2.1 WAI-ARIA Roles**§**

A WAI-ARIA [role](https://www.w3.org/TR/wai-aria-1.1/#dfn-role) is set on an [element](https://www.w3.org/TR/wai-aria-1.1/#dfn-element) using a role [attribute](https://www.w3.org/TR/wai-aria-1.1/#dfn-attribute), similar to the role attribute defined in [*Role Attribute*](https://www.w3.org/TR/role-attribute/)[[role-attribute](https://www.w3.org/TR/wai-aria-1.1/#bib-role-attribute)].

EXAMPLE 1

<li role="menuitem">Open file…</li>

The roles defined in this specification include a collection of [document landmarks](https://www.w3.org/TR/wai-aria-1.1/#landmark_roles) and the [WAI-ARIA role taxonomy](https://www.w3.org/TR/wai-aria-1.1/#role_definitions).

The roles in this [taxonomy](https://www.w3.org/TR/wai-aria-1.1/#dfn-taxonomy) and their expected behaviors are modeled using [*RDF/OWL*](https://www.w3.org/TR/2004/REC-owl-features-20040210/) [[owl-features](https://www.w3.org/TR/wai-aria-1.1/#bib-owl-features)]. Features of the role taxonomy provide the following information for each role:

* an informative description of the role;
* hierarchical information about related roles (e.g., a [directory](https://www.w3.org/TR/wai-aria-1.1/#directory) is a type of [list](https://www.w3.org/TR/wai-aria-1.1/#list));
* context of the role (e.g., a [listitem](https://www.w3.org/TR/wai-aria-1.1/" \l "listitem) is contained inside a list);
* references to related concepts in other specifications;
* use of OWL to provide a type hierarchy allowing for [semantic](https://www.w3.org/TR/wai-aria-1.1/#dfn-semantics) inheritance (similar to a [class](https://www.w3.org/TR/wai-aria-1.1/#dfn-class) hierarchy); and
* supported [states](https://www.w3.org/TR/wai-aria-1.1/#dfn-state) and [properties](https://www.w3.org/TR/wai-aria-1.1/#dfn-property) for each role (e.g., a [checkbox](https://www.w3.org/TR/wai-aria-1.1/#checkbox) supports being checked via [aria-checked](https://www.w3.org/TR/wai-aria-1.1/#aria-checked)).

Attaching a role gives [assistive technologies](https://www.w3.org/TR/wai-aria-1.1/#dfn-assistive-technology) information about how to handle each element.

2.2 WAI-ARIA States and Properties[**§**](https://www.w3.org/TR/wai-aria-1.1/#introstates)

WAI-ARIA provides a collection of accessibility [states](https://www.w3.org/TR/wai-aria-1.1/#dfn-state) and [properties](https://www.w3.org/TR/wai-aria-1.1/#dfn-property) which are used to support platform [accessibility APIs](https://www.w3.org/TR/wai-aria-1.1/#dfn-accessibility-api) on various operating system platforms. [Assistive technologies](https://www.w3.org/TR/wai-aria-1.1/#dfn-assistive-technology) may access this information through an exposed [user agent](https://www.w3.org/TR/wai-aria-1.1/#dfn-user-agent) DOM or through a mapping to the platform accessibility API. When combined with [roles](https://www.w3.org/TR/wai-aria-1.1/#dfn-role), 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](https://www.w3.org/TR/wai-aria-1.1/#dfn-event) will capture mouse and keyboard events to toggle the value of [aria-checked](https://www.w3.org/TR/wai-aria-1.1/#aria-checked). A role is used to make the behavior of this simple [widget](https://www.w3.org/TR/wai-aria-1.1/#dfn-widget) known to the user agent. [Attributes](https://www.w3.org/TR/wai-aria-1.1/#dfn-attribute) that change with user actions (such as aria-checked) are defined in the [states and properties](https://www.w3.org/TR/wai-aria-1.1/#states_and_properties) section.

EXAMPLE 2

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

Some accessibility states, called [*managed states*](https://www.w3.org/TR/wai-aria-1.1/#dfn-managed-state), 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](https://www.w3.org/TR/wai-aria-1.1/#aria-posinset) and [aria-setsize](https://www.w3.org/TR/wai-aria-1.1/#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*](https://www.w3.org/TR/css3-selectors/#attribute-selectors) ([[css3-selectors](https://www.w3.org/TR/wai-aria-1.1/#bib-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](https://www.w3.org/TR/wai-aria-1.1/" \l "menuitemcheckbox) is checked.

EXAMPLE 4

<li role="menuitemcheckbox" aria-checked="true">

<img src="checked.gif" role="presentation" alt="">

*<!-- note: additional scripts required to toggle image source -->*

Sort by Last Modified

</li>

For more: [https://www.w3.org/TR/wai-aria-1.1/](:%20https:/www.w3.org/TR/wai-aria-1.1/)

