowspan
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author

checkbox role

A checkable input that has three possible values: true, false, or mixed.

The [aria-checked attribute](#) of a checkbox indicates whether the input is checked (true), unchecked (false), or represents a group of [elements](#) that have a mixture of checked and unchecked values (mixed). Many checkboxes do not use the mixed value, and thus are effectively boolean checkboxes.

Characteristics:

Characteristic	Value
Superclass Role:	input
Subclass Roles:	switch
Related Concepts:	<input [type="checkbox"]> in [HTML] option
Required States and Properties:	aria-checked
Supported States and Properties:	aria-errormessage aria-expanded aria-invalid aria-readonly aria-required
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

code role

A section whose content represents a fragment of computer code.

The primary purpose of the code role is to inform assistive technologies that the content is computer code and thus may require special presentation, in particular with respect to synthesized speech. More specifically, screen readers and other tools which provide text-to-speech presentation of content ***SHOULD*** prefer full punctuation verbosity to ensure common symbols (e.g. "-") are spoken.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<code> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

columnheader role

A cell containing header information for a column.

columnheader can be used as a column header in a table or grid. It could also be used in a pie chart to show a similar [relationship](#) in the data.

The columnheader establishes a relationship between it and all cells in the corresponding column. It is the structural equivalent to an [HTML th element](#) with a column scope.

Authors **MUST** ensure [elements](#) with [role](#) columnheader are contained in, or [owned](#) by, an element with the role [row](#).

Applying the [aria-selected](#) state on a columnheader **MUST** not cause the user agent to automatically propagate the [aria-selected](#) state to all the cells in the corresponding column. An author **MAY** choose to propagate selection in this manner depending on the specific application.

While the columnheader role can be used in both interactive grids and non-interactive tables, the use of [aria-readonly](#) and [aria-required](#) is only applicable to interactive elements. Therefore, authors **SHOULD NOT** use [aria-required](#) or [aria-readonly](#) in a columnheader that descends from a [table](#), and user agents **SHOULD NOT** expose either property to [assistive technologies](#) unless the columnheader descends from a [grid](#).

NOTE

Because cells are organized into rows, there is not a single container element for the column. The column is the set of [gridcell](#) elements in a particular position within their respective [row](#) containers.

NOTE: Usage of aria-disabled

While [aria-disabled](#) is currently supported on [columnheader](#), in a future version the working group plans to prohibit its use on elements with role [columnheader](#) except when the element is in the context of a [grid](#) or [treegrid](#).

Characteristics:

Characteristic	Value
Superclass Role:	cell gridcell sectionhead
Base Concept:	<th[scope="col"]> in [HTML]
Required Context Role:	row
Supported States and Properties:	aria-sort
Inherited States and Properties:	aria-atomic aria-busy (state) aria-colindex aria-colspan aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) aria-dropeffect aria-errormessage aria-expanded (state) aria-flowto aria-grabbed (state) aria-haspopup aria-hidden (state) aria-invalid (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-readonly aria-relevant aria-required aria-roledescription aria-rowindex aria-rowspan aria-selected (state)
Name From:	contents author
Accessible Name Required:	True

combobox role

An [input](#) that controls another element, such as a [listbox](#) or [grid](#), that can dynamically pop up to help the user set the value of the [input](#).

EDITOR'S NOTE: Major Changes to combobox role in ARIA 1.2

The Guidance for [combobox](#) has changed significantly in ARIA 1.2 due to problems with implementation of the previous patterns. Authors and developers of User Agents, Assistive Technologies, and Conformance Checkers are advised to review this section carefully to understand the changes. Explanation of the changes is available in the [ARIA repository wiki](#).

A combobox functionally combines a named input field with the ability to assist value selection via a supplementary popup element. A combobox input **MAY** be either a single-line text field that supports editing and typing or an element that only displays the current value of the combobox. If the combobox supports text input and provides autocompletion behavior as described in [aria-autocomplete](#), authors **MUST** set [aria-autocomplete](#) on the combobox element to the value that corresponds to the provided behavior.

Typically, the initial state of a combobox is collapsed. In the collapsed state, only the combobox element and a separate, optional popup control [button](#) are visible. A combobox is said to be expanded when both the combobox element showing its current value and its associated popup element are visible. Authors **MUST** set [aria-expanded](#) to true on an element with role combobox when it is expanded and false when it is collapsed.

Authors **MUST** ensure the popup element associated with a combobox has a role of [listbox](#), [tree](#), [grid](#), or [dialog](#). Authors **MUST** set [aria-controls](#) on a combobox element to a value that refers to the combobox popup element.

Elements with the role combobox have an implicit [aria-haspopup](#) value of `listbox`. If the combobox popup element has a role other than [listbox](#), authors **MUST** specify a value for [aria-haspopup](#) that corresponds to the role of its popup.

If the user interface includes an additional icon that allows the visibility of the popup to be controlled via pointer and touch events, authors **SHOULD** ensure that element has role [button](#), that it is focusable but not included in the page Tab sequence, and that it is not a descendant of the element with role combobox. In addition, to be keyboard accessible, authors **SHOULD** provide keyboard mechanisms for moving focus between the combobox element and elements contained in the popup. For example, one common convention is that Down Arrow moves focus from the input to the first focusable descendant of the popup element. If the popup element supports [aria-activedescendant](#), in lieu of moving focus, such keyboard mechanisms can control the value of [aria-activedescendant](#) on the combobox element. When a descendant of the popup element is active, authors **MAY** set [aria-activedescendant](#) on the combobox to a value that refers to the active element within the popup while focus remains on the combobox element.

User agents **MUST** expose the value of elements with role combobox to [assistive technologies](#). The value of a combobox is represented by one of the following:

- If the combobox element is a host language element that provides a value, such as an `HTML` input element, the value of the combobox is the value of that element.
- Otherwise, the value of the combobox is represented by its descendant elements and can be determined using the same method used to compute the name of a [button](#) from its descendant content.

EXAMPLE 6

```
<label for="tag_combo">Tag</label>
<input type="text" id="tag_combo"
  role="combobox" aria-autocomplete="list"
  aria-haspopup="listbox" aria-expanded="true"
  aria-controls="popup_listbox" aria-activedescendant="selected_option">
<ul role="listbox" id="popup_listbox">
  <li role="option">Zebra</li>
  <li role="option" id="selected_option">Zoom</li>
</ul>
```

EDITOR'S NOTE: Validity changes combobox for ARIA 1.2

Please review the following carefully. As a result of these changes a combobox following the ARIA 1.1 combobox specification will no longer conform with the ARIA specification.

NOTE

The structural requirements for combobox defined by this version of the specification are different from the requirements defined by ARIA 1.0 and ARIA 1.1:

- The ARIA 1.0 specification required the input element with the combobox role to be a single-line text field and reference the popup element with [aria-owns](#) instead of [aria-controls](#).
- The ARIA 1.1 specification, which was not broadly supported by assistive technologies, required the combobox to be a non-focusable element with two required owned elements -- a focusable [textbox](#) and a popup element controlled by the [textbox](#).
- The changes introduced in ARIA 1.2 improve interoperability with assistive technologies and enable authors to create presentations of combobox that more closely imitate a native `HTML` `select` element.

The features and behaviors of combobox implementations vary widely. Consequently, there are many important authoring considerations. See the [WAI-ARIA Authoring Practices](#) for additional details on implementing combobox design patterns.

Characteristics:

Characteristic	Value
Superclass Role:	input
Related Concepts:	<select> in [HTML]
Required States and Properties:	aria-controls aria-expanded
Supported States and Properties:	aria-activedescendant aria-autocomplete aria-errormessage aria-haspopup aria-invalid aria-readonly aria-required
Inherited States and Properties:	aria-atomic aria-busy (state) aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

Characteristic	Value
Accessible Name Required:	True
Implicit Value for Role:	Default for aria-haspopup is listbox.

command role

A form of widget that performs an action but does not receive input data.

NOTE

command is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	widget
Subclass Roles:	button link menuitem
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state)

Characteristic	Value
	aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

complementary role

A [landmark](#) that is designed to be complementary to the main content at a similar level in the DOM hierarchy, but remaining meaningful when separated from the main content.

There are various types of content that would appropriately have this [role](#). For example, in the case of a portal, this may include but not be limited to show times, current weather, related articles, or stocks to watch. The complementary role indicates that contained content is relevant to the main content. If the complementary content is completely separable from the main content, it may be appropriate to use a more general role.

User agents **SHOULD** treat elements with the role of complementary as navigational [landmarks](#).

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state)

Characteristic	Value
	aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

composite role

A [widget](#) that may contain navigable descendants or [owned](#) children.

Authors **SHOULD** ensure that a composite widget exists as a single navigation stop within the larger navigation system of the web page. Once the composite widget has focus, authors **SHOULD** provide a separate navigation mechanism for users to navigate to [elements](#) that are descendants or owned children of the composite element.

NOTE

`composite` is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	widget
Subclass Roles:	grid select spinbutton tablist
Supported States and Properties:	aria-activedescendant aria-disabled
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts

Characteristic	Value
	aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

contentinfo role

A [landmark](#) that contains information about the parent document.

Examples of information included in this region of the page are copyrights and links to privacy statements.

User agents **SHOULD** treat elements with the role of contentinfo as navigational [landmarks](#).

Within any [document](#) or [application](#), the author **SHOULD** mark no more than one [element](#) with the contentinfo role.

NOTE

Because document and application elements can be nested in the DOM, they may have multiple contentinfo elements as DOM descendants, assuming each of those is associated with different document nodes, either by a DOM nesting (e.g., [document](#) within [document](#)) or by use of the [aria-owns](#) attribute.

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

definition role

A definition of a term or concept. See related [term](#).

Authors **SHOULD** identify the [element](#) being defined by giving that element a role of [term](#) and referencing it with the [aria-labelledby](#) attribute or by making the element with role [term](#) a descendant of the element with role definition.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

deletion role

A deletion contains content that is marked as removed or content that is being suggested for removal. See related [insertion](#).

Deletions are typically used to either mark differences between two versions of content or to designate content suggested for removal in scenarios where multiple people are revising content.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	 in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant

Characteristic	Value
	aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

dialog role

A dialog is a descendant window of the primary window of a web application. For HTML pages, the primary application window is the entire web document, i.e., the body element.

Dialogs are most often used to prompt the user to enter or respond to information. A dialog that is designed to interrupt workflow is usually modal. See related [alertdialog](#).

Authors **MUST** provide an accessible name for a dialog, which can be done with the [aria-label](#) or [aria-labelledby](#) attribute.

Authors **SHOULD** ensure that all dialogs (both modal and non-modal) have at least one focusable descendant element. Authors **SHOULD** focus an element in the modal dialog when it is displayed, and authors **SHOULD** manage focus of modal dialogs.

NOTE

In the description of this role, the term "web application" does not refer to the [application](#) role, which specifies specific assistive technology behaviors.

Characteristics:

Characteristic	Value
Superclass Role:	window
Subclass Roles:	alertdialog
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-modal aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

directory role

[Deprecated in ARIA 1.2] A list of references to members of a group, such as a static table of contents.

NOTE

As exposed by accessibility APIs, the `directory` [role](#) is essentially equivalent to the `list` [role](#). So, using `directory` does not provide any additional benefits to assistive technology users. Authors are advised to treat `directory` as deprecated and to use `list`, or a host language's equivalent semantics instead.

A `directory` is a static table of contents, whether linked or unlinked. This includes tables of contents built with lists, including nested lists. Dynamic tables of contents, however, might use a [tree](#) role instead.

Characteristics:

Characteristic	Value
Superclass Role:	list
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

document role

An [element](#) containing content that [assistive technology](#) users may want to browse in a reading mode.

When [user agent](#) focus moves to an element assigned the role of document, [assistive technologies](#) having a reading mode for browsing static content **MAY** switch to that reading mode and intercept standard input events, such as Up or Down arrow keyboard events, to control the reading cursor.

Because [assistive technologies](#) that have a reading mode default to that mode for all elements except for those with either a [widget](#) or [application](#) role, the only circumstance where the document role is useful for changing assistive technology behavior is when the element with role document is a focusable child element of a [widget](#) or [application](#). For example, given an [application](#) element which contains some static rich text, the author can apply role document to the element containing the text and give it a tabindex of 0. When a screen reader user presses the Tab key and places focus on the document element, the user will be able to read the text with the screen reader's reading cursor.

Characteristics:

Characteristic	Value
Superclass Role:	structure
Subclass Roles:	article
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	False

emphasis role

One or more emphasized characters. See related [strong](#).

The purpose of the `emphasis` role is to stress or emphasize content. It is not for communicating changes in typographical presentation that do not impact the meaning of the content. Authors **SHOULD** use the `emphasis` role only if its absence would change the meaning of the content.

The `emphasis` role is not intended to convey importance; for that purpose, the [strong](#) role is more appropriate.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	section
Related Concepts:	 in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

feed role

A scrollable [list](#) of [articles](#) where scrolling may cause [articles](#) to be added to or removed from either end of the list.

A **feed** enables users of [assistive technologies](#) that have a document browse mode, such as screen readers, to use the browse mode reading cursor to both read and scroll through a stream of rich content that may continue scrolling infinitely by loading more content as the user reads. In a **feed**, [assistive technologies](#) provide a web application with signals of the user's reading cursor movement by moving [user agent](#) focus, enabling the application to both add new content and visually position content as the user browses the page. The **feed** also lets authors inform assistive technologies when additions and removals are occurring so assistive technologies can more reliably update their reading view without disrupting reading or degrading performance.

For example, a **feed** could be used to present a stream of news stories where each [article](#) contains a story with text, links, images, and comments as well as widgets for sharing and commenting. As a screen reader user reads and interacts with each story and moves the screen reader reading cursor from story to story, each story scrolls into view and, as needed, new stories are loaded.

A **feed** is a container element whose children have role [article](#). When [articles](#) are added or removed from either or both ends of a **feed**, authors **SHOULD** set [aria-busy](#) to true on the **feed** element before the changes are made and set it to false after the changes are complete. Authors **SHOULD** avoid inserting or removing [articles](#) in the middle of a **feed**. These requirements help [assistive technologies](#) gracefully respond to changes in the **feed** content that occur simultaneously with user commands to move the reading cursor within the **feed**.

Authors **SHOULD** make each [article](#) in a **feed** focusable and ensure that the application scrolls an [article](#) into view when [user agent](#) focus is set on the [article](#) or one of its descendant elements. For example, in [HTML](#), each [article](#) element should have a [tabindex](#) value of either -1 or 0.

When an [assistive technology](#) reading cursor moves from one [article](#) to another, [assistive technologies](#) **SHOULD** set user agent focus on the [article](#) that contains the reading cursor. If the reading cursor lands on a focusable element inside the [article](#), the assistive technology **MAY** set focus on that element in lieu of setting focus on the containing [article](#).

Because the ability to scroll to another [article](#) with an [assistive technology](#) reading cursor depends on the presence of another [article](#) in the page, authors **SHOULD** attempt to load additional [articles](#) before [user agent](#) focus reaches an [article](#) at either end of the set of [articles](#) that has been loaded. Alternatively, authors **MAY** include an [article](#) at either or both ends of the loaded set of [articles](#) that includes an element, such as a [button](#), that lets the user request more [articles](#) to be loaded.

In addition to providing a brief label, authors **MAY** apply [aria-describedby](#) to [article](#) elements in a **feed** to suggest to screen readers which elements to speak after the label when users navigate by [article](#).

Screen readers **MAY** provide users with a way to quickly scan feed content by speaking both the label and [accessible description](#) when navigating by [article](#), enabling the user to ignore repetitive or less important elements, such as embedded interaction widgets, that the author has left out of the description.

Authors **SHOULD** provide keyboard commands for moving focus among [articles](#) in a feed so users who do not utilize an assistive technology that provides [article](#) navigation features can use the keyboard to navigate the feed.

If the number of articles available in a feed supply is static, authors **MAY** specify [aria-setsize](#) on [article](#) elements in that feed. However, if the total number is extremely large, indefinite, or changes often, authors **MAY** set [aria-setsize](#) to -1 to communicate the unknown size of the set.

See the [WAI-ARIA Authoring Practices](#) for additional details on implementing a feed design pattern.

Characteristics:

Characteristic	Value
Superclass Role:	list
Required Owned Elements:	article
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	False

figure role

A perceivable [section](#) of content that typically contains a [graphical document](#), images, code snippets, or example text. The parts of a figure **MAY** be user-navigable.

Authors **SHOULD** provide a reference to the figure from the main text, but the figure need not be displayed at the same location as the referencing element. Authors **MAY** reference text serving as a caption using [aria-describedby](#). Authors **MAY** provide a label using [aria-label](#) or **MAY** reference text serving as a label using [aria-labelledby](#).

[Assistive technologies](#) **SHOULD** enable users to quickly navigate to figures. Mainstream [user agents](#) **MAY** enable users to quickly navigate to figures.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<figure> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	False

form role

A [landmark](#) region that contains a collection of items and objects that, as a whole, combine to create a form. See related [search](#).

A form may contain a mix of host language form controls, scripted controls, and hyperlinks. Authors are reminded to use native host language semantics to create form controls whenever possible. If the purpose of a form is to submit search criteria, authors **SHOULD** use the [search](#) role instead of the generic form role.

Authors **MUST** give each element with role form a brief label that describes the purpose of the form.

Authors **SHOULD** reference a visible label with [aria-labelledby](#) if a visible label is present. Authors

SHOULD include the label inside of a heading whenever possible. The heading **MAY** be an instance of the standard host language heading element or an instance of an element with role [heading](#).

If an author uses a script to submit a form based on a user action that would otherwise not trigger an `onsubmit` event (for example, a form submission triggered by the user changing a form element's value), the author **SHOULD** provide the user with advance notification of the behavior.

User agents **SHOULD** treat elements with the role of `form` as navigational [landmarks](#).

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Base Concept:	<code><form></code> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	true

generic role

A nameless container [element](#) that has no semantic meaning on its own.

The `generic` role is intended for use as the implicit role of generic elements in host languages (such as `HTML div` or `span`), so is primarily for implementors of user agents. Authors ***SHOULD NOT*** use this role in content. Authors ***MAY*** use [presentation](#) or [none](#) to remove implicit accessibility semantics, or a semantic container role such as [group](#) to semantically group descendants in a named container.

Like an element with role [presentation](#), an element with role `generic` can provide a limited number of accessible states and properties for its descendants, such as [aria-live](#) attributes. However, unlike elements with role [presentation](#), `generic` elements are exposed in [accessibility APIs](#) so that assistive technologies can gather certain properties such as layout and bounds.

Characteristics:

Characteristic	Value
Superclass Role:	structure
Related Concepts:	HTML div , HTML span
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect

Characteristic	Value
	aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant
Prohibited States and Properties:	aria-label aria-labelledby aria-roledescription
Name From:	prohibited

grid role

A composite [widget](#) containing a collection of one or more rows with one or more cells where some or all cells in the grid are focusable by using methods of two-dimensional navigation, such as directional arrow keys.

The `grid` role does not imply a specific visual, e.g., tabular, presentation. It describes [relationships](#) among [elements](#). It may be used for purposes as simple as grouping a collection of checkboxes or navigation links or as complex as creating a full-featured spreadsheet application.

The cell elements of a `grid` have role [gridcell](#). Authors **MAY** designate a cell as a row or column header by using either the [rowheader](#) or [columnheader](#) role in lieu of the [gridcell](#) role. Authors **MUST** ensure elements with role [gridcell](#), [columnheader](#), or [rowheader](#) are [owned](#) by elements with role [row](#), which are in turn owned by an element with role [rowgroup](#), or `grid`.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants of a grid as described in [Managing Focus](#). When a user is navigating the grid content with a keyboard, authors **SHOULD** set focus as follows:

- If a [gridcell](#) contains a single interactive [widget](#) that will not consume arrow key presses when it receives focus, such as a [checkbox](#), [button](#), or [link](#), authors **MAY** set focus on the interactive element contained in that cell. This allows the contained widget to be directly operable.
- Otherwise, authors **SHOULD** ensure the element that receives focus is a [gridcell](#), [rowheader](#), or [columnheader](#) element.

Authors **SHOULD** provide a mechanism for changing to an interaction or edit mode that allows users to navigate and interact with content contained inside a focusable cell if that focusable cell contains any of the following:

- a widget that requires arrow keys to operate, e.g., a [combobox](#) or [radiogroup](#)
- multiple interactive elements
- editable content

For example, if a cell in a spreadsheet contains a [combobox](#) or editable text, the Enter key might be used to activate a cell interaction or editing mode when that cell has focus so the directional arrow keys can be used to operate the contained [combobox](#) or [textbox](#). Depending on the implementation, pressing Enter again, Tab, Escape, or another key may switch the application back to the grid navigation mode.

Authors **MAY** use a [gridcell](#) to display the result of a formula, which could be editable by the user. In a spreadsheet application, for example, a [gridcell](#) may show a value calculated from a formula until the user activates the [gridcell](#) for editing when a [textbox](#) appears in the [gridcell](#) containing the formula in an editable state.

If [aria-readonly](#) is set on an element with role grid, [user agents](#) **MUST** propagate the value to all [gridcell](#) elements [owned](#) by the grid and expose the value in the accessibility API. An author **MAY** override the propagated value of [aria-readonly](#) for an individual [gridcell](#) element.

In a grid that provides cell content editing functions, if the content of a focusable [gridcell](#) element is not editable, authors **MAY** set [aria-readonly](#) to true on the gridcell element. However, the value of [aria-readonly](#), whether specified for a grid or individual cells, only indicates whether the content contained in cells is editable. It does not represent availability of functions for navigating or manipulating the grid itself.

An unspecified value for [aria-readonly](#) does not imply that a grid or a [gridcell](#) contains editable content. For example, if a grid presents a collection of elements that are not editable, such as a collection of [link](#) elements representing dates in a datepicker, it is not necessary for the author to specify a value for [aria-readonly](#).

Authors **MAY** indicate that a focusable [gridcell](#) is selectable as the object of an action with the [aria-selected](#) attribute. If the grid allows multiple [gridcells](#) to be selected, the author **SHOULD** set [aria-multiselectable](#) to true on the element with role grid.

Since [WAI-ARIA](#) can augment an element of the host language, a grid can reuse the elements and attributes of a native table, such as an [HTML](#) table element. For example, if an author applies the grid role to an [HTML](#) table element, the author does not need to apply the [row](#) and [gridcell](#) roles to the descendant [HTML](#) tr and td elements because the [user agent](#) will automatically make the appropriate translations. When the author is reusing a native host language table element and needs a [gridcell](#) element to span multiple rows or columns, the author **SHOULD** apply the appropriate host language attributes instead of [WAI-ARIA](#) [aria-rowspan](#) or [aria-colspan](#) properties.

See the [WAI-ARIA Authoring Practices](#) for additional details on implementing grid design patterns.

Characteristics:

Characteristic	Value
Superclass Role:	composite table
Subclass Roles:	treegrid
Base Concept:	<table> in [HTML]
Required Owned Elements:	row rowgroup ,... row
Supported States and Properties:	aria-multiselectable aria-readonly
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-colcount aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state)

Characteristic	Value
	aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-rowcount
Name From:	author
Accessible Name Required:	True

gridcell role

A [cell](#) in a [grid](#) or [treegrid](#).

A gridcell may be focusable, editable, and selectable. A gridcell may have [relationships](#) such as [aria-controls](#) to address the application of functional relationships.

If an author intends a gridcell to have a row header, column header, or both, and if the relevant headers cannot be determined from the DOM structure, authors **SHOULD** explicitly indicate which header cells are relevant to the gridcell by applying [aria-describedby](#) on the gridcell and referencing [elements](#) with [role rowheader](#) or [columnheader](#).

In a [treegrid](#), authors **MAY** define a `gridcell` as expandable by using the [aria-expanded](#) attribute. If the [aria-expanded](#) attribute is provided, it applies only to the individual cell. It is not a proxy for the container [row](#), which also can be expanded. The main use case for providing this attribute on a `gridcell` is pivot table behavior.

Authors **MUST** ensure [elements](#) with [role](#) `gridcell` are contained in, or [owned](#) by, an element with the [role](#) `row`.

Characteristics:

Characteristic	Value
Superclass Role:	cell widget
Subclass Roles:	columnheader rowheader
Base Concept:	<td> in [HTML]
Required Context Role:	row
Supported States and Properties:	aria-disabled aria-errormessage aria-expanded aria-haspopup aria-invalid aria-readonly aria-required aria-selected
Inherited States and Properties:	aria-atomic aria-busy (state) aria-colindex aria-colspan aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-dropeffect aria-flowto aria-grabbed (state) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-rowindex aria-rowspan
Name From:	contents author

group role

A set of user interface [objects](#) that is not intended to be included in a page summary or table of contents by [assistive technologies](#).

Contrast with [region](#), which is a grouping of user interface objects that will be included in a page summary or table of contents.

Authors **SHOULD** use a group to form a logical collection of items in a [widget](#), such as children in a tree widget forming a collection of siblings in a hierarchy. However, when a group is used in the context of a [listbox](#), authors **MUST** limit its children to [option](#) elements. Therefore, proper handling of group by authors and assistive technologies is determined by the context in which it is provided.

Authors **MAY** nest group elements. If a section is significant enough to warrant inclusion in the web page's table of contents, the author **SHOULD** assign it a [role](#) of [region](#) or a [standard landmark role](#).

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	section
Subclass Roles:	row select toolbar
Related Concepts:	<fieldset> in [HTML]
Supported States and Properties:	aria-activedescendant aria-disabled

Characteristic	Value
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

heading role

A heading for a section of the page.

To ensure elements with a role of heading are organized into a logical outline, authors **MUST** use the [aria-level](#) attribute to indicate the proper nesting level.

Characteristics:

Characteristic	Value
Superclass Role:	sectionhead
Related Concepts:	<h1>, <h2>, <h3>, <h4>, <h5>, and <h6> in [HTML]
Required States and Properties:	aria-level
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents

Characteristic	Value
	author
Accessible Name Required:	True

img role

A container for a collection of [elements](#) that form an image.

An `img` can contain captions and descriptive text, as well as multiple image files that when viewed together give the impression of a single image. An `img` represents a single graphic within a document, whether or not it is formed by a collection of drawing [objects](#). In order for elements with a [role](#) of `img` to be [perceivable](#), authors **MUST** provide a label using the [aria-label](#) or [aria-labelledby](#) attribute.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	 in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state)

Characteristic	Value
	aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True
Children Presentational:	True

input role

A generic type of [widget](#) that allows user input.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	widget
Subclass Roles:	checkbox combobox option radio slider spinbutton textbox
Supported States and Properties:	aria-disabled
Inherited States and Properties:	aria-atomic aria-busy (state)

Characteristic	Value
	aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

insertion role

An insertion contains content that is marked as added or content that is being suggested for addition. See related [deletion](#).

Insertions are typically used to either mark differences between two versions of content or to designate content suggested for addition in scenarios where multiple people are revising content.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<ins> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

Landmark role

A perceivable [section](#) containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.

Authors designate the purpose of the content by assigning a role that is a subclass of the landmark role and, when needed, by providing a brief, descriptive label.

Elements with a role that is a subclass of the landmark role are known as landmark regions or navigational landmark regions. [Assistive technologies](#) **SHOULD** enable users to quickly navigate to landmark regions. Mainstream [user agents](#) **MAY** enable users to quickly navigate to landmark regions.

NOTE

landmark is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	section
Subclass Roles:	banner complementary contentinfo form main navigation region search
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	False

Link role

An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource. See related [button](#).

If this is a native link in the host language (such as an [HTML](#) anchor with an href value), activating the link causes the [user agent](#) to navigate to that resource. If this is a simulated link, the web application author is responsible for managing navigation.

NOTE

If pressing the link triggers an action but does not change browser focus or page location, authors are advised to consider using the [button](#) role instead of the link role.

Characteristics:

Characteristic	Value
Superclass Role:	command
Related Concepts:	<a> in [HTML] <link> in [HTML]
Supported States and Properties:	aria-disabled aria-expanded aria-haspopup
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live

Characteristic	Value
	aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True

list role

A [section](#) containing [listitem](#) elements. See related [listbox](#).

Lists contain children whose [role](#) is [listitem](#).

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	directory feed
Base Concept:	in [HTML] in [HTML]
Required Owned Elements:	listitem
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect

Characteristic	Value
	aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

listbox role

A [widget](#) that allows the user to select one or more items from a list of choices. See related [combobox](#) and [list](#).

Items within the list are static and, unlike standard HTML [select elements](#), may contain images. List boxes contain children whose [role](#) is [option](#) or elements whose [role](#) is [group](#) which in turn contains children whose [role](#) is [option](#).

To be [keyboard accessible](#), authors **SHOULD** manage focus of [option](#) descendants for all instances of this [role](#), as described in [Managing Focus](#).

Elements with the role `listbox` have an implicit [aria-orientation](#) value of `vertical`.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	select
Related Concepts:	list <select> in [HTML]
Required Owned Elements:	group , option option
Supported States and Properties:	aria-errormessage aria-expanded aria-invalid aria-multiselectable aria-readonly aria-required
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-orientation aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True
Implicit Value for Role:	Default for aria-orientation is vertical.

listitem role

A single item in a list or directory.

Authors **MUST** ensure [elements](#) whose [role](#) is listitem are contained in, or [owned](#) by, an [element](#) whose [role](#) is [list](#).

