Page Contents

Grid (Interactive Tabular Data and Layout Containers) Pattern

About This Pattern

A grid (https://w3c.github.io/aria/#grid) widget is a container that enables users to navigate the information or interactive elements it contains using directional navigation keys, such as arrow keys, `Home`, and `End`. As a generic container widget that offers flexible keyboard navigation, it can serve a wide variety of needs. It can be used for purposes as simple as grouping a collection of checkboxes or navigation links or as complex as creating a full-featured spreadsheet application. While the words "row" and "column" are used in the names of WAI-ARIA attributes and by assistive technologies when describing and presenting the logical structure of elements with the `grid` role, using the `grid` role on an element does not necessarily imply that its visual presentation is tabular.

When presenting content that is tabular, consider the following factors when choosing between implementing this `grid` pattern or the table pattern.

- A `grid` is a composite widget so it:
  - Always contains multiple focusable elements.
  - Only one of the focusable elements contained by the grid is included in the page tab sequence.

- Requires the author to provide code that manages focus movement inside it.
- All focusable elements contained in a table are included in the page tab sequence.

Uses of the `grid` pattern broadly fall into two categories: presenting tabular information (data grids) and grouping other widgets (layout grids). Even though both data grids and layout grids employ the same ARIA roles, states, and properties, differences in their content and purpose surface factors that are important to consider in keyboard interaction design. To address these factors, the following two sections describe separate keyboard interaction patterns for data and layout grids.

Examples

Layout Grid Examples: 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: 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: Example of a grid with behaviors and features similar to a typical spreadsheet, including cell and row selection.

Data Grids For Presenting Tabular Information

A `grid` can be used to present tabular information that has column titles, row titles, or both. The `grid` pattern is particularly useful if the tabular information is editable or interactive. For example, when data elements are links to more information, rather than presenting them in a static table and including the links in the tab sequence, implementing the `grid` pattern provides users with intuitive and efficient keyboard navigation of the grid contents as well as a shorter tab sequence for the page. A `grid` may also offer functions, such as cell content editing, selection, cut, copy, and paste.

In a grid, every cell contains a focusable element or is itself focusable, regardless of whether the cell content is editable or interactive. There is one exception: if column or row header cells do not provide functions, such as sort or filter, they do not need to be focusable. One reason it is important for all cells to be able to receive or contain keyboard focus is that screen readers will typically be in their application reading mode, rather than their document reading mode, when users are interacting with the grid. While in application mode, a screen reader user hears only focusable elements and content that labels focusable elements. So, screen reader users may unknowingly overlook elements contained in a grid that are either not focusable or not used to label a column or row.

Keyboard Interaction For Data Grids

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.

Note

- When the above grid navigation keys move focus, whether the focus is set on an element inside the cell or the grid cell depends on cell content. See Whether to Focus on a Cell or an Element Inside It(⇘ #gridNav_focus).
- While navigation keys, such as arrow keys, are moving focus from cell to cell, they are not available to do something like operate a combobox or move an editing caret inside of a cell. If this functionality is needed, see Editing and Navigating Inside a Cell(⇘ #gridNav_inside).
- If navigation functions can dynamically add more rows or columns to the DOM, key events that move focus to the beginning or end of the grid, such as `Control + End`, may move focus to the last row in the DOM rather than the last available row in the back-end data.

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.

> **Note**
>
> See Key Assignment Conventions for Common Functions for cut, copy, and paste key assignments.

Layout Grids for Grouping Widgets

The `grid` pattern can be used to group a set of interactive elements, such as links, buttons, or checkboxes. Since only one element in the entire grid is included in the tab sequence, grouping with a grid can dramatically reduce the number of tab stops on a page. This is especially valuable if scrolling through a list of elements dynamically loads more of those elements from a large data set, such as in a continuous list of suggested products on a shopping site. If elements in a list like this were in the tab sequence, keyboard users are effectively trapped in the list. If any elements in the group also have associated elements that appear on hover, the `grid` pattern is also useful for providing keyboard access to those contextual elements of the user interface.