| **Element** | **Keyboard Interaction** | **WAI-ARIA Roles, States, and Properties** | **Notes** |
| --- | --- | --- | --- |
| **Accordion** | **Enter or Space:** When focus is on the accordion header for a collapsed panel, expands the associated panel. If the implementation allows only one panel to be expanded, and if another panel is expanded, collapses that panel. When focus is on the accordion header for an expanded panel, collapses the panel if the implementation supports collapsing. Some implementations require one panel to be expanded at all times and allow only one panel to be expanded; so, they do not support a collapse function. **Tab:** Moves focus to the next focusable element; all focusable elements in the accordion are included in the page Tab sequence. **Shift + Tab**: Moves focus to the previous focusable element; all focusable elements in the accordion are included in the page Tab sequence. **Down Arrow** (Optional): If focus is on an accordion header, moves focus to the next accordion header. If focus is on the last accordion header, either does nothing or moves focus to the first accordion header. **Up Arrow (Optional)**: If focus is on an accordion header, moves focus to the previous accordion header. If focus is on the first accordion header, either does nothing or moves focus to the last accordion header. **Home (Optional)**: When focus is on an accordion header, moves focus to the first accordion header. **End (Optional)**: When focus is on an accordion header, moves focus to the last accordion header. | The title of each accordion header is contained in an element with role button.  Each accordion header button is wrapped in an element with role heading that has a value set for aria-level that is appropriate for the information architecture of the page.  If the native host language has an element with an implicit heading and aria-level, such as an HTML heading tag, a native host language element may be used. The button element is the only element inside the heading element. That is, if there are other visually persistent elements, they are not included inside the heading element.  If the accordion panel associated with an accordion header is visible, the header button element has aria-expanded set to true. If the panel is not visible, aria-expanded is set to false.  The accordion header button element has aria-controls set to the ID of the element containing the accordion panel content.  If the accordion panel associated with an accordion header is visible, and if the accordion does not permit the panel to be collapsed, the header button element has aria-disabled set to true.  Optionally, each element that serves as a container for panel content has role region and aria-labelled by with a value that refers to the button that controls display of the panel.  Avoid using the region role in circumstances that create landmark region proliferation, e.g., in an accordion that contains more than approximately 6 panels that can be expanded at the same time.  Role region is especially helpful to the perception of structure by screen reader users when panels contain heading elements or a nested accordion. | check Example @ https://www.w3.org/TR/wai-aria-practices/examples/accordion/accordion.html |
| **Alert** | An alert (WAI-ARIA live region) does not require any keyboard interaction. | The widget has a role of alert. | check Example @ https://www.w3.org/TR/wai-aria-practices/examples/alert/alert.html |
| **Dialog (Modal)** | When a dialog opens, focus moves to an element inside the dialog.  **Tab:** Moves focus to the next tabbable element inside the dialog. If focus is on the last tabbable element inside the dialog, moves focus to the first tabbable element inside the dialog.  **Shift + Tab:** Moves focus to the previous tabbable element inside the dialog. If focus is on the first tabbable element inside the dialog, moves focus to the last tabbable element inside the dialog.   **Escape**: Closes the dialog. | The element that serves as the dialog container has a role of dialog.  All elements required to operate the dialog are descendants of the element that has role dialog.  The dialog container element has aria-modal set to true.  The dialog has either: A value set for the aria-labelled by property that refers to a visible dialog title.  A label specified by aria-label.  Optionally, the aria-described by property is set on the element with the dialog role to indicate which element or elements in the dialog contain content that describes the primary purpose or message of the dialog. Specifying descriptive elements enables screen readers to announce the description along with the dialog title and initially focused element when the dialog opens. | check Example @ https://www.w3.org/TR/wai-aria-practices/examples/dialog-modal/alertdialog.html |
| **Breadcrumb** | Not applicable. | The set of links is structured using an ordered list .  A nav element labeled Breadcrumb identifies the structure as a breadcrumb trail and makes it a navigation landmark so that it is easy to locate.  To prevent screen reader announcement of the visual separators between links, they are added via CSS: The separators are part of the visual presentation that signifies the breadcrumb trail, which is already semantically represented by the nav element with its label of Breadcrumb. | check Example @ https://www.w3.org/TR/wai-aria-practices/examples/breadcrumb/index.html |
| **Button** | Space: Activates the button.  Enter: Activates the button. | The button has role of button.  The button has an accessible label. By default, the accessible name is computed from any text content inside the button element. However, it can also be provided with aria-labelled by or aria-label.  If a description of the button's function is present, the button element has aria-described by set to the ID of the element containing the description. When the action associated with a button is unavailable, the button has aria-disabled set to true.  If the button is a toggle button, it has an aria-pressed state. When the button is toggled on, the value of this state is true, and when toggled off, the state is false. | Following button activation, focus is set depending on the type of action the button performs. For example: If activating the button opens a dialog, the focus moves inside the dialog. (see dialog pattern)  If activating the button closes a dialog, focus typically returns to the button that opened the dialog unless the function performed in the dialog context logically leads to a different element.  For example, activating a cancel button in a dialog returns focus to the button that opened the dialog.  However, if the dialog were confirming the action of deleting the page from which it was opened, the focus would logically move to a new context.  If activating the button does not dismiss the current context, then focus typically remains on the button after activation, e.g., an Apply or Recalculate button.  If the button action indicates a context change, such as move to next step in a wizard or add another search criteria, then it is often appropriate to move focus to the starting point for that action.  If the button is activated with a shortcut key, the focus usually remains in the context from which the shortcut key was activated. For example, if Alt + U were assigned to an "Up" button that moves the currently focused item in a list one position higher in the list, pressing Alt + U when the focus is in the list would not move the focus from the list.  check Example @ https://www.w3.org/TR/wai-aria-practices/examples/button/button.html |
| **Checkbox** | When the checkbox has focus, pressing the Space key changes the state of the checkbox. | The checkbox has role checkbox.  The checkbox has an accessible label provided by one of the following: Visible text content contained within the element with role checkbox.  A visible label referenced by the value of aria-labelled by set on the element with role checkbox.  aria-label set on the element with role checkbox.  When checked, the checkbox element has state aria-checked set to true.  When not checked, it has state aria-checked set to false.  When partially checked, it has state aria-checked set to mixed.  If a set of checkboxes is presented as a logical group with a visible label, the checkboxes are included in an element with role group that has the property aria-labelled by set to the ID of the element containing the label.  If the presentation includes additional descriptive static text relevant to a checkbox or checkbox group, the checkbox or checkbox group has the property aria-described by set to the ID of the element containing the description. | check Examples  @ https://www.w3.org/TR/wai-aria-practices/examples/checkbox/checkbox-1/checkbox-1.html  https://www.w3.org/TR/wai-aria-practices/examples/checkbox/checkbox-2/checkbox-2.html |
| **Combo Box** | * Tab: The textbox is in the page Tab sequence. * Note: The popup indicator icon or button (if present), the popup, and the popup descendants are excluded from the page Tab sequence.  Keyboard InteractionTextbox Keyboard InteractionListbox Popup Keyboard Interaction | The role, state, and property guidance where the ARIA 1.1 and ARIA 1.0 patterns differ is listed first. The subsequent guidance applies to both patterns.   * In a combobox implementing the ARIA 1.1 pattern:   + The element that serves as the combobox container has role [combobox](https://www.w3.org/TR/wai-aria-1.1/" \l "combobox).   + The element with role combobox contains or owns a textbox element that has either role [textbox](https://www.w3.org/TR/wai-aria-1.1/#textbox) or role [searchbox](https://www.w3.org/TR/wai-aria-1.1/" \l "searchbox).   + When the combobox popup is visible, the combobox element contains or owns an element that has role [listbox](https://www.w3.org/TR/wai-aria-1.1/" \l "listbox), [tree](https://www.w3.org/TR/wai-aria-1.1/#tree), [grid](https://www.w3.org/TR/wai-aria-1.1/#grid), or [dialog](https://www.w3.org/TR/wai-aria-1.1/#dialog).   + If the combobox popup has a role other than listbox, the element with role combobox has [aria-haspopup](https://www.w3.org/TR/wai-aria-1.1/#aria-haspopup) set to a value that corresponds to the popup type. That is, aria-haspopup is set to grid, tree, or dialog. Note that elements with role combobox have an implicit aria-haspopup value of listbox.   + When the combobox popup is visible, the textbox element has [aria-controls](https://www.w3.org/TR/wai-aria-1.1/#aria-controls) set to a value that refers to the combobox popup element. * In a combobox implementing the ARIA 1.0 pattern:   + The element that serves as the textbox has role [combobox](https://www.w3.org/TR/wai-aria-1.1/" \l "combobox).   + When the combobox popup is visible, the element with role combobox has [aria-owns](https://www.w3.org/TR/wai-aria-1.1/#aria-owns) set to a value that refers to an element with role [listbox](https://www.w3.org/TR/wai-aria-1.1/" \l "listbox).   + the element with role combobox has a value for [aria-haspopup](https://www.w3.org/TR/wai-aria-1.1/#aria-haspopup) of listbox. Note that elements with role combobox have an implicit aria-haspopup value of listbox. * The textbox element has a value for [aria-multiline](https://www.w3.org/TR/wai-aria-1.1/#aria-multiline) of false. Note that the default value of aria-multiline is false. * When the combobox popup is not visible, the element with role combobox has [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) set to false. When the popup element is visible, aria-expanded is set to true. Note that elements with role combobox have a default value for aria-expanded of false. * When a combobox receives focus, DOM focus is placed on the textbox element. * When a descendant of a listbox, grid, or tree popup is focused, DOM focus remains on the textbox and the textbox has [aria-activedescendant](https://www.w3.org/TR/wai-aria-1.1/#aria-activedescendant) set to a value that refers to the focused element within the popup. * In a combobox with a listbox, grid, or tree popup, when a suggested value is visually indicated as the currently selected value, the option, gridcell, row, or treeitem containing that value has [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true. * If the combobox has a visible label, the element with role combobox has [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) set to a value that refers to the labeling element. Otherwise, the combobox element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * The textbox element has [aria-autocomplete](https://www.w3.org/TR/wai-aria-1.1/#aria-autocomplete) set to a value that corresponds to its autocomplete behavior: * none: When the popup is displayed, the suggested values it contains are the same regardless of the characters typed in the textbox. * list: When the popup is triggered, it presents suggested values that complete or logically correspond to the characters typed in the textbox. * both: When the popup is triggered, it presents suggested values that complete or logically correspond to the characters typed in the textbox. In addition, the portion of the selected suggestion that has not been typed by the user, known as the completion string, appears inline after the input cursor in the textbox. The inline completion string is visually highlighted and has a selected state.   NOTE   1. When referring to the roles, states, and properties documentation for the below list of patterns used for popups, keep in mind that a combobox is a single-select widget where selection always follows focus in the popup. 2. The roles, states, and properties for popup elements are defined in their respective design patterns:    * [Listbox Roles, States, and Properties](https://www.w3.org/TR/wai-aria-practices/#listbox_roles_states_props) . [Grid Roles, States, and Properties](https://www.w3.org/TR/wai-aria-practices/#grid_roles_states_props) , [Tree Roles, States, and Properties](https://www.w3.org/TR/wai-aria-practices/#tree_roles_states_props) , [Dialog Roles, States, and Properties](https://www.w3.org/TR/wai-aria-practices/#dialog_roles_states_props) | Examples  * [Examples of ARIA 1.1 Combobox with Listbox Popup](https://www.w3.org/TR/wai-aria-practices/examples/combobox/aria1.1pattern/listbox-combo.html): Comboboxes that demonstrate the various forms of autocomplete behavior using a listbox popup and use the ARIA 1.1 implementation pattern. * [Example of ARIA 1.1 Combobox with Grid Popup](https://www.w3.org/TR/wai-aria-practices/examples/combobox/aria1.1pattern/grid-combo.html): A combobox that presents suggestions in a grid, enabling users to navigate descriptive information about each suggestion. * [ARIA 1.0 Combobox with Both List and Inline Autocomplete](https://www.w3.org/TR/wai-aria-practices/examples/combobox/aria1.0pattern/combobox-autocomplete-both.html): A combobox that demonstrates the autocomplete behavior known as list with inline autocomplete and uses the ARIA 1.0 implementation pattern. * [ARIA 1.0 Combobox with List Autocomplete](https://www.w3.org/TR/wai-aria-practices/examples/combobox/aria1.0pattern/combobox-autocomplete-list.html): A combobox that demonstrates the autocomplete behavior known as list with manual selection and uses the ARIA 1.0 implementation pattern. * [ARIA 1.0 Combobox Without Autocomplete](https://www.w3.org/TR/wai-aria-practices/examples/combobox/aria1.0pattern/combobox-autocomplete-none.html): A combo box that demonstrates the behavior associated with aria-autocomplete=none and uses the ARIA 1.0 implementation pattern. |