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	treeitem
Base Concept:	in [HTML]
Required Context Role:	directory list
Supported States and Properties:	aria-level aria-posinset aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

log role

A type of [live region](#) where new information is added in meaningful order and old information may disappear. See related [marquee](#).

Examples include chat logs, messaging history, game log, or an error log. In contrast to other live regions, in this [role](#) there is a [relationship](#) between the arrival of new items in the log and the reading order. The log contains a meaningful sequence and new information is added only to the end of the log, not at arbitrary points.

Elements with the role `log` have an implicit [aria-live](#) value of `polite`.

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-live is polite.

main role

A [landmark](#) containing the main content of a document.

This marks the content that is directly related to or expands upon the central topic of the document. The [main role](#) is a non-obtrusive alternative for "skip to main content" links, where the navigation option to go to the main content (or other [landmarks](#)) is provided by the [user agent](#) through a dialog or by [assistive technologies](#).

User agents **SHOULD** treat elements with the role of main as navigational landmarks.

Within any [document](#) or [application](#), the author **SHOULD** mark no more than one [element](#) with the main role.

NOTE

Because document and application elements can be nested in the DOM, they may have multiple main elements as DOM descendants, assuming each of those is associated with different document nodes, either by a DOM nesting (e.g., [document](#) within [document](#)) or by use of the [aria-owns](#) attribute.

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state)

Characteristic	Value
	aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

marquee role

A type of [live region](#) where non-essential information changes frequently. See related [log](#).

Common usages of marquee include stock tickers and ad banners. The primary difference between a marquee and a [log](#) is that logs usually have a meaningful order or sequence of important content changes.

Elements with the role marquee have an implicit [aria-live](#) value of off.

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

math role

Content that represents a mathematical expression.

Content with the role `math` is intended to be marked up in an accessible format such as [MathML](#) [MathML3], or with another type of textual representation such as TeX or LaTeX, which can be converted to an accessible format by native browser implementations or a polyfill library.

While it is not ideal to use an image of a mathematical expression, there exists a significant amount of legacy content where images are used to represent mathematical expressions. Authors **SHOULD** ensure that images of math are labeled by text that describes the mathematical expression as it might be spoken.

NOTE

Browsers that support native implementations of **MathML**, are able to provide a more robust, accessible math experience than can be accomplished with plain text approximations of math. Some rendering engines have close integration with screen readers that allow spacial touch exploration of the formula and refreshable braille display output in the [Nemeth Braille](#) format. This level of integration is not supported with images of mathematical formulas, even if the author provides a plain text approximation.

At the time of this writing, some mainstream browsers do not support **MathML**, natively, and must be retrofit using a JavaScript polyfill library. When authoring math content, use native **MathML**, wherever possible, and test thoroughly. Use a polyfill library or provide a fallback image with a text alternative approximation if necessary.

§ MathML Example with Embedded TeX Annotation

EXAMPLE 7

```

<!-- Note: Use a JavaScript polyfill library to ensure
      this renders in user agents that do not support MathML. -->
<!-- The math element has an implicit role="math". -->
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
      <mrow>
        <mo form="prefix">-</mo>
        <mi>b</mi>
        <mo>±</mo>
        <msqrt>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>-</mo>
          <mn>4</mn>
          <mo>&#x2062;<!-- &InvisibleTimes; --></mo>
          <mi>a</mi>
          <mo>&#x2062;<!-- &InvisibleTimes; --></mo>
          <mi>c</mi>
        </msqrt>
      </mrow>
      <mrow>
        <mn>2</mn>
        <mo>&#x2062;<!-- &InvisibleTimes; --></mo>
        <mi>a</mi>
      </mrow>
    </mfrac>
  </mrow>
  <annotation encoding="TeX">
    x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}
  </annotation>
</math>

```

§ Plain HTML or Polyfill DOM Result of the MathML Quadratic Formula

If a rendering engine does not support a native math format such as MathML, authors **MAY** use JavaScript to downgrade the content to a format the browser can display, such as this [HTML](#) image using a data URI and plain text alternative.

EXAMPLE 8

```

```

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant

Characteristic	Value
	aria-roledescription
Name From:	author
Children Presentational:	False

meter role

An [element](#) that represents a scalar measurement within a known range, or a fractional value. See related [progressbar](#).

Authors **MAY** set [aria-valuemin](#) and [aria-valuemax](#) to indicate the minimum and maximum values for the meter. Otherwise, their implicit values follow the same rules as `<input [ type="range" ]>` in [HTML]:

- If [aria-valuemin](#) is missing or not a [number](#), it defaults to 0 (zero).
- If [aria-valuemax](#) is missing or not a [number](#), it defaults to 100.

The value of [aria-valuenow](#) **MUST NOT** fall below or exceed the computed values of [aria-valuemin](#) and [aria-valuemax](#), respectively.

Authors **SHOULD NOT** use the meter role to indicate progress; the [progressbar](#) role exists to address that need.

NOTE

Presently, there are no WAI-ARIA properties corresponding to the low, optimum, and high attributes supported on the `<meter>` element in [HTML]. The addition of these properties will be considered for ARIA version 1.3.

Characteristics:

Characteristic	Value
Superclass Role:	range
Related Concepts:	<code><meter></code> in [HTML]
Required States and Properties:	aria-valuenow
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls

Characteristic	Value
	aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-valuemax aria-valuemin aria-valuetext
Name From:	author
Accessible Name Required:	True
Children Presentational:	True
Implicit Value for Role:	Default for aria-valuemin is 0. Default for aria-valuemax is 100.

menu role

A type of [widget](#) that offers a list of choices to the user.

A menu is often a list of common actions or functions that the user can invoke. The menu [role](#) is appropriate when a list of menu items is presented in a manner similar to a menu on a desktop application.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

Elements with the role menu have an implicit [aria-orientation](#) value of vertical.

Characteristics:

Characteristic	Value
Superclass Role:	select
Subclass Roles:	menubar
Related Concepts:	list
Required Owned Elements:	group ... menuitem group ... menuitemradio group ... menuitemcheckbox menuitem menuitemcheckbox menuitemradio
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-orientation aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-orientation is vertical.

menubar role

A presentation of [menu](#) that usually remains visible and is usually presented horizontally.

The menubar [role](#) is used to create a menu bar similar to those found in Windows, Mac, and Gnome desktop applications. A menu bar is used to create a consistent set of frequently used commands. Authors **SHOULD** ensure that menubar interaction is similar to the typical menu bar interaction in a desktop graphical user interface.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

Elements with the role menubar have an implicit [aria-orientation](#) value of horizontal.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	menu
Related Concepts:	toolbar
Required Owned Elements:	group ... menuitem group ... menuitemradio group ... menuitemcheckbox menuitem menuitemcheckbox menuitemradio
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live

Characteristic	Value
	aria-orientation aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-orientation is horizontal.

menuitem role

An option in a set of choices contained by a [menu](#) or [menubar](#).

Authors **MAY** disable a menu item with the [aria-disabled](#) attribute. If the menu item has its [aria-haspopup](#) attribute set to `true`, it indicates that the menu item may be used to launch a sub-level menu, and authors **SHOULD** display a new sub-level menu when the menu item is activated.

In order to identify that they are related [widgets](#), authors **MUST** ensure that menu items are [owned](#) by an element with role [menu](#) or [menubar](#). Authors **MAY** separate menu items into sets by use of a [separator](#) or an element with an equivalent role from the native markup language.

Characteristics:

Characteristic	Value
Superclass Role:	command
Subclass Roles:	menuitemcheckbox
Related Concepts:	listitem option
Required Context Role:	group menu menubar
Supported States and Properties:	aria-disabled aria-expanded aria-haspopup aria-posinset

Characteristic	Value
	aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True

menuitemcheckbox role

A [menuitem](#) with a checkable state whose possible values are true, false, or mixed.

The [aria-checked](#) [attribute](#) of a `menuitemcheckbox` indicates whether the menu item is checked (`true`), unchecked (`false`), or represents a sub-level menu of other menu items that have a mixture of checked and unchecked values (`mixed`).

In order to identify that they are related [widgets](#), authors **MUST** ensure that menu item checkboxes are [owned](#) by an element with role [menu](#) or [menubar](#). Authors **MAY** separate menu items into sets by use of a [separator](#) or an element with an equivalent role from the native markup language.

Characteristics:

Characteristic	Value
Superclass Role:	menuitem
Subclass Roles:	menuitemradio
Related Concepts:	menuitem
Required Context Role:	group menu menubar
Required States and Properties:	aria-checked
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-expanded (state) aria-flowto aria-grabbed (state) aria-haspopup aria-hidden (state)

Characteristic	Value
	aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-posinset aria-relevant aria-roledescription aria-setsize
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

menuitemradio role

A checkable [menuitem](#) in a set of elements with the same role, only one of which can be checked at a time.

Authors **SHOULD** enforce that only one menuitemradio in a group can be checked at the same time. When one item in the group is checked, the previously checked item becomes unchecked (its [aria-checked](#) [attribute](#) becomes false).

In order to identify that they are related [widgets](#), authors **MUST** ensure that menu item radios are [owned](#) by an element with role [menu](#) or [menubar](#), or by a role [group](#) which itself is [owned](#) by an element with role [menu](#) or [menubar](#).

If a [menu](#) or [menubar](#) contains more than one group of menuitemradio elements, or if the menu contains one group and other, unrelated menu items, authors **SHOULD** contain each set of related menuitemradio elements in an element using the [group](#) role. Authors **MAY** also delimit the group from other menu items with an element using the [separator](#) role, or an element with an equivalent role from the native markup language.

Characteristics:

Characteristic	Value
Superclass Role:	menuitemcheckbox
Related Concepts:	menuitem
Required Context Role:	group menu menubar
Inherited States and Properties:	aria-atomic aria-busy (state) aria-checked (state) (required) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-expanded (state) aria-flowto aria-grabbed (state) aria-haspopup aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns

Characteristic	Value
	aria-posinset aria-relevant aria-roledescription aria-setsize
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

navigation role

A [landmark](#) containing a collection of navigational [elements](#) (usually links) for navigating the document or related documents.

User agents ***SHOULD*** treat elements with the role of navigation as navigational [landmarks](#).

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Related Concepts:	<nav> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto

Characteristic	Value
	aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

none role

An [element](#) whose implicit native role semantics will not be mapped to the [accessibility API](#). See synonym [presentation](#).

NOTE

Note regarding the ARIA 1.1 none role.

In ARIA 1.1, the working group introduced none as a synonym to the [presentation](#) role, due to author confusion surrounding the intended meaning of the word "presentation" or "presentational." Many individuals erroneously consider role="presentation" to be synonymous with aria-hidden="true", and we believe role="none" conveys the actual meaning more unambiguously.

note role

A section whose content is parenthetic or ancillary to the main content of the resource.

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription

Characteristic	Value
Name From:	author

option role

A selectable item in a [listbox](#).

Authors **MUST** ensure [elements](#) with [role](#) option are contained in, or [owned](#) by, an element with the [role](#) [listbox](#) or [group](#) within a listbox. Options not associated with a [listbox](#) might not be correctly mapped to an [accessibility API](#).

Elements with the role option have an implicit [aria-selected](#) value of false.

Characteristics:

Characteristic	Value
Superclass Role:	input
Subclass Roles:	treeitem
Base Concept:	<option> in [HTML]
Related Concepts:	listitem
Required Context Role:	group listbox
Required States and Properties:	aria-selected
Supported States and Properties:	aria-checked aria-posinset aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state)

Characteristic	Value
	aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True
Implicit Value for Role:	Default for aria-selected is false.

paragraph role

A paragraph of content.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<p> in [HTML]

Characteristic	Value
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

presentation role

An [element](#) whose implicit native role semantics will not be mapped to the [accessibility API](#). See synonym [none](#).

NOTE

Note regarding the ARIA 1.1 [none](#) role.

In ARIA 1.1, the working group introduced [none](#) as a synonym to the presentation role, due to author confusion surrounding the intended meaning of the word "presentation" or "presentational." Many individuals erroneously consider `role="presentation"` to be synonymous with `aria-hidden="true"`, and we believe `role="none"` conveys the actual meaning more unambiguously.

Until implementations include sufficient support for `role="none"`, web authors are advised to use the presentation role alone `role="presentation"` or redundantly as a fallback to the [none](#) role `role="none presentation"`.

The intended use is when an element is used to change the look of the page but does not have all the functional, interactive, or structural relevance implied by the element type, or may be used to provide for an accessible fallback in older browsers that do not support [WAI-ARIA](#).

Example use cases:

- An element whose content is completely presentational (like a spacer image, decorative graphic, or clearing element);
- An image that is in a container with the [img role](#) and where the full text alternative is available and is marked up with [aria-labelledby](#) and (if needed) [aria-describedby](#);
- An element used as an additional markup "hook" for [CSS](#); or
- A layout table and/or any of its associated rows, cells, etc.

For any element with a role of presentation and which is not focusable, the user agent **MUST NOT** expose the implicit native semantics of the element (the role and its states and properties) to accessibility [APIs](#). However, the user agent **MUST** expose content and descendant elements that do not have an explicit or inherited role of presentation. Thus, the presentation role causes a given element to be treated as having no role or to be removed from the accessibility tree, but does not cause the content contained within the element to be removed from the accessibility tree.

For example, according to an accessibility API, the following markup elements would appear to have identical role semantics (no role) and identical content.

EXAMPLE 9

```
<!-- 1. [role="presentation"] negates the implicit 'heading' role semantics but does not a
<h1 role="presentation"> Sample Content </h1>

<!-- 2. There is no implicit role for span, so only the contents are exposed. -->
<span> Sample Content </span>

<!-- 3. Depending on styling and other factors, this role declaration is redundant in some
<span role="presentation"> Sample Content </span>

<!-- 4. In all cases, the element contents are exposed to accessibility APIs without any i
<!-- <> --> Sample Content <!-- </> -->
```

The presentation role is used on an element that has implicit native semantics, meaning that there is a default accessibility API role for the element. Some elements are only complete when additional descendant elements are provided. For example, in HTML, table elements (matching the [table](#) role) require tr descendants (the [row](#) role), which in turn require th or td children (the [cell](#), [columnheader](#), [rowheader](#) roles). Similarly, lists require list item children. The descendant elements that complete the semantics of an element are described in WAI-ARIA as [required owned elements](#).

When an explicit or inherited role of presentation is applied to an element with the implicit semantic of a WAI-ARIA role that has [required owned elements](#), in addition to the element with the explicit role of presentation, the user agent **MUST** apply an inherited role of presentation to any owned elements that do not have an explicit role defined. Also, when an explicit or inherited role of presentation is applied to a host language element which has required children as defined by the host language specification, in addition to the element with the explicit role of presentation, the user agent **MUST** apply an inherited role of presentation to any required children that do not have an explicit role defined.

In HTML, the `<img>` [element](#) is treated as a single entity regardless of the type of image file. Consequently, using `role="presentation"` or `role="none"` on an HTML `img` is equivalent to using `aria-hidden="true"`. In order to make the image contents accessible, authors can embed the object using an `<object>` or `<iframe>` [element](#), or use inline SVG code, and follow the accessibility guidelines for the image content.

For any element with an explicit or inherited role of presentation and which is not focusable, user agents **MUST** ignore role-specific WAI-ARIA states and properties for that element. For example, in HTML, a `ul` or `ol` element with a role of presentation will have the implicit native semantics of its `li` elements removed because the [list](#) role to which the `ul` or `ol` corresponds has a [required owned element](#) of [listitem](#). Likewise, the implicit native semantics of an HTML table element's

thead/tbody/tfoot/tr/th/td descendants will also be removed, because the [HTML](#) specification indicates that these are required structural descendants of the table element.

NOTE

Only the implicit native semantics of elements that correspond to [WAI-ARIA required owned elements](#) are removed. All other content remains intact, including nested tables or lists, unless those elements also have an explicit role of presentation applied.

For example, according to an accessibility [API](#), the following markup elements would appear to have identical role semantics (no roles) and identical content.

EXAMPLE 10

```
<!-- 1. [role="presentation"] negates the implicit 'list' and 'listitem' role semantics bu
<ul role="presentation">
  <li> Sample Content </li>
  <li> More Sample Content </li>
</ul>

<!-- 2. There is no implicit role for "foo", so only the contents are exposed. -->
<foo>
  <foo> Sample Content </foo>
  <foo> More Sample Content </foo>
</foo>
```

NOTE

There are other [WAI-ARIA](#) roles with required children for which this situation is applicable (e.g., radiogroups and listboxes), but tables and lists are the most common real-world cases in which the presentation inheritance is likely to apply.

For any element with an explicit or inherited role of presentation, user agents **MUST** apply an inherited role of presentation to all host-language-specific labeling elements for the presentational element. For example, a table element with a role of presentation will have the implicit native semantics of its caption element removed, because the caption is merely a label for the presentational table.

Authors **SHOULD NOT** provide meaningful alternative text (for example, use alt="" in [HTML](#)) when the presentation role is applied to an image.

In the following code sample, the containing [img](#) and is appropriately labeled by the caption paragraph. In this example the img element can be marked as presentation because the role and the text alternatives are provided by the containing element.

EXAMPLE 11

```
<div role="img" aria-labelledby="caption">
  
  <p id="caption">A visible text caption labeling the image.</p>
</div>
```

In the following code sample, because the anchor (HTML `a` element) is acting as the treeitem, the list item (HTML `li` element) is assigned an explicit WAI-ARIA role of presentation to override the user agent's implicit native semantics for list items.

EXAMPLE 12

```
<ul role="tree">
  <li role="presentation">
    <a role="treeitem" aria-expanded="true">An expanded tree node</a>
  </li>
  ...
</ul>
```

§ Presentational Roles Conflict Resolution

There are a number of ways presentational role conflicts are resolved.

User agents **MUST NOT** expose [elements](#) having explicit or inherited presentational role in the accessibility tree, with these exceptions:

- If an element is focusable, or otherwise interactive, user agents **MUST** ignore the presentation role and expose the element with its implicit role, in order to ensure that the element is [operable](#).
- If a [required owned element](#) has an explicit non-presentational role, user agents **MUST** ignore an inherited presentational role and expose the element with its explicit role. If the action of exposing the explicit role causes the accessibility tree to be malformed, the expected results are undefined.
- If an element has global WAI-ARIA states or properties, user agents **MUST** ignore the presentation role and expose the element with its implicit role. However, if an element has only non-global, role-specific WAI-ARIA states or properties, the element **MUST NOT** be exposed unless the presentational role is inherited and an explicit non-presentational role is applied.

For example, [aria-describedby](#) is a global attribute and would always be applied; [aria-level](#) is not a global attribute and would therefore only apply if the element was not in a presentational state.

EXAMPLE 13

```
<!-- 1. [role="presentation"] is ignored due to the global aria-describedby property. -->
<h1 role="presentation" aria-describedby="comment-1"> Sample Content </h1>
<!-- 2. [role="presentation"] negates both the implicit 'heading' and the non-global aria-
<h1 role="presentation" aria-level="2"> Sample Content </h1>
```

Characteristics:

Characteristic	Value
Superclass Role:	structure
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant

Characteristic	Value
	aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

progressbar role

An [element](#) that displays the progress status for tasks that take a long time.

A progressbar indicates that the user's request has been received and the application is making progress toward completing the requested action.

Authors **MAY** set [aria-valuemin](#) and [aria-valuemax](#) to indicate the minimum and maximum progress indicator values. Otherwise, their implicit values follow the same rules as `<input [type="range"]>` in [HTML]:

- If [aria-valuemin](#) is missing or not a [number](#), it defaults to 0 (zero).
- If [aria-valuemax](#) is missing or not a [number](#), it defaults to 100.

The author **SHOULD** supply a value for [aria-valuenow](#) unless the value is indeterminate, in which case the author **SHOULD** omit the [aria-valuenow](#) attribute. Authors **SHOULD** update this value when the visual progress indicator is updated. If the progressbar is describing the loading progress of a particular region of a page, the author **SHOULD** use [aria-describedby](#) to point to the status, and set the [aria-busy](#) attribute to true on the region until it is finished loading. It is not possible for the user to alter the value of a progressbar because it is always read-only.

NOTE

Assistive technologies generally will render the value of [aria-valuenow](#) as a percent of a range between the value of [aria-valuemin](#) and [aria-valuemax](#), unless [aria-valuetext](#) is specified.

Characteristics:

Characteristic	Value
Superclass Role:	range widget
Related Concepts:	status

Characteristic	Value
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-valuemax aria-valuemin aria-valuenow aria-valuetext
Name From:	author

Characteristic	Value
Accessible Name Required:	True
Children Presentational:	True
Implicit Value for Role:	Default for aria-valuemin is 0. Default for aria-valuemax is 100.

radio role

A checkable input in a group of elements with the same role, only one of which can be checked at a time.

Authors **SHOULD** ensure that [elements](#) with role `radio` are explicitly grouped in order to indicate which ones affect the same value. This is achieved by enclosing the radio elements in an element with role [radiogroup](#). If it is not possible to make the radio buttons **DOM** children of the [radiogroup](#), authors **SHOULD** use the [aria-owns](#) attribute on the [radiogroup](#) element to indicate the [relationship](#) to its children.

Characteristics:

Characteristic	Value
Superclass Role:	input
Related Concepts:	<input [type="radio"]> in [HTML]
Required States and Properties:	aria-checked
Supported States and Properties:	aria-posinset aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

radiogroup role

A group of [radio](#) buttons.

A radiogroup is a type of [select](#) list that can only have a single entry checked at any one time. Authors **SHOULD** enforce that only one radio button in a group can be checked at the same time. When one item in the group is checked, the previously checked item becomes unchecked (its [aria-checked](#) attribute becomes false).

Characteristics:

Characteristic	Value
Superclass Role:	select
Related Concepts:	list

Characteristic	Value
Required Owned Elements:	radio
Supported States and Properties:	aria-errormessage aria-invalid aria-readonly aria-required
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-orientation aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

range role

An element representing a range of values.

NOTE

range is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	structure
Subclass Roles:	meter progressbar scrollbar slider spinbutton
Supported States and Properties:	aria-valuemax aria-valuemin aria-valuenow aria-valuetext
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto

Characteristic	Value
	aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

region role

A [landmark](#) containing content that is relevant to a specific, author-specified purpose and sufficiently important that users will likely want to be able to navigate to the section easily and to have it listed in a summary of the page. Such a page summary could be generated dynamically by a user agent or assistive technology.

Authors **SHOULD** limit use of the region role to sections containing content with a purpose that is not accurately described by one of the other [landmark](#) roles, such as [main](#), [complementary](#), or [navigation](#).

Authors **MUST** give each element with role region a brief label that describes the purpose of the content in the region. Authors **SHOULD** reference a visible label with [aria-labelledby](#) if a visible label is present. Authors **SHOULD** include the label inside of a heading whenever possible. The heading **MAY** be an instance of the standard host language heading element or an instance of an element with role [heading](#).

[Assistive technologies](#) **SHOULD** enable users to quickly navigate to elements with role region. Mainstream [user agents](#) **MAY** enable users to quickly navigate to elements with role region.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Superclass Role:	landmark
Related Concepts:	<section> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

roletype role

The base [role](#) from which all other roles inherit.

Properties of this role describe the structural and functional purpose of [objects](#) that are assigned this role. A role is a concept that can be used to understand and operate instances.

NOTE

roletype is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Subclass Roles:	structure widget window
Supported States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (Global use deprecated in ARIA 1.2) aria-dropeffect aria-errormessage (Global use deprecated in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (Global use deprecated in ARIA 1.2) aria-hidden (state) aria-invalid (state) (Global use deprecated in ARIA 1.2)

Characteristic	Value
	aria-keyshortcuts aria-label (Except where prohibited) aria-labelledby (Except where prohibited) aria-live aria-owns aria-relevant aria-roledescription
Name From:	n/a

row role

A row of cells in a tabular container.

Rows contain [cell](#) or [gridcell](#) [elements](#), and thus serve to organize a [table](#), [grid](#), or [treegrid](#).

While the row role can be used in a [table](#), [grid](#), or [treegrid](#), the semantics of [aria-expanded](#), [aria-posinset](#), [aria-setsize](#), and [aria-level](#) are only applicable to the hierarchical structure of an interactive tree grid. Therefore, authors **MUST NOT** apply [aria-expanded](#), [aria-posinset](#), [aria-setsize](#), and [aria-level](#) to a [row](#) that descends from a [table](#) or [grid](#), and user agents **SHOULD NOT** expose any of these four properties to assistive technologies unless the [row](#) descends from a [treegrid](#).

Authors **MUST** ensure [elements](#) with [role](#) row are contained in, or [owned](#) by, an element with the role [table](#), [grid](#), [rowgroup](#), or [treegrid](#).

NOTE: Usage of aria-disabled

While [aria-disabled](#) is currently supported on [row](#), in a future version the working group plans to prohibit its on elements with role [row](#) except when the element is in the context of a [grid](#) or [treegrid](#).

Characteristics:

Characteristic	Value
Superclass Role:	group widget
Base Concept:	<tr> in [HTML]

Characteristic	Value
Required Context Role:	grid rowgroup table treegrid
Required Owned Elements:	cell columnheader gridcell rowheader
Supported States and Properties:	aria-colindex aria-expanded aria-level aria-posinset aria-rowindex aria-setsize aria-selected
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state)

Characteristic	Value
	aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author

rowgroup role

A structure containing one or more row elements in a tabular container.

The rowgroup role establishes a [relationship](#) between [owned row](#) elements. It is a structural equivalent to the tthead, tfoot, and tbody elements in an [HTML table element](#).

Authors **MUST** ensure [elements](#) with [role](#) rowgroup are contained in, or [owned](#) by, an element with the role [grid](#), [table](#), or [treegrid](#).

NOTE

The rowgroup role exists, in part, to support role symmetry in [HTML](#), and allows for the propagation of presentation inheritance on [HTML](#) table elements with an explicit presentation role applied.

NOTE

This role does not differentiate between types of row groups (e.g., tthead vs. tbody), but an issue has been raised for [WAI-ARIA 2.0](#).

Characteristics:

Characteristic	Value
Superclass Role:	structure
Base Concept:	<tbody>, <tfoot> and <thead>in [HTML]
Required Context Role:	grid table treegrid
Required Owned Elements:	row
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns

Characteristic	Value
	aria-relevant aria-roledescription
Name From:	author

rowheader role

A cell containing header information for a row.

The [rowheader](#) role can be used to identify a cell as a header for a row in a [table](#), [grid](#), or [treegrid](#). The rowheader establishes a [relationship](#) between it and all cells in the corresponding row. It is a structural equivalent to setting `scope="row"` on an [HTML](#) [th](#) [element](#).

Authors **MUST** ensure [elements](#) with [role](#) rowheader are contained in, or [owned](#) by, an element with the role [row](#).

Applying the [aria-selected](#) state on a rowheader **MUST NOT** cause the user agent to automatically propagate the [aria-selected](#) state to all the cells in the corresponding row. An author **MAY** choose to propagate selection in this manner depending on the specific application.

While the rowheader role can be used in both interactive grids and non-interactive tables, the use of [aria-expanded](#), [aria-readonly](#), and [aria-required](#) is only applicable to interactive elements. Therefore, authors **SHOULD NOT** use [aria-expanded](#), [aria-readonly](#), or [aria-required](#) in a rowheader that descends from a [table](#), and user agents **SHOULD NOT** expose these properties to [assistive technologies](#) unless the rowheader descends from a [grid](#) or [treegrid](#).

NOTE: Usage of aria-disabled

While [aria-disabled](#) is currently supported on [rowheader](#), in a future version the working group plans to prohibit its use on elements with role [rowheader](#) except when the element is in the context of a [grid](#) or [treegrid](#).

Characteristics:

Characteristic	Value
Superclass Role:	cell gridcell sectionhead
Base Concept:	<th[scope="row"]> in [HTML]

Characteristic	Value
Required Context Role:	row
Supported States and Properties:	aria-expanded aria-sort
Inherited States and Properties:	aria-atomic aria-busy (state) aria-colindex aria-colspan aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage aria-flowto aria-grabbed (state) aria-haspopup aria-hidden (state) aria-invalid (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-readonly aria-relevant aria-required aria-roledescription aria-rowindex

Characteristic	Value
	aria-rowspan aria-selected (state)
Name From:	contents author
Accessible Name Required:	True

scrollbar role

A graphical object that controls the scrolling of content within a viewing area, regardless of whether the content is fully displayed within the viewing area.

A scrollbar represents the current value and range of possible values via the size of the scrollbar and position of the thumb with respect to the visible range of the orientation (horizontal or vertical) it controls. Its orientation represents the orientation of the scrollbar and the scrolling effect on the viewing area controlled by the scrollbar. It is typically possible to add or subtract to the current value by using directional keys such as arrow keys.

Authors **MUST** set the [aria-controls](#) attribute on the scrollbar element to reference the scrollable area it controls.

Authors **MAY** set [aria-valuemin](#) and [aria-valuemax](#) to indicate the minimum and maximum thumb position. Otherwise, their implicit values follow the same rules as `<input [ type="range" ]>` in [HTML]:

- If `aria-valuemin` is missing or not a [number](#), it defaults to 0 (zero).
- If `aria-valuemax` is missing or not a [number](#), it defaults to 100.

