SlotFill

- Usage
- Props
  - bubblesVirtually=false
  - bubblesVirtually=true

`Slot` and `Fill` are a pair of components which enable developers to render React UI elsewhere in a React element tree, a pattern often referred to as “portal” rendering. It is a pattern for component extensibility, where a single `Slot` may be occupied by multiple `Fill`s rendered in different parts of the application.

Slot/Fill was originally inspired by the `react-slot-fill` library.

Usage

At the root of your application, you must render a `SlotFillProvider` which coordinates `Slot` and `Fill` rendering.

Then, render a `Slot` component anywhere in your application, giving it a `name`. The `name` is either a `string` or a symbol. Symbol names are useful for slots that are supposed to be private, accessible only to clients that have access to the symbol value.

Any `Fill` will render its UI in this `Slot` space, even if rendered elsewhere in the application.

You can either use the `Fill` component directly, or create a wrapper component (as in the following example) to hide the slot name from the consumer.

```
import {
    SlotFillProvider,
    Slot,
    Fill,
    Panel,
    PanelBody,
} from '@wordpress/components';
```

```
const MySlotFillProvider = () => {
    const MyPanelSlot = () => (
        <Panel header="Panel with slot">
            <PanelBody>
                <Slot name="MyPanelSlot" />
            </PanelBody>
        </Panel>
    );

    MyPanelSlot.Content = () => <Fill name="MyPanelSlot">Panel body</Fill>;

    return (
        <SlotFillProvider>
            <MyPanelSlot />
            <MyPanelSlot.Content />
        </SlotFillProvider>
    );
};
```

There is also the `createSlotFill` helper method which was created to simplify the process of matching the corresponding `Slot` and `Fill` components:

```
const { Fill, Slot } = createSlotFill( 'Toolbar' );

const ToolbarItem = () => <Fill>My item</Fill>;

const Toolbar = () => (
    <div className="toolbar">
        <Slot />
    </div>
);
```

Props

The `SlotFillProvider` component does not accept any props (except `children`).

Both `Slot` and `Fill` accept a `name` string prop, where a `Slot` with a given `name` will render the `children` of any associated `Fill`s.

`Slot` accepts a `bubblesVirtually` prop which changes the method how the `Fill` children are rendered. With `bubblesVirtually`, the `Fill` is rendered using a React portal. That affects the event bubbling and React context propagation behaviour:

`bubblesVirtually=false`

- events will bubble to their parents on the DOM hierarchy (native event bubbling)
- the React elements inside the `Fill` will be rendered with React context of the `Slot`
- renders the `Fill` elements directly, inside a `Fragment`, with no wrapper DOM element

`bubblesVirtually=true`

- events will bubble to their virtual (React) parent in the React elements hierarchy
- the React elements inside the `Fill` will keep the React context of the `Fill` and its parents
- renders a wrapper DOM element inside which the `Fill` elements are rendered (used as an argument for React `createPortal`)

`Slot` with `bubblesVirtually=true` renders a wrapper DOM element (a `div` by default) and accepts additional props that customize this element, like `className` or `style`. You can also replace the `div` with another element by passing an `as` prop.

`Slot` **without** `bubblesVirtually` accepts an optional `children` prop, which is a function that receives `fills` array as a param. It allows you to perform additional processing: render a placeholder when there are no fills, or render a wrapper only when there are fills.

Example:

```
const Toolbar = ( { isMobile } ) => (
    <div className="toolbar">
        <Slot name="Toolbar">
            { ( fills ) => {
                return isMobile && fills.length > 3 ? (
                    <div className="toolbar__mobile-long">{ fills }</div>
                ) : (
                    fills
                );
            } }
        </Slot>
    </div>
);
```

Additional information (props) can also be passed from a `Slot` to a `Fill` by a combination of:
1. Adding a `fillProps` prop to the `Slot`.
2. Passing a function as `children` to the `Fill`. This function will receive the `fillProps` as an argument.

```
const { Fill, Slot } = createSlotFill( 'Toolbar' );

const ToolbarItem = () => (
    <Fill>
        { ( { hideToolbar } ) => {
            <Button onClick={ hideToolbar }>Hide</Button>;
        } }
    </Fill>
);

const Toolbar = () => {
    const hideToolbar = () => {
        console.log( 'Hide toolbar' );
    };
    return (
        <div className="toolbar">
            <Slot fillProps={ { hideToolbar } } />
        </div>
    );
};
```

□□□ #2
□ □□□ □□□ 2026-10-31 18:14:43 UTC
□ □□□ □□□ 2026-10-31 18:14:45 UTC