| **Dialog (Modal)** | In the following description, the term tabbable element refers to any element with a tabindex value of zero or greater. Note that values greater than 0 are strongly discouraged.   * When a dialog opens, focus moves to an element inside the dialog. See notes below regarding initial focus placement. * Tab:   + Moves focus to the next tabbable element inside the dialog.   + If focus is on the last tabbable element inside the dialog, moves focus to the first tabbable element inside the dialog. * Shift + Tab:   + Moves focus to the previous tabbable element inside the dialog.   + If focus is on the first tabbable element inside the dialog, moves focus to the last tabbable element inside the dialog. * Escape: Closes the dialog. | * The element that serves as the dialog container has a role of [dialog](https://www.w3.org/TR/wai-aria-1.1/#dialog). * All elements required to operate the dialog are descendants of the element that has role dialog. * The dialog container element has [aria-modal](https://www.w3.org/TR/wai-aria-1.1/#aria-modal) set to true. * The dialog has either:   + A value set for the [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) property that refers to a visible dialog title.   + A label specified by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * Optionally, the [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) property is set on the element with the dialog role to indicate which element or elements in the dialog contain content that describes the primary purpose or message of the dialog. Specifying descriptive elements enables screen readers to announce the description along with the dialog title and initially focused element when the dialog opens. | check example @ https://www.w3.org/TR/wai-aria-practices/examples/dialog-modal/dialog.html  NOTE   1. When a dialog opens, focus placement depends on 2. When a dialog closes, focus returns to the element that invoked the dialog unless either: 3. It is strongly recommended that the tab sequence of all dialogs include a visible element with role button that closes the dialog, such as a close icon or cancel button. |
| **Disclosure (Show/Hide)** | When the disclosure control has focus:   * Enter: activates the disclosure control and toggles the visibility of the disclosure content. * Space: activates the disclosure control and toggles the visibility of the disclosure content. | * The element that shows and hides the content has role [button](https://www.w3.org/TR/wai-aria-1.1/#button). * When the content is visible, the element with role button has [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) set to true. When the content area is hidden, it is set to false. * Optionally, the element with role button has a value specified for [aria-controls](https://www.w3.org/TR/wai-aria-1.1/#aria-controls) that refers to the element that contains all the content that is shown or hidden. | Check example @   * [Disclosure (Show/Hide) of Image Description](https://www.w3.org/TR/wai-aria-practices/examples/disclosure/disclosure-img-long-description.html) * [Disclosure (Show/Hide) of Answers to Frequently Asked Questions](https://www.w3.org/TR/wai-aria-practices/examples/disclosure/disclosure-faq.html) |
| **Feed** | The feed pattern is not based on a desktop GUI widget so the feed role is not associated with any well-established keyboard conventions. Supporting the following, or a similar, interface is recommended.  When focus is inside the feed:   * Page Down: Move focus to next article. * Page Up: Move focus to previous article. * Control + End: Move focus to the first focusable element after the feed. * Control + Home: Move focus to the first focusable element before the feed. | * The element that contains the set of feed articles has role [feed](https://www.w3.org/TR/wai-aria-1.1/#feed). * If the feed has a visible title, the feed element has [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) referring to the element containing the title. Otherwise, the feed element has a label specified with [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * Each unit of content in a feed is contained in an element with role [article](https://www.w3.org/TR/wai-aria-1.1/#article). All content inside the feed is contained in an article element. * Each article element has [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) referring to elements inside the article that can serve as a distinguishing label. * It is optional but strongly recommended for each article element to have [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) referring to one or more elements inside the article that serve as the primary content of the article. * Each article element has [aria-posinset](https://www.w3.org/TR/wai-aria-1.1/#aria-posinset) set to a value that represents its position in the feed. * Each article element has [aria-setsize](https://www.w3.org/TR/wai-aria-1.1/#aria-setsize) set to a value that represents either the total number of articles that have been loaded or the total number in the feed, depending on which value is deemed more helpful to users. If the total number in the feed is undetermined, it can be represented by a aria-setsize value of -1. * When article elements are being added to or removed from the feed container, and if the operation requires multiple DOM operations, the feed element has [aria-busy](https://www.w3.org/TR/wai-aria-1.1/#aria-busy) set to true during the update operation. Note that it is extremely important that aria-busy is set to false when the operation is complete or the changes may not become visible to some assistive technology users. | Check example @   [Example Implementation of Feed Pattern](https://www.w3.org/TR/wai-aria-practices/examples/feed/feed.html) |
| **Grids : Interactive Tabular Data and Layout Containers** | The following keys provide grid navigation by moving focus among cells of the grid. Implementations of grid make these key commands available when an element in the grid has received focus, e.g., after a user has moved focus to the grid with Tab.   * Right Arrow: Moves focus one cell to the right. If focus is on the right-most cell in the row, focus does not move. * Left Arrow: Moves focus one cell to the left. If focus is on the left-most cell in the row, focus does not move. * Down Arrow: Moves focus one cell down. If focus is on the bottom cell in the column, focus does not move. * Up Arrow: Moves focus one cell Up. If focus is on the top cell in the column, focus does not move. * Page Down: Moves focus down an author-determined number of rows, typically scrolling so the bottom row in the currently visible set of rows becomes one of the first visible rows. If focus is in the last row of the grid, focus does not move. * Page Up: Moves focus up an author-determined number of rows, typically scrolling so the top row in the currently visible set of rows becomes one of the last visible rows. If focus is in the first row of the grid, focus does not move. * Home: moves focus to the first cell in the row that contains focus. * End: moves focus to the last cell in the row that contains focus. * Control + Home: moves focus to the first cell in the first row. * Control + End: moves focus to the last cell in the last row.   If a grid supports selection of cells, rows, or columns, the following keys are commonly used for these functions.   * Control + Space: selects the column that contains the focus. * Shift + Space: Selects the row that contains the focus. If the grid includes a column with checkboxes for selecting rows, this key can serve as a shortcut for checking the box when focus is not on the checkbox. * Control + A: Selects all cells. * Shift + Right Arrow: Extends selection one cell to the right. * Shift + Left Arrow: Extends selection one cell to the left. * Shift + Down Arrow: Extends selection one cell down. * Shift + Up Arrow: Extends selection one cell Up. | * The grid container has role [grid](https://www.w3.org/TR/wai-aria-1.1/#grid). * Each row container has role [row](https://www.w3.org/TR/wai-aria-1.1/#row) and is either a DOM descendant of or owned by the grid element or an element with role [rowgroup](https://www.w3.org/TR/wai-aria-1.1/" \l "rowgroup). * Each cell is either a DOM descendant of or owned by a row element and has one of the following roles:   + [columnheader](https://www.w3.org/TR/wai-aria-1.1/#columnheader) if the cell contains a title or header information for the column.   + [rowheader](https://www.w3.org/TR/wai-aria-1.1/#rowheader) if the cell contains title or header information for the row.   + [gridcell](https://www.w3.org/TR/wai-aria-1.1/#gridcell) if the cell does not contain column or row header information. * If there is an element in the user interface that serves as a label for the grid, [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) is set on the grid element with a value that refers to the labeling element. Otherwise, a label is specified for the grid element using [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the grid has a caption or description, [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) is set on the grid element with a value referring to the element containing the description. * If the grid provides sort functions, [aria-sort](https://www.w3.org/TR/wai-aria-1.1/#aria-sort) is set to an appropriate value on the header cell element for the sorted column or row as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). * If the grid supports selection, when a cell or row is selected, the selected element has [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set true. If the grid supports column selection and a column is selected, all cells in the column have aria-selected set to true. * If the grid provides content editing functionality and contains cells that may have edit capabilities disabled in certain conditions, [aria-readonly](https://www.w3.org/TR/wai-aria-1.1/#aria-readonly) may be set true on cells where editing is disabled. If edit functions are disabled for all cells, aria-readonly may be set true on the grid element. Grids that do not provide editing functions do not include the aria-readonly attribute on any of their elements. * If there are conditions where some rows or columns are hidden or not present in the DOM, e.g., data is dynamically loaded when scrolling or the grid provides functions for hiding rows or columns, the following properties are applied as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties).   + [aria-colcount](https://www.w3.org/TR/wai-aria-1.1/#aria-colcount) or [aria-rowcount](https://www.w3.org/TR/wai-aria-1.1/#aria-rowcount) is set to the total number of columns or rows, respectively.   + [aria-colindex](https://www.w3.org/TR/wai-aria-1.1/#aria-colindex) or [aria-rowindex](https://www.w3.org/TR/wai-aria-1.1/#aria-rowindex) is set to the position of a cell within a row or column, respectively. * If the grid includes cells that span multiple rows or multiple columns, and if the grid role is NOT applied to an HTML table element, then [aria-rowspan](https://www.w3.org/TR/wai-aria-1.1/#aria-rowspan) or [aria-colspan](https://www.w3.org/TR/wai-aria-1.1/#aria-colspan) is applied as described in [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). | * Check example @   [Layout Grid Examples](https://www.w3.org/TR/wai-aria-practices/examples/grid/LayoutGrids.html): Three example implementations of grids that are used to lay out widgets, including a collection of navigation links, a message recipients list, and a set of search results. * [Data Grid Examples](https://www.w3.org/TR/wai-aria-practices/examples/grid/dataGrids.html): Three example implementations of grid that include features relevant to presenting tabular information, such as content editing, sort, and column hiding. * [Advanced Data Grid Example](https://www.w3.org/TR/wai-aria-practices/examples/grid/advancedDataGrid.html): Example of a grid with behaviors and features similar to a typical spreadsheet, including cell and row selection. |
| **Link** | * Enter: Executes the link and moves focus to the link target. * Shift + F10 (Optional): Opens a context menu for the link. | The element containing the link text or graphic has role of [link](https://www.w3.org/TR/wai-aria-1.1/#link). | * Check examples @  [Link Examples](https://www.w3.org/TR/wai-aria-practices/examples/link/link.html): Link widgets constructed from HTML span and div elements. |