Authors **MUST** set the [aria-valuenow](#) attribute to indicate the current thumb position. If `aria-valuenow` is missing or has an unexpected value, browsers **MAY** implement the repair techniques specified in the [section describing handling author errors in states and properties](#), which are equivalent to the repair techniques for `<input [ type="range" ]>` in [HTML].

Elements with the role `scrollbar` have an implicit [aria-orientation](#) value of `vertical`.

NOTE

Assistive technologies generally will render the value of [aria-valuenow](#) as a percent of a range between the value of [aria-valuemin](#) and [aria-valuemax](#), unless [aria-valuetext](#) is specified. It is best to set the values for [aria-valuemin](#), [aria-valuemax](#), and [aria-valuenow](#) in a manner that is appropriate for this calculation.

Characteristics:

Characteristic	Value
Superclass Role:	range widget
Required States and Properties:	aria-controls aria-valuenow
Supported States and Properties:	aria-disabled aria-orientation aria-valuemax aria-valuemin
Inherited States and Properties:	aria-atomic aria-busy (state) aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-valuetext
Name From:	author
Accessible Name Required:	False
Children Presentational:	True
Implicit Value for Role:	Default for aria-orientation is vertical. Default for aria-valuemin is 0. Default for aria-valuemax is 100.

search role

A [landmark](#) region that contains a collection of items and objects that, as a whole, combine to create a search facility. See related [form](#) and [searchbox](#).

A search region may be a mix of host language form controls, scripted controls, and hyperlinks.

User agents **SHOULD** treat elements with the role of search as navigational [landmarks](#).

Characteristics:

Characteristic	Value
Superclass Role:	landmark
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details

Characteristic	Value
	aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

searchbox role

A type of textbox intended for specifying search criteria. See related [textbox](#) and [search](#).

Characteristics:

Characteristic	Value
Superclass Role:	textbox
Base Concept:	<input [type="search"]> in [HTML]
Inherited States and Properties:	aria-activedescendant aria-atomic

Characteristic	Value
	aria-autocomplete aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage aria-flowto aria-grabbed (state) aria-haspopup aria-hidden (state) aria-invalid (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-multiline aria-owns aria-placeholder aria-readonly aria-relevant aria-required aria-roledescription
Name From:	author
Accessible Name Required:	True

section role

A renderable structural containment unit in a document or application.

NOTE

section is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	structure
Subclass Roles:	alert blockquote caption cell code definition deletion emphasis figure group img insertion landmark list listitem log marquee math note paragraph status

Characteristic	Value
	strong subscript superscript table tabpanel term time tooltip
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live

Characteristic	Value
	aria-owns aria-relevant aria-roledescription
Name From:	n/a

sectionhead role

A structure that labels or summarizes the topic of its related section.

NOTE

sectionhead is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	structure
Subclass Roles:	columnheader heading rowheader tab
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	contents author

select role

A form widget that allows the user to make selections from a set of choices.

NOTE

`select` is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	composite group
Subclass Roles:	listbox

Characteristic	Value
	menu radiogroup tree
Supported States and Properties:	aria-orientation
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

separator role

A divider that separates and distinguishes sections of content or groups of menuitems.

There are two types of separators: a static [structure](#) that provides only a visible boundary and a focusable, interactive [widget](#) that is also moveable. If a separator is not focusable, it is revealed to [assistive technologies](#) as a static structural element. For example, a static separator can be used to help visually divide two groups of menu items in a menu or to provide a horizontal rule between two sections of a page.

Authors **MAY** make a separator focusable to create a [widget](#) that both provides a visible boundary between two sections of content and enables the user to change the relative size of the sections by changing the position of the separator. A variable separator widget can be moved continuously within a range, whereas a fixed separator widget supports only two discrete positions. Typically, a fixed separator widget is used to toggle one of the sections between expanded and collapsed states.

If the separator is focusable, authors **MUST** set the value of [aria-valuenow](#) to a [number](#) reflecting the current position of the separator and update that value when it changes. Authors **SHOULD** also provide the value of [aria-valuemin](#) if it is not 0 and the value of [aria-valuemax](#) if it is not 100. If missing or not a number, the implicit values of these attributes are as follows:

- The implicit value of `aria-valuemin` is 0.
- The implicit value of `aria-valuemax` is 100.

In applications where there is more than one focusable separator, authors **SHOULD** provide an accessible name for each one.

Elements with the role separator have an implicit [aria-orientation](#) value of horizontal.

Characteristics:

Characteristic	Value
Superclass Role:	structure (if not focusable) widget (if focusable)
Related Concepts:	<hr> in [HTML]
Required States and Properties:	aria-valuenow (if focusable)

Characteristic	Value
Supported States and Properties:	aria-disabled (if focusable) aria-orientation aria-valuemax (if focusable) aria-valuemin (if focusable) aria-valuetext (if focusable)
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Children Presentational:	True

Characteristic	Value
Implicit Value for Role:	Default for aria-orientation is horizontal. Default for aria-valuemin is 0. Default for aria-valuemax is 100.

slider role

An input where the user selects a value from within a given range.

A slider represents the current value and range of possible values via the size of the slider and position of the thumb. It is typically possible to add or subtract to the value by using directional keys such as arrow keys.

Authors **MAY** set the [aria-valuemin](#) and [aria-valuemax](#) attributes. Otherwise, their implicit values follow the same rules as `<input [type="range"]>` in [HTML]:

- If [aria-valuemin](#) is missing or not a [number](#), it defaults to 0 (zero).
- If [aria-valuemax](#) is missing or not a [number](#), it defaults to 100.

Authors **MUST** set the [aria-valuenow](#) attribute. If [aria-valuenow](#) is missing or has an unexpected value, browsers **MAY** implement the repair techniques specified in the [section describing handling author errors in states and properties](#), which are equivalent to the repair techniques for `<input [type="range"]>` in [HTML].

Elements with the role `slider` have an implicit [aria-orientation](#) value of `horizontal`.

Characteristics:

Characteristic	Value
Superclass Role:	input range
Required States and Properties:	aria-valuenow
Supported States and Properties:	aria-errormessage aria-haspopup aria-invalid aria-orientation aria-readonly aria-valuemax

Characteristic	Value
	aria-valuemin
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription aria-valuetext
Name From:	author
Accessible Name Required:	True
Children Presentational:	True
Implicit Value for Role:	Default for aria-orientation is horizontal. Default for aria-valuemin is 0. Default for aria-valuemax is 100.

spinbutton role

A form of [range](#) that expects the user to select from among discrete choices.

A `spinbutton` typically allows users to change its displayed value by activating increment and decrement buttons that step through a set of allowed values. Some implementations display the value in an text field that allows editing and typing but typically limits input in ways that help prevent invalid values.

Although a `spinbutton` is similar in appearance to many presentations of `select`, it is advisable to use `spinbutton` when working with known ranges (especially in the case of large ranges) as opposed to distinct options. For example, a `spinbutton` representing a range from 1 to 1,000,000 would provide much better performance than a `select` [widget](#) representing the same values.

Authors **MAY** create a `spinbutton` with children or owned elements, but **MUST** limit those elements to a [textbox](#) and/or two [buttons](#). Alternatively, authors **MAY** apply the [spinbutton](#) role to a text input and create sibling buttons to support the increment and decrement functions.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#). When a `spinbutton` receives focus, authors **SHOULD** ensure focus is placed on the [textbox](#) element if one is present, and on the `spinbutton` itself otherwise. Authors **SHOULD** also ensure the up and down arrows on a keyboard perform the increment and decrement functions and that the increment and decrement [button](#) elements are **NOT** included in the primary navigation ring, e.g., the Tab ring in [HTML](#).

Authors **SHOULD** set the [aria-valuenow](#) attribute when the [spinbutton](#) has a value. Authors **SHOULD** set the [aria-valuemin](#) attribute when there is a minimum value, and the [aria-valuemax](#) attribute when there is a maximum value.

Characteristics:

Characteristic	Value
Superclass Role:	composite input range
Supported States and Properties:	aria-errormessage aria-invalid aria-readonly aria-required aria-valuemax aria-valuemin aria-valuenow aria-valuetext

Characteristic	Value
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True
Implicit Value for Role:	Default for aria-valuemin is that there is no minimum value. Default for aria-valuemax is that there is no maximum value. Default for aria-valuenow is 0

status role

A type of [live region](#) whose content is advisory information for the user but is not important enough to justify an [alert](#), often but not necessarily presented as a status bar.

Authors **SHOULD** ensure an element with role status does not receive focus as a result of change in status.

Status is a form of [live region](#). If another part of the page controls what appears in the status, authors **SHOULD** make the [relationship](#) explicit with the [aria-controls](#) [attribute](#).

[Assistive technologies](#) **MAY** reserve some cells of a Braille display to render the status.

Elements with the role status have an implicit [aria-live](#) value of polite and an implicit [aria-atomic](#) value of true.

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	timer
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state)

Characteristic	Value
	aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-live is polite. Default for aria-atomic is true.

strong role

Content that is important, serious, or urgent. See related [emphasis](#).

The purpose of the strong role is to communicate strong importance, seriousness, or urgency. It is not for communicating changes in typographical presentation that are not important to the meaning of the content. Authors **SHOULD** use the strong role only if its absence would change the meaning of the content.

The strong role is not intended to convey stress or emphasis; for that purpose, the [emphasis](#) role is more appropriate.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	 in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

structure role

A document structural [element](#).

[Roles](#) for document structure support the accessibility of dynamic web content by helping [assistive technologies](#) determine active content versus static document content. Structural roles by themselves do not all map to [accessibility APIs](#), but are used to create [widget](#) roles or assist content adaptation for assistive technologies.

NOTE

`structure` is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	roletype
Subclass Roles:	application document generic presentation range rowgroup section sectionhead separator
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	n/a

subscript role

One or more subscripted characters. See related [superscript](#).

The subscript role is intended to be used only to mark up typographical conventions that have specific meanings; not for typographical presentation for presentation's sake. In general, authors **SHOULD** use this role only if the absence of the subscript would change the meaning of the content.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<sub> and <sup> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

superscript role

One or more superscripted characters. See related [superscript](#).

The superscript role is intended to be used only to mark up typographical conventions that have specific meanings; not for typographical presentation for presentation's sake. In general, authors **SHOULD** use this role only if the absence of the superscript would change the meaning of the content.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<sub> and <sup> in [HTML]
Inherited States and Properties:	aria-atomic

Characteristic	Value
	aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-live aria-owns aria-relevant aria-roledescription
Prohibited States and Properties:	aria-label aria-labelledby
Name From:	prohibited

switch role

A type of checkbox that represents on/off values, as opposed to checked/unchecked values. See related [checkbox](#).

The [aria-checked attribute](#) of a switch indicates whether the input is on (true) or off (false). The mixed value is invalid, and user agents **MUST** treat a mixed value as equivalent to false for this role.

NOTE

A switch provides approximately the same functionality as a checkbox and toggle button, but makes it possible for assistive technologies to present the widget in a fashion consistent with its on-screen appearance.

Characteristics:

Characteristic	Value
Superclass Role:	checkbox
Related Concepts:	button
Required States and Properties:	aria-checked
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage aria-expanded (state) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-owns aria-readonly aria-relevant aria-required aria-roledescription
Name From:	contents author
Accessible Name Required:	True
Children Presentational:	True

tab role

A grouping label providing a mechanism for selecting the tab content that is to be rendered to the user.

If a [tabpanel](#) or item in a [tabpanel](#) has focus, the associated **tab** is the currently active tab in the [tablist](#), as defined in [Managing Focus](#). [tablist](#) elements, which contain a set of associated [tab](#) elements, are typically placed near a series of [tabpanel](#) elements, usually preceding it. See the [WAI-ARIA Authoring Practices](#) for details on implementing a tab set design pattern.

Authors **MUST** ensure [elements](#) with [role tab](#) are contained in, or [owned](#) by, an element with the role [tablist](#).

Authors **SHOULD** ensure the [tabpanel](#) associated with the currently active tab is [perceivable](#) to the user.

For a single-selectable [tablist](#), authors **SHOULD** hide other [tabpanel](#) [elements](#) from the user until the user selects the tab associated with that [tabpanel](#). For a multi-selectable [tablist](#), authors **SHOULD** ensure that the [tab](#) for each visible [tabpanel](#) has the [aria-expanded attribute](#) set to `true`, and that the tabs associated with the remaining hidden [tabpanel](#) elements have their [aria-expanded](#) attributes set to `false`.

In either case, authors **SHOULD** ensure that a selected tab has its [aria-selected](#) attribute set to `true`, that inactive tab elements have their [aria-selected](#) attribute set to `false`, and that the currently selected tab provides a visual indication that it is selected. In the absence of an [aria-selected](#) attribute on the current tab, [user agents](#) **SHOULD** indicate to [assistive technologies](#) through the platform [accessibility API](#) that the currently focused tab is selected.

Characteristics:

Characteristic	Value
Superclass Role:	sectionhead widget
Required Context Role:	tablist
Supported States and Properties:	aria-disabled aria-expanded aria-haspopup aria-posinset aria-selected aria-setsize
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns

Characteristic	Value
	aria-relevant aria-roledescription
Name From:	contents author
Children Presentational:	True
Implicit Value for Role:	Default for aria-selected is false.

table role

A [section](#) containing data arranged in rows and columns. See related [grid](#).

The `table` role is intended for tabular containers which are not interactive. If the tabular container maintains a selection state, provides its own two-dimensional navigation, or allows the user to rearrange or otherwise manipulate its contents or the display thereof, authors **SHOULD** use [grid](#) or [treegrid](#) instead.

Authors **SHOULD** prefer the use of the host language's semantics for table whenever possible, such as the `<table>` element in [HTML](#).

Characteristics:

Characteristic	Value
Superclass Role:	section
Subclass Roles:	grid
Base Concept:	<code><table></code> in HTML
Required Owned Elements:	row rowgroup , row
Supported States and Properties:	aria-colcount aria-rowcount
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

tablist role

A list of [tab elements](#), which are references to [tabpanel](#) elements.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

For a single-selectable tablist, authors **SHOULD** hide other [tabpanel elements](#) from the user until the user selects the tab associated with that tabpanel. For a multi-selectable [tablist](#), authors **SHOULD** ensure

each visible `tabpanel` has its `aria-expanded` attribute set to `true`, and that the remaining hidden `tabpanel` elements have their `aria-expanded` attributes set to `false`.

`tablist` elements are typically placed near usually preceding, a series of `tabpanel` elements. See the [WAI-ARIA Authoring Practices](#) for details on implementing a tab set design pattern.

Elements with the role `tablist` have an implicit `aria-orientation` value of `horizontal`.

Characteristics:

Characteristic	Value
Superclass Role:	composite
Required Owned Elements:	tab
Supported States and Properties:	aria-multiselectable aria-orientation
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts

Characteristic	Value
	aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-orientation is horizontal.

tabpanel role

A container for the resources associated with a [tab](#), where each [tab](#) is contained in a [tablist](#).

Authors **SHOULD** associate a `tabpanel` [element](#) with its [tab](#), either by using the [aria-controls](#) attribute on the tab to reference the tab panel, or by using the [aria-labelledby](#) attribute on the tab panel to reference the tab.

[tablist](#) elements are typically placed near, usually preceding, a series of [tabpanel](#) elements. See the [WAI-ARIA Authoring Practices](#) for details on implementing a tab set design pattern.

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect

Characteristic	Value
	aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

term role

A word or phrase with a corresponding definition. See related [definition](#).

The `term` role is used to explicitly identify a word or phrase for which a [definition](#) has been provided by the author or is expected to be provided by the user.

Authors **SHOULD NOT** use the `term` role on interactive elements such as links because doing so could prevent users of [assistive technologies](#) from interacting with those elements.

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<dfn> in [HTML]

Characteristic	Value
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

textbox role

A type of input that allows free-form text as its value.

If the [aria-multiline attribute](#) is true, the [widget](#) accepts line breaks within the input, as in an [HTML](#) [textarea](#). Otherwise, this is a simple text box. The intended use is for languages that do not have a text input [element](#), or cases in which an element with different [semantics](#) is repurposed as a text field.

NOTE

In most user agent implementations, the default behavior of the ENTER or RETURN key is different between the single-line and multi-line text fields in [HTML](#). When user has focus in a single-line `<input type="text">` element, the keystroke usually submits the form. When user has focus in a multi-line `<textarea>` element, the keystroke inserts a line break. The [WAI-ARIA](#) `textbox` role differentiates these types of boxes with the [aria-multiline](#) attribute, so authors are advised to be aware of this distinction when designing the field.

Characteristics:

Characteristic	Value
Superclass Role:	input
Subclass Roles:	searchbox
Related Concepts:	<code><textarea></code> in [HTML] <code><input[type="text"]></code> in [HTML]
Supported States and Properties:	aria-activedescendant aria-autocomplete aria-errormessage aria-haspopup aria-invalid aria-multiline aria-placeholder aria-readonly aria-required
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True

time role

An element that represents a specific point in time.

NOTE

At the present time, there are no WAI-ARIA properties corresponding to the `datetime` attribute supported on `<time>` in [HTML]. The addition of this property will be considered for ARIA version 1.3.

Authors **SHOULD** limit text contents to a valid date- or time-related string, or apply this future `datetime`-equivalent property to the element which has role `time`.

Examples of valid date- or time-related strings as text contents of an element with the `time` role:

- A valid month string: 2019-11
- A valid date string: 2019-11-18
- A valid yearless date string: 11-18
- A valid time string: 09:54:39

- A valid floating date and time string: 2019-11-18T14:54
- A valid time-zone offset string: -08:00
- A valid global date and time string: 2019-11-18T14:54Z
- A valid week string: 2019-W47
- Four or more ASCII digits, at least one of which is not U+0030 DIGIT ZERO (0): 0001
- A valid duration string: 4h 18m 3s

Characteristics:

Characteristic	Value
Superclass Role:	section
Related Concepts:	<time> in [HTML]
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

timer role

A type of [live region](#) containing a numerical counter which indicates an amount of elapsed time from a start point, or the time remaining until an end point.

The text contents of the timer [object](#) indicate the current time measurement, and are updated as that amount changes. The timer value is not necessarily machine parsable, but authors **SHOULD** update the text contents at fixed intervals, except when the timer is paused or reaches an end-point.

Elements with the role `timer` have an implicit [aria-live](#) value of `off`.

Characteristics:

Characteristic	Value
Superclass Role:	status
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state)

Characteristic	Value
	aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

toolbar role

A collection of commonly used function buttons or controls represented in compact visual form.

The toolbar is often a subset of functions found in a [menubar](#), designed to reduce user effort in using these functions. Authors **MUST** supply a label on each toolbar when the application contains more than one toolbar.

Authors **MAY** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

Elements with the role toolbar have an implicit [aria-orientation](#) value of horizontal.

Characteristics:

Characteristic	Value
Superclass Role:	group
Related Concepts:	menubar
Supported States and Properties:	aria-orientation
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state)

Characteristic	Value
	aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author
Implicit Value for Role:	Default for aria-orientation is horizontal.

tooltip role

A contextual popup that displays a description for an element.

The tooltip typically becomes visible, after a short delay, in response to a mouse hover, or after the owning element receives keyboard focus. The use of a WAI-ARIA tooltip is a supplement to the normal tooltip behavior of the user agent.

NOTE

Typical tooltip delays last from one to five seconds.

Authors **SHOULD** ensure that elements with the [role](#) tooltip are referenced through the use of [aria-describedby](#) before or at the time the tooltip is displayed.

Characteristics:

Characteristic	Value
Superclass Role:	section
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns

Characteristic	Value
	aria-relevant aria-roledescription
Name From:	contents author
Accessible Name Required:	True

tree role

A [widget](#) that allows the user to select one or more items from a hierarchically organized collection.

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

Elements with the role tree have an implicit [aria-orientation](#) value of vertical.

Characteristics:

Characteristic	Value
Superclass Role:	select
Subclass Roles:	treegrid
Required Owned Elements:	group , treeitem treeitem
Supported States and Properties:	aria-errormessage aria-invalid aria-multiselectable aria-required
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) aria-dropeffect aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-keyshortcuts aria-label aria-labelledby aria-live aria-orientation aria-owns aria-relevant aria-roledescription
Name From:	author
Accessible Name Required:	True
Implicit Value for Role:	Default for aria-orientation is vertical.

treegrid role

A [grid](#) whose rows can be expanded and collapsed in the same manner as for a [tree](#).

If [aria-readonly](#) is set on an [element](#) with [role](#) treegrid, [user agents](#) **MUST** propagate the value to all [gridcell](#) elements owned by the treegrid and expose the value in the accessibility API. An author **MAY** override the propagated value of [aria-readonly](#) for an individual [gridcell](#) element.

When the [aria-readonly](#) attribute is applied to a focusable [gridcell](#), it indicates whether the content contained in the [gridcell](#) is editable. The [aria-readonly](#) attribute does not represent availability of functions for navigating or manipulating the treegrid itself.

In a treegrid that provides content editing functions, if the content of a focusable [gridcell](#) element is not editable, authors **MAY** set [aria-readonly](#) to true on the [gridcell](#) element. However, if a

treegrid presents a collection of elements that do not support [aria-readonly](#), such as a collection of [link](#) elements, it is not necessary for the author to specify a value for [aria-readonly](#).

To be [keyboard accessible](#), authors **SHOULD** manage focus of descendants for all instances of this [role](#), as described in [Managing Focus](#).

Characteristics:

Characteristic	Value
Superclass Role:	grid tree
Required Owned Elements:	row rowgroup ... row
Inherited States and Properties:	aria-activedescendant aria-atomic aria-busy (state) aria-colcount aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) aria-keyshortcuts aria-label aria-labelledby

Characteristic	Value
	aria-live aria-multiselectable aria-orientation aria-owns aria-readonly aria-relevant aria-required aria-roledescription aria-rowcount
Name From:	author
Accessible Name Required:	True

treeitem role

An option item of a [tree](#). This is an [element](#) within a tree that may be expanded or collapsed if it contains a sub-level group of tree item elements.

A collection of `treeitem` elements to be expanded and collapsed are enclosed in an element with the [group](#) [role](#).

Authors **MUST** ensure [elements](#) with [role](#) `treeitem` are contained in, or [owned](#) by, an element with the role [group](#) or [tree](#).

Characteristics:

Characteristic	Value
Superclass Role:	listitem option
Required Context Role:	group tree
Supported States and Properties:	aria-expanded aria-haspopup
Inherited States and Properties:	aria-atomic

Characteristic	Value
	aria-busy (state) aria-checked (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-level aria-live aria-owns aria-posinset aria-relevant aria-roledescription aria-selected (state) (required) aria-setsize
Name From:	contents author
Accessible Name Required:	True

widget role

An interactive component of a graphical user interface (GUI).

Widgets are discrete user interface objects with which the user can interact. Widget [roles](#) map to standard features in [accessibility APIs](#). When the user navigates an element assigned any of the non-abstract subclass roles of `widget`, [assistive technologies](#) that typically intercept standard keyboard events **SHOULD** switch to an application browsing mode, and pass keyboard events through to the web application. The intent is to hint to certain [assistive technologies](#) to switch from normal browsing mode into a mode more appropriate for interacting with a web application; some [user agents](#) have a browse navigation mode where keys, such as up and down arrows, are used to browse the document, and this native behavior prevents the use of these keys by a web application.

NOTE

`widget` is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	roletype
Subclass Roles:	command composite gridcell input progressbar row scrollbar separator tab
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby

Characteristic	Value
	aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2) aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	n/a

window role

A browser or application window.

[Elements](#) with this [role](#) have a window-like behavior in a graphical user interface (GUI) context, regardless of whether they are implemented as a native window in the operating system, or merely as a section of the document styled to look like a window.

NOTE

In the description of this role, the term "application" does not refer to the [application](#) role, which specifies specific assistive technology behaviors.

NOTE

window is an abstract role used for the ontology. Authors should not use this role in content.

Characteristics:

Characteristic	Value
Is Abstract:	True
Superclass Role:	roletype
Subclass Roles:	dialog
Supported States and Properties:	aria-modal
Inherited States and Properties:	aria-atomic aria-busy (state) aria-controls aria-current (state) aria-describedby aria-details aria-disabled (state) (deprecated on this role in ARIA 1.2) aria-dropeffect aria-errormessage (deprecated on this role in ARIA 1.2) aria-flowto aria-grabbed (state) aria-haspopup (deprecated on this role in ARIA 1.2) aria-hidden (state) aria-invalid (state) (deprecated on this role in ARIA 1.2)

Characteristic	Value
	aria-keyshortcuts aria-label aria-labelledby aria-live aria-owns aria-relevant aria-roledescription
Name From:	author

§ 6. Supported States and Properties

§ 6.1 Clarification of States versus Properties

The terms "states" and "properties" refer to similar features. Both provide specific information about an [object](#), and both form part of the definition of the nature of [roles](#). In this document, states and properties are both treated as aria-prefixed markup [attributes](#). However, they are maintained conceptually distinct to clarify subtle differences in their meaning. One major difference is that the values of properties (such as [aria-labelledby](#)) are often less likely to change throughout the application life-cycle than the values of states (such as [aria-checked](#)) which may change frequently due to user interaction. Note that the frequency of change difference is not a rule; a few properties, such as [aria-valuetext](#) are expected to change often. Because the distinction between states and properties is of little consequence to most web content authors, this specification refers to both "states" and "properties" simply as "attributes" whenever possible. See the definitions of *state* and *property* for more information.

§ 6.2 Characteristics of States and Properties

States and properties have the characteristics described in the following sections.

§ 6.2.1 Related Concepts

Advisory information about features from this or other languages that correspond to this [state](#) or [property](#). While the correspondence may not be exact, it is useful to help understand the intent of the state or property.

§ 6.2.2 Used in Roles

Advisory information about [roles](#) that use this [state](#) or [property](#). This information is provided to help understand the appropriate usage of the state or property. Use of a given state or property is not defined when used on roles other than those listed.

§ 6.2.3 Inherits into Roles

Advisory information about [roles](#) that inherit the [state](#) or [property](#) from an ancestor role.

§ 6.2.4 Value

Value type of the [state](#) or [property](#). The value may be one of the following types:

true/false

Value representing either true or false. The default value for this value type is false unless otherwise specified.

tristate

Value representing true, false, mixed, or undefined values. The default value for this value type is undefined unless otherwise specified.

true/false/undefined

Value representing true, false, or undefined (not applicable). The default value for this value type is undefined unless otherwise specified. For example, an element with [aria-expanded](#) set to false is not currently expanded; an element with [aria-expanded](#) set to undefined is not expandable.

ID reference

Reference to the ID of another [element](#) in the same document

ID reference list

A list of one or more ID references.

integer

A numerical value without a fractional component.

number

Any real numerical value.

string

Unconstrained value type.

token

One of a limited set of allowed values. The default value is defined in each attribute's Values table, as specified in the [Attribute Values](#) section.

token list

A list of one or more tokens.

These are generic types for states and properties, but do not define specific representation. See [State and Property Attribute Processing](#) for details on how these values are expressed and handled in host languages.

§ 6.3 ARIA Attributes

§ 6.3.1 Multi-value Attribute Values

When the ARIA attribute definition includes a table listing the attribute's allowed values, that attribute is a multi-value nullable attribute. Each value in the table is a keyword for the attribute, mapping to a state of the same name.

§ 6.3.2 IDL reflection of ARIA attributes

All ARIA attributes reflect in IDL as [nullable DOMString](#) attributes. This includes the boolean-like [true/false](#) type, and all other ARIA attributes.

Default values from the ARIA values tables **MUST NOT** reflect to IDL as the [missing value default](#) or the [invalid value default](#) for the attribute. On getting, a missing ARIA attribute will return `null`. ARIA attributes are not validated on get. If an ARIA value is invalid, on getting, it will return its set value as a literal string, and will not return an invalid value default.

§ 6.3.3 Operating System Accessibility API mapping of multi-value ARIA attributes

Unlike IDL reflection, operating system accessibility API mappings of ARIA attributes can have defaults. Any default values from the ARIA values tables are exposed to the operating system accessibility API as described in [5.2.3 Supported States and Properties](#), and in [Core Accessibility API Mappings 1.1](#).

§ 6.3.4 ARIA nullable DOMString Attributes

As noted in [A. Mapping WAI-ARIA Value types to languages](#), attributes are included in host languages, and the syntax for representation of WAI-ARIA types is governed by the host language.

The following algorithm should be used for ARIA nullable [DOMString](#) attributes in [HTML](#):

On getting, if the corresponding content attribute is not present, then the IDL attribute must return null, otherwise, the IDL attribute must get the value in a transparent, case-preserving manner. On setting, if the new value is null, the content attribute must be removed, and otherwise, the content attribute must be set to the specified new value in a transparent, case-preserving manner.

NOTE

Note: As of ARIA 1.2, all ARIA attributes exposed via IDL are defined as nullable [DOMStrings](#). This matches the current implementation of all major rendering engines. This specification change should result in no implementation changes; it will merely represent the current reality of web engines. However, in a future draft, the ARIA Working Group intends to change several ARIA attributes to non-nullable DOMStrings, and seek implementations. The proposed change will bring ARIA into alignment with the [HTML](#)'s usage of [enumerated attributes](#).

§ 6.3.4.1 Example Attribute Usage

This section is non-normative.