Unlike grids used to present data, A `grid` used for layout does not necessarily have header cells for labelling rows or columns and might contain only a single row or a single column. Even if it has multiple rows and columns, it may present a single, logically homogenous set of elements. For example, a list of recipients for a message may be a grid where each cell contains a link that represents a recipient. The grid may initially have a single row but then wrap into multiple rows as recipients are added. In such circumstances, grid navigation keys may also wrap so the user can read the list from beginning to end by pressing either `Right Arrow` or `Down Arrow`. While This type of focus movement wrapping can be very

helpful in a layout grid, it would be disorienting if used in a data grid, especially for users of assistive technologies.

Because arrow keys are used to move focus inside of a `grid`, a `grid` is both easier to build and use if the components it contains do not require the arrow keys to operate. If a cell contains an element like a listbox, then an extra key command to focus and activate the listbox is needed as well as a command for restoring the grid navigation functionality. Approaches to supporting this need are described in the section on Editing and Navigating Inside a Cell(⇘ #gridNav_inside).

Keyboard Interaction For Layout Grids

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. Optionally, if focus is on the right-most cell in the row, focus may move to the first cell in the following row. If focus is on the last cell in the grid, focus does not move.
- `Left Arrow`: Moves focus one cell to the left. Optionally, if focus is on the left-most cell in the row, focus may move to the last cell in the previous row. If focus is on the first cell in the grid, focus does not move.
- `Down Arrow`: Moves focus one cell down. Optionally, if focus is on the bottom cell in the column, focus may move to the top cell in the following column. If focus is on the last cell in the grid, focus does not move.
- `Up Arrow`: Moves focus one cell up. Optionally, if focus is on the top cell in the column, focus may move to the bottom cell in the previous column. If focus is on the first cell in the grid, focus does not move.
- `Page Down` (Optional): 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` (Optional): 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. Optionally, if the grid has a single column or fewer than three cells per row, focus may instead move to the first cell in the grid.
- `End`: moves focus to the last cell in the row that contains focus. Optionally, if the grid has a single column or fewer than three cells per row, focus may instead move to the last cell in the grid.

- `Control + Home` (optional): moves focus to the first cell in the first row.
- `Control + End` (Optional): moves focus to the last cell in the last row.

> ### Note
>
> - When the above grid navigation keys move focus, whether the focus is set on an element inside the cell or the grid cell depends on cell content. See Whether to Focus on a Cell or an Element Inside It(⇘ #gridNav_focus).
> - While navigation keys, such as arrow keys, are moving focus from cell to cell, they are not available to do something like operate a combobox or move an editing caret inside of a cell. If this functionality is needed, see Editing and Navigating Inside a Cell(⇘ #gridNav_inside).
> - If navigation functions can dynamically add more rows or columns to the DOM, key events that move focus to the beginning or end of the grid, such as `control + End`, may move focus to the last row in the DOM rather than the last available row in the back-end data.

It would be unusual for a layout grid to provide functions that require cell selection. If it did, though, 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.

> ### Note
>
> See Key Assignment Conventions for Common Functions for cut, copy, and paste key assignments.

Keyboard Interaction - Setting Focus and Navigating Inside Cells

This section describes two important aspects of keyboard interaction design shared by both data and layout grid patterns:

1. Choosing whether a cell or an element inside a cell receives focus in response to grid navigation key events.
2. Enabling grid navigation keys to be used to interact with elements inside of a cell.

Whether to Focus on a Cell Or an Element Inside It

For assistive technology users, the quality of experience when navigating a grid heavily depends on both what a cell contains and on where keyboard focus is set. For example, if a cell contains a button and a grid navigation key places focus on the cell instead of the button, screen readers announce the button label but do not tell users a button is present.

There are two optimal cell design and focus behavior combinations:

1. A cell contains one widget whose operation does not require arrow keys and grid navigation keys set focus on that widget. Examples of such widgets include link, button, menubutton, toggle button, radio button (not radio group), switch, and checkbox.
2. A cell contains text or a single graphic and grid navigation keys set focus on the cell.

While any combination of widgets, text, and graphics may be included in a single cell, grids that do not follow one of these two cell design and focus movement patterns add complexity for authors or users or both. The reference implementations included in the example section below demonstrate some strategies for making other cell designs as accessible as possible, but the most widely accessible experiences are likely to come by applying the above two patterns.

Editing and Navigating Inside a Cell

While navigation keys, such as arrow keys, are moving focus from cell to cell, they are not available to perform actions like operate a combobox or move an editing caret inside of a cell. The user may need keys that are used for grid navigation to operate elements inside a cell if a cell contains:

1. Editable content.
2. Multiple widgets.

3. A widget that utilizes arrow keys in its interaction model, such as a radio group or slider.

Following are common keyboard conventions for disabling and restoring grid navigation functions.