| **ListBox** | For a vertically oriented listbox:   * When a single-select listbox receives focus:   + If none of the options are selected before the listbox receives focus, the first option receives focus. Optionally, the first option may be automatically selected.   + If an option is selected before the listbox receives focus, focus is set on the selected option. * When a multi-select listbox receives focus:   + If none of the options are selected before the listbox receives focus, focus is set on the first option and there is no automatic change in the selection state.   + If one or more options are selected before the listbox receives focus, focus is set on the first option in the list that is selected. * Down Arrow: Moves focus to the next option. Optionally, in a single-select listbox, selection may also move with focus. * Up Arrow: Moves focus to the previous option. Optionally, in a single-select listbox, selection may also move with focus. * Home (Optional): Moves focus to first option. Optionally, in a single-select listbox, selection may also move with focus. Supporting this key is strongly recommended for lists with more than five options. * End (Optional): Moves focus to last option. Optionally, in a single-select listbox, selection may also move with focus. Supporting this key is strongly recommended for lists with more than five options. * Type-ahead is recommended for all listboxes, especially those with more than seven options:   + Type a character: focus moves to the next item with a name that starts with the typed character.   + Type multiple characters in rapid succession: focus moves to the next item with a name that starts with the string of characters typed. * **Multiple Selection**: Authors may implement either of two interaction models to support multiple selection: a recommended model that does not require the user to hold a modifier key, such as Shift or Control, while navigating the list or an alternative model that does require modifier keys to be held while navigating in order to avoid losing selection states.   + Recommended selection model -- holding modifier keys is not necessary:     - Space: changes the selection state of the focused option .     - Shift + Down Arrow (Optional): Moves focus to and toggles the selected state of the next option.     - Shift + Up Arrow (Optional): Moves focus to and toggles the selected state of the previous option.     - Shift + Space (Optional): Selects contiguous items from the most recently selected item to the focused item.     - Control + Shift + Home (Optional): Selects the focused option and all options up to the first option. Optionally, moves focus to the first option.     - Control + Shift + End (Optional): Selects the focused option and all options down to the last option. Optionally, moves focus to the last option.     - Control + A (Optional): Selects all options in the list. Optionally, if all options are selected, it may also unselect all options.   + Alternative selection model -- moving focus without holding a Shift or Control modifier unselects all selected nodes except the focused node:     - Shift + Down Arrow: Moves focus to and toggles the selection state of the next option.     - Shift + Up Arrow: Moves focus to and toggles the selection state of the previous option.     - Control + Down Arrow: Moves focus to the next option without changing its selection state.     - Control + Up Arrow: Moves focus to the previous option without changing its selection state.     - Control + Space Changes the selection state of the focused option.     - Shift + Space (Optional): Selects contiguous items from the most recently selected item to the focused item.     - Control + Shift + Home (Optional): Selects the focused option and all options up to the first option. Optionally, moves focus to the first option.     - Control + Shift + End (Optional): Selects the focused option and all options down to the last option. Optionally, moves focus to the last option.     - Control + A (Optional): Selects all options in the list. Optionally, if all options are selected, it may also unselect all options. | * An element that contains or owns all the listbox options has role [listbox](https://www.w3.org/TR/wai-aria-1.1/" \l "listbox). * Each option in the listbox has role [option](https://www.w3.org/TR/wai-aria-1.1/#option) and is a DOM descendant of the element with role listbox or is referenced by an [aria-owns](https://www.w3.org/TR/wai-aria-1.1/#aria-owns) property on the listbox element. * If the listbox is not part of another widget, then it has a visible label referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the element with role listbox. * In a single-select listbox, the selected option has [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true. * if the listbox supports multiple selection:   + The element with role listbox has [aria-multiselectable](https://www.w3.org/TR/wai-aria-1.1/#aria-multiselectable) set to true.   + All selected options have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true.   + All options that are not selected have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to false. * If the complete set of available options is not present in the DOM due to dynamic loading as the user scrolls, their [aria-setsize](https://www.w3.org/TR/wai-aria-1.1/#aria-setsize) and [aria-posinset](https://www.w3.org/TR/wai-aria-1.1/#aria-posinset) attributes are set appropriately. * If options are arranged horizontally, the element with role listbox has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to horizontal. The default value of aria-orientation for listbox is vertical. | * Examples @   [Scrollable Listbox Example](https://www.w3.org/TR/wai-aria-practices/examples/listbox/listbox-scrollable.html): Single-select listbox that scrolls to reveal more options, similar to HTML selectwith size attribute greater than one. * [Collapsible Dropdown Listbox Example](https://www.w3.org/TR/wai-aria-practices/examples/listbox/listbox-collapsible.html): Single-select collapsible listbox that expands when activated, similar to HTML select with the attribute size="1". * [Example Listboxes with Rearrangeable Options](https://www.w3.org/TR/wai-aria-practices/examples/listbox/listbox-rearrangeable.html): Examples of both single-select and multi-select listboxes with accompanying toolbars where options can be added, moved, and removed. |
| **Menu or Menu Bar** | * The following description of keyboard behaviors assumes: * A horizontal menubar containing several menuitem elements. * All items in the menubar have child submenus that contain multiple vertically arranged items. * Some of the menuitem elements in the submenus have child submenus with items that are also vertically arranged. * When reading the following descriptions, also keep in mind that: * Focusable elements, which may have role menuitem, menuitemradio, or menuitemcheckbox, are referred to as items. * If a behavior applies to only certain types of items, e.g., menuitem elements, the specific role name is used. * Submenus, also known as pop-up menus, are elements with role menu. * Except where noted, menus opened from a menubutton behave the same as menus opened from a menubar. * When a menu opens, or when a menubar receives focus, keyboard focus is placed on the first item. All items are focusable as described in [5.6 Keyboard Navigation Inside Components](https://www.w3.org/TR/wai-aria-practices/#kbd_general_within). * Enter:   + When focus is on a menuitem that has a submenu, opens the submenu and places focus on its first item.   + Otherwise, activates the item and closes the menu. * Space:   + (Optional): When focus is on a menuitemcheckbox, changes the state without closing the menu.   + (Optional): When focus is on a menuitemradio that is not checked, without closing the menu, checks the focused menuitemradio and unchecks any other checked menuitemradio element in the same group.   + (Optional): When focus is on a menuitem that has a submenu, opens the submenu and places focus on its first item.   + (Optional): When focus is on a menuitem that does not have a submenu, activates the menuitem and closes the menu. * Down Arrow:   + When focus is on a menuitem in a menubar, opens its submenu and places focus on the first item in the submenu.   + When focus is in a menu, moves focus to the next item, optionally wrapping from the last to the first. * Up Arrow:   + When focus is in a menu, moves focus to the previous item, optionally wrapping from the first to the last.   + (Optional): When focus is on a menuitem in a menubar, opens its submenu and places focus on the last item in the submenu. * Right Arrow:   + When focus is in a menubar, moves focus to the next item, optionally wrapping from the last to the first.   + When focus is in a menu and on a menuitem that has a submenu, opens the submenu and places focus on its first item.   + When focus is in a menu and on an item that does not have a submenu, performs the following 3 actions:     - Closes the submenu and any parent menus.     - Moves focus to the next menuitem in the menubar.     - Either: (Recommended) opens the submenu of that menuitem without moving focus into the submenu, or opens the submenu of that menuitem and places focus on the first item in the submenu. * Note that if the menubar were not present, e.g., the menus were opened from a menubutton, Right Arrow would not do anything when focus is on an item that does not have a submenu. * Left Arrow:   + When focus is in a menubar, moves focus to the previous item, optionally wrapping from the first to the last.   + When focus is in a submenu of an item in a menu, closes the submenu and returns focus to the parent menuitem.   + When focus is in a submenu of an item in a menubar, performs the following 3 actions:     - Closes the submenu.     - Moves focus to the previous menuitem in the menubar.     - Either: (Recommended) opens the submenu of that menuitem without moving focus into the submenu, or opens the submenu of that menuitem and places focus on the first item in the submenu. * Home: If arrow key wrapping is not supported, moves focus to the first item in the current menu or menubar. * End: If arrow key wrapping is not supported, moves focus to the last item in the current menu or menubar. * Any key that corresponds to a printable character (Optional): Move focus to the next menu item in the current menu whose label begins with that printable character. * Escape: Close the menu that contains focus and return focus to the element or context, e.g., menu button or parent menuitem, from which the menu was opened. * Tab: Moves focus to the next element in the tab sequence, and if the item that had focus is not in a menubar, closes its menu and all open parent menu containers. * Shift + Tab: Moves focus to the previous element in the tab sequence, and if the item that had focus is not in a menubar, closes its menu and all open parent menu containers. | * A menu is a container of items that represent choices. The element serving as the menu has a role of either [menu](https://www.w3.org/TR/wai-aria-1.1/#menu) or [menubar](https://www.w3.org/TR/wai-aria-1.1/" \l "menubar). * The items contained in a menu are child elements of the containing menu or menubar and have any of the following roles:   + [menuitem](https://www.w3.org/TR/wai-aria-1.1/#menuitem)   + [menuitemcheckbox](https://www.w3.org/TR/wai-aria-1.1/#menuitemcheckbox)   + [menuitemradio](https://www.w3.org/TR/wai-aria-1.1/#menuitemradio) * If activating a [menuitem](https://www.w3.org/TR/wai-aria-1.1/" \l "menuitem) opens a submenu, the menuitem is known as a parent menuitem. A submenu's menu element is:   + Contained inside the same menu element as its parent menuitem.   + Is the sibling element immediately following its parent menuitem. * A parent menuitem has [aria-haspopup](https://www.w3.org/TR/wai-aria-1.1/#aria-haspopup) set to either menu or true. * A parent menuitem has [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) set to false when its child menu is not visible and set to true when the child menu is visible. * One of the following approaches is used to enable scripts to move focus among items in a menu as described in [5.6 Keyboard Navigation Inside Components](https://www.w3.org/TR/wai-aria-practices/#kbd_general_within):   + The menu container has tabindex set to -1 or 0 and [aria-activedescendant](https://www.w3.org/TR/wai-aria-1.1/#aria-activedescendant) set to the ID of the focused item.   + Each item in the menu has tabindex set to -1, except in a menubar, where the first item has tabindexset to 0. * When a [menuitemcheckbox](https://www.w3.org/TR/wai-aria-1.1/" \l "menuitemcheckbox) or [menuitemradio](https://www.w3.org/TR/wai-aria-1.1/" \l "menuitemradio) is checked, [aria-checked](https://www.w3.org/TR/wai-aria-1.1/#aria-checked) is set to true. * When a menu item is disabled, [aria-disabled](https://www.w3.org/TR/wai-aria-1.1/#aria-disabled) is set to true. * Items in a menu may be divided into groups by placing an element with a role of [separator](https://www.w3.org/TR/wai-aria-1.1/#separator) between groups. For example, this technique should be used when a menu contains a set of [menuitemradio](https://www.w3.org/TR/wai-aria-1.1/" \l "menuitemradio) items. * All [separators](https://www.w3.org/TR/wai-aria-1.1/#separator) should have [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) consistent with the separator's orientation. * If a menubar has a visible label, the element with role menubar has [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) set to a value that refers to the labeling element. Otherwise, the menubar element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If a menubar is vertically oriented, it has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to vertical. The default value of aria-orientation for a menubar is horizontal. * An element with role menu either has:   + [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) set to a value that refers to the menuitem or button that controls its display.   + A label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If a menu is horizontally oriented, it has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to horizontal. The default value of aria-orientation for a menu is vertical. | * Examples @ * [Navigation Menubar Example](https://www.w3.org/TR/wai-aria-practices/examples/menubar/menubar-1/menubar-1.html): Demonstrates a menubar that provides site navigation. * [Editor Menubar Example](https://www.w3.org/TR/wai-aria-practices/examples/menubar/menubar-2/menubar-2.html): Demonstrates menu radios and menu checkboxes in submenus of a menubar that provides text formatting commands for a text field. |