EXAMPLE 14

```
// HTML hidden="" example (not aria-hidden="true")
// Actual boolean type; defaults to false.

// Note: Actual boolean assignment and return value.
el.hidden = true;
el.hidden; // true

// Removal of content attribute results in missing value default: boolean false.
el.removeAttribute("hidden");
el.hidden; // false
```

EXAMPLE 15

```
// aria-busy example
// true/false ~ boolean-like nullable string; returns null unless set

el.ariaBusy; // null

// Note: String assignment and return value.
el.ariaBusy = "true";
el.ariaBusy; // "true"

// Removal of content attribute results in missing value default: string "false".
el.removeAttribute("aria-busy");
el.ariaBusy; // null

// Assignment of invalid "busy" value. Not validated on set or get and the literal string
el.setAttribute("aria-busy", "busy");
el.ariaBusy; // "busy"
```

EXAMPLE 16

```
// aria-pressed example
// Tristate ~ true/false/mixed/undefined string; null if unspecified

// no value has been defined
button.ariaPressed; // null

// A value of "true", "false", or "mixed" for aria-pressed on a button denotes a toggle button
button.setAttribute("aria-pressed", "true"); // Content attribute assignment.
button.ariaPressed; // "true"
button.ariaPressed = "false"; // DOM property assignment.
button.ariaPressed; // "false"

// Assignment of invalid "foo" value. Not validated on set or get and the literal string value is returned
button.ariaPressed = "foo";
button.ariaPressed; // "foo" (Note: button is no longer a toggle button.)

// Removal of content attribute results in a null value
button.removeAttribute("aria-pressed");
button.ariaPressed; // null
```

§ 6.4 Translatable States and Properties

The [HTML](#) specification states that other specifications can define [translatable attributes](#). In order to be understandable by assistive technology users, the values of the following [states](#) and [properties](#) are [translatable attributes](#) and should be translated when a page is localized:

- [aria-label](#)
- [aria-placeholder](#)
- [aria-roledescription](#)
- [aria-valuetext](#)

§ 6.5 Global States and Properties

Some [states](#) and [properties](#) are applicable to all host language [elements](#) regardless of whether a [role](#) is applied. The following global states and properties are supported by all roles and by all base markup elements unless otherwise prohibited. If a role prohibits use of any global states or properties, those states or properties are listed as prohibited in the characteristics table included in the section that defines the role.

- [aria-atomic](#)
- [aria-busy \(state\)](#)
- [aria-controls](#)
- [aria-current \(state\)](#)
- [aria-describedby](#)
- [aria-details](#)
- [aria-disabled \(state\)](#) (Global use deprecated in ARIA 1.2)
- [aria-dropeffect](#)
- [aria-errormessage](#) (Global use deprecated in ARIA 1.2)
- [aria-flowto](#)
- [aria-grabbed \(state\)](#)
- [aria-haspopup](#) (Global use deprecated in ARIA 1.2)
- [aria-hidden \(state\)](#)
- [aria-invalid \(state\)](#) (Global use deprecated in ARIA 1.2)
- [aria-keyshortcuts](#)
- [aria-label](#) (Except where prohibited)
- [aria-labelledby](#) (Except where prohibited)
- [aria-live](#)
- [aria-owns](#)
- [aria-relevant](#)
- [aria-roledescription](#)

§ 6.6 Taxonomy of WAI-ARIA States and Properties

States and properties are categorized as follows:

1. [Widget Attributes](#)
2. [Live Region Attributes](#)
3. [Drag-and-Drop Attributes](#)
4. [Relationship Attributes](#)

§ 6.6.1 Widget Attributes

This section contains [attributes](#) specific to common user interface [elements](#) found on GUI systems or in rich internet applications which receive user input and process user actions. These attributes are used to support the [widget roles](#).

- [aria-autocomplete](#)
- [aria-checked](#)
- [aria-disabled](#)
- [aria-errormessage](#)
- [aria-expanded](#)
- [aria-haspopup](#)
- [aria-hidden](#)
- [aria-invalid](#)
- [aria-label](#)
- [aria-level](#)
- [aria-modal](#)
- [aria-multiline](#)
- [aria-multiselectable](#)
- [aria-orientation](#)
- [aria-placeholder](#)
- [aria-pressed](#)
- [aria-readonly](#)
- [aria-required](#)
- [aria-selected](#)
- [aria-sort](#)
- [aria-valuemax](#)
- [aria-valuemin](#)
- [aria-valuenow](#)
- [aria-valuetext](#)

Widget attributes might be mapped by a [user agent](#) to platform [accessibility API state](#), for access by [assistive technologies](#), or they might be accessed directly from the [DOM](#). User agents **MUST** provide a way for assistive technologies to be notified when states change, either through [DOM](#) attribute change [events](#) or platform accessibility [API](#) events.

§ 6.6.2 Live Region Attributes

This section contains [attributes](#) specific to [live regions](#) in rich internet applications. These attributes may be applied to any [element](#). The purpose of these attributes is to indicate that content changes may occur without the element having focus, and to provide [assistive technologies](#) with information on how to process those content updates. Some [roles](#) specify a default value for the [aria-live](#) attribute specific to that role. An example of a live region is a ticker section that lists updating stock quotes.

- [aria-atomic](#)
- [aria-busy](#)
- [aria-live](#)
- [aria-relevant](#)

§ 6.6.3 Drag-and-Drop Attributes

This section lists [attributes](#) which indicate information about drag-and-drop interface [elements](#), such as draggable elements and their drop targets. Drop target information will be rendered visually by the author and provided to [assistive technologies](#) through an alternate modality.

- [aria-dropeffect](#)
- [aria-grabbed](#)

§ 6.6.4 Relationship Attributes

This section lists [attributes](#) that indicate [relationships](#) or associations between [elements](#) which cannot be readily determined from the document structure.

- [aria-activedescendant](#)
- [aria-colcount](#)

- [aria-colindex](#)
- [aria-colspan](#)
- [aria-controls](#)
- [aria-describedby](#)
- [aria-details](#)
- [aria-errormessage](#)
- [aria-flowto](#)
- [aria-labelledby](#)
- [aria-owns](#)
- [aria-posinset](#)
- [aria-rowcount](#)
- [aria-rowindex](#)
- [aria-rowspan](#)
- [aria-setsize](#)

§ 6.7 Definitions of States and Properties (all aria-* attributes)

Below is an alphabetical list of WAI-ARIA [states](#) and [properties](#) to be used by rich internet application authors. A detailed definition of each WAI-ARIA state and [property](#) follows this compact list.

[aria-activedescendant](#)

Identifies the currently active element when DOM focus is on a [composite](#) widget, [combobox](#), [textbox](#), [group](#), or [application](#).

[aria-atomic](#)

Indicates whether [assistive technologies](#) will present all, or only parts of, the changed region based on the change notifications defined by the [aria-relevant](#) attribute.

[aria-autocomplete](#)

Indicates whether inputting text could trigger display of one or more predictions of the user's intended value for a [combobox](#), [searchbox](#), or [textbox](#) and specifies how predictions would be presented if they were made.

[aria-busy](#)

Indicates an element is being modified and that assistive technologies **MAY** want to wait until the modifications are complete before exposing them to the user.

[aria-checked](#)

Indicates the current "checked" [state](#) of checkboxes, radio buttons, and other [widgets](#). See related [aria-pressed](#) and [aria-selected](#).

[aria-colcount](#)

Defines the total number of columns in a [table](#), [grid](#), or [treegrid](#). See related [aria-colindex](#).

[aria-colindex](#)

Defines an [element's](#) column index or position with respect to the total number of columns within a [table](#), [grid](#), or [treegrid](#). See related [aria-colcount](#) and [aria-colspan](#).

[aria-colspan](#)

Defines the number of columns spanned by a cell or gridcell within a [table](#), [grid](#), or [treegrid](#). See related [aria-colindex](#) and [aria-rowspan](#).

[aria-controls](#)

Identifies the [element](#) (or elements) whose contents or presence are controlled by the current element. See related [aria-owns](#).

[aria-current](#)

Indicates the [element](#) that represents the current item within a container or set of related elements.

[aria-describedby](#)

Identifies the [element](#) (or elements) that describes the [object](#). See related [aria-labelledby](#).

[aria-details](#)

Identifies the [element](#) that provides a detailed, extended description for the [object](#). See related [aria-describedby](#).

[aria-disabled](#)

Indicates that the [element](#) is [perceivable](#) but disabled, so it is not editable or otherwise [operable](#). See related [aria-hidden](#) and [aria-readonly](#).

[aria-dropeffect](#)

[Deprecated in ARIA 1.1] Indicates what functions can be performed when a dragged object is released on the drop target.

[aria-errormessage](#)

Identifies the [element](#) that provides an error message for an [object](#). See related [aria-invalid](#) and [aria-describedby](#).

[aria-expanded](#)

Indicates whether a grouping element owned or controlled by this element is expanded or collapsed.

[aria-flowto](#)

Identifies the next [element](#) (or elements) in an alternate reading order of content which, at the user's discretion, allows assistive technology to override the general default of reading in document source order.

[aria-grabbed](#)

[Deprecated in ARIA 1.1] Indicates an element's "grabbed" [state](#) in a drag-and-drop operation.

[aria-haspopup](#)

Indicates the availability and type of interactive popup element, such as menu or dialog, that can be triggered by an [element](#).

[aria-hidden](#)

Indicates whether the [element](#) is exposed to an accessibility API. See related [aria-disabled](#).

[aria-invalid](#)

Indicates the entered value does not conform to the format expected by the application. See related [aria-errormessage](#).

[aria-keyshortcuts](#)

Indicates keyboard shortcuts that an author has implemented to activate or give focus to an element.

[aria-label](#)

Defines a string value that labels the current element. See related [aria-labelledby](#).

[aria-labelledby](#)

Identifies the [element](#) (or elements) that labels the current element. See related [aria-describedby](#).

[aria-level](#)

Defines the hierarchical level of an [element](#) within a structure.

[aria-live](#)

Indicates that an [element](#) will be updated, and describes the types of updates the [user agents](#), [assistive technologies](#), and user can expect from the [live region](#).

[aria-modal](#)

Indicates whether an [element](#) is modal when displayed.

[aria-multiline](#)

Indicates whether a text box accepts multiple lines of input or only a single line.

[aria-multiselectable](#)

Indicates that the user may select more than one item from the current selectable descendants.

[aria-orientation](#)

Indicates whether the element's orientation is horizontal, vertical, or unknown/ambiguous.

[aria-owns](#)

Identifies an [element](#) (or elements) in order to define a visual, functional, or contextual parent/child [relationship](#) between DOM elements where the DOM hierarchy cannot be used to represent the relationship. See related [aria-controls](#).

[aria-placeholder](#)

Defines a short hint (a word or short phrase) intended to aid the user with data entry when the control has no value. A hint could be a sample value or a brief description of the expected format.

[aria-posinset](#)

Defines an [element](#)'s number or position in the current set of listitems or treeitems. Not required if all elements in the set are present in the DOM. See related [aria-setsize](#).

[aria-pressed](#)

Indicates the current "pressed" [state](#) of toggle buttons. See related [aria-checked](#) and [aria-selected](#).

[aria-readonly](#)

Indicates that the [element](#) is not editable, but is otherwise [operable](#). See related [aria-disabled](#).

[aria-relevant](#)

Indicates what notifications the user agent will trigger when the accessibility tree within a live region is modified. See related [aria-atomic](#).

[aria-required](#)

Indicates that user input is required on the [element](#) before a form may be submitted.

[aria-roledescription](#)

Defines a human-readable, author-localized description for the [role](#) of an [element](#).

[aria-rowcount](#)

Defines the total number of rows in a [table](#), [grid](#), or [treegrid](#). See related [aria-rowindex](#).

[aria-rowindex](#)

Defines an [element's](#) row index or position with respect to the total number of rows within a [table](#), [grid](#), or [treegrid](#). See related [aria-rowcount](#) and [aria-rowspan](#).

[aria-rowspan](#)

Defines the number of rows spanned by a cell or gridcell within a [table](#), [grid](#), or [treegrid](#). See related [aria-rowindex](#) and [aria-colspan](#).

[aria-selected](#)

Indicates the current "selected" [state](#) of various [widgets](#). See related [aria-checked](#) and [aria-pressed](#).

[aria-setsize](#)

Defines the number of items in the current set of listitems or treeitems. Not required if all elements in the set are present in the **DOM**. See related [aria-posinset](#).

[aria-sort](#)

Indicates if items in a table or grid are sorted in ascending or descending order.

[aria-valuemax](#)

Defines the maximum allowed value for a range [widget](#).

[aria-valuemin](#)

Defines the minimum allowed value for a range [widget](#).

[aria-valuenow](#)

Defines the current value for a range [widget](#). See related [aria-valuetext](#).

[aria-valuetext](#)

Defines the human readable text alternative of [aria-valuenow](#) for a range [widget](#).

aria-activedescendant property

Identifies the currently active element when **DOM** focus is on a [composite](#) widget, [combobox](#), [textbox](#), [group](#), or [application](#).

The `aria-activedescendant` property provides an alternative method of managing focus for interactive elements that may contain multiple focusable descendants, such as menus, grids, and toolbars. Instead of moving **DOM** focus among [owned](#) elements, authors **MAY** set **DOM** focus on a container [element](#) that supports `aria-activedescendant` and then use `aria-activedescendant` to refer to the element that is active.

Authors **MUST** ensure that one of the following two sets of conditions is met when setting the value of `aria-activedescendant` on an element with **DOM** focus:

1. The value of `aria-activedescendant` refers to an [owned](#) element. An owned element is either a descendant of the element with **DOM** focus or a logical descendant as indicated by the [aria-owns](#) attribute.
2. The element with **DOM** focus is a [combobox](#), [textbox](#) or [searchbox](#) with [aria-controls](#) referring to an element that supports `aria-activedescendant`, and the value of `aria-activedescendant` refers to an owned element of the controlled element. For example, in a [combobox](#), focus may remain on the [combobox](#) while the value of `aria-activedescendant` on the [combobox](#) element refers to a descendant of a popup [listbox](#) that is controlled by the [combobox](#).

Authors **SHOULD** also ensure that the currently active descendant is visible and in view (or scrolls into view) when focused.

Characteristics:

Characteristic	Value
Related Concepts:	SVG [SVG2] and DOM [DOM] active
Used in Roles:	application combobox composite group textbox
Inherits into Roles:	grid listbox menu menubar

Characteristic	Value
	radiogroup row searchbox select spinbutton tablist toolbar tree treegrid
Value:	ID reference

aria-atomic property

Indicates whether [assistive technologies](#) will present all, or only parts of, the changed region based on the change notifications defined by the [aria-relevant](#) attribute.

Both [accessibility APIs](#) and the [Document Object Model](#) [DOM] provide events to allow the assistive technologies to determine changed areas of the document.

When the content of a [live region](#) changes, user agents **SHOULD** examine the changed [element](#) and traverse the ancestors to find the first element with [aria-atomic](#) set, and apply the appropriate behavior for the cases below.

1. If none of the ancestors have explicitly set [aria-atomic](#), the default is that [aria-atomic](#) is false, and assistive technologies will only present the changed node to the user.
2. If [aria-atomic](#) is explicitly set to false, assistive technologies will stop searching up the ancestor chain and present only the changed node to the user.
3. If [aria-atomic](#) is explicitly set to true, assistive technologies will present the entire contents of the element, including the author-defined live region label if one exists.

When [aria-atomic](#) is true, assistive technologies **MAY** choose to combine several changes and present the entire changed region at once.

Characteristics:

Characteristic	Value
----------------	-------

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	true/false

Values:

Value	Description
false (default)	Assistive technologies will present only the changed node or nodes.
true	Assistive technologies will present the entire changed region as a whole, including the author-defined label if one exists.

aria-autocomplete property

Indicates whether inputting text could trigger display of one or more predictions of the user's intended value for a [combobox](#), [searchbox](#), or [textbox](#) and specifies how predictions would be presented if they were made.

The `aria-autocomplete` property describes the type of interaction model a [textbox](#), [searchbox](#), or [combobox](#) employs when dynamically helping users complete text input. It distinguishes between two models: the inline model (`aria-autocomplete="inline"`) that presents a value completion prediction inside the text input and the list model (`aria-autocomplete="list"`) that presents a collection of possible values in a separate element that pops up adjacent to the text input. It is possible for an input to offer both models at the same time (`aria-autocomplete="both"`).

The `aria-autocomplete` property is limited to describing predictive behaviors of an input element. Authors **SHOULD** either omit specifying a value for `aria-autocomplete` or set `aria-autocomplete` to `none` if an input element provides one or more input proposals where none of the proposals are dependent on the specific input provided by the user. For instance, a combobox where the value of `aria-autocomplete` would be `none` is a search field that displays suggested values by listing the 5 most recently used search terms without any filtering of the list based on the user's input. Elements with a role that supports `aria-autocomplete` have a default value for `aria-autocomplete` of `none`.

When an inline suggestion is made as a user types in an input, suggested text for completing the value of the field dynamically appears in the field after the input cursor, and the suggested value is accepted as the value of the input if the user performs an action that causes focus to leave the field. When an element has `aria-autocomplete` set to `inline` or `both`, authors **SHOULD** ensure that the automatically suggested portion of the text is presented as selected text. This enables assistive technologies to distinguish between a user's

input and the automatic suggestion and, in the event that the suggestion is not the desired value, enables the user to easily delete the suggestion or replace it by continuing to type.

If an element has `aria-autocomplete` set to `list` or `both`, authors **MUST** ensure both of the following conditions are met:

1. The element has a value specified for [aria-controls](#) that refers to the element that contains the collection of suggested values.
2. The element has a value for [aria-haspopup](#) that matches the role of the element that contains the collection of suggested values.

Some implementations of the list model require the user to perform an action, such as moving focus to the suggestion with the Down Arrow or clicking on the suggestion, in order to choose the suggestion. In such implementations, authors **MAY** manage focus by either using [aria-activedescendant](#) if the collection container supports it or by moving `DOM` focus to the suggestion. However, other implementations of the list model automatically highlight one suggestion as the selected value that will be accepted when the field loses focus, e.g., when the user presses the Tab key or clicks on a different field. If an element has `aria-autocomplete` set to `list` or `both`, and if a suggestion is automatically selected as the user provides input, authors **MUST** ensure all the following conditions are met:

1. The collection of suggestions is presented in an element with a role that supports [aria-activedescendant](#).
2. The value of `aria-activedescendant` set on the input field is dynamically adjusted to refer to the element containing the selected suggestion as described in the definition of [aria-activedescendant](#).
3. `DOM` focus remains on the text input while the suggestions are displayed.

The `aria-autocomplete` property is not intended to indicate the presence of a completion suggestion, and authors **SHOULD NOT** dynamically change its value in order to communicate the presence of a suggestion. When an element has `aria-autocomplete` set to `list` or `both`, authors **SHOULD** use the [aria-expanded](#) state to communicate whether the element that presents the suggestion collection is displayed.

Characteristics:

Characteristic	Value
Used in Roles:	combobox textbox
Inherits into Roles:	searchbox
Value:	token

Values:

Value	Description
inline	When a user is providing input, text suggesting one way to complete the provided input may be dynamically inserted after the caret.
list	When a user is providing input, an element containing a collection of values that could complete the provided input may be displayed.
both	When a user is providing input, an element containing a collection of values that could complete the provided input may be displayed. If displayed, one value in the collection is automatically selected, and the text needed to complete the automatically selected value appears after the caret in the input.
none (default)	When a user is providing input, an automatic suggestion that attempts to predict how the user intends to complete the input is not displayed.

aria-busy state

Indicates an element is being modified and that assistive technologies **MAY** want to wait until the modifications are complete before exposing them to the user.

The default value of `aria-busy` is `false` for all elements. When `aria-busy` is `true` for an element, assistive technologies **MAY** ignore changes to content owned by that element and then process all changes made during the busy period as a single, atomic update when `aria-busy` becomes `false`.

If it is necessary to make multiple additions, modifications, or removals within a container element that is already either partially or fully rendered, authors **MAY** set `aria-busy` to `true` on the container element before the first change, and then set it to `false` when the last change is complete. For example, if multiple changes to a [live region](#) should be spoken as a single unit of speech, authors **MAY** set `aria-busy` to `true` while the changes are being made and then set it to `false` when the changes are complete and ready to be spoken.

If an element with role [feed](#) is marked busy, assistive technologies **MAY** defer rendering changes that occur inside the `feed` with the exception of user-initiated changes that occur inside the [article](#) that the user is reading during the busy period.

If changes to a rendered [widget](#) would create a state where the [widget](#) is missing [required owned elements](#) during script execution, authors **MUST** set `aria-busy` to `true` on the [widget](#) during the update process.

For example, if a rendered tree grid required a set of simultaneous updates to multiple discontinuous branches, an alternative to replacing the complete tree element with a single update would be to mark the tree busy while each of the branches are modified.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	true/false

Values:

Value	Description
false (default):	There are no expected updates for the element.
true	The element is being updated.

aria-checked state

Indicates the current "checked" [state](#) of checkboxes, radio buttons, and other [widgets](#). See related [aria-pressed](#) and [aria-selected](#).

The [aria-checked attribute](#) indicates whether the [element](#) is checked (true), unchecked (false), or represents a group of other elements that have a mixture of checked and unchecked values (mixed). Most inputs only support values of true and false, but the mixed value is supported by certain tri-state inputs such as a [checkbox](#) or [menuitemcheckbox](#).

The mixed value is *not* supported on [radio](#), [menuitemradio](#), [switch](#) or any element that inherits from these, and [user agents](#) **MUST** treat a mixed value as equivalent to false for those [roles](#).

Examples using the mixed value of tri-state inputs are covered in the [WAI-ARIA Authoring Practices](#).

Characteristics:

Characteristic	Value
Used in Roles:	checkbox menuitemcheckbox option radio switch

Characteristic	Value
Inherits into Roles:	menuitemradio switch treeitem
Value:	tristate

Values:

Value	Description
false	The element supports being checked but is not currently checked.
mixed	Indicates a mixed mode value for a tri-state checkbox or menuitemcheckbox.
true	The element is checked.
undefined (default)	The element does not support being checked.

aria-colcount property

Defines the total number of columns in a [table](#), [grid](#), or [treegrid](#). See related [aria-colindex](#).

If all of the columns are present in the **DOM**, it is not necessary to set this [attribute](#) as the [user agent](#) can automatically calculate the total number of columns. However, if only a portion of the columns is present in the **DOM** at a given moment, this attribute is needed to provide an explicit indication of the number of columns in the full table.

Authors **MUST** set the value of [aria-colcount](#) to an integer equal to the number of columns in the full table. If the total number of columns is unknown, authors **MUST** set the value of [aria-colcount](#) to -1 to indicate that the value should not be calculated by the user agent.

The following example shows a grid with 16 columns, of which columns 2, 3, 4, and 9 are displayed to the user.

EXAMPLE 17

```
<div role="grid" aria-colcount="16">
  <div role="rowgroup">
    <div role="row">
      <span role="columnheader" aria-colindex="2">First Name</span>
      <span role="columnheader" aria-colindex="3">Last Name</span>
      <span role="columnheader" aria-colindex="4">Company</span>
      <span role="columnheader" aria-colindex="9">Phone</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row">
      <span role="gridcell" aria-colindex="2">Fred</span>
      <span role="gridcell" aria-colindex="3">Jackson</span>
      <span role="gridcell" aria-colindex="4">Acme, Inc.</span>
      <span role="gridcell" aria-colindex="9">555-1234</span>
    </div>
    <div role="row">
      <span role="gridcell" aria-colindex="2">Sara</span>
      <span role="gridcell" aria-colindex="3">James</span>
      <span role="gridcell" aria-colindex="4">Acme, Inc.</span>
      <span role="gridcell" aria-colindex="9">555-1235</span>
    </div>
    ...
  </div>
</div>
```

Characteristics:

Characteristic	Value
Used in Roles:	table
Inherits into Roles:	grid treegrid
Value:	integer

aria-colindex property

Defines an [element's](#) column index or position with respect to the total number of columns within a [table](#), [grid](#), or [treegrid](#). See related [aria-colcount](#) and [aria-colspan](#).

If all of the columns are present in the **DOM**, it is not necessary to set this [attribute](#) as the [user agent](#) can automatically calculate the column index of each cell or [gridcell](#). However, if only a portion of the columns is present in the **DOM** at a given moment, this attribute is needed to provide an explicit indication of the column of each cell or gridcell with respect to the full table.

Authors **MUST** set the value for [aria-colindex](#) to an integer greater than or equal to 1, greater than the [aria-colindex](#) value of any previous elements within the same row, and less than or equal to the number of columns in the full table. For a cell or gridcell which spans multiple columns, authors **MUST** set the value of [aria-colindex](#) to the start of the span.

If the set of columns which is present in the DOM is contiguous, and if there are no cells which span more than one row or column in that set, then authors **MAY** place [aria-colindex](#) on each row, setting the value to the index of the first column of the set. Otherwise, authors **SHOULD** place [aria-colindex](#) on all of the children or [owned](#) elements of each row.

The following example shows a grid with 16 columns, of which columns 2 through 5 are displayed to the user. Because the set of columns is contiguous, [aria-colindex](#) can be placed on each row.

EXAMPLE 18

```
<div role="grid" aria-colcount="16">
  <div role="rowgroup">
    <div role="row" aria-colindex="2">
      <span role="columnheader">First Name</span>
      <span role="columnheader">Last Name</span>
      <span role="columnheader">Company</span>
      <span role="columnheader">Address</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row" aria-colindex="2">
      <span role="gridcell">Fred</span>
      <span role="gridcell">Jackson</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">123 Broad St.</span>
    </div>
    <div role="row" aria-colindex="2">
      <span role="gridcell">Sara</span>
      <span role="gridcell">James</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">123 Broad St.</span>
    </div>
    ...
  </div>
</div>
```

The following example shows a grid with 16 columns, of which columns 2 through 5 are displayed to the user. While the set of columns is contiguous, some of the cells span multiple rows. As a result, [aria-colindex](#) needs to be placed on all of the owned elements of each row.

EXAMPLE 19

```

<div role="grid" aria-colcount="16">
  <div role="rowgroup">
    <div role="row">
      <span role="columnheader" aria-colindex="2">First Name</span>
      <span role="columnheader" aria-colindex="3">Last Name</span>
      <span role="columnheader" aria-colindex="4">Company</span>
      <span role="columnheader" aria-colindex="5">Address</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row">
      <span role="gridcell" aria-colindex="2">Fred</span>
      <span role="gridcell" aria-colindex="3">Jackson</span>
      <span role="gridcell" aria-colindex="4" aria-rowspan="2">Acme, Inc.</span>
      <span role="gridcell" aria-colindex="5" aria-rowspan="2">123 Broad St.</span>
    </div>
    <div role="row">
      <span role="gridcell" aria-colindex="2">Sara</span>
      <span role="gridcell" aria-colindex="3">James</span>
    </div>
    ...
  </div>
</div>

```

The following example shows a grid with 16 columns, of which columns 2, 3, 4, and 9 are displayed to the user. Because the set of columns is non-contiguous, [aria-colindex](#) needs to be placed on all of the owned elements of each row.

EXAMPLE 20

```
<div role="grid" aria-colcount="16">
  <div role="rowgroup">
    <div role="row">
      <span role="columnheader" aria-colindex="2">First Name</span>
      <span role="columnheader" aria-colindex="3">Last Name</span>
      <span role="columnheader" aria-colindex="4">Company</span>
      <span role="columnheader" aria-colindex="9">Phone</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row">
      <span role="gridcell" aria-colindex="2">Fred</span>
      <span role="gridcell" aria-colindex="3">Jackson</span>
      <span role="gridcell" aria-colindex="4">Acme, Inc.</span>
      <span role="gridcell" aria-colindex="9">555-1234</span>
    </div>
    <div role="row">
      <span role="gridcell" aria-colindex="2">Sara</span>
      <span role="gridcell" aria-colindex="3">James</span>
      <span role="gridcell" aria-colindex="4">Acme, Inc.</span>
      <span role="gridcell" aria-colindex="9">555-1235</span>
    </div>
    ...
  </div>
</div>
```

Characteristics:

Characteristic	Value
Used in Roles:	cell row
Inherits into Roles:	columnheader gridcell rowheader
Value:	integer

aria-colspan property

Defines the number of columns spanned by a cell or gridcell within a [table](#), [grid](#), or [treegrid](#). See related [aria-colindex](#) and [aria-rowspan](#).

This [attribute](#) is intended for cells and gridcells which are not contained in a native table. When defining the column span of cells or gridcells in a native table, authors **SHOULD** use the host language's attribute instead of [aria-colspan](#). If [aria-colspan](#) is used on an element for which the host language provides an equivalent attribute, [user agents](#) **MUST** ignore the value of [aria-colspan](#) and instead expose the value of the host language's attribute to [assistive technologies](#).

Authors **MUST** set the value of [aria-colspan](#) to an integer greater than or equal to 1 and less than the value which would cause the cell or gridcell to overlap the next cell or gridcell in the same row.

Characteristics:

Characteristic	Value
Used in Roles:	cell
Inherits into Roles:	columnheader rowheader
Value:	integer

aria-controls property

Identifies the [element](#) (or elements) whose contents or presence are controlled by the current element. See related [aria-owns](#).

For example:

- A table of contents tree view may control the content of a neighboring document pane.
- A group of checkboxes may control what commodity prices are tracked live in a table or graph.
- A tab controls the display of its associated tab panel.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	ID reference list

aria-current state

Indicates the [element](#) that represents the current item within a container or set of related elements.

