Accessible Name and Description Computation 1.2

W3C Editor's Draft 13 December 2023

▼ **More details about this document**

This version:

https://w3c.github.io/accname/

Latest published version:

https://www.w3.org/TR/accname-1.2/

Latest editor's draft:

https://w3c.github.io/accname/

History:

https://www.w3.org/standards/history/accname-1.2/

Commit history

Latest Recommendation:

https://www.w3.org/TR/accname/

Editors:

Bryan Garaventa (Level Access)

Melanie Sumner (Invited Expert)

Michael Cooper (W3C)

Former editors:

Joanmarie Diggs (Igalia, S.L.) (Editor until March 2021)

Richard Schwerdtfeger (Knowbility) (Editor until October 2017)

Joseph Scheuhammer (Inclusive Design Research Centre, OCAD University) (Editor until May 2017)

James Craig (Apple Inc.) (Editor until May 2016)

Andi Snow-Weaver (IBM) (Editor until December 2012)

Aaron Leventhal (IBM) (Editor until January 2009)

Feedback:

GitHub w3c/accname (pull requests, new issue, open issues)

Abstract

This document describes how *user agents* determine the *names* and *descriptions* of *accessible objects* from web content languages. This information is in turn exposed through *accessibility APIs* so that *assistive technologies* can identify these objects and present their names or descriptions to users. Documenting the algorithm through which names and descriptions are to be determined promotes interoperable exposure of these properties among different accessibility APIs and helps to ensure that this information appears in a manner consistent with author intent.

The accessible name and description computation specification defines support that applies across multiple content technologies. This includes accessible name and description provided by general-purpose *WAI-ARIA* [WAI-ARIA] *roles*, *states*, and properties as well as features specific to individual content languages.

This document updates and will eventually supersede the accessible name and description guidance in the Accessible Name and Description Computation 1.1 [ACCNAME-1.1] W3C Recommendation. It 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/.

This document was published by the Accessible Rich Internet Applications Working Group as an Editor's Draft.

Publication as an Editor's Draft does not imply endorsement by W3C and its Members.

This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

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 03 November 2023 W3C Process Document.

Table of Contents

§ 1. Introduction

This section is non-normative.

User agents acquire information from the DOM [DOM] and create a parallel structure called the *accessibility tree*, made up of *accessible objects*. An accessible object provides information about its *role*, *states*, and properties. An example is an accessible object whose role is `menuitem`, is currently in an `enabled` state, with a `haspopup` property, indicating that it leads to a sub-menu.

The two properties of accessible objects described in this document are its *accessible name* and *accessible description*. The name is a short label that provides information about the purpose of the object. An example of an accessible name for a menu item is `New`, signifying that the menu item provides for the creation of new documents, windows, and so on.

The description is a short explanation that further clarifies the nature of the accessible object. It is not always necessary to provide a description if the name is sufficient, but it can help a user better understand the use of the object.

Accessibility APIs currently support flat, unstructured strings for accessible names and descriptions. The result of the name/description computation is thus a flat string.

The terms "accessible name" and "accessible description" are used to emphasize that they are properties of *accessible objects* as exposed by *Accessibility APIs*. However, they are frequently referred to hereafter as simply "name" and "description".

§ 2. Important Terms

This section is non-normative.

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

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.

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.

§ 3. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words ***MAY***, ***MUST***, and ***MUST NOT*** in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

§ 3.1 RFC-2119 Keywords

RFC-2119 keywords are formatted in uppercase and in bold type font. 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.

§ 3.2 Normative and Informative Sections

The indication whether a section is normative or non-normative (informative) applies to the entire section including sub-sections.

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.

§ 4. Name and Description

The starting point of the name and description computation is a DOM element. The output is a flat, unstructured string that can be as simple as a single word, or a string of space-separated tokens. Examples include `Save` and `Reload from disk`.