- `Enter`: Disables grid navigation and:
  - If the cell contains editable content, places focus in an input field, such as a textbox (https://w3c.github.io/aria/#textbox). If the input is a single-line text field, a subsequent press of `Enter` may either restore grid navigation functions or move focus to an input field in a neighboring cell.
  - If the cell contains one or more widgets, places focus on the first widget.
- `F2`:
  - If the cell contains editable content, places focus in an input field, such as a textbox (https://w3c.github.io/aria/#textbox). A subsequent press of `F2` restores grid navigation functions.
  - If the cell contains one or more widgets, places focus on the first widget. A subsequent press of `F2` restores grid navigation functions.
- Alphanumeric keys: If the cell contains editable content, places focus in an input field, such as a textbox (https://w3c.github.io/aria/#textbox).

When grid navigation is disabled, conventional changes to navigation behaviors include:

- `Escape`: restores grid navigation. If content was being edited, it may also undo edits.
- `Right Arrow` or `Down Arrow`: If the cell contains multiple widgets, moves focus to the next widget inside the cell, optionally wrapping to the first widget if focus is on the last widget. Otherwise, passes the key event to the focused widget.
- `Left Arrow` or `Up Arrow`: If the cell contains multiple widgets, moves focus to the previous widget inside the cell, optionally wrapping to the first widget if focus is on the last widget. Otherwise, passes the key event to the focused widget.
- `Tab`: moves focus to the next widget in the grid. Optionally, the focus movement may wrap inside a single cell or within the grid itself.
- `Shift + Tab`: moves focus to the previous widget in the grid. Optionally, the focus movement may wrap inside a single cell or within the grid itself.

WAI-ARIA Roles, States, and Properties

- The grid container has role grid (https://w3c.github.io/aria/#grid).
- Each row container has role row (https://w3c.github.io/aria/#row) and is either a DOM descendant of or owned by the `grid` element or an element with role rowgroup

(https://w3c.github.io/aria/#rowgroup).

- Each cell is either a DOM descendant of or owned by a `row` element and has one of the following roles:
  - columnheader (https://w3c.github.io/aria/#columnheader) if the cell contains a title or header information for the column.
  - rowheader (https://w3c.github.io/aria/#rowheader) if the cell contains title or header information for the row.
  - gridcell (https://w3c.github.io/aria/#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-labelledby (https://w3c.github.io/aria/#aria-labelledby) is set on the grid element with a value that refers to the labelling element. Otherwise, a label is specified for the grid element using aria-label (https://w3c.github.io/aria/#aria-label).
- If the grid has a caption or description, aria-describedby (https://w3c.github.io/aria/#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://w3c.github.io/aria/#aria-sort) is set to an appropriate value on the header cell element for the sorted column or row as described in the Grid and Table Properties Practice.
- If the grid supports selection, when a cell or row is selected, the selected element has aria-selected (https://w3c.github.io/aria/#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://w3c.github.io/aria/#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 Grid and Table Properties Practice.
  - aria-colcount (https://w3c.github.io/aria/#aria-colcount) or aria-rowcount (https://w3c.github.io/aria/#aria-rowcount) is set to the total number of columns or rows, respectively.
  - aria-colindex (https://w3c.github.io/aria/#aria-colindex) or aria-rowindex (https://w3c.github.io/aria/#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://w3c.github.io/aria/#aria-rowspan) or aria-colspan (https://w3c.github.io/aria/#aria-colspan) is applied as described in the Grid and Table Properties Practice.

Note

- If the element with the `grid` role is an HTML `table` element, then it is not necessary to use ARIA roles for rows and cells because the HTML elements have implied ARIA semantics. For example, an HTML `<TR>` has an implied ARIA role of `row`. A `grid` built from an HTML `table` that includes cells that span multiple rows or columns must use HTML `rowspan` and `colspan` and must not use `aria-rowspan` or `aria-colspan`.
- If rows or cells are included in a grid via aria-owns (https://w3c.github.io/aria/#aria-owns), they will be presented to assistive technologies after the DOM descendants of the `grid` element unless the DOM descendants are also included in the `aria-owns` attribute.