The [aria-current attribute](#) is a token type. Any value not included in the list of allowed values **SHOULD** be treated by [assistive technologies](#) as if the value `true` had been provided. If the attribute is not present or its value is an empty string or undefined, the default value of `false` applies and the [aria-current state](#) **MUST NOT** be exposed by user agents or assistive technologies.

The [aria-current](#) attribute is used when an element within a set of related elements is visually styled to indicate it is the current item in the set. For example:

- A `page` token used to indicate a link within a set of pagination links, where the link is visually styled to represent the currently-displayed page.
- A `step` token used to indicate a link within a step indicator for a step-based process, where the link is visually styled to represent the current step.
- A `location` token used to indicate the image that is visually highlighted as the current component of a flow chart.
- A `date` token used to indicate the current date within a calendar.
- A `time` token used to indicate the current time within a timetable.

Authors **SHOULD** only mark one element in a set of elements as current with [aria-current](#).

Authors **SHOULD NOT** use the [aria-current](#) attribute as a substitute for [aria-selected](#) in widgets where [aria-selected](#) has the same meaning. For example, in a [tablist](#), [aria-selected](#) is used on a [tab](#) to indicate the currently-displayed [tabpanel](#).

NOTE

In some use cases for widgets that support [aria-selected](#), current and selected can have different meanings and can both be used within the same set of elements. For example, `aria-current="page"` can be used in a navigation [tree](#) to indicate which page is currently displayed, while `aria-selected="true"` indicates which page will be displayed if the user activates the [treeitem](#). Furthermore, the same tree may support operating on one or more selected pages ([treeitems](#)) by way of a context menu containing options such as "delete" and "move."

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	token

Values:

Value	Description
-------	-------------

Value	Description
page	Represents the current page within a set of pages.
step	Represents the current step within a process.
location	Represents the current location within an environment or context.
date	Represents the current date within a collection of dates.
time	Represents the current time within a set of times.
true	Represents the current item within a set.
false (default)	Does not represent the current item within a set.

aria-describedby property

Identifies the [element](#) (or elements) that describes the [object](#). See related [aria-labelledby](#).

The [aria-labelledby](#) attribute is similar to the [aria-describedby](#) in that both reference other elements to calculate a text alternative, but a label should be concise, where a description is intended to provide more verbose information.

The element or elements referenced by the `aria-describedby` comprise the entire description. Include ID references to multiple elements if necessary, or enclose a set of elements (e.g., paragraphs) with the element referenced by the ID.

Characteristics:

Characteristic	Value
Related Concepts:	<code><label></code> in [HTML] online help HTML , table cell headers
Used in Roles:	All elements of the base markup
Value:	ID reference list

aria-details property

Identifies the [element](#) that provides a detailed, extended description for the [object](#). See related [aria-describedby](#).

The `aria-details` attribute references a single element that provides more detailed information than would normally be provided by [aria-describedby](#). It enables [assistive technologies](#) to make users aware of the availability of an extended description as well as navigate to it. Authors **SHOULD** ensure the element referenced by `aria-details` is visible to all users.

Unlike elements referenced by `aria-describedby`, the element referenced by `aria-details` is not used in either the Accessible [Name Computation](#) or the Accessible [Description Computation](#) as defined in the Accessible Name and Description specification. Thus, the content of an element referenced by `aria-details` is not flattened to a string when presented to assistive technology users. This makes `aria-details` particularly useful when converting the information to a string would cause a loss of information or make the extended description more difficult to understand.

In some user agents, multiple reference relationships for descriptive information are not supported by the accessibility API. In such cases, if both [aria-describedby](#) and `aria-details` are provided on an element, `aria-details` takes precedence.

A common use for `aria-details` is in digital publishing where an extended description needs to be conveyed in a book that requires structural markup or the embedding of other technology to provide illustrative content. The following example demonstrates this scenario.

EXAMPLE 21

```
<!-- Provision of an extended description -->

<details id="det">
  <summary>Example</summary>
  <p>
    The Pythagorean Theorem is a relationship in Euclidean Geometry between the three side
    a right triangle, where the square of the hypotenuse is the sum of the squares of the
    opposing sides.
  </p>
  <p>
    The following drawing illustrates an application of the Pythagorean Theorem when used
    construct a skateboard ramp.
  </p>
  <object data="skatebd-ramp.svg" type="image/svg+xml"></object>
  <p>
    In this example you will notice a skateboard with a base and vertical board whose width
    is the width of the ramp. To compute how long the ramp must be, simply calculate the
    base length, square it, sum it with the square of the height of the ramp, and take the
    square root of the sum.
  </p>
</details>
```

Alternatively, `aria-details` may refer to a link to a web page having the extended description, as shown in the following example.

EXAMPLE 22

```
<!-- Provision of an extended description -->

<p>
  See an <a href="http://foo.com/pt.html" id="det">Application of the Pythagorean Theorem<
</p>
```

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	ID reference

aria-disabled state

Indicates that the [element](#) is [perceivable](#) but disabled, so it is not editable or otherwise [operable](#). See related [aria-hidden](#) and [aria-readonly](#).

For example, irrelevant options in a radio group may be disabled. Disabled elements might not receive focus from the tab order. For some disabled elements, applications might choose not to support navigation to descendants. In addition to setting the [aria-disabled](#) attribute, authors **SHOULD** change the appearance (grayed out, etc.) to indicate that the item has been disabled.

The [state](#) of being disabled applies to the current element and all focusable descendant elements of the element on which the [aria-disabled attribute](#) is applied.

NOTE

While [aria-disabled](#) and proper scripting can successfully disable an element with role [link](#), fully disabling a host language equivalent can be problematic. Authors are advised not to use [aria-disabled](#) on elements that cannot be disabled through features of the host language alone.

NOTE: Usage on columnheader, rowheader and row

While [aria-disabled](#) is currently supported on [columnheader](#), [rowheader](#), and [row](#), in a future version the working group plans to prohibit its use on elements with any of those three roles except when they are in the context of a [grid](#) or [treegrid](#).

NOTE

This state is being deprecated as a global state in ARIA 1.2. In future versions it will only be allowed on roles where it is specifically supported.

Characteristics:

Characteristic	Value
Used in Roles:	application button composite gridcell group input link menuitem scrollbar separator tab
Inherits into Roles:	checkbox columnheader combobox grid listbox menu menubar menuitemcheckbox menuitemradio option radio radiogroup row

Characteristic	Value
	rowheader searchbox select slider spinbutton switch tablist textbox toolbar tree treegrid treeitem
Value:	true/false

Values:

Value	Description
false (default)	The element is enabled.
true	The element and all focusable descendants are disabled and its value cannot be changed by the user.

aria-dropeffect property

[Deprecated in ARIA 1.1] Indicates what functions can be performed when a dragged object is released on the drop target.

NOTE

The `aria-dropeffect` property is expected to be replaced by a new feature in a future version of WAI-ARIA. Authors are therefore advised to treat `aria-dropeffect` as [deprecated](#).

This [property](#) allows assistive technologies to convey the possible drag options available to users, including whether a pop-up menu of choices is provided by the application. Typically, drop effect functions can only be provided once an object has been grabbed for a drag operation as the drop effect functions available are dependent on the object being dragged.

More than one drop effect may be supported for a given [element](#). Therefore, the value of this [attribute](#) is a space-separated set of tokens indicating the possible effects, or none if there is no supported operation. In addition to setting the [aria-dropeffect](#) attribute, authors **SHOULD** show a visual indication of potential drop targets.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	token list

Values:

Value	Description
copy	A duplicate of the source object will be dropped into the target.
execute	A function supported by the drop target is executed, using the drag source as an input.
link	A reference or shortcut to the dragged object will be created in the target object.
move	The source object will be removed from its current location and dropped into the target.
none (default)	No operation can be performed; effectively cancels the drag operation if an attempt is made to drop on this object. Ignored if combined with any other token value. e.g., 'none copy' is equivalent to a 'copy' value.
popup	There is a popup menu or dialog that allows the user to choose one of the drag operations (copy, move, link, execute) and any other drag functionality, such as cancel.

aria-errormessage property

Identifies the [element](#) that provides an error message for an [object](#). See related [aria-invalid](#) and [aria-describedby](#).

The `aria-errormessage` attribute references another element that contains error message text. Authors **MUST** use [aria-invalid](#) in conjunction with `aria-errormessage`.

When the value of an object is not valid, [aria-invalid](#) is set to `true`, which indicates that the message contained by an element referenced by `aria-errormessage` is pertinent.

When an object is in a valid state, it has either [aria-invalid](#) set to `false` or it does not have the [aria-invalid](#) attribute. Authors **MAY** use `aria-errormessage` on an object that is currently valid, but only if the element referenced by `aria-errormessage` is [hidden](#), because the message it contains is not pertinent.

When `aria-errormessage` is pertinent, authors **MUST** ensure the content is not hidden so users can navigate to and examine the error message. Similarly, when `aria-errormessage` is not pertinent, authors **MUST** either ensure the content is [hidden](#) or remove the `aria-errormessage` attribute or its value.

User agents **MUST NOT** expose `aria-errormessage` for an object with an [aria-invalid](#) value of `false`.

Authors **MAY** call attention to a newly rendered error message with a live region by either applying an [aria-live](#) property or using one of the [live region roles](#), such as [alert](#). A live region is appropriate when an error message is displayed to users after they have provided an invalid value.

A typical message describes what is wrong and informs users what is required. For example, an error message might be, “Invalid time: the time must be between 9:00 AM and 5:00 PM.” The following example code shows markup for an initial valid state and for a subsequent invalid state. Note the changes to [aria-invalid](#) on the text input [object](#), and to [aria-live](#) on the [element](#) containing the text of the error message:

EXAMPLE 23

```
<!-- Initial valid state -->
<label for="startTime"> Please enter a start time for the meeting: </label>
<input id="startTime" type="text" aria-errormessage="msgID" value="" aria-invalid="false">
<span id="msgID" aria-live="assertive"><span style="visibility:hidden">Invalid time: the t

<!-- User has input an invalid value -->
<label for="startTime"> Please enter a start time for the meeting: </label>
<input id="startTime" type="text" aria-errormessage="msgID" aria-invalid="true" value="11:
<span id="msgID" aria-live="assertive"><span style="visibility:visible">Invalid time: the
```

NOTE

This example uses `aria-live="assertive"` to indicate that assistive technologies should immediately announce the error message rather than completing other queued announcements first. This increases the likelihood that users are aware of the error message before they move focus out of the input.

NOTE

This state is being deprecated as a global state in ARIA 1.2. In future versions it will only be allowed on roles where it is specifically supported.

Characteristics:

Characteristic	Value
Used in Roles:	application checkbox combobox gridcell listbox radiogroup slider spinbutton textbox tree
Inherits into Roles:	columnheader rowheader searchbox switch treegrid
Value:	ID reference

aria-expanded state

Indicates whether a grouping element owned or controlled by this element is expanded or collapsed.

The [aria-expanded](#) attribute is applied to a focusable, interactive element that toggles visibility of content in another element. For example, it is applied to a parent [treeitem](#) to indicate whether its child branch of the tree is shown. Similarly, it can be applied to a [button](#) that controls visibility of a section of page content.

If a grouping container that can be expanded or collapsed is not [owned](#) by the element that has the [aria-expanded](#) attribute, the author **SHOULD** identify the controlling relationship by referencing the container from the element that has [aria-expanded](#) with the [aria-controls](#) property.

Characteristics:

Characteristic	Value
Used in Roles:	application button checkbox combobox gridcell link listbox menuitem row rowheader tab treeitem
Inherits into Roles:	columnheader menuitemcheckbox menuitemradio rowheader switch
Value:	true/false/undefined

Values:

Value	Description
false	The grouping element this element owns or controls is collapsed.
true	The grouping element this element owns or controls is expanded.

Value	Description
undefined (default)	The element does not own or control a grouping element that is expandable.

aria-flowto property

Identifies the next [element](#) (or elements) in an alternate reading order of content which, at the user's discretion, allows assistive technology to override the general default of reading in document source order.

When [aria-flowto](#) has a single ID reference, it allows [assistive technologies](#) to, at the user's request, forego normal document reading order and go to the targeted [object](#). However, when [aria-flowto](#) is provided with multiple ID references, assistive technologies **SHOULD** present the referenced elements as path choices.

In the case of one or more ID references, [user agents](#) or assistive technologies **SHOULD** give the user the option of navigating to any of the targeted elements. The name of the path can be determined by the name of the target element of the [aria-flowto](#) attribute. [Accessibility APIs](#) can provide named path [relationships](#).

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	ID reference list

aria-grabbed state

[Deprecated in ARIA 1.1] Indicates an element's "grabbed" [state](#) in a drag-and-drop operation.

NOTE

The `aria-grabbed` state is expected to be replaced by a new feature in a future version of [WAI-ARIA](#). Authors are therefore advised to treat `aria-grabbed` as [deprecated](#).

Setting `aria-grabbed` to `true` indicates that the [element](#) has been selected for dragging. Setting `aria-grabbed` to `false` indicates that the element can be grabbed for a drag-and-drop operation, but is not currently grabbed. If `aria-grabbed` is unspecified or set to `undefined` (default), the element cannot be grabbed.

When [aria-grabbed](#) is set to `true`, authors **SHOULD** update the [aria-dropeffect](#) attribute of all potential drop targets. When an element is not grabbed (the value is set to `false` or `undefined`, or the

attribute is removed), authors **SHOULD** revert the [aria-dropeffect](#) attributes of the associated drop targets to none.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	true/false/undefined

Values:

Value	Description
false	Indicates that the element supports being dragged.
true	Indicates that the element has been "grabbed" for dragging.
undefined (default)	Indicates that the element does not support being dragged.

aria-haspopup property

Indicates the availability and type of interactive popup element, such as menu or dialog, that can be triggered by an [element](#).

A popup element usually appears as a block of content that is on top of other content. Authors **MUST** ensure that the role of the element that serves as the container for the popup content is [menu](#), [listbox](#), [tree](#), [grid](#), or [dialog](#), and that the value of `aria-haspopup` matches the role of the popup container.

For the popup element to be keyboard accessible, authors **SHOULD** ensure that the element that can trigger the popup is focusable, that there is a keyboard mechanism for opening the popup, and that the popup element manages focus of all its descendants as described in [Managing Focus](#).

The `aria-haspopup` property is a token type. [User agents](#) **MUST** treat any value of `aria-haspopup` that is not included in the list of allowed values, including an empty string, as if the value `false` had been provided. To provide backward compatibility with ARIA 1.0 content, user agents **MUST** treat an `aria-haspopup` value of `true` as equivalent to a value of `menu`.

[Assistive technologies](#) **SHOULD NOT** expose the `aria-haspopup` property if it has a value of `false`.

NOTE

A [tooltip](#) is not considered to be a popup in this context.

NOTE

`aria-haspopup` is most relevant to use when there is a visual indicator in the element that triggers the popup. For example, many controls styled with a downward pointing triangle, chevron, or ellipsis (three consecutive dots) have become standard visual indicators that a popup will display when the control is activated. If some functional difference is relevant to display to a sighted user by means of a different visual style, that functional difference is usually relevant to convey to users of assistive technology. If there is no visual indication that an element will trigger a popup, authors are advised to consider whether use of `aria-haspopup` is necessary, and avoid using it when it's not.

NOTE

This property is being deprecated as a global property in ARIA 1.2. In future versions it will only be allowed on roles where it is specifically supported.

Characteristics:

Characteristic	Value
Related Concepts:	aria-controls
Used in Roles:	application button combobox gridcell link menuitem slider tab textbox treeitem
Inherits into Roles:	columnheader menuitemcheckbox menuitemradio rowheader searchbox

Characteristic	Value
Value:	token

Values:

Value	Description
false (default)	Indicates the element does not have a popup.
true	Indicates the popup is a menu .
menu	Indicates the popup is a menu .
listbox	Indicates the popup is a listbox .
tree	Indicates the popup is a tree .
grid	Indicates the popup is a grid .
dialog	Indicates the popup is a dialog .

aria-hidden state

Indicates whether the [element](#) is exposed to an accessibility API. See related [aria-disabled](#).

User agents determine an element's [hidden](#) status based on whether it is rendered, and the rendering is usually controlled by CSS. For example, an element whose `display` property is set to `none` is not rendered. An element is considered [hidden](#) if it, or any of its ancestors are not rendered or have their `aria-hidden` attribute value set to `true`.

Authors **MAY**, with caution, use `aria-hidden` to hide visibly rendered content from assistive technologies *only* if the act of hiding this content is intended to improve the experience for users of assistive technologies by removing redundant or extraneous content. Authors using `aria-hidden` to hide visible content from screen readers **MUST** ensure that identical or equivalent meaning and functionality is exposed to assistive technologies.

NOTE

Authors are advised to use extreme caution and consider a wide range of disabilities when hiding visibly rendered content from assistive technologies. For example, a sighted, dexterity-impaired individual may use voice-controlled assistive technologies to access a visual interface. If an author hides visible link text "Go to checkout" and exposes similar, yet non-identical link text "Check out now" to the accessibility API, the user may be unable to access the interface they perceive using voice control. Similar problems may also arise for screen reader users. For example, a sighted telephone support technician may attempt to have the blind screen reader user click the "Go to checkout" link, which they may be unable to find using a type-ahead item search ("Go to...").

NOTE

At the time of this writing, [aria-hidden="false"](#) is known to work inconsistently in browsers. As future implementations improve, use caution and test thoroughly before relying on this approach.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	true/false/undefined

Values:

Value	Description
false	The element is exposed to the accessibility API as if it was rendered.
true	The element is hidden from the accessibility API.
undefined (default)	The element's hidden state is determined by the user agent based on whether it is rendered.

aria-invalid state

Indicates the entered value does not conform to the format expected by the application. See related [aria-errormessage](#).

If the value is computed to be invalid or out-of-range, the application author **SHOULD** set this [attribute](#) to true. [User agents](#) **SHOULD** inform the user of the error. Application authors **SHOULD** provide suggestions for corrections if they are known.

When the user attempts to submit data involving a field for which [aria-required](#) is true, authors **MAY** use the [aria-invalid](#) attribute to signal there is an error. However, if the user has not attempted to submit the form, authors **SHOULD NOT** set the [aria-invalid](#) attribute on required [widgets](#) simply because the user has not yet entered data.

For future expansion, the [aria-invalid](#) attribute is a token type. Any value not recognized in the list of allowed values **MUST** be treated by user agents as if the value true had been provided. If the attribute is not present, or its value is false, or its value is an empty string, the default value of false applies.

NOTE

This state is being deprecated as a global state in ARIA 1.2. In future versions it will only be allowed on roles where it is specifically supported.

Characteristics:

Characteristic	Value
Used in Roles:	application checkbox combobox gridcell listbox radiogroup slider spinbutton textbox tree
Inherits into Roles:	columnheader rowheader searchbox switch treegrid
Value:	token

Values:

Value	Description
grammar	A grammatical error was detected.
false (default)	There are no detected errors in the value.
spelling	A spelling error was detected.
true	The value entered by the user has failed validation.

aria-keyshortcuts property

Indicates keyboard shortcuts that an author has implemented to activate or give focus to an element.

The value of the `aria-keyshortcuts` attribute is a space-separated list of keyboard shortcuts that can be pressed to activate a command or textbox widget. The keys defined in the shortcuts represent the physical keys pressed and not the actual characters generated. Each keyboard shortcut consists of one or more tokens delimited by the plus sign ("+") representing zero or more modifier keys and exactly one non-modifier key that must be pressed simultaneously to activate the given shortcut.

Authors **MUST** specify modifier keys exactly according to the [UI Events KeyboardEvent key Values](#) spec [[uievents-key](#)] - for example, "Alt", "Control", "Shift", "Meta", or "AltGraph". Note that Meta corresponds to the Command key, and Alt to the Option key, on Apple computers.

The valid names for non-modifier keys are any printable character such as "A", "B", "1", "2", "\$", "Plus" for a plus sign, "Space" for the spacebar, or the names of any other non-modifier key specified in the [UI Events KeyboardEvent key Values](#) spec [[uievents-key](#)] - for example, "Enter", "Tab", "ArrowRight", "PageDown", "Escape", or "F1". The use of "Space" for the spacebar is an exception to the [UI Events KeyboardEvent key Values](#) spec [[uievents-key](#)] as the space or spacebar key is encoded as ' ' and would be treated as a whitespace character.

Authors **MUST** ensure modifier keys come first when they are part of a keyboard shortcut. Authors **MUST** ensure that required non-modifier keys come last when they are part of a shortcut. The order of the modifier keys is not otherwise significant, so "Alt+Shift+T" and "Shift+Alt+T" are equivalent, but "T+Shift+Alt" is not valid because all of the modifier keys don't come first, and "Alt" is not valid because it doesn't include at least one non-modifier key.

When specifying an alphabetic key, both the uppercase and lowercase variants are considered equivalent: "a" and "A" are the same.

When implementing keyboard shortcuts authors should consider the keyboards they intend to support to avoid unintended results. Keyboard designs vary significantly based on the device used and the languages supported. For example, many modifier keys are used in conjunction with other keys to create common punctuation symbols, create number characters, swap keyboard sides on bilingual keyboards to switch languages, and perform a number of other functions.

For many supported keyboards, authors can prevent conflicts by avoiding keys other than ASCII letters, as number characters and common punctuation often require modifiers. Here, the keyboard shortcut entered does not equate to the key generated. For example, in French keyboard layouts, the number characters are not available until you press the Control key, so a keyboard shortcut defined as "Control+2" would be ambiguous as this is how one would type the "2" character on a French keyboard.

If the character used is determined by a modifier key, the author **MUST** specify the actual key used to generate the character, that is generated by the key, and not the resulting character. This convention enables

the assistive technology to accurately convey what keys must be used to generate the shortcut. For example, on most U.S. English keyboards, the percent sign "%" can be input by pressing Shift+5. The correct way to specify this shortcut is "Shift+5". It is incorrect to specify "%" or "Shift+%". However, note that on some international keyboards the percent sign may be an unmodified key, in which case "%" and "Shift+%" could be correct on those keyboards.

If the key that needs to be specified is illegal in the host language or would cause a string to be terminated, authors **MUST** use the string escaping sequence of the host language to specify it. For example, the double-quote character can be encoded as "Shift+'" in HTML.

Examples of valid keyboard shortcuts include:

- "A"
- "Shift+Space"
- "Control+Alt+."
- "Control+Shift+'"
- "Alt+Shift+P Control+F"
- "Meta+C Meta+Shift+C"

User agents **MUST NOT** change keyboard behavior in response to the `aria-keyshortcuts` attribute. Authors **MUST** handle scripted keyboard events to process `aria-keyshortcuts`. The `aria-keyshortcuts` attribute exposes the existence of these shortcuts so that assistive technologies can communicate this information to users.

Authors **SHOULD** provide a way to expose keyboard shortcuts so that all users may discover them, such as through the use of a tooltip. Authors **MUST** ensure that `aria-keyshortcuts` applied to disabled elements are unavailable.

Authors **SHOULD** avoid implementing shortcut keys that inhibit operating system, user agent, or assistive technology functionality. This requires the author to carefully consider both which keys to assign and the contexts and conditions in which the keys are available to the user. For guidance, see the keyboard shortcuts section of the [WAI-ARIA Authoring Practices](#).

Characteristics:

Characteristic	Value
Related Concepts:	Keyboard shortcut
Used in Roles:	All elements of the base markup
Value:	string

aria-label property

Defines a string value that labels the current element. See related [aria-labelledby](#).

The purpose of [aria-label](#) is the same as that of [aria-labelledby](#). It provides the user with a recognizable name of the object. The most common [accessibility API](#) mapping for a label is the [accessible name](#) property.

If the label text is available in the DOM (i.e. typically visible text content), authors **SHOULD** use [aria-labelledby](#) and **SHOULD NOT** use [aria-label](#). There may be instances where the name of an element cannot be determined programmatically from the DOM, and there are cases where referencing DOM content is not the desired user experience. Most host languages provide an attribute that could be used to name the element (e.g., the `title` attribute in [HTML]), yet this could present a browser tooltip. In the cases where DOM content or a tooltip is undesirable, authors **MAY** set the accessible name of the element using [aria-label](#). As required by the [accessible name and description computation](#), user agents give precedence to [aria-labelledby](#) over [aria-label](#) when computing the accessible name property.

Characteristics:

Characteristic	Value
Related Concepts:	<code>title</code> attribute in [HTML]
Used in Roles:	All elements of the base markup except for the following roles: caption , code , deletion , emphasis , generic , insertion , paragraph , presentation , strong , subscript , superscript
Value:	string

aria-labelledby property

Identifies the [element](#) (or elements) that labels the current element. See related [aria-describedby](#).

The purpose of [aria-labelledby](#) is the same as that of [aria-label](#). It provides the user with a recognizable name of the object. The most common [accessibility API](#) mapping for a label is the [accessible name](#) property.

If the interface is such that it is not possible to have a visible label on the screen, authors **SHOULD** use [aria-label](#) and **SHOULD NOT** use [aria-labelledby](#). As required by the [accessible name and description computation](#), user agents give precedence to [aria-labelledby](#) over [aria-label](#) when computing the accessible name property.

The [aria-labelledby](#) attribute is similar to [aria-describedby](#) in that both reference other elements to calculate a text alternative, but a label should be concise, where a description is intended to provide more

verbose information.

NOTE

The expected spelling of this property in U.S. English is "labeledby." However, the [accessibility API](#) features to which this property is mapped have established the "labelledby" spelling. This property is spelled that way to match the convention and minimize the difficulty for developers.

Characteristics:

Characteristic	Value
Related Concepts:	<label> in [HTML]
Used in Roles:	All elements of the base markup except for the following roles: caption , code , deletion , emphasis , generic , insertion , paragraph , presentation , strong , subscript , superscript
Value:	ID reference list

aria-level property

Defines the hierarchical level of an [element](#) within a structure.

This can be applied inside trees to tree items, to headings inside a document, to nested grids, nested tablists and to other structural items that may appear inside a container or participate in an ownership hierarchy. The value for [aria-level](#) is an integer greater than or equal to 1.

Levels increase with depth. If the [DOM](#) ancestry does not accurately represent the level, authors **SHOULD** explicitly define the [aria-level](#) attribute.

This attribute is applied to elements that act as leaf nodes within the orientation of the set, for example, on elements with role [treeitem](#) rather than elements with role [group](#). This means that multiple elements in a set may have the same value for this attribute. Although it would be less repetitive to provide a single value on the container, restricting this to leaf nodes ensures that there is a single way for [assistive technologies](#) to use the attribute.

If the [DOM](#) ancestry accurately represents the level, the [user agent](#) can calculate the level of an item from the document structure. This attribute can be used to provide an explicit indication of the level when that is not possible to calculate from the document structure or the [aria-owns](#) attribute. User agent support for automatic calculation of level may vary; authors **SHOULD** test with [user agents](#) and assistive technologies to determine whether this attribute is needed. If the author intends for the user agent to calculate the level, the author **SHOULD** omit this attribute.

NOTE

In the case of a [treegrid](#), [aria-level](#) is supported on elements with the role [row](#), not elements with role [gridcell](#). At first glance, this may seem inconsistent with the application of [aria-level](#) on [treeitem](#) elements, but it is consistent in that the [row](#) acts as the leaf node within the vertical orientation of the [grid](#), whereas the [gridcell](#) is a leaf node within the horizontal orientation of each [row](#). Level is not supported on sets of cells within rows, so the [aria-level](#) attribute is applied to the element with the role [row](#).

Characteristics:

Characteristic	Value
Used in Roles:	heading listitem row
Inherits into Roles:	treeitem
Value:	integer

aria-live property

Indicates that an [element](#) will be updated, and describes the types of updates the [user agents](#), [assistive technologies](#), and user can expect from the [live region](#).

The values of this [attribute](#) are expressed in degrees of importance. When regions are specified as `polite`, assistive technologies will notify users of updates but generally do not interrupt the current task, and updates take low priority. When regions are specified as `assertive`, assistive technologies will immediately notify the user, and could potentially clear the speech queue of previous updates.

Politeness levels are essentially an ordering mechanism for updates and serve as a strong suggestion to user agents or assistive technologies. The value may be overridden by user agents, assistive technologies, or the user. For example, if assistive technologies can determine that a change occurred in response to a key press or a mouse click, the assistive technologies may present that change immediately even if the value of the [aria-live](#) attribute states otherwise.

Since different users have different needs, it is up to the user to tweak his or her assistive technologies' response to a live region with a certain politeness level from the commonly defined baseline. Assistive technologies may choose to implement increasing and decreasing levels of granularity so that the user can exercise control over queues and interruptions.

When the [property](#) is not set on an [object](#) that needs to send updates, the politeness level is the value of the nearest ancestor that sets the [aria-live](#) attribute.

The [aria-live](#) attribute is the primary determination for the order of presentation of changes to live regions. Implementations will also consider the default level of politeness in a [role](#) when the [aria-live](#) attribute is not set in the ancestor chain (e.g., [log](#) changes are polite by default). Items which are assertive will be presented immediately, followed by polite items. User agents or assistive technologies **MAY** choose to clear queued changes when an assertive change occurs. (e.g., changes in an assertive region may remove all currently queued changes)

When live regions are marked as polite, assistive technologies **SHOULD** announce updates at the next graceful opportunity, such as at the end of speaking the current sentence or when the user pauses typing. When live regions are marked as assertive, assistive technologies **SHOULD** notify the user immediately. Because an interruption may disorient users or cause them to not complete their current task, authors **SHOULD NOT** use the assertive value unless the interruption is imperative.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	token

Values:

Value	Description
assertive	Indicates that updates to the region have the highest priority and should be presented the user immediately.
off (default)	Indicates that updates to the region should not be presented to the user unless the user is currently focused on that region.
polite	Indicates that updates to the region should be presented at the next graceful opportunity, such as at the end of speaking the current sentence or when the user pauses typing.

aria-modal property

Indicates whether an [element](#) is modal when displayed.