An important factor is the element's *role*, that determines which content contributes to the name string. Roles have a `nameFrom` RDF property, with three possible values:

author
: name is generated from values provided by the author in explicit markup features such as the `aria-label` and `aria-labelledby` attribute, or a host language labeling mechanism, such as the `alt` or `title` attribute in HTML, or the `desc` element in SVG.

contents
: name is generated from the Text nodes associated with the element. Although this may be allowed in addition to "author" in some *roles*, "content" is used only if higher priority "author" features are not provided. Priority is defined by the text equivalent computation algorithm.

prohibited
: the element has no name. Authors ***MUST NOT*** use the `aria-label` or `aria-labelledby` attributes to name the element.

The *Accessible Rich Internet Applications (WAI-ARIA) 1.2* [WAI-ARIA] specification provides lists of roles that support name from author, roles that support name from content and roles that cannot be named.

§ 4.1 Name Computation

User agents ***MUST*** compute an *accessible name* using the rules outlined below in the section titled Text Equivalent Computation.

§ 4.2 Description Computation

The following table provides the order of precedence for markup that can be applied to compute an *accessible description*. *User agents* ***MUST*** use the first applicable entry from the table where the listed conditions are met, as described in the last column. The user agent ***MUST NOT*** use any markup other that the first relevant markup found, even if that markup results in an empty description:

Precedence	Attribute	Applicable conditions	How used to compute description
1	`aria-describedby` attribute	Use on any element	Name computation on all nodes referenced by aria-describedby on the element, concatenated, and separated by a space character
2	`aria-description` attribute	Use on any element	As a flat string
3	host language features which participate in the description calculation	Unique host language features ***MAY*** participate in the description computation for an element, only if they were not already used for the accessible name of the applicable element. See HTML AAM: Accessible Description Computation for the HTML elements which meet this condition.	Either a text equivalent computation of the host language element, or the string value of the host language attribute.
4	host language tooltip attribute or equivalent feature	• Any element • Only use if not already used for the accessible name of this node	As a flat string

Precedence	Attribute	Applicable conditions	How used to compute description
	(e.g., HTML `title` attribute)		

§ 4.3 Text Equivalent Computation

The text equivalent computation is used by both the *accessible name* and *accessible description*. There are different rules provided for several different types of elements, nodes, and combinations of markup. Text alternatives are built up, when appropriate, from all the relevant content contained within an element. This is accomplished via steps 2B and 2F, which are recursive, using the full set of rules to retrieve text from its own children or nodes it references.

The purpose of the computation is to create a *perceivable* label or description for alternative presentations, in the form of a flat string of space separated textual tokens.

§ 4.3.1 Terminology

Root node
: The DOM node or element for which the text alternative is sought.

Current node
: The DOM node currently traversed to compute the `root node`'s text equivalent. Initially, the `current node` is the `root node`, but at later stages is either some descendant of the `root node`, or another referenced node.

Flat string
: A string of characters where all carriage returns, newlines, tabs, and form-feeds are replaced with a single space, and multiple spaces are reduced to a single space. The string contains only character data; it does not contain any markup.

Total accumulated text
: The text equivalent computed up to, but not including the `current node`.

Accumulated text
: Text accumulated at a step or sequence of steps described below. It is temporary storage for those steps.

Result
: The text equivalent computed at one of the steps described below.

Append the result, without a space, to X

- If X is empty, copy the `result` to X.
- If X is non-empty, copy the `result` to the end of X.

Append the result, with a space, to X

- If X is empty, copy the `result` to X.
- If X is non-empty, add a space to the end of X and then copy the `result` to X after the space.

Prepend result, without a space, to X

- If X is empty, copy the `result` to X.
- If X is non-empty, copy the `result` to the start of X.

Prepend the result, with a space, to X

- If X is empty, copy the `result` to X.
- If X is non-empty, copy the `result` to the start of X, and add a space after the copy.