| **Menu Button** | * With focus on the button:   + Enter: opens the menu and places focus on the first menu item.   + Space: Opens the menu and places focus on the first menu item.   + (Optional) Down Arrow: opens the menu and moves focus to the first menu item.   + (Optional) Up Arrow: opens the menu and moves focus to the last menu item. * The keyboard behaviors needed after the menu is open are described in [3.14 Menu or Menu bar](https://www.w3.org/TR/wai-aria-practices/#menu). | * The element that opens the menu has role [button](https://www.w3.org/TR/wai-aria-1.1/#button). * The element with role button has [aria-haspopup](https://www.w3.org/TR/wai-aria-1.1/#aria-haspopup) set to either menu or true. * When the menu is displayed, the element with role button has [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) set to true. When the menu is hidden, it is recommended that aria-expanded is not present. If aria-expanded is specified when the menu is hidden, it is set to false. * The element that contains the menu items displayed by activating the button has role [menu](https://www.w3.org/TR/wai-aria-1.1/#menu). * Optionally, the element with role button has a value specified for [aria-controls](https://www.w3.org/TR/wai-aria-1.1/#aria-controls) that refers to the element with role menu. * Additional roles, states, and properties needed for the menu element are described in [3.14 Menu or Menu bar](https://www.w3.org/TR/wai-aria-practices/#menu). | Examples @   * [Navigation Menu Button](https://www.w3.org/TR/wai-aria-practices/examples/menu-button/menu-button-links.html): A menu button made from an HTML a element that opens a menu of items that behave as links. * [Action Menu Button Example Using element.focus()](https://www.w3.org/TR/wai-aria-practices/examples/menu-button/menu-button-actions.html): A menu button made from an HTML button element that opens a menu of actions or commands where focus in the menu is managed using element.focus(). * [Action Menu Button Example Using aria-activedescendant](https://www.w3.org/TR/wai-aria-practices/examples/menu-button/menu-button-actions-active-descendant.html): A button that opens a menu of actions or commands where focus in the menu is managed using aria-activedescendant. |
| **Radio Group** | * When a radio group receives focus:   + If a radio button is checked, focus is set on the checked button.   + If none of the radio buttons are checked, focus is set on the first radio button in the group. * Space: checks the focused radio button if it is not already checked. * Right Arrow and Down Arrow: move focus to the next radio button in the group, uncheck the previously focused button, and check the newly focused button. If focus is on the last button, focus moves to the first button. * Left Arrow and Up Arrow: move focus to the previous radio button in the group, uncheck the previously focused button, and check the newly focused button. If focus is on the first button, focus moves to the last button. | * The radio buttons are contained in or owned by an element with role [radiogroup](https://www.w3.org/TR/wai-aria-1.1/" \l "radiogroup) . * Each radio button element has role [radio](https://www.w3.org/TR/wai-aria-1.1/#radio) . * If a radio button is checked, the radio element has [aria-checked](https://www.w3.org/TR/wai-aria-1.1/#aria-checked) set to true. If it is not checked, it has [aria-checked](https://www.w3.org/TR/wai-aria-1.1/#aria-checked) set to false. * Each radio element is labeled by its content, has a visible label referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) , or has a label specified with [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label) . * The radiogroup element has a visible label referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) or has a label specified with [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label) . * If elements providing additional information about either the radio group or each radio button are present, those elements are referenced by the radiogroup element or radio elements with the [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) property. | * Examples @  [Radio Group Example Using Roving tabindex](https://www.w3.org/TR/wai-aria-practices/examples/radio/radio-1/radio-1.html) * [Radio Group Example Using aria-activedescendant](https://www.w3.org/TR/wai-aria-practices/examples/radio/radio-2/radio-2.html) |
| **Slider** | * Right Arrow: Increase the value of the slider by one step. * Up Arrow: Increase the value of the slider by one step. * Left Arrow: Decrease the value of the slider by one step. * Down Arrow: Decrease the value of the slider by one step. * Home: Set the slider to the first allowed value in its range. * End: Set the slider to the last allowed value in its range. * Page Up (Optional): Increment the slider by an amount larger than the step change made by Up Arrow. * Page Down (Optional): Decrement the slider by an amount larger than the step change made by Down Arrow. | * The element serving as the focusable slider control has role [slider](https://www.w3.org/TR/wai-aria-1.1/#slider). * The slider element has the [aria-valuenow](https://www.w3.org/TR/wai-aria-1.1/#aria-valuenow) property set to a decimal value representing the current value of the slider. * The slider element has the [aria-valuemin](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemin) property set to a decimal value representing the minimum allowed value of the slider. * The slider element has the [aria-valuemax](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemax) property set to a decimal value representing the maximum allowed value of the slider. * If the value of aria-valuenow is not user-friendly, e.g., the day of the week is represented by a number, the[aria-valuetext](https://www.w3.org/TR/wai-aria-1.1/#aria-valuetext) property is set to a string that makes the slider value understandable, e.g., "Monday". * If the slider has a visible label, it is referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the slider element. Otherwise, the slider element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the slider is vertically oriented, it has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to vertical. The default value of aria-orientation for a slider is horizontal. | * Examples: * [Horizontal Slider Examples](https://www.w3.org/TR/wai-aria-practices/examples/slider/slider-1.html): Demonstrates using three horizontally aligned sliders to make a color picker. * [Slider Examples with aria-orientation and aria-valuetext](https://www.w3.org/TR/wai-aria-practices/examples/slider/slider-2.html): Three thermostat control sliders that demonstrate using aria-orientation and aria-valuetext. |
| **Slider (Multi-** | * Each thumb is in the page tab sequence and has the same keyboard interaction as a [single-thumb slider](https://www.w3.org/TR/wai-aria-practices/#slider_kbd_interaction). * The tab order remains constant regardless of thumb value and visual position within the slider. For example, if the value of a thumb changes such that it moves past one of the other thumbs, the tab order does not change. | * Each element serving as a focusable slider thumb has role [slider](https://www.w3.org/TR/wai-aria-1.1/#slider). * Each slider element has the [aria-valuenow](https://www.w3.org/TR/wai-aria-1.1/#aria-valuenow) property set to a decimal value representing the current value of the slider. * Each slider element has the [aria-valuemin](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemin) property set to a decimal value representing the minimum allowed value of the slider. * Each slider element has the [aria-valuemax](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemax) property set to a decimal value representing the maximum allowed value of the slider. * When the range (e.g. minimum and/or maximum value) of another slider is dependent on the current value of a slider, the values of [aria-valuemin](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemin) or [aria-valuemax](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemax) of the dependent sliders are updated when the value changes. * If a value of aria-valuenow is not user-friendly, e.g., the day of the week is represented by a number, the[aria-valuetext](https://www.w3.org/TR/wai-aria-1.1/#aria-valuetext) property is set to a string that makes the slider value understandable, e.g., "Monday". * If a slider has a visible label, it is referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the slider element. Otherwise, the slider element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If a slider is vertically oriented, it has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to vertical. The default value of aria-orientation for a slider is horizontal. | * Examples:   [Multi-Thumb Slider Examples](https://www.w3.org/TR/wai-aria-practices/examples/slider/multithumb-slider.html): Demonstrates two-thumb sliders for picking price ranges for an airline flight and hotel reservation. |
| **Spinbutton** | * Up Arrow: Increases the value. * Down Arrow: Decreases the value. * Home: If the spinbutton has a minimum value, sets the value to its minimum. * End: If the spinbutton has a maximum value, sets the value to its maximum. * Page Up (Optional): Increases the value by a larger step than Up Arrow. * Page Down (Optional): Decreases the value by a larger step than Down Arrow. * If the spinbutton text field allows directly editing the value, the following keys are supported:   + Standard single line text editing keys appropriate for the device platform (see note below).   + Printable Characters: Type characters in the textbox. Note that many implementations allow only certain characters as part of the value and prevent input of any other characters. For example, an hour-and-minute spinner would allow only integer values from 0 to 59, the colon ':', and the letters 'AM' and 'PM'. Any other character input does not change the contents of the text field nor the value of the spinbutton. | * The focusable element serving as the spinbutton has role [spinbutton](https://www.w3.org/TR/wai-aria-1.1/" \l "spinbutton). This is typically an element that supports text input. * The spinbutton element has the [aria-valuenow](https://www.w3.org/TR/wai-aria-1.1/#aria-valuenow) property set to a decimal value representing the current value of the spinbutton. * The spinbutton element has the [aria-valuemin](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemin) property set to a decimal value representing the minimum allowed value of the spinbutton if it has a known minimum value. * The spinbutton element has the [aria-valuemax](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemax) property set to a decimal value representing the maximum allowed value of the spinbutton if it has a known maximum value. * If the value of aria-valuenow is not user-friendly, e.g., the day of the week is represented by a number, the[aria-valuetext](https://www.w3.org/TR/wai-aria-1.1/#aria-valuetext) property is set on the spinbutton element to a string that makes the spinbutton value understandable, e.g., "Monday". * If the spinbutton has a visible label, it is referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the spinbutton element. Otherwise, the spinbutton element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). | * Work to develop a spinbutton example is tracked by [issue 125.](https://github.com/w3c/aria-practices/issues/125) |
| **Table** | * Not applicable. | * The table container has role [table](https://www.w3.org/TR/wai-aria-1.1/#table). * Each row container has role [row](https://www.w3.org/TR/wai-aria-1.1/#row) and is either a DOM descendant of or owned by the table element or an element with role [rowgroup](https://www.w3.org/TR/wai-aria-1.1/" \l "rowgroup). * Each cell is either a DOM descendant of or owned by a row element and has one of the following roles:   + [columnheader](https://www.w3.org/TR/wai-aria-1.1/#columnheader) if the cell contains a title or header information for the column.   + [rowheader](https://www.w3.org/TR/wai-aria-1.1/#rowheader) if the cell contains title or header information for the row.   + [cell](https://www.w3.org/TR/wai-aria-1.1/#cell) if the cell does not contain column or row header information. * If there is an element in the user interface that serves as a label for the table, [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) is set on the table element with a value that refers to the labeling element. Otherwise, a label is specified for the table element using [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the table has a caption or description, [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) is set on the table element with a value referring to the element containing the description. * If the table contains sortable columns or rows, [aria-sort](https://www.w3.org/TR/wai-aria-1.1/#aria-sort) is set to an appropriate value on the header cell element for the sorted column or row as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). * If there are conditions where some rows or columns are hidden or not present in the DOM, e.g., there are widgets on the page for hiding rows or columns, the following properties are applied as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties).   + [aria-colcount](https://www.w3.org/TR/wai-aria-1.1/#aria-colcount) or [aria-rowcount](https://www.w3.org/TR/wai-aria-1.1/#aria-rowcount) is set to the total number of columns or rows, respectively.   + [aria-colindex](https://www.w3.org/TR/wai-aria-1.1/#aria-colindex) or [aria-rowindex](https://www.w3.org/TR/wai-aria-1.1/#aria-rowindex) is set to the position of a cell within a row or column, respectively. * If the table includes cells that span multiple rows or multiple columns, then [aria-rowspan](https://www.w3.org/TR/wai-aria-1.1/#aria-rowspan) or [aria-colspan](https://www.w3.org/TR/wai-aria-1.1/#aria-colspan) is applied as described in [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). | * [Table Example](https://www.w3.org/TR/wai-aria-practices/examples/table/table.html): ARIA table made using HTML div and span elements. |