The [aria-modal](#) attribute is used to indicate that the presence of a "modal" element precludes usage of other content on the page. For example, when a modal dialog is displayed, it is expected that the user's

interaction is limited to the contents of the dialog, until the modal dialog loses focus or is no longer displayed.

When a modal element is displayed, assistive technologies **SHOULD** navigate to the element unless focus has explicitly been set elsewhere. Assistive technologies **MAY** limit navigation to the modal element's contents. If focus moves to an element outside the modal element, assistive technologies **SHOULD NOT** limit navigation to the modal element.

When a modal element is displayed, authors **MUST** ensure the interface can be controlled using only descendants of the modal element. In other words, if a modal dialog has a close button, the button should be a descendant of the dialog. When a modal element is displayed, authors **SHOULD** mark all other contents as inert (such as "inert subtrees" in HTML) if the ability to do so exists in the host language.

Characteristics:

Characteristic	Value
Used in Roles:	window
Inherits into Roles:	alertdialog dialog
Value:	true/false

Values:

Value	Description
false (default)	Element is not modal.
true	Element is modal.

aria-multiline property

Indicates whether a text box accepts multiple lines of input or only a single line.

NOTE

In most user agent implementations, the default behavior of the ENTER or RETURN key is different between the single-line and multi-line text fields in HTML. When user has focus in a single-line <input type="text"> element, the keystroke usually submits the form. When user has focus in a multi-line <textarea> element, the keystroke inserts a line break. The WAI-ARIA textbox role differentiates these types of boxes with the [aria-multiline](#) attribute, so authors are advised to be aware of this distinction when designing the field.

Characteristics:

Characteristic	Value
Used in Roles:	textbox
Inherits into Roles:	searchbox
Value:	true/false

Values:

Value	Description
false (default)	This is a single-line text box.
true	This is a multi-line text box.

aria-multiselectable property

Indicates that the user may select more than one item from the current selectable descendants.

Authors **SHOULD** ensure that selected descendants have the [aria-selected attribute](#) set to `true`, and selectable descendant have the [aria-selected](#) attribute set to `false`. Authors **SHOULD NOT** use the [aria-selected](#) attribute on descendants that are not selectable.

NOTE

Lists and trees are examples of roles that might allow users to select more than one item at a time.

Characteristics:

Characteristic	Value
Used in Roles:	grid listbox tablist tree
Inherits into Roles:	treegrid
Value:	true/false

Values:

Value	Description
false (default)	Only one item can be selected.
true	More than one item in the widget may be selected at a time.

aria-orientation property

Indicates whether the element's orientation is horizontal, vertical, or unknown/ambiguous.

NOTE

In ARIA 1.1, the default value for [aria-orientation](#) changed from horizontal to undefined. Implicit defaults are defined on some roles (e.g., [slider](#) defaults to horizontal; [scrollbar](#) defaults to vertical) but remain undefined on roles where an expected default orientation is ambiguous (e.g., [radiogroup](#)).

Characteristics:

Characteristic	Value
Used in Roles:	scrollbar select separator slider tablist toolbar
Inherits into Roles:	listbox menu menubar radiogroup tree treegrid
Value:	token

Values:

Value	Description
horizontal	The element is oriented horizontally.
undefined (default)	The element's orientation is unknown/ambiguous.
vertical	The element is oriented vertically.

aria-owns property

Identifies an [element](#) (or elements) in order to define a visual, functional, or contextual parent/child [relationship](#) between [DOM](#) elements where the [DOM](#) hierarchy cannot be used to represent the relationship. See related [aria-controls](#).

The value of the [aria-owns attribute](#) is a space-separated ID reference list that references one or more elements in the document by ID. The reason for adding [aria-owns](#) is to expose a parent/child contextual relationship to [assistive technologies](#) that is otherwise impossible to infer from the [DOM](#).

If an element has both [aria-owns](#) and [DOM](#) children then the order of the child elements with respect to the parent/child relationship is the [DOM](#) children first, then the elements referenced in [aria-owns](#). If the author intends that the [DOM](#) children are not first, then list the [DOM](#) children in [aria-owns](#) in the desired order. Authors **SHOULD NOT** use [aria-owns](#) as a replacement for the [DOM](#) hierarchy. If the relationship is represented in the [DOM](#), do not use [aria-owns](#). Authors **MUST** ensure that an element's ID is not specified in more than one other element's [aria-owns](#) attribute at any time. In other words, an element can have only one explicit owner.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	ID reference list

aria-placeholder property

Defines a short hint (a word or short phrase) intended to aid the user with data entry when the control has no value. A hint could be a sample value or a brief description of the expected format.

Authors **SHOULD NOT** use [aria-placeholder](#) instead of a label as their purposes are different: The label indicates what kind of information is expected. The placeholder text is a hint about the expected value. See related [aria-labelledby](#) and [aria-label](#).

Authors **SHOULD** present this hint to the user by displaying the hint text at any time the control's value is the empty string. This includes cases where the control first receives focus, and when users remove a

previously-entered value.

NOTE

As is the case with the related `placeholder` attribute in [\[HTML\]](#), use of placeholder text as a replacement for a displayed label can reduce the accessibility and usability of the control for a range of users including older users and users with cognitive, mobility, fine motor skill or vision impairments. While the hint given by the control's label is shown at all times, the short hint given in the placeholder attribute is only shown before the user enters a value. Furthermore, placeholder text may be mistaken for a pre-filled value, and as commonly implemented the default color of the placeholder text provides insufficient contrast and the lack of a separate visible label reduces the size of the hit region available for setting focus on the control.

NOTE

The following examples do not use the `HTML` `label` element as it cannot be used to label `HTML` elements with `contenteditable`.

The following example shows a [searchbox](#) in which the user has entered a value:

EXAMPLE 24

```
<span id="label">Birthday:</span>
<div contenteditable role="searchbox" aria-labelledby="label" aria-placeholder="MM-DD-YYYY">
  01-01-2000
</div>
```

The following example shows the same [searchbox](#) in which the user has not yet entered a value or has removed a previously-entered value:

EXAMPLE 25

```
<span id="label">Birthday:</span>
<div contenteditable role="searchbox" aria-labelledby="label" aria-placeholder="MM-DD-YYYY">
  MM-DD-YYYY
</div>
```

Characteristics:

Characteristic	Value
Related Concepts:	placeholder attribute in [HTML]
Used in Roles:	textbox

Characteristic	Value
Inherits into Roles:	searchbox
Value:	string

aria-posinset property

Defines an [element](#)'s number or position in the current set of listitems or treeitems. Not required if all elements in the set are present in the DOM. See related [aria-setsize](#).

If all items in a set are present in the document structure, it is not necessary to set this [attribute](#), as the [user agent](#) can automatically calculate the set size and position for each item. However, if only a portion of the set is present in the document structure at a given moment, this [property](#) is needed to provide an explicit indication of an element's position.

The following example shows items 5 through 8 in a set of 16.

EXAMPLE 26

```
<h2 id="label_fruit"> Available Fruit </h2>
<ul role="listbox" aria-labelledby="label_fruit">
  <li role="option" aria-setsize="16" aria-posinset="5"> apples </li>
  <li role="option" aria-setsize="16" aria-posinset="6"> bananas </li>
  <li role="option" aria-setsize="16" aria-posinset="7"> cantaloupes </li>
  <li role="option" aria-setsize="16" aria-posinset="8"> dates </li>
</ul>
```

Authors **MUST** set the value for [aria-posinset](#) to an integer greater than or equal to 1, and less than or equal to the size of the set when that size is known. Authors **SHOULD** use [aria-setsize](#).

When exposing [aria-posinset](#) on a [menuitem](#), [menuitemcheckbox](#), or [menuitemradio](#), authors **SHOULD** set the value of [aria-posinset](#) with respect to the total number of items in the [menu](#), excluding any separators.

Characteristics:

Characteristic	Value
Used in Roles:	article listitem menuitem option

Characteristic	Value
	radio row tab
Inherits into Roles:	menuitemcheckbox menuitemradio treeitem
Value:	integer

aria-pressed state

Indicates the current "pressed" [state](#) of toggle buttons. See related [aria-checked](#) and [aria-selected](#).

Toggle buttons require a full press-and-release cycle to change their value. Activating it once changes the value to `true`, and activating it another time changes the value back to `false`. A value of `mixed` means that the values of more than one item controlled by the button do not all share the same value. If the [attribute](#) is not present, the button is not a toggle button.

The [aria-pressed](#) attribute is similar but not identical to the [aria-checked](#) attribute. Operating systems support pressed on buttons and checked on checkboxes.

Characteristics:

Characteristic	Value
Used in Roles:	button
Value:	tristate

Values:

Value	Description
false	The element supports being pressed but is not currently pressed.
mixed	Indicates a mixed mode value for a tri-state toggle button.
true	The element is pressed.
undefined (default)	The element does not support being pressed.

aria-readonly property

Indicates that the [element](#) is not editable, but is otherwise [operable](#). See related [aria-disabled](#).

This means the user can read but not set the value of the [widget](#). Readonly elements are relevant to the user, and application authors ***SHOULD NOT*** restrict navigation to the element or its focusable descendants. Other actions such as copying the value of the element are also supported. This is in contrast to disabled elements, to which applications might not allow user navigation to descendants.

Examples include:

- A form element which represents a constant.
- Row or column headers in a spreadsheet grid.
- The result of a calculation such as a shopping cart total.

Characteristics:

Characteristic	Value
Related Concepts:	readonly attribute in [HTML]
Used in Roles:	checkbox combobox grid gridcell listbox radiogroup slider spinbutton textbox
Inherits into Roles:	columnheader rowheader searchbox switch treegrid
Value:	true/false

Values:

Value	Description
false (default)	The user can set the value of the element.
true	The user cannot change the value of the element.

aria-relevant property

Indicates what notifications the user agent will trigger when the accessibility tree within a live region is modified. See related [aria-atomic](#).

The [attribute](#) is represented as a space-separated list of the following values: additions, removals, text; or a single catch-all value all.

This is used to describe [semantically](#) meaningful changes, as opposed to merely presentational ones. For example, nodes that are removed from the top of a log are merely removed for purposes of creating room for other entries, and the removal of them does not have meaning. However, in the case of a buddy list, removal of a buddy name indicates that they are no longer online, and this is a meaningful [event](#). In that case [aria-relevant](#) will be set to all. When the [aria-relevant](#) attribute is not provided, the default value, additions text, indicates that text modifications and node additions are relevant, but that node removals are irrelevant.

NOTE

[aria-relevant](#) values of removals or all are to be used sparingly. Assistive technologies only need to be informed of content removal when its removal represents an important change, such as a buddy leaving a chat room.

NOTE

Text removals should only be considered relevant if one of the specified values is 'removals' or 'all'. For example, for a text change from 'foo' to 'bar' in a live region with a default [aria-relevant](#) value, the text addition ('bar') would be spoken, but the text removal ('foo') would not.

[aria-relevant](#) is an optional attribute of live regions. This is a suggestion to [assistive technologies](#), but assistive technologies are not required to present changes of all the relevant types.

When [aria-relevant](#) is not defined, an element's value is inherited from the nearest ancestor with a defined value. Although the value is a [token list](#), inherited values are not additive; the value provided on a descendant element completely overrides any inherited value from an ancestor element.

When text changes are denoted as relevant, user agents **MUST** monitor any descendant node change that affects the [accessible name and description computation](#) of the live region as if the accessible name were

determined from contents ([nameFrom: contents](#)). For example, a text change would be triggered if the HTML alt attribute of a contained image changed. However, no change would be triggered if there was a text change to a node outside the live region, even if that node was referenced (via [aria-labelledby](#)) by an element contained in the live region.

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	token list

Values:

Value	Description
additions	Element nodes are added to the accessibility tree within the live region.
additions text (default)	Equivalent to the combination of values, "additions text".
all	Equivalent to the combination of all values, "additions removals text".
removals	Text content, a text alternative, or an element node within the live region is removed from the accessibility tree.
text	Text content or a text alternative is added to any descendant in the accessibility tree of the live region.

aria-required property

Indicates that user input is required on the [element](#) before a form may be submitted.

For example, if the user needs to fill in an address field, the author will need to set the field's [aria-required](#) attribute to true.

NOTE

The fact that the element is required is often presented visually (such as a sign or symbol after the [widget](#)). Using the [aria-required attribute](#) allows the author to explicitly convey to [assistive technologies](#) that an element is required.

Unless an exactly equivalent native attribute is available, host languages ***SHOULD*** allow authors to use the [aria-required](#) attribute on host language form elements that require input or selection by the user.

Characteristics:

Characteristic	Value
Related Concepts:	required attribute in [HTML]
Used in Roles:	checkbox combobox gridcell listbox radiogroup spinbutton textbox tree
Inherits into Roles:	columnheader rowheader searchbox switch treegrid
Value:	true/false

Values:

Value	Description
false (default)	User input is not necessary to submit the form.
true	Users need to provide input on an element before a form is submitted.

aria-roledescription property

Defines a human-readable, author-localized description for the [role](#) of an [element](#).

Some [assistive technologies](#), such as screen readers, present the role of an element as part of the user experience. Such assistive technologies typically localize the name of the role, and they may customize it as

well. Users of these assistive technologies depend on the presentation of the role name, such as "region," "button," or "slider," for an understanding of the purpose of the element and, if it is a widget, how to interact with it.

The `aria-roledescription` property gives authors the ability to override how assistive technologies localize and express the name of a role. Thus inappropriately using `aria-roledescription` may inhibit users' ability to understand or interact with an element. Authors **SHOULD** limit use of `aria-roledescription` to clarifying the purpose of non-interactive container roles like [group](#) or [region](#), or to providing a *more specific* description of a [widget](#).

When using `aria-roledescription`, authors **SHOULD** also ensure that:

1. The element to which `aria-roledescription` is applied has a valid [WAI-ARIA](#) role or has an implicit [WAI-ARIA](#) role semantic.
2. The value of `aria-roledescription` is not empty or does not contain only whitespace characters.

User agents **MUST NOT** expose the `aria-roledescription` property if any of the following conditions exist:

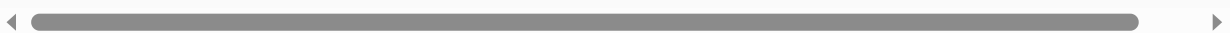
1. The element to which `aria-roledescription` is applied does not have a valid [WAI-ARIA](#) role or does not have an implicit [WAI-ARIA](#) role semantic.
2. The element to which `aria-roledescription` is applied has an explicit or implicit [WAI-ARIA](#) role where `aria-roledescription` is [prohibited](#).
3. The value of `aria-roledescription` is empty or contains only whitespace characters.

[Assistive technologies](#) **SHOULD** use the value of `aria-roledescription` when presenting the role of an element, but **SHOULD NOT** change other functionality based on the role of an element that has a value for `aria-roledescription`. For example, an assistive technology that provides functions for navigating to the next [region](#) or [button](#) **SHOULD** allow those functions to navigate to regions and buttons that have an `aria-roledescription`.

The following two examples show the use of `aria-roledescription` to indicate that a non-interactive container is a "slide" in a web-based presentation application.

[EXAMPLE 27](#)

```
<div role="region" aria-roledescription="slide" id="slide42" aria-labelledby="slide42headi
<h1 id="slide42heading">Quarterly Report</h1>
<!-- remaining slide contents -->
</div>
```



EXAMPLE 28

```
<section aria-roledescription="slide" id="slide42" aria-labelledby="slide42heading">
<h1 id="slide42heading">Quarterly Report</h1>
<!-- remaining slide contents -->
</section>
```

In the previous examples, a screen reader user may hear "Quarterly Report, slide" rather than the more vague "Quarterly Report, region" or "Quarterly Report, group."

Characteristics:

Characteristic	Value
Used in Roles:	All elements of the base markup
Value:	string

aria-rowcount property

Defines the total number of rows in a [table](#), [grid](#), or [treegrid](#). See related [aria-rowindex](#).

If all of the rows are present in the **DOM**, it is not necessary to set this [attribute](#) as the [user agent](#) can automatically calculate the total number of rows. However, if only a portion of the rows is present in the **DOM** at a given moment, this attribute is needed to provide an explicit indication of the number of rows in the full table.

Authors **MUST** set the value of [aria-rowcount](#) to an integer equal to the number of rows in the full table. If the total number of rows is unknown, authors **MUST** set the value of [aria-rowcount](#) to -1 to indicate that the value should not be calculated by the user agent.

The following example shows a grid with 2000 rows, of which the first row and rows 100 through 102 are displayed to the user.

EXAMPLE 29

```
<div role="grid" aria-rowcount="2000">
  <div role="rowgroup">
    <div role="row" aria-rowindex="1">
      <span role="columnheader">First Name</span>
      <span role="columnheader">Last Name</span>
      <span role="columnheader">Company</span>
      <span role="columnheader">Phone</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row" aria-rowindex="100">
      <span role="gridcell">Fred</span>
      <span role="gridcell">Jackson</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1234</span>
    </div>
    <div role="row" aria-rowindex="101">
      <span role="gridcell">Sara</span>
      <span role="gridcell">James</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1235</span>
    </div>
    <div role="row" aria-rowindex="102">
      <span role="gridcell">Taylor</span>
      <span role="gridcell">Johnson</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1236</span>
    </div>
  </div>
</div>
```

Characteristics:

Characteristic	Value
Used in Roles:	table
Inherits into Roles:	grid treegrid
Value:	integer

aria-rowindex property

Defines an [element's](#) row index or position with respect to the total number of rows within a [table](#), [grid](#), or [treegrid](#). See related [aria-rowcount](#) and [aria-rowspan](#).

If all of the rows are present in the **DOM**, it is not necessary to set this [attribute](#) as the [user agent](#) can automatically calculate the index of each row. However, if only a portion of the rows is present in the **DOM** at a given moment, this attribute is needed to provide an explicit indication of each row's position with respect to the full table.

Authors **MUST** set the value for [aria-rowindex](#) to an integer greater than or equal to 1, greater than the [aria-rowindex](#) value of any previous rows, and less than or equal to the number of rows in the full table. For a cell or gridcell which spans multiple rows, authors **MUST** set the value of [aria-rowindex](#) to the start of the span.

Authors **SHOULD** place [aria-rowindex](#) on each row. Authors **MAY** also place [aria-rowindex](#) on all of the children or [owned elements](#) of each row.

The following example shows a grid with 2000 rows, of which the first row and rows 100 through 102 are displayed to the user.

EXAMPLE 30

```
<div role="grid" aria-rowcount="2000">
  <div role="rowgroup">
    <div role="row" aria-rowindex="1">
      <span role="columnheader">First Name</span>
      <span role="columnheader">Last Name</span>
      <span role="columnheader">Company</span>
      <span role="columnheader">Phone</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row" aria-rowindex="100">
      <span role="gridcell">Fred</span>
      <span role="gridcell">Jackson</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1234</span>
    </div>
    <div role="row" aria-rowindex="101">
      <span role="gridcell">Sara</span>
      <span role="gridcell">James</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1235</span>
    </div>
    <div role="row" aria-rowindex="102">
      <span role="gridcell">Taylor</span>
      <span role="gridcell">Johnson</span>
      <span role="gridcell">Acme, Inc.</span>
      <span role="gridcell">555-1236</span>
    </div>
  </div>
</div>
```

The following example shows the grid from the previous example with [aria-rowindex](#) also placed on all of the owned elements of each row.

EXAMPLE 31

```
<div role="grid" aria-rowcount="2000">
  <div role="rowgroup">
    <div role="row" aria-rowindex="1">
      <span role="columnheader" aria-rowindex="1">First Name</span>
      <span role="columnheader" aria-rowindex="1">Last Name</span>
      <span role="columnheader" aria-rowindex="1">Company</span>
      <span role="columnheader" aria-rowindex="1">Phone</span>
    </div>
  </div>
  <div role="rowgroup">
    <div role="row" aria-rowindex="100">
      <span role="gridcell" aria-rowindex="100">Fred</span>
      <span role="gridcell" aria-rowindex="100">Jackson</span>
      <span role="gridcell" aria-rowindex="100">Acme, Inc.</span>
      <span role="gridcell" aria-rowindex="100">555-1234</span>
    </div>
    <div role="row" aria-rowindex="101">
      <span role="gridcell" aria-rowindex="101">Sara</span>
      <span role="gridcell" aria-rowindex="101">James</span>
      <span role="gridcell" aria-rowindex="101">Acme, Inc.</span>
      <span role="gridcell" aria-rowindex="101">555-1235</span>
    </div>
    <div role="row" aria-rowindex="102">
      <span role="gridcell" aria-rowindex="102">Taylor</span>
      <span role="gridcell" aria-rowindex="102">Johnson</span>
      <span role="gridcell" aria-rowindex="102">Acme, Inc.</span>
      <span role="gridcell" aria-rowindex="102">555-1236</span>
    </div>
  </div>
</div>
```

Characteristics:

Characteristic	Value
Used in Roles:	cell row
Inherits into Roles:	columnheader gridcell rowheader
Value:	integer

aria-rowspan property

Defines the number of rows spanned by a cell or gridcell within a [table](#), [grid](#), or [treegrid](#). See related [aria-rowindex](#) and [aria-colspan](#).

This [attribute](#) is intended for cells and gridcells which are not contained in a native table. When defining the row span of cells or gridcells in a native table, authors **SHOULD** use the host language's attribute instead of [aria-rowspan](#). If [aria-rowspan](#) is used on an element for which the host language provides an equivalent attribute, [user agents](#) **MUST** ignore the value of [aria-rowspan](#) and instead expose the value of the host language's attribute to [assistive technologies](#).

Authors **MUST** set the value of [aria-rowspan](#) to an integer greater than or equal to 0 and less than the value which would cause the cell or gridcell to overlap the next cell or gridcell in the same column. Setting the value to 0 indicates that the cell or gridcell is to span all the remaining rows in the row group.

Characteristics:

Characteristic	Value
Used in Roles:	cell
Inherits into Roles:	columnheader rowheader
Value:	integer

aria-selected state

Indicates the current "selected" [state](#) of various [widgets](#). See related [aria-checked](#) and [aria-pressed](#).

This [attribute](#) is used with single-selection and multiple-selection widgets:

1. Single-selection containers where the currently focused item is not selected. The selection normally follows the focus, and is managed by the [user agent](#).
2. Multiple-selection containers. Authors **SHOULD** ensure that any selectable descendant of a container in which the [aria-multiselectable](#) attribute is true specifies a value of either true or false for the [aria-selected](#) attribute.

Any explicit assignment of [aria-selected](#) takes precedence over the implicit selection based on focus. If no **DOM** element in the widget is explicitly marked as selected, assistive technologies **MAY** convey implicit selection which follows the keyboard focus of the [managed focus](#) widget. If any **DOM** element in the widget is explicitly marked as selected, the user agent **MUST NOT** convey implicit selection for the widget.

Characteristics:

Characteristic	Value
Used in Roles:	gridcell option row tab
Inherits into Roles:	columnheader rowheader treeitem
Value:	true/false/undefined

Values:

Value	Description
false	The selectable element is not selected.
true	The selectable element is selected.
undefined (default)	The element is not selectable.

aria-setsize property

Defines the number of items in the current set of listitems or treeitems. Not required if all elements in the set are present in the DOM. See related [aria-posinset](#).

This [property](#) is marked on the members of a set, not the container element that collects the members of the set. To orient the user by saying an element is "item X out of Y," the [assistive technologies](#) would use X equal to the [aria-posinset attribute](#) and Y equal to the `aria-setsize` attribute.

If all items in a set are present in the document structure, it is not necessary to set this property, as the [user agent](#) can automatically calculate the set size and position for each item. However, if only a portion of the set is present in the document structure at a given moment (in order to reduce document size), this property is needed to provide an explicit indication of set size.

Authors **MUST** set the value of `aria-setsize` to an integer equal to the number of items in the set. If the total number of items is unknown, authors **SHOULD** set the value of `aria-setsize` to -1.

When exposing `aria-setsize` on a [menuitem](#), [menuitemcheckbox](#), or [menuitemradio](#), authors **SHOULD** set the value of `aria-setsize` based on the total number of items in the [menu](#), excluding any

separators.

The following example shows items 5 through 8 in a set of 16.

EXAMPLE 32

```
<h2 id="label_fruit"> Available Fruit </h2>
<ul role="listbox" aria-labelledby="label_fruit">
  <li role="option" aria-setsize="16" aria-posinset="5"> apples </li>
  <li role="option" aria-setsize="16" aria-posinset="6"> bananas </li>
  <li role="option" aria-setsize="16" aria-posinset="7"> cantaloupes </li>
  <li role="option" aria-setsize="16" aria-posinset="8"> dates </li>
</ul>
```

The following example shows items 5 through 8 in a set whose total size is unknown.

EXAMPLE 33

```
<h2 id="label_fruit"> Available Fruit </h2>
<ul role="listbox" aria-labelledby="label_fruit">
  <li role="option" aria-setsize="-1" aria-posinset="5"> apples </li>
  <li role="option" aria-setsize="-1" aria-posinset="6"> bananas </li>
  <li role="option" aria-setsize="-1" aria-posinset="7"> cantaloupes </li>
  <li role="option" aria-setsize="-1" aria-posinset="8"> dates </li>
</ul>
```

Characteristics:

Characteristic	Value
Used in Roles:	article listitem menuitem option radio row tab
Inherits into Roles:	menuitemcheckbox menuitemradio treeitem

Characteristic	Value
Value:	integer

aria-sort property

Indicates if items in a table or grid are sorted in ascending or descending order.

Authors **SHOULD** only apply this [property](#) to table headers or grid headers. If the property is not provided, there is no defined sort order. For each table or grid, authors **SHOULD** apply [aria-sort](#) to only one header at a time.

Characteristics:

Characteristic	Value
Used in Roles:	columnheader rowheader
Value:	token

Values:

Value	Description
ascending	Items are sorted in ascending order by this column.
descending	Items are sorted in descending order by this column.
none (default)	There is no defined sort applied to the column.
other	A sort algorithm other than ascending or descending has been applied.

aria-valuemax property

Defines the maximum allowed value for a range [widget](#).

Authors **MUST** ensure the value of [aria-valuemax](#) is greater than or equal to the value of [aria-valuemin](#). If the [aria-valuenow](#) has a known maximum and minimum, the author **SHOULD** provide properties for [aria-valuemax](#) and [aria-valuemin](#).

NOTE

A range widget starts with a given value, which can be increased until reaching the maximum value, defined by this [property](#). Declaring the minimum and maximum values allows assistive technology to convey the size of the range to users.

Characteristics:

Characteristic	Value
Related Concepts:	<range> element max attribute in [HTML]
Used in Roles:	range scrollbar separator slider spinbutton
Inherits into Roles:	meter progressbar scrollbar slider spinbutton
Value:	number

aria-valuemin property

Defines the minimum allowed value for a range [widget](#).

Authors **MUST** ensure the value of [aria-valuemin](#) is less than or equal to the value of [aria-valuemax](#). If the [aria-valuenow](#) has a known maximum and minimum, the author **SHOULD** provide properties for [aria-valuemax](#) and [aria-valuemin](#).

NOTE

A range widget starts with a given value, which can be decreased until reaching the minimum value, defined by this [property](#). Declaring the minimum and maximum values allows assistive technology to convey the size of the range to users.

Characteristics:

Characteristic	Value
Related Concepts:	<range> element min attribute in [HTML]
Used in Roles:	range scrollbar separator slider spinbutton
Inherits into Roles:	meter progressbar scrollbar slider spinbutton
Value:	number

aria-valuenow property

Defines the current value for a range [widget](#). See related [aria-valuetext](#).

This property is used, for example, on a range widget such as a slider or progress bar.

If the current value is not known (for example, an indeterminate progress bar), the author **SHOULD NOT** set the [aria-valuenow](#) attribute. If the [aria-valuenow](#) attribute is absent, no information is implied about the current value. If the [aria-valuenow](#) has a known maximum and minimum, the author **SHOULD** provide properties for [aria-valuemax](#) and [aria-valuemin](#).

The value of [aria-valuenow](#) is a decimal number. If the range is a set of numeric values, then [aria-valuenow](#) is one of those values. For example, if the range is [0, 1], a valid [aria-valuenow](#) is 0.5. A value outside the range, such as -2.5 or 1.1, is invalid.

For [progressbar](#) elements and [scrollbar](#) elements, assistive technologies **SHOULD** render the value to users as a percent, calculated as a position on the range from [aria-valuemin](#) to [aria-valuemax](#) if both are defined, otherwise the actual value with a percent indicator. For elements with role [slider](#) and [spinbutton](#), assistive technologies **SHOULD** render the actual value to users.

When the rendered value cannot be accurately represented as a number, authors **SHOULD** use the [aria-valuetext](#) attribute in conjunction with [aria-valuenow](#) to provide a user-friendly representation of the range's current value. For example, a slider may have rendered values of small, medium, and large. In this case, the values of [aria-valuetext](#) would be one of the strings: small, medium, or large.