§ 4.3.2 Computation steps

The text alternative for a given element is computed as follows:

1. *Initialization:* Set the `root node` to the given element, the `current node` to the `root node`, and the `total accumulated text` to the empty string (""). If the `root node`'s role prohibits naming, return the empty string ("").
2. *Computation:* Compute the text alternative for the `current node`:
   A. *Hidden Not Referenced:* If the `current node` is hidden **and is**:
      i. **Not** part of an `aria-labelledby` or `aria-describedby` traversal, where the node directly referenced by that relation was hidden.
      ii. **Nor** part of a native host language text alternative *element* (e.g. `label` in HTML) or *attribute* traversal, where the root of that traversal was hidden.

   Return the empty string.

NOTE

It's important to clarify the broad definition of hidden for the purposes of accessible name calculation:

i. Nodes with CSS properties `display:none`, `visibility:hidden`, `visibility:collapse` or `content-visibility:hidden`: They are considered hidden, as they match the guidelines "not perceivable" and "explicitly hidden".

ii. Nodes with CSS properties `opacity:0` or `filter:opacity(0%)`, or similar SVG mechanisms: They are not considered hidden. Text hidden with these methods can still be selected or copied, and user agents still expose it in their accessibility trees.

iii. Nodes with the `aria-hidden="true"` property: it is considered hidden, matching the "explicitly hidden" guideline.

iv. Nodes hidden off screen or behind another object: they are not considered hidden. They are exposed in the accessibility tree and they can even name on-screen objects.

NOTE

By default, *assistive technologies* do not relay hidden information, but an author can explicitly override that and include hidden text as part of the *accessible name* or *accessible description* by using `aria-labelledby` or `aria-describedby`.

EXAMPLE 1

The following examples show the meaning of the clause "Not part of an `aria-labelledby` or `aria-describedby` traversal, where the node directy referenced by that relation was hidden.".

First, `element1`'s *accessible name* is "hello" because, although `element3` is hidden, it is part of an `aria-labelledby` traversal started in `element2`, which is hidden too.

EXAMPLE

```
<element1 id="el1" role="button" aria-labelledby="el2" />
  <element2 id="el2" class="hidden">
    <element3 id="el3" class="hidden">hello</element3>
  </element2>
```

EXAMPLE 2

Conversely, `element1` has no *accessible name* if `element3` is hidden and it is part of an `aria-labelledby` traversal started in `element2`, but `element2` is not hidden.

```
<element1 id="el1" role="button" aria-labelledby="el2" />
<element2 id="el2">
  <element3 id="el3" class="hidden">hello</element3>
</element2>
```

B. *LabelledBy:* Otherwise, if the `current node` has an `aria-labelledby` attribute that contains at least one valid IDREF, and the `current node` is not already part of an ongoing `aria-labelledby` or `aria-describedby` traversal, process its IDREFs in the order they occur:

   i. Set the `accumulated text` to the empty string.

   ii. For each IDREF:

      a. Set the `current node` to the node referenced by the IDREF.

      b. *LabelledBy Recursion:* Compute the text alternative of the `current node` beginning with the overall Computation step. Set the `result` to that text alternative.

      c. Append a space character and the `result` to the `accumulated text`.

iii. Return the `accumulated text` if it is not the empty string ("").

The result of LabelledBy Recursion in combination with Hidden Not Referenced means that *user agents* ***MUST*** include all nodes in the subtree as part of the *accessible name* or *accessible description*, when the node referenced by `aria-labelledby` or `aria-describedby` is hidden.

> EXAMPLE 3
>
> The following example shows the meaning of the clause "… and the `current node` is not already part of an `aria-labelledby` traversal …" .
>
> i. `element1`'s *accessible name* is "hello" because this is a first traversal of its `aria-labelledby`, leading to `element3`.
>
> ii. `element2` has no *accessible name*. The computation involves a first traversal of its `aria-labelledby` leading to `element1`, but `element1`'s `aria-labelledby` is not subsequently followed.

```
<element1 id="el1" aria-labelledby="el3" />
<element2 id="el2" aria-labelledby="el1" />
<element3 id="el3"> hello </element3>
```

