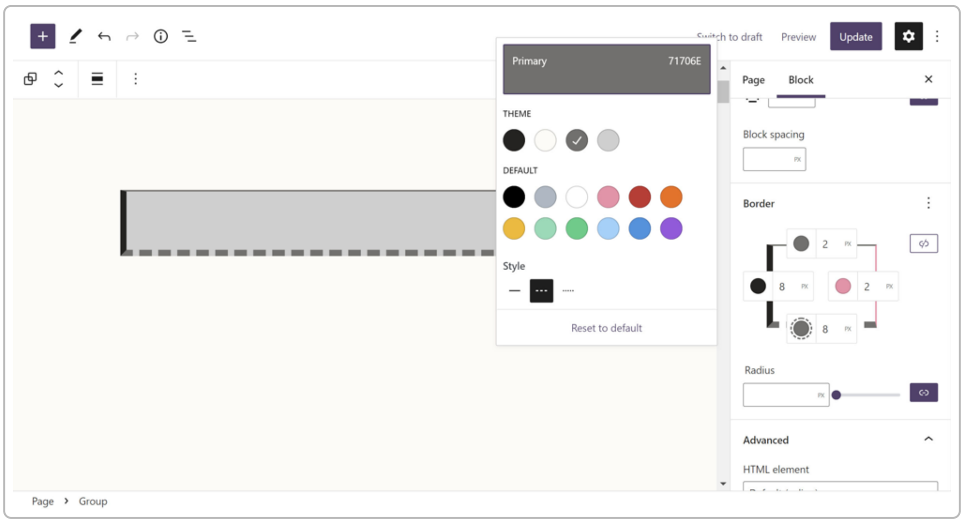
ChatGPT

Implementing Layered Sidebar Controls with ToolsPanel (Gutenberg)



Example of the WordPress block editor's Border panel using a layered ToolsPanel UI (note the three-dot dropdown menu next to "Border"). Users can toggle specific sub-controls (like individual border sides, color, style, radius) to show or hide them in the panel.

Gutenberg (the WordPress block editor) has moved away from static accordion panels (using `PanelBody`) in the inspector sidebar. Instead, it now uses the **ToolsPanel** approach for groups of design controls like Border, Color, Dimensions, etc [1]. This ToolsPanel UI provides a layered, progressive-disclosure experience – the panel header includes a dropdown (the three dots) that lets users **toggle individual settings** on or off within that group [2]. This means only the controls a user enables (or those with preset values) will be shown, reducing clutter from unused settings [3].

ToolsPanel and ToolsPanelItem Components

To implement a similar layered UI in the sidebar, you can use the **ToolsPanel** and **ToolsPanelItem** components from `@wordpress/components` (these are currently marked as experimental). A `ToolsPanel` acts as a container with a header label (e.g. "Border") and automatically generates a dropdown menu listing its child items [2]. Each child should be a `ToolsPanelItem`, which wraps an actual control (like a color picker, input, slider, etc.) and defines how it appears in the ToolsPanel menu. The block editor will handle the dropdown menu for you – each `ToolsPanelItem` with a unique label becomes a menu option that the user can toggle on to reveal that control, or toggle off to hide it.

Key properties of ToolsPanelItem: Every `ToolsPanelItem` requires a `label` (human-readable, and also used as an identifier) and a `hasValue` callback that returns a boolean indicating if that control currently has a value [4]. This is used to determine the menu item's checked state (for example, a border color control might be considered "active" if a color value is set). You can also specify `onSelect` and `onDeselect` callbacks to run custom logic when the user shows or hides the control [5] [6]. Typically, **onDeselect** is used to reset the attribute when a control is turned off (e.g. clear the border color if the user hides that control) [7]. There's also an optional `isShownByDefault` prop – if set to true, that item will *always* be shown by default (without needing the user to toggle it on) [8]. In the core design panels, a few essential sub-settings are shown by default, while others remain hidden until enabled by the user.

Notably, whether a ToolsPanel item is visible initially depends on its value and default settings. If an item has a saved value (or is marked `isShownByDefault`), it will appear in the panel automatically; otherwise it stays hidden until toggled on [9]. This way, previously configured options remain visible to the user, but unused options don't clutter the UI.

Usage Example

Implementing a ToolsPanel in your custom block's sidebar involves a few steps:

1. **Import the components:** In your block's edit component file, import `__experimentalToolsPanel` and `__experimentalToolsPanelItem` from `@wordpress/components` (and any specific controls you need, like color pickers, range inputs, etc). For example:

```
import {
    __experimentalToolsPanel as ToolsPanel,
    __experimentalToolsPanelItem as ToolsPanelItem,
    ColorPicker,
    __experimentalUnitControl as UnitControl,
    // ...other controls
} from '@wordpress/components';
```

   (The experimental prefix indicates the API is subject to change, but it's the same approach used by core blocks.)