| **Tabs** | For the tab list:   * Tab: When focus moves into the tab list, places focus on the active tab element . When the tab list contains the focus, moves focus to the next element in the page tab sequence outside the tablist, which is typically either the first focusable element inside the tab panel or the tab panel itself. * When focus is on a tab element in a horizontal tab list:   + Left Arrow: moves focus to the previous tab. If focus is on the first tab, moves focus to the last tab. Optionally, activates the newly focused tab (See note below).   + Right Arrow: Moves focus to the next tab. If focus is on the last tab element, moves focus to the first tab. Optionally, activates the newly focused tab (See note below). * When focus is on a tab in a tablist with either horizontal or vertical orientation:   + Space or Enter: Activates the tab if it was not activated automatically on focus.   + Home (Optional): Moves focus to the first tab. Optionally, activates the newly focused tab (See note below).   + End (Optional): Moves focus to the last tab. Optionally, activates the newly focused tab (See note below).   + Shift + F10: If the tab has an associated pop-up menu, opens the menu.   + Delete (Optional): If deletion is allowed, deletes (closes) the current tab element and its associated tab panel, sets focus on the tab following the tab that was closed, and optionally activates the newly focused tab. If there is not a tab that followed the tab that was deleted, e.g., the deleted tab was the right-most tab in a left-to-right horizontal tab list, sets focus on and optionally activates the tab that preceded the deleted tab. If the application allows all tabs to be deleted, and the user deletes the last remaining tab in the tab list, the application moves focus to another element that provides a logical work flow. As an alternative to Delete, or in addition to supporting Delete, the delete function is available in a context menu. | * The element that serves as the container for the set of tabs has role [tablist](https://www.w3.org/TR/wai-aria-1.1/" \l "tablist). * Each element that serves as a tab has role [tab](https://www.w3.org/TR/wai-aria-1.1/#tab) and is contained within the element with role tablist. * Each element that contains the content panel for a tab has role [tabpanel](https://www.w3.org/TR/wai-aria-1.1/" \l "tabpanel). * If the tab list has a visible label, the element with role tablist has [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) set to a value that refers to the labeling element. Otherwise, the tablist element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * Each element with role tab has the property [aria-controls](https://www.w3.org/TR/wai-aria-1.1/#aria-controls) referring to its associated tabpanel element. * The active tab element has the state [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true and all other tab elements have it set to false. * Each element with role tabpanel has the property [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) referring to its associated tab element. * If a tab element has a pop-up menu, it has the property [aria-haspopup](https://www.w3.org/TR/wai-aria-1.1/#aria-haspopup) set to either menu or true. * If the tablist element is vertically oriented, it has the property [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to vertical. The default value of aria-orientation for a tablist element is horizontal. | * [Tabs With Automatic Activation](https://www.w3.org/TR/wai-aria-practices/examples/tabs/tabs-1/tabs.html): A tabs widget where tabs are automatically activated and their panel is displayed when they receive focus. * [Tabs With Manual Activation](https://www.w3.org/TR/wai-aria-practices/examples/tabs/tabs-2/tabs.html): A tabs widget where users activate a tab and display its panel by pressing Space or Enter. |
| **ToolBar** | * When the toolbar receives focus, focus is set on the first control that is not disabled. Optionally, if the toolbar has previously contained focus, focus is set on the control that last had focus. * For a horizontal toolbar (the default):   + Left Arrow: Moves focus to the previous control. Optionally, focus movement may wrap from the first element to the last element.   + Right Arrow: Moves focus to the next control. Optionally, focus movement may wrap from the last element to the first element. * Home (Optional): Moves focus to first element. * End (Optional): Moves focus to last element. | * The element that serves as the toolbar container has role [toolbar](https://www.w3.org/TR/wai-aria-1.1/#toolbar). * If the toolbar has a visible label, it is referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the toolbar element. Otherwise, the toolbar element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the controls are arranged vertically, the toolbar element has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to vertical. The default orientation is horizontal. | * [Toolbar Example](https://www.w3.org/TR/wai-aria-practices/examples/toolbar/toolbar.html): A toolbar that uses roving tabindex to manage focus and contains several buttons, including a menu button. |
| **Tooltip Widget** | * Escape: Dismisses the Tooltip. | * The element that serves as the tooltip container has role [tooltip](https://www.w3.org/TR/wai-aria-1.1/#tooltip). * The element that triggers the tooltip references the tooltip element with [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby). | * Work to develop a tooltip example is tracked by [issue 127.](https://github.com/w3c/aria-practices/issues/127) * **NOTE:** This design pattern is work in progress; it does not yet have task force consensus. Progress and discussions are captured in [issue 128.](https://github.com/w3c/aria-practices/issues/128) |
| **Tree View** | For a vertically oriented tree:   * When a single-select tree receives focus:   + If none of the nodes are selected before the tree receives focus, focus is set on the first node.   + If a node is selected before the tree receives focus, focus is set on the selected node. * When a multi-select tree receives focus:   + If none of the nodes are selected before the tree receives focus, focus is set on the first node.   + If one or more nodes are selected before the tree receives focus, focus is set on the first selected node. * Right arrow:   + When focus is on a closed node, opens the node; focus does not move.   + When focus is on a open node, moves focus to the first child node.   + When focus is on an end node, does nothing. * Left arrow:   + When focus is on an open node, closes the node.   + When focus is on a child node that is also either an end node or a closed node, moves focus to its parent node.   + When focus is on a root node that is also either an end node or a closed node, does nothing. * Down Arrow: Moves focus to the next node that is focusable without opening or closing a node. * Up Arrow: Moves focus to the previous node that is focusable without opening or closing a node. * Home: Moves focus to the first node in the tree without opening or closing a node. * End: Moves focus to the last node in the tree that is focusable without opening a node. * Enter: activates a node, i.e., performs its default action. For parent nodes, one possible default action is to open or close the node. In single-select trees where selection does not follow focus (see note below), the default action is typically to select the focused node. * Type-ahead is recommended for all trees, especially for trees with more than 7 root nodes:   + Type a character: focus moves to the next node with a name that starts with the typed character.   + Type multiple characters in rapid succession: focus moves to the next node with a name that starts with the string of characters typed. * \* (Optional): Expands all siblings that are at the same level as the current node. * **Selection in multi-select trees:** Authors may implement either of two interaction models to support multiple selection: a recommended model that does not require the user to hold a modifier key, such as Shift or Control, while navigating the list or an alternative model that does require modifier keys to be held while navigating in order to avoid losing selection states.   + Recommended selection model -- holding a modifier key while moving focus is not necessary:     - Space: Toggles the selection state of the focused node.     - Shift + Down Arrow (Optional): Moves focus to and toggles the selection state of the next node.     - Shift + Up Arrow (Optional): Moves focus to and toggles the selection state of the previous node.     - Shift + Space (Optional): Selects contiguous nodes from the most recently selected node to the current node.     - Control + Shift + Home (Optional): Selects the node with focus and all nodes up to the first node. Optionally, moves focus to the first node.     - Control + Shift + End (Optional): Selects the node with focus and all nodes down to the last node. Optionally, moves focus to the last node.     - Control + A (Optional): Selects all nodes in the tree. Optionally, if all nodes are selected, it can also unselect all nodes.   + Alternative selection model -- Moving focus without holding the Shift or Control modifier unselects all selected nodes except for the focused node:     - Shift + Down Arrow: Moves focus to and toggles the selection state of the next node.     - Shift + Up Arrow: Moves focus to and toggles the selection state of the previous node.     - Control + Down Arrow: Without changing the selection state, moves focus to the next node.     - Control + Up Arrow: Without changing the selection state, moves focus to the previous node.     - Control + Space: Toggles the selection state of the focused node.     - Shift + Space (Optional): Selects contiguous nodes from the most recently selected node to the current node.     - Control + Shift + Home (Optional): Selects the node with focus and all nodes up to the first node. Optionally, moves focus to the first node.     - Control + Shift + End (Optional): Selects the node with focus and all nodes down to the last node. Optionally, moves focus to the last node.     - Control + A (Optional): Selects all nodes in the tree. Optionally, if all nodes are selected, it can also unselect all nodes. | * All tree nodes are contained in or owned by an element with role [tree](https://www.w3.org/TR/wai-aria-1.1/#tree). * Each element serving as a tree node has role [treeitem](https://www.w3.org/TR/wai-aria-1.1/" \l "treeitem). * Each root node is contained in the element with role tree or referenced by an [aria-owns](https://www.w3.org/TR/wai-aria-1.1/#aria-owns) property set on the tree element. * Each parent node contains or owns an element with role [group](https://www.w3.org/TR/wai-aria-1.1/#group). * Each child node is contained in or owned by an element with role [group](https://www.w3.org/TR/wai-aria-1.1/#group) that is contained in or owned by the node that serves as the parent of that child. * Each element with role treeitem that serves as a parent node has [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) set to false when the node is in a closed state and set to true when the node is in an open state. End nodes do not have the aria-expanded attribute because, if they were to have it, they would be incorrectly described to assistive technologies as parent nodes. * If the tree supports selection of more than one node, the element with role tree has [aria-multiselectable](https://www.w3.org/TR/wai-aria-1.1/#aria-multiselectable) set to true. Otherwise, aria-multiselectable is either set to false or the default value of false is implied. * If the tree does not support multiple selection, [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) is set to true for the selected node and it is not present on any other node in the tree. * if the tree supports multiple selection:   + All selected nodes have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true.   + All nodes that are selectable but not selected have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to false.   + If the tree contains nodes that are not selectable, those nodes do not have the aria-selected state. * The element with role tree has either a visible label referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) or a value specified for [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the complete set of available nodes is not present in the DOM due to dynamic loading as the user moves focus in or scrolls the tree, each node has [aria-level](https://www.w3.org/TR/wai-aria-1.1/#aria-level), [aria-setsize](https://www.w3.org/TR/wai-aria-1.1/#aria-setsize), and [aria-posinset](https://www.w3.org/TR/wai-aria-1.1/#aria-posinset) specified. * If the tree element is horizontally oriented, it has [aria-orientation](https://www.w3.org/TR/wai-aria-1.1/#aria-orientation) set to horizontal. The default value of aria-orientation for a tree is vertical. | * [File Directory Treeview Example Using Computed Properties](https://www.w3.org/TR/wai-aria-practices/examples/treeview/treeview-1/treeview-1a.html): A file selector tree that demonstrates browser support for automatically computing aria-level, aria-posinset and aria-setsize based on DOM structure. * [File Directory Treeview Example Using Declared Properties](https://www.w3.org/TR/wai-aria-practices/examples/treeview/treeview-1/treeview-1b.html): A file selector tree that demonstrates how to explicitly define values for aria-level, aria-posinset and aria-setsize. * [Navigation Treeview Example Using Computed Properties](https://www.w3.org/TR/wai-aria-practices/examples/treeview/treeview-2/treeview-2a.html): A tree that provides navigation to a set of web pages and demonstrates browser support for automatically computing aria-level, aria-posinset and aria-setsize based on DOM structure. * [Navigation Treeview Example Using Declared Properties](https://www.w3.org/TR/wai-aria-practices/examples/treeview/treeview-2/treeview-2b.html): A tree that provides navigation to a set of web pages and demonstrates how to explicitly define values for aria-level, aria-posinset and aria-setsize. |