NOTE

If [aria-valuetext](#) is specified, assistive technologies render that instead of the value of [aria-valuenow](#).

Characteristics:

Characteristic	Value
Related Concepts:	<range> element value attribute in [HTML]
Used in Roles:	meter range scrollbar separator slider spinbutton
Inherits into Roles:	meter progressbar scrollbar slider spinbutton
Value:	number

aria-valuetext property

Defines the human readable text alternative of [aria-valuenow](#) for a range [widget](#).

This property is used, for example, on a range widget such as a slider or progress bar.

If the [aria-valuetext](#) attribute is set, authors **SHOULD** also set the [aria-valuenow](#) attribute, unless that value is unknown (for example, on an indeterminate [progressbar](#)).

Authors **SHOULD** only set the [aria-valuetext](#) attribute when the rendered value cannot be meaningfully represented as a number. For example, a slider may have rendered values of small, medium, and large. In this case, the values of [aria-valuenow](#) could range from 1 through 3, which indicate the position of each value in the value space, but the [aria-valuetext](#) would be one of the strings: small, medium, or large. If the [aria-valuetext](#) attribute is absent, the [assistive technologies](#) will rely solely on the [aria-valuenow](#) attribute for the current value.

If [aria-valuetext](#) is specified, assistive technologies **SHOULD** render that value instead of the value of [aria-valuenow](#).

Characteristics:

Characteristic	Value
Used in Roles:	range separator spinbutton
Inherits into Roles:	meter progressbar scrollbar slider spinbutton
Value:	string

§ 7. Accessibility Tree

The [accessibility tree](#) and the [DOM](#) tree are parallel structures. The [accessibility tree](#) includes the user interface objects of the [user agent](#) and the objects of the document. [Accessible objects](#) are created in the accessibility tree for every [DOM](#) element that should be exposed to an [assistive technology](#), either because it may fire an accessibility [event](#) or because it has a [property](#), [relationship](#) or feature which needs to be exposed.

§ 7.1 Excluding Elements from the Accessibility Tree

The following [elements](#) are not exposed via the [accessibility API](#) and user agents **MUST NOT** include them in the [accessibility tree](#):

- Elements, including their descendent elements, that have host language semantics specifying that the element is not displayed, such as [CSS](#) `display:none`, `visibility:hidden`, or the [HTML](#) `hidden` attribute.
- Elements with [none](#) or [presentation](#) as the first role in the role attribute. However, their exclusion is conditional. In addition, the element's descendants and text content are generally included. These exceptions and conditions are documented in the [presentation \(role\)](#) section.

If not already excluded from the accessibility tree per the above rules, user agents **SHOULD NOT** include the following elements in the accessibility tree:

- Elements, including their descendants, that have [aria-hidden](#) set to `true`. In other words, `aria-hidden="true"` on a parent overrides `aria-hidden="false"` on descendants.
- Any descendants of elements that have the characteristic "[Children Presentational: True](#)" unless the descendant is not allowed to be presentational because it meets one of the conditions for exception described in [Presentational Roles Conflict Resolution](#). However, the text content of any excluded descendants is included.

Elements with the following roles have the characteristic "Children Presentational: True":

- [button](#)
- [checkbox](#)
- [img](#)
- [menuitemcheckbox](#)
- [menuitemradio](#)
- [meter](#)
- [option](#)
- [progressbar](#)
- [radio](#)
- [scrollbar](#)
- [separator](#)

- [slider](#)
- [switch](#)
- [tab](#)

§ 7.2 Including Elements in the Accessibility Tree

If not excluded from or marked as hidden in the accessibility tree per the rules above in [Excluding Elements in the Accessibility Tree](#), user agents **MUST** provide an [accessible object](#) in the [accessibility tree](#) for [DOM elements](#) that meet any of the following criteria:

- Elements that are not [hidden](#) and may fire an [accessibility API event](#), including:
 - Elements that are currently focused, even if the element or one of its ancestor elements has its [aria-hidden](#) attribute set to true.
 - Elements that are a valid target of an [aria-activedescendant](#) attribute.
- Elements that have an explicit role or a global [WAI-ARIA](#) attribute and do not have [aria-hidden](#) set to true. (See [Excluding Elements in the Accessibility Tree](#) for additional guidance on [aria-hidden](#).)
- Elements that are not [hidden](#) and have an ID that is referenced by another element via a [WAI-ARIA](#) property.

NOTE

Text equivalents for hidden referenced objects may still be used in the [name and description computation](#) even when not included in the accessibility tree.

§ 8. Implementation in Host Languages

The [roles](#), [state](#), and [properties](#) defined in this specification do not form a complete web language or format. They are intended to be used in the context of a host language. This section discusses how host languages are to implement [WAI-ARIA](#), to ensure that the markup specified here will integrate smoothly and effectively with the host language markup.

Although markup languages look alike superficially, they do not share language definition infrastructure. To accommodate differences in language-building approaches, the requirements are both general and modularization-specific. While allowing for differences in how the specifications are written, the intent is to

maintain consistency in how the WAI-ARIA information looks to authors and how it is manipulated in the DOM by scripts.

WAI-ARIA roles, states, and properties are implemented as attributes of elements. Roles are applied by placing their names among the tokens appearing in the value of a host-language-provided `role` attribute. States and properties each get their own attribute, with values as defined for each particular state or property in this specification. The name of the attribute is the aria-prefixed name of the state or property.

§ 8.1 Role Attribute

An implementing host language will provide an attribute with the following characteristics:

- The attribute name **MUST** be `role`;
- The attribute value **MUST** allow a token list as the value;
- The appearance of the name literal of any concrete WAI-ARIA role as one of these tokens **MUST NOT** in and of itself make the attribute value illegal in the host-language syntax; and
- The first name literal of a non-abstract WAI-ARIA role in the list of tokens in the role attribute defines the role according to which the user agent **MUST** process the element. User Agent processing for roles is defined in the Core Accessibility API Mappings [CORE-AAM-1.2].

§ 8.2 State and Property Attributes

An implementing host language **MUST** allow attributes with the following characteristics:

- The attribute name is the name of any state or property identified in the Supported States and Properties section, such as `aria-busy`, `aria-selected`, `aria-activedescendant`, `aria-valuetext`;
- The syntax does **NOT** prevent the attribute from appearing anywhere that it is applicable, as specified in this specification;
- When these attributes appear in a document instance, the attributes will be processed as defined in this specification.

Host languages that support XML Namespaces [XML-NAMES] **MAY** require that WAI-ARIA attributes be used with a namespace. In this case, the namespace for WAI-ARIA state and property attributes **MUST** be `http://www.w3.org/ns/wai-aria/`. To use WAI-ARIA in host languages that do not explicitly describe support for it, authors **SHOULD** use this namespace as well, if the host language supports namespaces and

there is expectation that user agents will recognize the `WAI-ARIA` namespace. The namespace prefix is not defined by this specification but generally is expected to be `"aria"`.

NOTE

The `WAI-ARIA` state and property attributes have a naming convention such that they all begin with the string `"aria-"`. This is *not* a namespace prefix, it is a part of the state or property name. Therefore, when using `WAI-ARIA` states and properties with namespace prefixes, the complete attribute name will be like `"aria:aria-foo"`.

Some host languages do not use namespaces with `WAI-ARIA` state and property attributes, either because the host language does not support namespaces or because the designers wish to incorporate `WAI-ARIA` into the core feature set. In these host languages, the namespace name for these attributes has no value. The names of these attributes do not have a prefix offset by a colon; in the terms of namespaces they are unprefixed attribute names. The ECMAScript binding of the `DOM` interface `getAttributeNS` for example, treats an empty string (`""`) as representing this condition, so that both `getAttribute("aria-busy")` and `getAttributeNS("", "aria-busy")` access the same [aria-busy](#) attribute in the `DOM`.

NOTE

According to the requirements of this section, some user agents recognize `WAI-ARIA` state and property attributes *with* namespaces, some *without* namespaces, and some might recognize both. Authors are advised to be aware of which form is supported for the host language they are using. Unless the host language and supporting user agents explicitly indicate that the namespace is required, authors are advised to use the attribute without namespaces. Even user agents that support namespaces generally do not publish namespaced `WAI-ARIA` states and properties to accessibility APIs. In particular, current implementations of `HTML`, including `XHTML`, do not support this namespace.

§ 8.3 Focus Navigation

An implementing host language **MUST** provide support for the author to make all interactive elements focusable, that is, any renderable or event-receiving elements. An implementing host language **MUST** provide a facility to allow web authors to define whether these focusable, interactive elements appear in the default tab navigation order. The `tabindex` [attribute](#) in `HTML` is an example of one implementation.

§ 8.4 Implicit WAI-ARIA Semantics

WAI-ARIA is designed to provide [semantic](#) information about objects when host languages lack native semantics for the object. WAI-ARIA is designed, however, to provide additional semantics for many host languages. Furthermore, host languages over time can evolve and provide new native features that correspond to WAI-ARIA features. Therefore, there are many situations in which WAI-ARIA semantics are redundant with host language semantics.

These host language features can be viewed as having "implicit WAI-ARIA semantics". User agent processing of features with implicit WAI-ARIA semantics would be similar to the processing for the WAI-ARIA feature. The processing might not be identical because of lexical differences between the host language feature and the WAI-ARIA feature, but generally the user agent would expose the same information to the accessibility API. Features with implicit WAI-ARIA semantics satisfy WAI-ARIA structural requirements such as required owned elements, required states and properties, etc. and do not require explicit WAI-ARIA semantics to be provided. On elements with implicit WAI-ARIA roles, authors can also use WAI-ARIA states and properties supported by those roles *without* requiring explicit indication of the WAI-ARIA role.

For example, if an element with the functionality already exists, such as a checkbox or radio button, use the native semantics of the host language. WAI-ARIA markup is only intended to be used to enhance the native semantics (e.g., indicating that the element is required with [aria-required](#)), or to change the semantics to a different purpose from the standard functionality of the element.

Implicit WAI-ARIA semantics affect the conflict resolution procedures in the following section, Conflicts with Host Language Semantics. Therefore, implicit WAI-ARIA semantics need to be defined in a normative specification, such as the host language specification or the [Core Accessibility API Mappings](#).

§ 8.5 Conflicts with Host Language Semantics

WAI-ARIA roles, states, and properties are intended to add [semantic](#) information when native host language elements with these semantics are not available, and are generally used on elements that have no native semantics of their own. They can also be used on elements that have similar but non-identical semantics (for example, a nested list could be used to represent a tree structure). This method can be part of a fallback strategy for older browsers that have no WAI-ARIA implementation, or because native presentation of the repurposed element reduces the amount of style and/or script needed. Except for the cases outlined below, user agents **MUST** always use the WAI-ARIA semantics to define how it exposes the element to accessibility APIs, rather than using the host language semantics.

In addition to these normal situations in which `WAI-ARIA` is expected to override native semantics, there are elements that are inappropriate to override with `WAI-ARIA`. This could be because identical host language semantics exist, so `WAI-ARIA` is not needed, or because semantics from `WAI-ARIA` directly conflict with host language semantics. When a feature in the host language with identical role semantics and values is available, and the author has no compelling reason to avoid using the host language feature, authors **SHOULD** use the host language features rather than repurpose other elements with `WAI-ARIA`.

Host languages can have features that have implicit `WAI-ARIA` semantics corresponding to roles. When a `WAI-ARIA` role is provided, user agents **MUST** use the semantic of the `WAI-ARIA` role for processing, not the native semantic, unless the role requires `WAI-ARIA` states and properties whose attributes are explicitly forbidden on the native element by the host language. Values for roles do not conflict in the same way as values for states and properties (for example, the `HTML` 'checked' attribute and the 'aria-checked' attribute could have conflicting values), and authors are expected to have valid reason to provide a `WAI-ARIA` role even on elements that would not normally be repurposed.

When `WAI-ARIA` states and properties correspond to host language features that have the same [implicit `WAI-ARIA` semantic](#), it can be particularly problematic to use the `WAI-ARIA` feature. If the `WAI-ARIA` feature and the host language feature are both provided but their values are not kept in sync, user agents and assistive technologies cannot know which value to use. Therefore, to prevent providing conflicting states and properties to assistive technologies, host languages **MUST** explicitly declare where the use of `WAI-ARIA` attributes on each host language element conflicts with native attributes for that element. When a host language declares a `WAI-ARIA` attribute to be in direct semantic conflict with a native attribute for a given element, user agents **MUST** ignore the `WAI-ARIA` attribute and instead use the host language attribute with the same implicit semantic.

Host languages **MAY** document features that cannot be overridden with `WAI-ARIA` (these are called "strong native semantics"). These can be features that have implicit `WAI-ARIA` semantics, as well as features where the processing would be uncertain if the semantics were changed with `WAI-ARIA`. Conformance checkers **MAY** signal an error or warning when a `WAI-ARIA` role is used on elements with strong native semantics, but as described above, user agents **MUST** still use the value of the semantic of the `WAI-ARIA` role when exposing the element to accessibility `APIs` unless the native host language semantic is permanently presentational.

The opportunity for host languages to create exceptions to the `WAI-ARIA` override of native features is meant to avoid potential author errors or problems with intrinsic processing of host language features. Author errors could happen when a host language and `WAI-ARIA` provide similar but not identical features, where it might not be clear how changing one but not the other affects the accessibility `API`. Intrinsic processing refers to the way a feature is processed, beyond simple rendering and exposure to the Accessibility `API`, that cannot reasonably be changed in response to an ARIA feature, and would lead to unpredictable results were ARIA allowed. In these situations, there is good reason for host languages to limit the scope of `WAI-ARIA`. However, this provision does not give blanket permission for host languages to forbid the use of `WAI-ARIA` simply by documenting, feature by feature, that it may not be used. Host

languages should create restrictions on the use of ARIA only when it is critical to effective processing of content.

Certain ARIA features are critical to building a complete model in the accessibility API. Such features are not expected to conflict with native host language semantics (though they may complement them).

Therefore, host languages **MUST NOT** declare strong native semantics that prevent use of the following ARIA features:

- [aria-describedby](#)
- [aria-label](#)
- [aria-labelledby](#)

§ 8.6 State and Property Attribute Processing

State and property attributes are included in host languages, and therefore syntax for representation of their value types is governed by the host language. For each of the value types defined in [Value](#), an appropriate value type from the host language is used. Recommended correspondences between WAI-ARIA value types and various host language value types are listed in [Mapping WAI-ARIA Value types to languages](#). This is a non-normative mapping in order to accommodate new host languages supporting WAI-ARIA.

The list value types—ID reference list and token list—allow more than one value of the given type to be provided. The values are separated by delimiter characters recognized by the host language for list attributes, such as space characters, commas, etc. Some languages may require a specific, single delimiter, while others may allow various delimiters.

Global states and properties are supported on any element in the host language. However, authors **MUST** only use non-global states and properties on elements with a role supporting the state or property; either defined as an explicit WAI-ARIA role, or as defined by the host language implicit WAI-ARIA semantic matching an appropriate WAI-ARIA role. When a role attribute is added to an element, the [semantics](#) and behavior of the element, including support for WAI-ARIA states and properties, are augmented or overridden by the role behavior. User agents **MUST** ignore non-global states and properties used on an element without a role supporting the state or property; either defined as an explicit WAI-ARIA role, or as defined by the host language WAI-ARIA semantic matching an appropriate WAI-ARIA role. For example, the [aria-valuetext](#) attribute may be used on a [progressbar](#).

WAI-ARIA roles have associated states and properties that are qualified as "supported" or "required". An example of a property *supported* by the [combobox](#) role is [aria-autocomplete](#). The property is designated "supported" in this case because a given combobox might or might not implement auto completion. In contrast, the combobox role *requires* the [aria-expanded](#) state in order to indicate that it is expandable.

Comboboxes have a controlled popup element, such as a `listbox`, that is either open or closed. If the `listbox` is open, the combobox is in its expanded state; otherwise it is collapsed.

When WAI-ARIA roles are used, *supported* states and properties that are not present in the DOM are treated according to their default value. Keeping with the combobox example, a missing `aria-autocomplete` attribute is equivalent to `aria-autocomplete="none"`, meaning the combobox does not offer auto completion.

However, *required* states and properties that are absent are an author error. Missing required states and properties are treated as if they were present and have an implicit neutral value that is not necessarily their default value. For example, the default value of `aria-expanded` is undefined, meaning neither expandable nor collapsible. But that does not apply to the case of a combobox. In this case, `aria-expanded` is needed to convey the expandable/collapsible nature of the combobox. Thus, the implicit value of `aria-expanded` for the combobox role is `false`, meaning expandable (and currently collapsed). The characteristics table associated with each WAI-ARIA role has an "[Implicit Value for Role](#)" entry that specifies the value of a state or property to use in the context of that role when the state or property is missing.

Elements that have implicit WAI-ARIA semantics support the full set of WAI-ARIA states and properties supported by the corresponding role. Therefore, authors **MAY** omit the role when setting states and properties. The role is only needed when the implicit WAI-ARIA role of the element needs to be changed.

Sometimes states and properties are present in the DOM but have a zero-length string ("") as their value. Authors **MAY** specify a zero-length string ("") for any supported (but not required) state or property. User agents **SHOULD** treat state and property attributes with a value of "" the same as they treat an absent attribute. For supported states and properties, this corresponds to the default value, but if it is a required attribute, it signals an author error, and the implicit value for the role is used.

§ 8.6.1 ID Reference Error Processing

User agents **SHOULD** ignore ID references that do not match the ID of another element in the same document.

It is the web author's responsibility to ensure that IDs are unique. If more than one element has the same ID, the user agent **SHOULD** use the first element found with the given ID. The behavior will be the same as `getElementById`.

If the same element is specified multiple times in a single WAI-ARIA relation, user agents **SHOULD** return multiple pointers to the same element.

[aria-activedescendant](#) is defined as referencing only a single ID reference. Any `aria-activedescendant` value that does not match an existing ID reference exactly is an author error and will not match any element in the [DOM](#).

§ 8.7 [CSS Selectors](#)

NOTE

This section might be removed in a future version.

Support for [attribute](#) selectors **MUST** include [WAI-ARIA](#) attributes. For example, `.fooMenuItem[aria-haspopup="true"]` would select all [elements](#) with class `fooMenuItem`, and [WAI-ARIA](#) property [aria-haspopup](#) with value of `true`. The presentation **MUST** be updated for dynamic changes to [WAI-ARIA](#) attributes. This allows authors to match styling with [WAI-ARIA semantics](#).

§ 9. Handling Author Errors

§ 9.1 Roles

User agents are expected to perform validation of [WAI-ARIA roles](#).

As stated in the [Definition of Roles](#) section, it is considered an authoring error to use [abstract roles](#) in content. User agents **MUST NOT** map abstract roles via the standard role mechanism of the accessibility [API](#).

If the `role` attribute contains no tokens matching the name of a non-abstract [WAI-ARIA](#) role, the user agent **MUST** treat the element as if no [role](#) had been provided. For example, `<table role="foo">` should be exposed in the same way as `<table>` and `<input type="text" role="structure">` in the same way as `<input type="text">`.

§ 9.2 States and Properties

In general, [user agents](#) do not do much validation of [WAI-ARIA properties](#). User agents **MAY** do some minor validation on request, such as making sure [valid IDs](#) are specified for [WAI-ARIA](#) relations, and enforcing things like [aria-posinset](#) being within 1 and [aria-setsize](#), inclusive. User agents are not responsible for logical validation, such as the following:

1. Circular references created by relations, such as specifying that two [elements](#) own each other.
2. Correct usage with regard to [DOM](#) tree structure, such as an [element](#) being owned by more than one other element.
3. Elements with [WAI-ARIA roles](#) correctly implement the behavior of the specified role. For example, user agents do not verify that an element with a role of [checkbox](#) actually behaves like a checkbox.
4. Elements that do not correctly observe required child / parent role relationships or that appear elsewhere than in their required parent.
5. Determining whether [aria-activedescendant](#) actually points to an [owned](#) element of the container widget.
6. Determining implicit values of [aria-setsize](#) and [aria-posinset](#) when they are specified on some but not all the elements of the set.

If the author specifies a non-numeric value for a decimal or integer value type, the user agent **SHOULD** do the following:

- When asked for the string version of the property, return the string if specified by the author.
- When asked for the numeric version:
 - Follow the guidance in the [Fallback values for missing required attributes](#) table below, if applicable.
 - Otherwise, return a fallback value of 0.0 for decimal value types and 0 for integer value types.

If a [WAI-ARIA](#) property contains an unknown or disallowed value, the user agent **SHOULD** expose to platform [accessibility APIs](#) as follows:

- When exposing as a platform accessibility [API](#) attribute, expose the unknown value — do not vet it against possible values.
- When exposing as a platform [API](#) Boolean state:
 - For values of "" (empty string), "undefined" or no [attribute](#) present:
 - Follow the guidance in the [Fallback values for missing required attributes](#) table below, if applicable.

- Otherwise, treat as false.
- Treat any other value as true.
- Otherwise, ignore the value and treat the property as not present.

NOTE

In UIA, the user agent might leave the corresponding property set to "unsupported."

User agents **MUST NOT** expose WAI-ARIA attributes that reference unresolved IDs. For example:

- When the state or property has only one ID reference that cannot be resolved, treat as if the state or property is not present.
- When the state or property has a list of ID references, ignore any that can't be resolved. If none in the list can be resolved, treat as if the state or property is not present.

User Agents **MUST NOT** expose aria-roledescription when:

- The element it is applied to has an invalid WAI-ARIA role, or
- The element does not have an implicit WAI-ARIA role

If a required WAI-ARIA attribute for a given role is missing, user agents **SHOULD** process the attribute as if the values given in the following table were provided.

Fallback values for missing required attributes

<u>WAI-ARIA</u> role	Required Attribute	Fallback value
<u>checkbox</u>	<u>aria-checked</u>	false
<u>combobox</u>	<u>aria-controls</u>	no mapping
<u>combobox</u>	<u>aria-expanded</u>	false
<u>heading</u>	<u>aria-level</u>	2
<u>menuitemcheckbox</u>	<u>aria-checked</u>	false
<u>menuitemradio</u>	<u>aria-checked</u>	false

WAI-ARIA role	Required Attribute	Fallback value
radio	aria-checked	false
scrollbar	aria-controls	no mapping
scrollbar	aria-valuenow	If missing or not a number , (aria-valuemax - aria-valuemin) / 2. If present but less than aria-valuemin, the value of aria-valuemin. If present but greater than aria-valuemax, the value of aria-valuemax.
separator (if focusable)	aria-valuenow	If missing or not a number , (aria-valuemax - aria-valuemin) / 2. If present but less than aria-valuemin, the value of aria-valuemin. If present but greater than aria-valuemax, the value of aria-valuemax.
slider	aria-valuenow	If missing or not a number , (aria-valuemax - aria-valuemin) / 2. If present but less than aria-valuemin, the value of aria-valuemin. If present but greater than aria-valuemax, the value of aria-valuemax.
switch	aria-checked	false
meter	aria-valuenow	A value matching the implicit or explicitly set aria-valuemin.

NOTE

Implicit Values for non-required states and properties appear in the characteristics table for each role. These are not considered fallback values so are not included here.

§ 10. IDL Interface

Conforming user agents **MUST** implement the following IDL interface.

§ 10.1 Interface Mixin *ARIAMixin*

WebIDL

```
interface mixin ARIAMixin {
    attribute DOMString? role;

    attribute DOMString? ariaAtomic;
    attribute DOMString? ariaAutoComplete;
    attribute DOMString? ariaBusy;
    attribute DOMString? ariaChecked;
    attribute DOMString? ariaColCount;
    attribute DOMString? ariaColIndex;

    attribute DOMString? ariaColSpan;

    attribute DOMString? ariaCurrent;

    attribute DOMString? ariaDisabled;

    attribute DOMString? ariaExpanded;

    attribute DOMString? ariaHasPopup;
    attribute DOMString? ariaHidden;
    attribute DOMString? ariaInvalid;
    attribute DOMString? ariaKeyShortcuts;
    attribute DOMString? ariaLabel;

    attribute DOMString? ariaLevel;
    attribute DOMString? ariaLive;
    attribute DOMString? ariaModal;
    attribute DOMString? ariaMultiLine;
    attribute DOMString? ariaMultiSelectable;
    attribute DOMString? ariaOrientation;

    attribute DOMString? ariaPlaceholder;
    attribute DOMString? ariaPosInSet;
    attribute DOMString? ariaPressed;
    attribute DOMString? ariaReadOnly;

    attribute DOMString? ariaRequired;
    attribute DOMString? ariaRoleDescription;
    attribute DOMString? ariaRowCount;
    attribute DOMString? ariaRowIndex;

    attribute DOMString? ariaRowSpan;
    attribute DOMString? ariaSelected;
    attribute DOMString? ariaSetSize;
    attribute DOMString? ariaSort;
    attribute DOMString? ariaValueMax;
    attribute DOMString? ariaValueMin;
```

```
    attribute DOMString? ariaValueNow;  
    attribute DOMString? ariaValueText;  
};
```

Interfaces that include `ARIAMixin` must provide the following algorithms:

- **ARIAMixin getter steps**, which take the host interface instance, IDL attribute name, and content attribute name, and must return a string value; and
- **ARIAMixin setter steps**, which take the host interface instance, IDL attribute name, content attribute name, and string value, and must return nothing.

For every IDL attribute *idlAttribute* defined in `ARIAMixin`, on getting, it must perform the following steps:

1. Let *contentAttribute* be the ARIA content attribute determined by looking up *idlAttribute* in the ARIA Attribute Correspondence table.
2. Return the result of running the [ARIAMixin getter steps](#), given this, *idlAttribute*, and *contentAttribute*.

Similarly, on setting, it must perform the following steps:

1. Let *contentAttribute* be the ARIA content attribute determined by looking up *idlAttribute* in the ARIA Attribute Correspondence table.
2. Run the [ARIAMixin setter steps](#), given this, *idlAttribute*, *contentAttribute*, and the given value.

NOTE

This very general framework is motivated by the desire for different host interfaces, such as `Element` and `ElementInternals`, to give these IDL attributes different behaviors. The alternative is requiring each host interface to duplicate the IDL attributes independently, so that they can specify independent behaviors, but that comes with a high risk of them getting out of sync.

§ 10.2 ARIA Attribute Correspondence

The following table provides a correspondence between IDL attribute names and content attribute names, for use by `ARIAMixin`.

IDL Attribute	Reflected ARIA Content Attribute
<i>role</i>	role
<i>ariaAtomic</i>	aria-atomic
<i>ariaAutoComplete</i>	aria-autocomplete

<i>ariaBusy</i>	<u>aria-busy</u>
<i>ariaChecked</i>	<u>aria-checked</u>
<i>ariaColCount</i>	<u>aria-colcount</u>
<i>ariaColIndex</i>	<u>aria-colindex</u>
<i>ariaColSpan</i>	<u>aria-colspan</u>
<i>ariaCurrent</i>	<u>aria-current</u>
<i>ariaDisabled</i>	<u>aria-disabled</u>
<i>ariaExpanded</i>	<u>aria-expanded</u>
<i>ariaHasPopup</i>	<u>aria-haspopup</u>
<i>ariaHidden</i>	<u>aria-hidden</u>
<i>ariaInvalid</i>	<u>aria-invalid</u>
<i>ariaKeyShortcuts</i>	<u>aria-keyshortcuts</u>
<i>ariaLabel</i>	<u>aria-label</u>
<i>ariaLevel</i>	<u>aria-level</u>
<i>ariaLive</i>	<u>aria-live</u>
<i>ariaModal</i>	<u>aria-modal</u>
<i>ariaMultiLine</i>	<u>aria-multiline</u>
<i>ariaMultiSelectable</i>	<u>aria-multiselectable</u>
<i>ariaOrientation</i>	<u>aria-orientation</u>
<i>ariaPlaceholder</i>	<u>aria-placeholder</u>
<i>ariaPosInSet</i>	<u>aria-posinset</u>
<i>ariaPressed</i>	<u>aria-pressed</u>
<i>ariaReadOnly</i>	<u>aria-readonly</u>
<i>ariaRequired</i>	<u>aria-required</u>
<i>ariaRoleDescription</i>	<u>aria-roledescription</u>
<i>ariaRowCount</i>	<u>aria-rowcount</u>
<i>ariaRowIndex</i>	<u>aria-rowindex</u>
<i>ariaRowSpan</i>	<u>aria-rowspan</u>
<i>ariaSelected</i>	<u>aria-selected</u>
<i>ariaSetSize</i>	<u>aria-setsize</u>
<i>ariaSort</i>	<u>aria-sort</u>
<i>ariaValueMax</i>	<u>aria-valuemax</u>
<i>ariaValueMin</i>	<u>aria-valuemin</u>

<i>ariaValueNow</i>	aria-valuenow
<i>ariaValueText</i>	aria-valuetext

NOTE

Note: Attributes [aria-dropeffect](#) and [aria-grabbed](#) were deprecated in ARIA 1.1 and do not have corresponding IDL attributes.

§ 10.2.1 Disambiguation Pattern

This section is non-normative.

Though specification authors may make exceptions to this pattern, the following rules were used to disambiguate names and case of the IDL attributes listed above.