C. *Embedded Control:* Otherwise, if the `current node` is a control embedded within the label (e.g. any element directly referenced by `aria-labelledby`) for another *widget*, where the user can adjust the embedded control's value, then return the embedded control as part of the text alternative in the following manner:

i. *Textbox:* If the embedded control has role textbox, return its value.

ii. *Combobox/Listbox:* If the embedded control has role combobox or listbox, return the text alternative of the chosen option.

iii. *Range:* If the embedded control has role range (e.g., a spinbutton or slider):

a. If the `aria-valuetext` property is present, return its value,

b. Otherwise, if the `aria-valuenow` property is present, return its value,

c. Otherwise, use the value as specified by a host language attribute.

EXAMPLE 4

Consider a check box label that contains a text input field: "Flash the screen [input] times". If the input field has the value "5", the complete label is "Flash the screen 5 times", e.g.:

```
<label for="flash">
  <input type="checkbox" id="flash">
  Flash the screen <span tabindex="0" role="textbox" aria-label="number of t
</label>
```

D. *AriaLabel:* Otherwise, if the `current node` has an `aria-label` attribute whose value is not undefined, not the empty string, nor, when trimmed of whitespace, is not the empty string:

i. If traversal of the `current node` is due to recursion **and** the `current node` is an embedded control, ignore `aria-label` and skip to rule Embedded Control.

ii. Otherwise, return the value of `aria-label`.

EXAMPLE 5

The following example shows the interaction of `aria-labelledby` and `aria-label` when a node has an `aria-labelledby` that refers to itself. The `<span role="button">` elements have the *accessible names* "Delete Documentation.pdf" and "Delete HolidayLetter.pdf", respectively.

EXAMPLE

```
<h1>Files</h1>
<ul>
  <li>
    <a id="file_row1" href="./files/Documentation.pdf">Documentation.pdf</
    <span role="button" tabindex="0" id="del_row1" aria-label="Delete" ar
  </li>
  <li>
    <a id="file_row2" href="./files/HolidayLetter.pdf">HolidayLetter.pdf</
    <span role="button" tabindex="0" id="del_row2" aria-label="Delete" ar
  </li>
</ul>
```

E. *Host Language Label:* Otherwise, if the `current node`'s native markup provides an attribute (e.g. `alt`) or element (e.g. HTML `label` or SVG `title`) that defines a text alternative, return that alternative in the form of a `flat string` as defined by the host language, unless the element is marked as presentational (`role="presentation"` or `role="none"`).

> NOTE
> See HTML-AAM, SVG-AAM, or other host language documentation for more information on markup that defines a text alternative.

> NOTE
>
> For example, in HTML, the `img` element's `alt` attribute defines a text alternative string, and the `label` element provides text for the referenced form element. In SVG2, the `desc` and `title` elements provide a description of their parent element.

F. *Name From Content:* Otherwise, if the `current node's` *role* allows name from content, or if the `current node` is referenced by `aria-labelledby`, `aria-describedby`, or is a native host language text alternative element (e.g. `label` in HTML), or is a descendant of a native host language text alternative element:

i. *Name From Content Reset:* Set the `accumulated text` to the empty string.

ii. *Name From Generated Content:* Check for CSS generated textual content associated with the `current node` and include it in the `accumulated text`. The CSS `::before` and `::after` pseudo elements [CSS2] can provide textual content for elements that have a content model.

a. For `::before` pseudo elements, *User agents* ***MUST*** prepend CSS textual content, without a space, to the textual content of the `current node`.

b. For `::after` pseudo elements, *User agents* ***MUST*** append CSS textual content, without a space, to the textual content of the `current node`.

iii. *Name From Each Child:* For each child node of the `current node`:

a. Set the `current node` to the child node.

b. Compute the text alternative of the `current node` beginning with the overall Computation step. Set the `result` to that text alternative.

c. Append the `result` to the `accumulated text`.

iv. Return the `accumulated text` if it is not the empty string ("").