| **Treegrid** | The following keys provide treegrid navigation by moving focus among rows and cells of the grid. Implementations of treegrid make these key commands available when an element in the grid has received focus, e.g., after a user has moved focus to the grid with Tab. Moving focus into the grid may result in the first cell or the first row being focused. Whether focus goes to a cell or the row depends on author preferences and whether row focus is supported, since some treegrids may not provide focus to rows.   * Enter: If cell-only focus is enabled and focus is on the first cell with the aria-expanded property, opens or closes the child rows.,Otherwise, performs the default action for the cell. * Tab: If the row containing focus contains focusable elements (e.g., inputs, buttons, links, etc.), moves focus to the next input in the row. If focus is on the last focusable element in the row, moves focus out of the treegrid widget to the next focusable element. * Right Arrow:   + If focus is on a collapsed row, expands the row.   + If focus is on an expanded row or is on a row that does not have child rows, moves focus to the first cell in the row.   + If focus is on the right-most cell in a row, focus does not move.   + If focus is on any other cell, moves focus one cell to the right. * Left Arrow:   + If focus is on an expanded row, collapses the row.   + If focus is on a collapsed row or on a row that does not have child rows, focus does not move.   + If focus is on the first cell in a row and row focus is supported, moves focus to the row.   + If focus is on the first cell in a row and row focus is not supported, focus does not move.   + If focus is on any other cell, moves focus one cell to the left. * Down Arrow:   + If focus is on a row, moves focus one row down. If focus is on the last row, focus does not move.   + If focus is on a cell, moves focus one cell down. If focus is on the bottom cell in the column, focus does not move. * Up Arrow:   + If focus is on a row, moves focus one row up. If focus is on the first row, focus does not move.   + If focus is on a cell, moves focus one cell up. If focus is on the top cell in the column, focus does not move. * Page Down:   + If focus is on a row, moves focus down an author-determined number of rows, typically scrolling so the bottom row in the currently visible set of rows becomes one of the first visible rows. If focus is in the last row, focus does not move.   + If focus is on a cell, moves focus down an author-determined number of cells, typically scrolling so the bottom row in the currently visible set of rows becomes one of the first visible rows. If focus is in the last row, focus does not move. * Page Up:   + If focus is on a row, moves focus up an author-determined number of rows, typically scrolling so the top row in the currently visible set of rows becomes one of the last visible rows. If focus is in the first row, focus does not move.   + If focus is on a cell, moves focus up an author-determined number of cells, typically scrolling so the top row in the currently visible set of rows becomes one of the last visible rows. If focus is in the first row, focus does not move. * Home:   + If focus is on a row, moves focus to the first row. If focus is in the first row, focus does not move.   + If focus is on a cell, moves focus to the first cell in the row. If focus is in the first cell of the row, focus does not move. * End:   + If focus is on a row, moves focus to the last row. If focus is in the last row, focus does not move.   + If focus is on a cell, moves focus to the last cell in the row. If focus is in the last cell of the row, focus does not move. * Control + Home:   + If focus is on a row, moves focus to the first row. If focus is in the first row, focus does not move.   + If focus is on a cell, moves focus to the first cell in the column. If focus is in the first row, focus does not move. * Control + End:   + If focus is on a row, moves focus to the last row. If focus is in the last row, focus does not move.   + If focus is on a cell, moves focus to the last cell in the column. If focus is in the last row, focus does not move.   If a treegrid supports selection of cells, rows, or columns, the following keys are commonly used for these functions.   * Control + Space:   + If focus is on a row, selects all cells.   + If focus is on a cell, selects the column that contains the focus. * Shift + Space:   + If focus is on a row, selects the row.   + If focus is on a cell, selects the row that contains the focus. If the treegrid includes a column with checkboxes for selecting rows, this key can serve as a shortcut for checking the box when focus is not on the checkbox. * Control + A: Selects all cells. * Shift + Right Arrow:   + If focus is on a row, does not change selection.   + if focus is on a cell, extends selection one cell to the right. * Shift + Left Arrow:   + If focus is on a row, does not change selection.   + if focus is on a cell, extends selection one cell to the left. * Shift + Down Arrow:   + If focus is on a row, extends selection to all the cells in the next row.   + If focus is on a cell, extends selection one cell down. * Shift + Up Arrow:   + If focus is on a row, extends selection to all the cells in the previous row.   + If focus is on a cell, extends selection one cell up. | * The treegrid container has role [treegrid](https://www.w3.org/TR/wai-aria-1.1/" \l "treegrid). * Each row container has role [row](https://www.w3.org/TR/wai-aria-1.1/#row) and is either a DOM descendant of or owned by the treegrid element or an element with role [rowgroup](https://www.w3.org/TR/wai-aria-1.1/" \l "rowgroup). * Each cell is either a DOM descendant of or owned by a row element and has one of the following roles:   + [columnheader](https://www.w3.org/TR/wai-aria-1.1/#columnheader) if the cell contains a title or header information for the column.   + [rowheader](https://www.w3.org/TR/wai-aria-1.1/#rowheader) if the cell contains title or header information for the row.   + [gridcell](https://www.w3.org/TR/wai-aria-1.1/#gridcell) if the cell does not contain column or row header information. * A row that can be expanded or collapsed to show or hide a set of child rows is a parent row. Each parent row has the [aria-expanded](https://www.w3.org/TR/wai-aria-1.1/#aria-expanded) state set on either the row element or on a cell contained in therow. The aria-expanded state is set to false when the child rows are not displayed and set to true when the child rows are displayed. Rows that do not control display of child rows do not have the aria-expanded attribute because, if they were to have it, they would be incorrectly described to assistive technologies as parent rows. * If the treegrid supports selection of more than one row or cell, it is a multi-select treegrid and the element with role treegrid has [aria-multiselectable](https://www.w3.org/TR/wai-aria-1.1/#aria-multiselectable) set to true. Otherwise, it is a single-select treegrid, and aria-multiselectable is either set to false or the default value of false is implied. * If the treegrid is a single-select treegrid, [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) is set to true on the selected row or cell, and it is not present on any other row or cell in the treegrid. * if the treegrid is a multi-select treegrid:   + All selected rows or cells have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to true.   + All rows and cells that are not selected have [aria-selected](https://www.w3.org/TR/wai-aria-1.1/#aria-selected) set to false. * If there is an element in the user interface that serves as a label for the treegrid, [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) is set on the grid element with a value that refers to the labeling element. Otherwise, a label is specified for the grid element using [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * If the treegrid has a caption or description, [aria-described by](https://www.w3.org/TR/wai-aria-1.1/#aria-describedby) is set on the grid element with a value referring to the element containing the description. * If the treegrid provides sort functions, [aria-sort](https://www.w3.org/TR/wai-aria-1.1/#aria-sort) is set to an appropriate value on the header cell element for the sorted column or row as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). * If the treegrid provides content editing functionality and contains cells that may have edit capabilities disabled in certain conditions, [aria-readonly](https://www.w3.org/TR/wai-aria-1.1/#aria-readonly) is set to true on cells where editing is disabled. If edit functions are disabled for all cells, instead of setting aria-readonly to true on every cell, aria-readonly may be set to true on the treegrid element. Treegrids that do not provide cell content editing functions do not include the aria-readonly attribute on any of their elements. * If there are conditions where some rows or columns are hidden or not present in the DOM, e.g., data is dynamically loaded when scrolling or the grid provides functions for hiding rows or columns, the following properties are applied as described in the section on [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties).   + [aria-colcount](https://www.w3.org/TR/wai-aria-1.1/#aria-colcount) or [aria-rowcount](https://www.w3.org/TR/wai-aria-1.1/#aria-rowcount) is set to the total number of columns or rows, respectively.   + [aria-colindex](https://www.w3.org/TR/wai-aria-1.1/#aria-colindex) or [aria-rowindex](https://www.w3.org/TR/wai-aria-1.1/#aria-rowindex) is set to the position of a cell within a row or column, respectively. * If the treegrid includes cells that span multiple rows or multiple columns, and if the treegrid role is NOT applied to an HTML table element, then [aria-rowspan](https://www.w3.org/TR/wai-aria-1.1/#aria-rowspan) or [aria-colspan](https://www.w3.org/TR/wai-aria-1.1/#aria-colspan) is applied as described in [grid and table properties](https://www.w3.org/TR/wai-aria-practices/#gridAndTableProperties). | * [E-mail Inbox treegrid Example](https://www.w3.org/TR/wai-aria-practices/examples/treegrid/treegrid-1.html): A treegrid for navigating an e-mail inbox that demonstrates three keyboard navigation models -- rows first, cells first, and cells only. |
| **Window Splitter** | * Left Arrow: Moves a vertical splitter to the left. * Right Arrow: Moves a vertical splitter to the right. * Up Arrow: Moves a horizontal splitter up. * Down Arrow: Moves a horizontal splitter down. * Enter: If the primary pane is not collapsed, collapses the pane. If the pane is collapsed, restores the splitter to its previous position. * Home (Optional): Moves splitter to the position that gives the primary pane its smallest allowed size. This may completely collapse the primary pane. * End (Optional): Moves splitter to the position that gives the primary pane its largest allowed size. This may completely collapse the secondary pane. * F6 (Optional): Cycle through window panes. | * The element that serves as the focusable splitter has role [separator](https://www.w3.org/TR/wai-aria-1.1/#separator). * The separator element has the [aria-valuenow](https://www.w3.org/TR/wai-aria-1.1/#aria-valuenow) property set to a decimal value representing the current position of the separator. * The separator element has the [aria-valuemin](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemin) property set to a decimal value that represents the position where the primary pane has its minimum size. This is typically 0. * The separator element has the [aria-valuemax](https://www.w3.org/TR/wai-aria-1.1/#aria-valuemax) property set to a decimal value that represents the position where the primary pane has its maximum size . This is typically 100. * If the primary pane has a visible label, it is referenced by [aria-labelled by](https://www.w3.org/TR/wai-aria-1.1/#aria-labelledby) on the separator element. Otherwise, the separator element has a label provided by [aria-label](https://www.w3.org/TR/wai-aria-1.1/#aria-label). * The separator element has [aria-controls](https://www.w3.org/TR/wai-aria-1.1/#aria-controls) referring to the primary pane. | * **NOTE:** ARIA 1.1 introduced changes to the separator role so it behaves as a widget when focusable. While this pattern has been revised to match the ARIA 1.1 specification, the task force will not complete its review until a functional example that matches the ARIA 1.1 specification is complete. Progress on this pattern is tracked by[issue 129.](https://github.com/w3c/aria-practices/issues/129) * Work to develop an example window splitter widget is tracked by [issue 130.](https://github.com/w3c/aria-practices/issues/130) |