- Any attribute name referencing concepts that are combinations of two or more words (such as "described by") becomes a camel-cased IDL attribute capitalizing each word boundary. For example, [aria-describedby](#) becomes `ariaDescribedBy` with both the D and B capitalized.
- Likewise, any attribute name referencing concepts that can be hyphenated (such as "multi-selectable") becomes a camel-cased IDL attribute capitalizing each hyphenation boundary. For example, the only valid spelling for "multi-selectable" is hyphenated, so [aria-multiselectable](#) becomes `ariaMultiSelectable` with both the M and S capitalized.
- When trusted dictionary sources list both hyphenated or non-hyphenated spellings (e.g. "multi-line" and "multiline" are both valid spellings) use the hyphenated version and apply the hyphenation rule above. For example, [aria-multiline](#) becomes `ariaMultiLine` with both the M and L capitalized.
- If all trusted dictionary sources list a single spelling of a compound word with no spaces or hyphens, only the first letter of the term is capitalized. For example, neither "place-holder" nor "place holder" are considered valid spellings of the term "placeholder," so [aria-placeholder](#) becomes `ariaPlaceholder` with only the P capitalized.
- There are currently no acronym-based ARIA attributes, but if future attributes include acronym usage, attempt to match existing `DOM` conventions (e.g. ID becomes `Id`).

§ 10.2.2 IDL Attribute Name Notes or Exceptions

This section is non-normative.

Any notes or exceptions for specific attribute names will be listed here.

- `ariaPosInSet`: The [aria-posinset](#) attribute refers to an item's position in a set (two words: "in set") rather than the "inset" of an item from the beginning of the collection. Therefore the IDL attribute name is `ariaPosInSet` with the P, I, and second S capitalized, *not* `ariaPosInset`.

§ 10.3 ARIAMixin Mixed in to Element

User agents **MUST** include ARIAMixin on Element:

WebIDL

[Element](#) includes [ARIAMixin](#);

For Element:

- The [ARIAMixin getter steps](#) given *element*, *idlAttribute*, and *contentAttribute* are to return the result of the getter algorithm for *idlAttribute* [reflecting](#) *contentAttribute* on *element*.
- The [ARIAMixin setter steps](#) given *element*, *idlAttribute*, *contentAttribute*, and *value* are to perform the setter algorithm for *idlAttribute* [reflecting](#) *contentAttribute* on *element*, given *value*.

NOTE

In practice, this means that, e.g., the `role` IDL on `Element` reflects the `role` content attribute; the `ariaValueMin` IDL attribute reflects the `aria-valuemin` content attribute; etc.

§ 10.4 Example IDL Attribute Usage

This section is non-normative.

The primary purpose of ARIA IDL attribute reflection is to ease JavaScript-based manipulation of values. The following examples demonstrate its usage.

EXAMPLE 34

```
<div id="inaccessibleButton">
  <!-- Use semantic markup instead. This is just a retrofit example. -->
</div>
```

```
// Get a reference to the element.
let el = document.getElementById('inaccessibleButton');
el.tabIndex = 0; // Make it focusable.

// Set the role and label.
el.role = "button";
el.ariaLabel = "Edit";

// Get the role and label.
el.role; // Returns "button"
el.ariaLabel; // Returns "Edit"

// These are interchangeable with the more verbose setAttribute and getAttribute methods.
el.setAttribute("role", "button");
el.setAttribute("aria-label", "Edit");
el.getAttribute("role"); // Returns "button"
el.getAttribute("aria-label"); // Returns "Edit"

// Changes via either interface are reflected by the other.
el.setAttribute("aria-label", "Delete");
el.ariaLabel; // Returns "Delete"
el.ariaLabel = "Publish";
el.getAttribute("aria-label"); // Returns "Publish"
```

§ 11. Privacy and Security Considerations

This section is non-normative.

This specification introduces no new security considerations.

In accordance with [Web Platform Design Principles](#), this specification provides no programmatic interface to determine if information is being used by Assistive Technologies. However, this specification does allow an author to present different information to users of Assistive Technologies from the information available to users who do not use Assistive Technologies. This is possible using many features of the ARIA specification, just as this is possible using many other parts of the web technology stack. This content disparity could be abused to perform [active fingerprinting](#) of users of Assistive Technologies.

§ A. Mapping WAI-ARIA Value types to languages

This section is non-normative.

NOTE

The HTML column of the table below is advisory. Guidance on use of WAI-ARIA state and properties in HTML is provided in [Allowed ARIA roles, states and properties](#) ([HTML-ARIA]).

NOTE

The suggested mappings for true/false values in HTML use [Keyword and enumerated attributes](#) with allowed values of true and false, instead of using the HTML boolean value type.

The table below provides recommended mappings between WAI-ARIA state and property types and attribute types from [HTML], [XML Schema Datatypes](#) [XMLSCHEMA11-2], [SVG2], and SGML.

Languages not listed below might have appropriate value types defined in the language. If they do not, we recommend XML Schema Datatypes for general purpose XML languages. Documents using DTDs instead of schemas will not be able to validate automatically and require additional processing on WAI-ARIA attributes.

WAI-ARIA type	HTML	XML Schema
true/false	Keyword and enumerated attributes with allowed values of "true" and "false"	boolean
true/false/undefined	Keyword and enumerated attributes with allowed values of true, false, and undefined	NMTOKEN with an enumeration constraint allowing values of true, false, and undefined
tristate	Keyword and enumerated attributes with allowed values of "true", "false", and "mixed"	NMTOKEN with an enumeration constraint allowing values of "true", "false", and "mixed"
number	Floating-point numbers	decimal
integer	Non-negative integer	integer
token	Keyword and enumerated attributes	NMTOKEN with an enumeration constraint allowing values listed in the state or property definition
token list	Space-separated tokens	NMTOKENS with an enumeration constraint allowing values listed in the state or property definition

ID reference	The value of a defined id attribute on another element	IDREF
ID reference list	The value of one or more defined id attributes on other element(s), represented as Space-separated tokens	IDREFS
string	No value constraints	string

§ B. Substantive changes since the [WAI-ARIA 1.1 Recommendation](#)

- 16-Feb-2023: Resolved At-Risk items from CR
- 16-Feb-2023: Reverted [spinbutton](#): Change the default value of [aria-valuenow](#) from 0 to "there is no current value." Also add [aria-valuetext](#) as a supported property.
- 17-Sep-2021: Revised IDL and enumerated attribute section to reflect implementations
- 30-Aug-2021: removed ariaDescription from IDL section as was added erroneously
- 14-May-2021: Added Privacy and Security Considerations section
- 05-May-2021: clarify accessible name prohibited definition
- 10-Feb-2021: clarify including elements in accessibility tree to only require elements when actually focused
- 08-Sep-2020: remove [aria-level](#) from [tablist](#)
- 08-Sep-2020: Remove contents as a supported name source for [rowgroup](#).
- 08-Sep-2020: prohibit [aria-roledescription](#) on [generic](#)
- 08-Sep-2020: Require user agents to expose a value for [combobox](#) elements
- 08-Sep-2020: Remove multiple inheritance from [menuitemcheckbox](#) and [menuitemradio](#)
- 08-Sep-2020: Add missing implicit value for [progressbar](#)
- 27-Jul-2020: Update to define owned and container for
- 10-Jul-2020: Re-add [aria-haspopup](#) on links
- 15-May-2020: Remove nullable from IDL [DOMStrings](#), add enumerated attributes prose and examples, and remove ariaRelevant IDL until Issue #1267 can be resolved.
- 07-May-2020: Deprecate [aria-disabled](#), [aria-errormessage](#), [aria-haspopup](#) and [aria-invalid](#) as globals rather than removing them.
- 03-Apr-2020: Clarify default values
- 03-Apr-2020: Revise [meter](#) authoring advice

- 26-Mar-2020: remove recommendation to use `role="none presentation"`
- 26-Mar-2020: Add info about layout and bounds to [generic](#)
- 03-Mar-2020: Clean up of Presentational roles conflict resolution section
- 20-Feb-2020: Update [combobox](#) to remove aria-multiline reference
- 01-Nov-2019: Modify [combobox](#) to new ARIA 1.2 pattern.
- 25-Oct-2019: Modify [caption](#) authoring advice
- 25-Oct-2019: Change [aria-disabled](#), [aria-errormessage](#), [aria-haspopup](#) and [aria-invalid](#) from global to widget specific.
- 24-Oct-2019: Prohibits Labeling of [caption](#), [code](#), [deletion](#), [emphasis](#), [insertion](#), [paragraph](#), [presentation](#), [strong](#), [subscript](#), [superscript](#)
- 24-Oct-2019: Remove accessible name required from [log](#) and [timer](#)
- 24-Oct-2019: Allow group as child of [listbox](#)
- 24-Oct-2019: Add [code](#) role
- 24-Oct-2019: Add [time](#) role
- 24-Oct-2019: Add [subscript](#) and [superscript](#) roles
- 24-Oct-2019: Add [meter](#) role
- 23-Oct-2019: Resolve inconsistencies around group ownership of [menuitem](#), [menuitemcheckbox](#) and [menuitemradio](#).
- 23-Oct-2019: Add [generic](#) role
- 22-Oct-2019: Clarify use of [alertdialog](#) and [alert](#) roles
- 22-Oct-2019: Add [insertion](#) and [deletion](#) roles
- 18-Oct-2019: Remove references to taxonomy file
- 18-Oct-2019: Remove implicit value from [aria-checked](#) on [checkbox](#)
- 17-Oct-2019: Add [strong](#) and [emphasis](#) roles
- 11-Oct-2019: Deprecate [directory](#) role
- 11-Oct-2019: Make [form](#) role accessible name required true
- 11-Oct-2019: Remove allowance of [group](#) in [list](#)
- 04-Sep-2019: Add [aria-required](#) as a supported property of [checkbox](#)
- 04-Sep-2019: Allow [aria-posinset](#) and [aria-setsize](#) on [row](#) when used in a [treegrid](#).
- 04-Sep-2019: Add [aria-expanded](#) support to [application](#) and [checkbox](#) roles.

- 04-Sep-2019: Remove [aria-expanded](#) support from the following roles: [alert](#), [alertdialog](#), [article](#), [banner](#), [blockquote](#), [caption](#), [cell](#), [complementary](#), [contentinfo](#), [definition](#), [deletion](#), [dialog](#), [directory](#), [feed](#), [figure](#), [form](#), [grid](#), [group](#), [heading](#), [img](#), [insertion](#), [landmark](#), [list](#), [listitem](#), [log](#), [main](#), [marquee](#), [math](#), [menu](#), [menubar](#), [navigation](#), [note](#), [paragraph](#), [radiogroup](#), [region](#), [search](#), [select](#), [status](#), [subscript](#), [superscript](#), [table](#), [tabpanel](#), [term](#), [time](#), [timer](#), [toolbar](#), [tooltip](#), [tree](#), [treegrid](#).
- 04-Sep-2019: Remove children-presentational=true from [math](#) role
- 22-Aug-2019: Remove [aria-level](#) from [grid](#)
- 23-Jul-2019: Add [generic](#) role
- 11-Jul-2019: Remove advice against changing roles
- 11-Jul-2019: Set Accessible Name Required to false on [gridcell](#)
- 04-Jun-2019: Make [aria-valuemin](#) and [aria-valuemax](#) supported, rather than required, properties of focusable [separator](#), [slider](#), and [scrollbar](#). Make [aria-orientation](#) a supported, rather than required, property of [scrollbar](#).
- 27-Mar-2019: Add Translatable States and Properties Section
- 31-Jan-2019: Change the superclass of [range](#) from widget to structure.
- 23-Jan-2019: Removed Default value of [aria-checked](#) from [menuitemcheckbox](#) and [menuitemradio](#) roles
- 09-Jan-2019: Removed Default value of [aria-checked](#) from [switch](#) and [checkbox](#) roles
- 05-Oct-2018: Role [spinbutton](#): Change the default value of [aria-valuenow](#) from 0 to "there is no current value." Also add [aria-valuetext](#) as a supported property.
- 05-Oct-2018: Role [spinbutton](#): allow empty values, no min, no max, and structure with sibling steppers
- 21-Aug-2018: Correct normative language in [rowheader](#) to be consistent with required states and properties.
- 21-Jun-2018: Allow [group](#) as child of [listbox](#).
- 31-May-2018: Add [blockquote](#), [caption](#), and [paragraph](#) roles.
- 01-Apr-2018: Added ARIA IDL Section (JavaScript interfaces).
- 06-Dec-2017: Make [aria-expanded](#) a supported state of [menuitem](#). This change also makes it a supported property of [menuitemcheckbox](#) and [menuitemradio](#) via inheritance.
- 06-Dec-2017: When aria-errormessage is not pertinent, authors **MUST** either ensure the content is not rendered or remove the aria-errormessage attribute or its value. User agents **MUST NOT** expose aria-errormessage for an object with an [aria-invalid](#) value of false.

§ C. Acknowledgments

This section is non-normative.

The following people contributed to the development of this document.

§ C.1 Participants active in the ARIA WG at the time of publication

- Sina Bahram (Invited Expert)
- Curt Bellew (Oracle Corporation)
- Zoë Bijl (Invited Expert)
- Shari Butler (Pearson plc)
- Dominic Cooney (Meta)
- Michael Cooper (W3C Staff)
- James Craig (Apple Inc.)
- Joanmarie Diggs (Igalia)
- Isaac Durazo (Bocoup)
- Howard Edwards (Bocoup)
- Frank Elavsky (Invited Expert)
- Mayuri Faldu (Navy Federal Credit Union)
- Steve Faulkner (TPGi)
- Reinaldo Ferraz (NIC.br)
- Alexander Flenniken (Bocoup)
- Bryan Garaventa (Level Access)
- Rashmi Garimella (Google LLC)
- Matt Garrish (DAISY Consortium)
- Jaunita George (Navy Federal Credit Union)
- Ariella Gilmore (IBM Corporation)
- Raghavendra Giriyappa (IBM Corporation)
- Michael Goddard (Bocoup)

- Glen Gordon (TPGi)
- Shirisha Gubba (Google LLC)
- Jon Gunderson (University of Illinois at Urbana-Champaign)
- Markku Hakkinen (Educational Testing Service)
- Sarah Higley (Microsoft Corporation)
- Hans Hillen (TPGi)
- Isabel Holdsworth (TPGi)
- Stanley Hon (Microsoft Corporation)
- Patrick Hung (University of Ontario Institute of Technology)
- Matthew King (Meta)
- Greta Krafsig (The Washington Post)
- Peter Krautzberger (Invited Expert)
- JaEun Jemma Ku (University of Illinois at Urbana-Champaign)
- Christopher Lane (VMWare)
- Charles LaPierre (Benetech)
- Gez Lemon (TPGi)
- Aaron Leventhal (Google LLC)
- Brian Liu Xu (Microsoft Corporation)
- David MacDonald (Invited Expert)
- Carolyn MacLeod (IBM Corporation)
- Mark McCarthy (University of Illinois at Urbana-Champaign)
- Jan McSorley (Pearson plc)
- Erika Miguel (Bocoup)
- Daniel Montalvo (W3C)
- Sheila Moussavi (Bocoup)
- James Nurthen (Adobe)
- Scott O'Hara (Microsoft Corporation)
- Adam Page (Intel Corporation)
- Michael Pennisi (Bocoup)
- Roberto Perez (Microsoft Corporation)
- Janina Sajka (Invited Expert, The Linux Foundation)

- Trish Salas (Level Access)
- Stefan Schnabel (SAP SE)
- Harris Schneiderman (Deque Systems, Inc.)
- Boaz Sender (Bocoup)
- Cynthia Shelly (Google LLC)
- Tzviya Siegman (Wiley)
- Avneesh Singh (DAISY Consortium)
- Neil Soiffer (Invited Expert)
- Francis Storr (Intel Corporation)
- Melanie Sumner (Invited Expert)
- Alexander Surkov (Igalia)
- James Teh (Mozilla Foundation)
- Seth Thompson (Bocoup)
- Jan Williams (TPGi)
- Benjamin Young (Wiley)
- Valerie Young (Igalia)
- Helen Zhou (University of Illinois)
- 骅 杨 (Shenzhen Accessibility Research Association)

§ C.2 Other ARIA contributors, commenters, and previously active participants

Ann Abbott (Invited Expert), Shadi Abou-Zahra (W3C), Irfan Ali (Educational Testing Service), Jim Allan (TSB), CB Averitt (Deque Systems, Inc), Jonny Axelsson (Opera Software), David Baron (Mozilla Foundation), Art Barstow (Nokia Corporation), Simon Bates, Amelia Bellamy-Royds (Invited Expert), Alex Bernier (Association BrailleNet), Jorge Blazquez Alonso (IBM Corporation), Christy Blew (University of Illinois at Urbana-Champaign), Chris Blouch (AOL), David Bolter (Mozilla Foundation), Alice Boxhall (Igalia), Judy Brewer (W3C/MIT), Mark Birbeck (Sidewinder Labs), Matthew Brennan (Facebook), Bogdan Brinza (Microsoft Corporation), Kim Bunge (TPGi), Sally Cain (Royal National Institute of Blind People (RNIB)), Ben Caldwell (Trace), Thaddeus Cambron (Invited Expert), Tammy Campoverde (UnitedHealth Group), Gerardo Capiel (Benetech), David Caro (Wikimedia Foundation), Sofia Celic-Li, Jaesik Chang (Samsung Electronics Co., Ltd.), Alex Qiang Chen (University of Manchester),

Charles Chen (Google, Inc.), Gerard K. Cohen, Christian Cohrs, Timothy Cole (University of Illinois at Urbana-Champaign), Jory Cunningham (Salesforce), Deborah Dahl, Erik Dahlström (Opera Software), Jes Daigle (Bocoup), Dimitar Denev (Frauenhofer Gesellschaft), Jason Duan (IBM Corporation), Micah Dubinko (Invited Expert), Mandana Eibegger, Beth Epperson (Websense), Fred Esch (IBM Corporation), Donald Evans (AOL), Chris Fleizach (Apple Inc.), John Foliot (Deque Systems, Inc.), Kelly Ford (Microsoft Corporation), Geoff Freed (Invited Expert, NCAM), Kentarou Fukuda (IBM Corporation), Christopher Gallelo (Microsoft Corporation), Billy Gregory (The Paciello Group, LLC), Karl Groves (The Paciello Group, LLC), Birkir Gunnarsson (Deque Systems, Inc.), Guido Geloso, Ali Ghassemi, Becky Gibson (Invited Expert), Alfred S. Gilman, Andres Gonzalez (Adobe Systems Inc.), Scott González (jQuery Foundation), James Graham, Georgios Grigoriadis (SAP AG), Jeff Grimes (Oracle), Loretta Guarino Reid (Google, Inc.), Markus Gylling (DAISY Consortium), Katie Haritos-Shea (Knowbility), Barbara Hartel, James Hawkins (Google, Inc.), Benjamin Hawkes-Lewis, Sean Hayes (Microsoft Corporation), Mona Heath (University of Illinois at Urbana-Champaign), Jan Heck, Shawn Henry, Tina Homboe, Nicholas Hoyt (University of Illinois at Urbana-Champaign), John Hrvatin (Microsoft Corporation), Takahiro Inada, Masayasu Ishikawa (W3C), Jim Jewitt, Kenny Johar (Microsoft Corporation), Earl Johnson (Sun), Masahiko Kaneko (Microsoft Corporation), Shilpi Kapoor (BarrierBreak Technologies), Marjolein Katsma, Susann Keohane (IBM Corporation), George Kerscher (International Digital Publishing Forum), Jason Kiss (Department of Internal Affairs, New Zealand Government), Todd Kloots, Jamie Knight (British Broadcasting Corporation), Johannes Koch, Sam Kuper, Jael Kurz, Rajesh Lal (Nokia Corporation), Diego La Monica (International Webmasters Association / ~~HTML~~ Writers Guild (IWA-HWG)), Lori Lane (University of Illinois at Urbana-Champaign), Alex Li (SAP), Chris Lilley, Thomas Logan (HiSoftware Inc.), Brian Loh, William Loughborough (Invited Expert), Krzysztof Maczyński, Linda Mao (Microsoft), Anders Markussen (Opera Software), Daniel Marques (WIRIS Science), Matthew May (Adobe Systems Inc.), Dominic Mazzoni (Google LLC), Shane McCarron (Invited Expert, Aptest), Charles McCathie Nevile (Yandex), Juliette McShane (Access2online Inc.), Heather Migliorisi (Invited Expert), Mary Jo Mueller (IBM Corporation), Alexandre Morgaut (4D), Ann Navarro (Invited Expert), Rich Noah (Bocoup), Joshue O Connor (Invited Expert, CFIT), Achraf Othman (MADA Center), Artur Ortega (Microsoft Corporation), Sailesh Panchang (Deque), Lisa Pappas (Society for Technical Communication (STC)), Marta Pawlowlska (Samsung Electronics Co., Ltd.), Dave Pawson (RNIB), Steven Pemberton (CWI Amsterdam), Vijaya Gowri Perumal (Newgen Knowledgeworks), Christos Petrou (Centre for Inclusive Design), Simon Pieters (Bocoup), Jean-Bernard Piot (4D), David Poehlman (Opera Software), Ian Pouncey (TetraLogical Services Ltd), Sarah Pulis (Media Access Australia), T.V. Raman (Google, Inc.), Ruoxi Ran (W3C Staff), Melanie Richards (Microsoft Corporation), Jan Richards, Adrian Roselli (TPGi),

Gregory Rosmaita (Invited Expert), Tony Ross (Microsoft Corporation), Alex Russell (Dojo Foundation), Mark Sadecki (Invited Expert), Mario Sánchez Prada (Samsung Electronics Co., Ltd. and Gnome Foundation), Martin Schaus (SAP AG), Doug Schepers (W3C), Cynthia Shelly (Microsoft Corporation), Joseph Scheuhammer (Invited Expert, Inclusive Design Research Centre, OCAD University), Matthias Schmitt, Richard Schwerdtfeger (IBM, Knowbility), Lisa Seeman-Kestenbaum (Invited Expert), Marc Silbey (Microsoft Corporation), Leif Halvard Sili, Henri Sivonen (Mozilla), Ville Skyttä, Sharon Snider (IBM Corporation), Michael Smith (W3C), Andi Snow-Weaver (IBM Corporation), Volker Sorge (Invited Expert), Vitaly Sourikov, Mike Squillace (IBM), Maciej Stachowiak (Apple Inc.), Christophe Strobbe, Henny Swan (BBC), Suzanne Taylor (Pearson plc), William Tennis (Navy Federal Credit Union), Terrill Thompson, David Todd, Gregg Vanderheiden (Invited Expert, Trace), Job van Achterberg (Invited Expert), Anne van Kesteren, Scott Vinkle (Shopify), Wen He (Tencent), Can Wang (Zhejiang University), Wei Wang (Zhejiang University), Léonie Watson (TetraLogical Services Ltd), Wu Wei (W3C / RITT), Jason White (Educational Testing Service), Sam White (Apple Inc.), Ryan Williams (Oracle), Tom Wlodkowski, Evan Yamanishi (W. W. Norton), Marco Zehe (Mozilla Foundation), Gottfried Zimmermann (Invited Expert, Access Technologies Group)

§ C.3 Enabling funders

This publication has been funded in part with U.S. Federal funds from the Department of Education, National Institute on Disability, Independent Living, and Rehabilitation Research (NIDILRR), initially under contract number ED-OSE-10-C-0067, then under contract number HHSP23301500054C, and now under HHS75P00120P00168. The content of this publication does not necessarily reflect the views or policies of the U.S. Department of Education, nor does mention of trade names, commercial products, or organizations imply endorsement by the U.S. Government.

§ D. References

§ D.1 Normative references

[ACCNAME-1.2]

Accessible Name and Description Computation 1.2. Bryan Garaventa; Joanmarie Diggs; Michael Cooper. W3C. 11 July 2019. W3C Working Draft. URL: <https://www.w3.org/TR/accname-1.2/>

[CORE-AAM]

[*Core Accessibility API Mappings 1.1*](#). Joanmarie Diggs; Joseph Scheuhammer; Richard Schwerdtfeger; Michael Cooper; Andi Snow-Weaver; Aaron Leventhal. W3C. 14 December 2017. W3C Recommendation. URL: <https://www.w3.org/TR/core-aam-1.1/>

[CORE-AAM-1.2]

[*Core Accessibility API Mappings 1.2*](#). Valerie Young; Alexander Surkov; Michael Cooper. W3C. 18 May 2023. W3C Candidate Recommendation. URL: <https://www.w3.org/TR/core-aam-1.2/>

[CSS3-SELECTORS]

[*Selectors Level 3*](#). Tantek Çelik; Erika Etemad; Daniel Glazman; Ian Hickson; Peter Linss; John Williams. W3C. 6 November 2018. W3C Recommendation. URL: <https://www.w3.org/TR/selectors-3/>

[DOM]

[*DOM Standard*](#). Anne van Kesteren. WHATWG. Living Standard. URL: <https://dom.spec.whatwg.org/>

[HTML]

[*HTML Standard*](#). Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard. URL: <https://html.spec.whatwg.org/multipage/>

[MathML3]

[*Mathematical Markup Language \(MathML\) Version 3.0 2nd Edition*](#). David Carlisle; Patrick D F Ion; Robert R Miner. W3C. 10 April 2014. W3C Recommendation. URL: <https://www.w3.org/TR/MathML3/>

[RFC2119]

[*Key words for use in RFCs to Indicate Requirement Levels*](#). S. Bradner. IETF. March 1997. Best Current Practice. URL: <https://www.rfc-editor.org/rfc/rfc2119>

[ROLE-ATTRIBUTE]

[*Role Attribute 1.0*](#). Shane McCarron et al. W3C. 28 March 2013. W3C Recommendation. URL: <https://www.w3.org/TR/role-attribute/>

[SVG2]

[*Scalable Vector Graphics \(SVG\) 2*](#). Amelia Bellamy-Royds; Bogdan Brinza; Chris Lilley; Dirk Schulze; David Storey; Eric Willigers. W3C. 4 October 2018. W3C Candidate Recommendation. URL: <https://www.w3.org/TR/SVG2/>

[uievents-key]

[*UI Events KeyboardEvent key Values*](#). Travis Leithead; Gary Kacmarcik. W3C. 30 May 2023. W3C Candidate Recommendation. URL: <https://www.w3.org/TR/uievents-key/>

[webidl]

[*Web IDL Standard*](#). Edgar Chen; Timothy Gu. WHATWG. Living Standard. URL: <https://webidl.spec.whatwg.org/>

[XML-NAMES]

[*Namespaces in XML 1.0 \(Third Edition\)*](#). Tim Bray; Dave Hollander; Andrew Layman; Richard Tobin; Henry Thompson et al. W3C. 8 December 2009. W3C Recommendation. URL: <https://www.w3.org/TR/xml-names/>

§ D.2 Informative references

[AT-SPI]

Assistive Technology Service Provider Interface. The GNOME Project. URL: <https://developer-old.gnome.org/libatspi/stable/>

[ATK]

ATK - Accessibility Toolkit. The GNOME Project. URL: <https://developer.gnome.org/atk/stable/>

[AXAPI]

The NSAccessibility Protocol for macOS. Apple, Inc. URL: <https://developer.apple.com/documentation/appkit/nsaccessibility>

[HTML-ARIA]

ARIA in HTML. Steve Faulkner; Scott O'Hara; Patrick Lauke. W3C. 31 May 2023. W3C Recommendation. URL: <https://www.w3.org/TR/html-aria/>

[IAccessible2]

IAccessible2. Linux Foundation. URL: <https://wiki.linuxfoundation.org/accessibility/iaccessible2/>

[MSAA]

Microsoft Active Accessibility (MSAA). Microsoft Corporation. URL: <https://docs.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility>

[UI-AUTOMATION]

UI Automation. Microsoft Corporation. URL: <https://docs.microsoft.com/en-us/windows/win32/winauto/ui-automation-specification>

[UIA-EXPRESS]

The IAccessibleEx Interface. Microsoft Corporation. URL: <https://docs.microsoft.com/en-us/windows/win32/winauto/iaccessibleex>

[wai-aria-1.1]

Accessible Rich Internet Applications (WAI-ARIA) 1.1. Joanmarie Diggs; Shane McCarron; Michael Cooper; Richard Schwerdtfeger; James Craig. W3C. 14 December 2017. W3C Recommendation. URL: <https://www.w3.org/TR/wai-aria-1.1/>

[WAI-ARIA-PRACTICES-1.2]

WAI-ARIA Authoring Practices 1.2. Matthew King; JaEun Jemma Ku; James Nurthen; Zoë Bijl; Michael Cooper. W3C. 19 May 2022. W3C Working Group Note. URL: <https://www.w3.org/TR/wai-aria-practices-1.2/>

[WCAG21]

Web Content Accessibility Guidelines (WCAG) 2.1. Andrew Kirkpatrick; Joshue O'Connor; Alastair Campbell; Michael Cooper. W3C. 5 June 2018. W3C Recommendation. URL: <https://www.w3.org/TR/WCAG21/>

[XMLSCHEMA11-2]

[*W3C XML Schema Definition Language \(XSD\) 1.1 Part 2: Datatypes*](#). David Peterson; Sandy Gao; Ashok Malhotra; Michael Sperberg-McQueen; Henry Thompson; Paul V. Biron et al. W3C. 5 April 2012. W3C Recommendation. URL: <https://www.w3.org/TR/xmlschema11-2/>