Important: Each node in the subtree is consulted only once. If text has been collected from a descendant, but is referenced by another IDREF in some descendant node, then that second, or subsequent, reference is not followed. This is done to avoid infinite loops.

> NOTE
>
> This step can apply to the child nodes themselves, which means the computation is recursive and results in text collected from all the elements in the `current node`'s subtree, no matter how deep it is. However, any given descendant node's text alternative can result from higher precedent markup described in steps B through D above, where "Namefrom: author" attributes provide the text alternative for the entire subtree.

G. *Text Node:* Otherwise, if the `current node` is a Text Node, return its textual contents.

H. *Recursive Name From Content:* Otherwise, if the `current node` is a descendant of an element whose *Accessible Name* or *Accessible Description* is being computed, and contains descendants, proceed to Name From Content Reset.

I. *Tooltip:* Otherwise, if the `current node` has a *Tooltip attribute*, return its value.

> NOTE
>
> Tooltip attributes are used only if nothing else, including subtree content, has provided results.

J. Append a space character and the `result` of each step above to the `total accumulated text`.

3. After all steps are completed, the `total accumulated text` is used as the *accessible name* or *accessible description* of the element that initiated the computation.

§ 5. Accessible Name and Description Mapping

Information concerning name and description accessibility API mappings, including relationships, such as labelled-by/label-for and described-by/description-for, is documented in the Core Accessibility API Mappings specification [CORE-AAM-1.2]. See the mapping table entries for `aria-label`, `aria-labelledby`, and `aria-describedby`.

§ 6. Appendices

§ 6.1 Change Log

§ 6.1.1 Substantive changes since the last public working draft

§ 6.1.2 Substantive changes since the Accessible Name and Description Computation 1.1 Recommendation

- 27-June-2019: Add statement allowing for the possibility of naming being prohibited on the root node. Note: This change in and of itself has no implementation impact, but it allows other specifications to optionally prohibit naming for a top-level element. Furthermore, even if this prohibition is made within a specification, that prohibition will not have any impact on calculating name from contents.

§ 6.2 Acknowledgments

This section is non-normative.

The following people contributed to the development of this document.

§ 6.2.1 Participants active in the ARIA WG at the time of publication

- Irfan Ali (Educational Testing Service)
- CB Averitt (Deque Systems, Inc)
- Sina Bahram (Invited Expert)
- Shirisha Balusani (Invited Expert)

- Amelia Bellamy-Royds (Invited Expert)
- Curt Bellew (Oracle Corporation)
- Alex Bernier (Association BrailleNet)
- Zoë Bijl (Invited Expert)
- Jorge Blazquez Alonso (IBM Corporation)
- Alice Boxhall (Google LLC)
- Matthew Brennan (Facebook)
- Kim Bunge (TPGi)
- Shari Butler (Pearson plc)
- Tammy Campoverde (UnitedHealth Group)
- David Caro (Wikimedia Foundation)
- Timothy Cole (University of Illinois at Urbana-Champaign)
- Dominic Cooney (Facebook)
- Michael Cooper (W3C Staff)
- James Craig (Apple Inc.)
- Jory Cunningham (Salesforce)
- Jes Daigle (Bocoup)
- Joanmarie Diggs (Igalia)
- Jason Duan (IBM Corporation)
- Isaac Durazo (Bocoup)
- Howard Edwards (Bocoup)
- Steve Faulkner (TPGi)
- Reinaldo Ferraz (NIC.br)
- Alexander Flenniken (Bocoup)
- Bryan Garaventa (Level Access)
- Matt Garrish (DAISY Consortium)
- Jaunita George (Navy Federal Credit Union)
- Raghavendra Giriyappa (IBM Corporation)
- Michael Goddard (Bocoup)