2. **Wrap controls in a ToolsPanel:** Inside your block's sidebar inspector (typically within `<InspectorControls>`), add a `<ToolsPanel>` element. Give it a descriptive `label` (this text appears as the panel title and in the dropdown's aria-label) [10]. You can also provide a `resetAll` callback to define what happens if the user chooses "Reset all" from the panel's menu (e.g. clear all related attributes) [11]. For example:

```
<InspectorControls group="styles"> {/* or "advanced", etc., as
appropriate */}
  <ToolsPanel label="My Panel" resetAll={ resetAllFunction }>
    {/* ToolsPanelItems will go here */}
  </ToolsPanel>
</InspectorControls>
```

3. **Add ToolsPanelItem controls:** For each sub-property you want to include, add a `<ToolsPanelItem>` inside the ToolsPanel. Provide the required props:

4. `label`: Unique name for this control (e.g. "Border Color").

5. `hasValue`: a function that returns true if the corresponding attribute is set (this helps ToolsPanel decide if the control should be initially shown or marked active) [4] .
6. `onDeselect`: a function to execute when the user hides the control – usually this should reset the attribute to undefined or a default [6] .
7. (Optional) `onSelect`: if you want to initialize something when the control is first shown (not always needed, but you could use it to set a default value).
8. (Optional) `resetAllFilter`: a function that returns an object to reset this attribute; ToolsPanel will call all these when “Reset all” is triggered [12] .
9. (Optional) `isShownByDefault`: set to true for any control that should appear by default even with no value [8] .

Inside each `<ToolsPanelItem>`, render the actual UI control for that property (such as a `<ColorPicker>` for color, `<UnitControl>` for size, etc.), binding it to your block’s state/attributes. For example:

```
<ToolsPanelItem
    hasValue={ () => borderColor !== undefined }
    label="Border Color"
    onDeselect={ () => setAttributes({ borderColor: undefined }) }
>
    <ColorPicker
        color={ borderColor }
        onChange={ (newColor) => setAttributes({ borderColor: newColor }) }
        enableAlpha={ false }
    />
</ToolsPanelItem>
```

You would repeat similar blocks for “Border Style”, “Border Width”, etc., each with their own state and controls. If a control should always be visible (like a crucial setting), add `isShownByDefault` to its ToolsPanelItem [13] . Otherwise, it will start hidden until the user toggles it on via the menu.

1. **Reset All behavior (optional):** If you provided a `resetAll` on ToolsPanel, implement it to clear all the related attributes at once. The ToolsPanel’s menu will automatically include a “Reset all” option if you pass this prop [11] . Usually, `resetAll` can call each item’s onDeselect or use their `resetAllFilter` outputs. For example:

   ```
   const resetAll = () => {
     setAttributes({
       borderColor: undefined,
       borderStyle: undefined,
       borderWidth: undefined,
       // ...etc
     });
   };
   // ... <ToolsPanel label="Border" resetAll={ resetAll }> ...
   ```

With the above setup, your custom panel will function like the core **Border** panel: the header shows a **menu icon** (…) that lists all sub-controls (by label). Users can check or uncheck each item to show/hide that control in the UI. When an item is deselected, the `onDeselect` runs (e.g. resetting the value) and the control disappears from view. If an item has a value saved (or `isShownByDefault`), it will appear automatically in the panel so the user can see and adjust it [9]. This layered approach keeps the sidebar clean – only relevant controls are shown – while still allowing users to configure advanced properties on demand.

Finally, remember to consult the official documentation and examples. The WordPress Developer Handbook provides a **ToolsPanel reference** with usage examples [14] [15], and the 10up Gutenberg Best Practices guide offers a detailed example of adding custom ToolsPanel controls to blocks [1] [16]. Using ToolsPanel/ToolsPanelItem will give your sidebar UI the same layered, extensible structure seen in core blocks (e.g. the Border panel with its sub-controls for color, style, radius, etc.) [2].

Sources: WordPress Gutenberg component docs on ToolsPanel and ToolsPanelItem [2] [13] [15], Gutenberg best practices guide by 10up [1], and the official Gutenberg code examples [16] [17].

---

[1] [3] [11] [12] [16] ToolsPanel and ToolsPanelItem | 10up - WP Block Editor Best Practices
https://gutenberg.10up.com/guides/tools-panel/

[2] [4] [5] [9] [10] [13] [14] [15] [17] ToolsPanel – Block Editor Handbook | Developer.WordPress.org
https://developer.wordpress.org/block-editor/reference-guides/components/tools-panel/

[6] [7] [8] ToolsPanelItem – Block Editor Handbook | Developer.WordPress.org
https://developer.wordpress.org/block-editor/reference-guides/components/tools-panel-item/