Landmark Regions:

|  |  |  |  |
| --- | --- | --- | --- |
| Element | HTML5 Technique | WAI-ARIA Roles, States, and Properties | Notes |
| Banner | * The HTML5 header element defines a banner landmark when its context is the body element. * The HTML5 header element is not considered a banner landmark when it is descendant of any of following elements (see [HTML Accessibility Mappings](http://w3c.github.io/aria/html-aam/html-aam.html)):   + article   + aside   + main   + nav   + section | * If the HTML5 header element technique is not being used, a role="banner" attribute should be used to define a banner landmark. | *Examples* [Banner Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/banner.html) |
| Complementary | * Use the HTML5 aside element to define a complementary landmark. | * If the HTML5 aside element technique is not being used, use a role="complementary" attribute to define a complementary landmark. | *Examples* [Complementary Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/complementary.html) |
| ContentInfo | * The HTML5 footer element defines a contentinfo landmark when its context is the body element. * The HTML5 footer element is not considered a contentinfo landmark when it is descendant of any of following elements (see [HTML Accessibility Mappings](http://w3c.github.io/aria/html-aam/html-aam.html)):   + article   + aside   + main   + nav   + section | * If the HTML5 footer element technique is not being used, a role="contentinfo" attribute should be used to define a contentinfo landmark. | *Examples* [Contentinfo Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/contentinfo.html) |
| Form | * The HTML5 form element defines a form landmark when it has an accessible name (e.g. aria-labelled by, aria-label or title). | * Use the role="form" to identify a region of the page; do not use it to identify every form field. | *Examples* [Form Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/form.html) |
| Main | * Use the HTML5 main element to define a main landmark. | * If the HTML5 main element technique is not being used, use a role="main" attribute to define a main landmark. | *Examples* [Main Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/main.html) |
| Navigation | * Use the HTML5 nav element to define a navigation landmark. | * If the HTML5 nav element technique is not being used, use a role="navigation" attribute to define a navigation landmark. | *Examples* [Navigation Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/navigation.html) |
| Region | * The HTML5 section element defines a region landmark when it has an accessible name (e.g. aria-labelled by, aria-label or title). | * If the HTML5 section element technique is not being used, use a role="region" attribute to define a regionlandmark. | *Examples* [Region Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/region.html) |
| Search | * There is no HTML5 element that defines a search landmark. | * The role="search" attribute defines a search landmark. | *Examples* [Search Landmark Example](https://www.w3.org/TR/wai-aria-practices/examples/landmarks/search.html) |