- Glen Gordon (TPGi)
- 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)
- Nicholas Hoyt (University of Illinois at Urbana-Champaign)
- Shilpi Kapoor (BarrierBreak Technologies)
- Matthew King (Facebook)
- Greta Krafsig (The Washington Post)
- Peter Krautzberger (Invited Expert)
- JaEun Jemma Ku (University of Illinois at Urbana-Champaign)
- Lori Lane (University of Illinois at Urbana-Champaign)
- Charles LaPierre (Benetech)
- Gez Lemon (TPGi)
- Aaron Leventhal (Google LLC)
- Brian Liu Xu (Microsoft Corporation)
- David MacDonald (Invited Expert)
- Carolyn MacLeod (IBM Corporation)
- Daniel Marques (WIRIS Science)
- Dominic Mazzoni (Google LLC)
- Mark McCarthy (University of Illinois at Urbana-Champaign)
- Jan McSorley (Pearson plc)
- Erika Miguel (Bocoup)
- Sheila Moussavi (Bocoup)
- Rich Noah (Bocoup)
- James Nurthen (Adobe)
- Scott O'Hara (Microsoft Corporation)

- Achraf Othman (MADA Center)
- Vijaya Gowri Perumal (Newgen Knowledgeworks)
- Christos Petrou (Centre for Inclusive Design)
- Simon Pieters (Bocoup)
- Ian Pouncey (TetraLogical Services Ltd)
- Ruoxi Ran (W3C Staff)
- Adrian Roselli (TPGi)
- Janina Sajka (Invited Expert, The Linux Foundation)
- Stefan Schnabel (SAP SE)
- Harris Schneiderman (Deque Systems, Inc.)
- Lisa Seeman-Kestenbaum (Invited Expert)
- Boaz Sender (Bocoup)
- Cynthia Shelly (Google LLC)
- Tzviya Siegman (Wiley)
- Sharon Snider (IBM Corporation)
- Neil Soiffer (Invited Expert)
- Volker Sorge (Invited Expert)
- Francis Storr (Intel Corporation)
- Melanie Sumner (Invited Expert)
- Alexander Surkov (Igalia)
- William Tennis (Navy Federal Credit Union)
- Seth Thompson (Bocoup)
- Scott Vinkle (Shopify)
- Can Wang (Zhejiang University)
- Wei Wang (Zhejiang University)
- Léonie Watson (TetraLogical Services Ltd)
- Jason White (Educational Testing Service)
- Jan Williams (TPGi)
- Evan Yamanishi (W. W. Norton)

- Benjamin Young (Wiley)
- Valerie Young (Bocoup)
- Marco Zehe (Mozilla Foundation)

§ 6.2.2 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.

§ A. References

§ A.1 Normative references

[CORE-AAM-1.2]
Core Accessibility API Mappings 1.2. Valerie Young; Alexander Surkov; Michael Cooper. W3C. 2 November 2023. W3C Candidate Recommendation. URL: https://www.w3.org/TR/core-aam-1.2/

[CSS2]
Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Bert Bos; Tantek Çelik; Ian Hickson; Håkon Wium Lie. W3C. 7 June 2011. W3C Recommendation. URL: https://www.w3.org/TR/CSS21/

[DOM]
DOM Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://dom.spec.whatwg.org/

[html-aam-1.0]
HTML Accessibility API Mappings 1.0. Scott O'Hara. W3C. 4 December 2023. W3C Working Draft. URL: https://www.w3.org/TR/html-aam-1.0/

[infra]
Infra Standard. Anne van Kesteren; Domenic Denicola. WHATWG. Living Standard. URL: https://infra.spec.whatwg.org/

[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

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[WAI-ARIA]

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/

§ A.2 Informative references

[ACCNAME-1.1]

Accessible Name and Description Computation 1.1. Joanmarie Diggs; Bryan Garaventa; Michael Cooper. W3C. 18 December 2018. W3C Recommendation. URL: https://www.w3.org/TR/accname-1.1/

[wai-aria-1.2]

Accessible Rich Internet Applications (WAI-ARIA) 1.2. Joanmarie Diggs; James Nurthen; Michael Cooper; Carolyn MacLeod. W3C. 6 June 2023. W3C Recommendation. URL: https://www.w3.org/TR/wai-aria-1.2/

↥