SlotFills Reference

- Usage overview
- Conditionally rendering SlotFill content
  - Restricting fills to the Post Editor
  - Restricting fills to certain post types.
  - Restricting fills to the Side Editor
  - Restricting fills to certain screens in the Site Editor.
- How do they work?
- Currently available SlotFills and examples

Slot and Fill are components that have been exposed to allow developers to inject items into some predefined places in the Gutenberg admin experience.

Please see the SlotFill component docs for more details.

In order to use them, we must leverage the @wordpress/plugins api to register a plugin that will inject our items.

Usage overview

In order to access the SlotFills, we need to do four things:

1. Import the `registerPlugin` method from the `@wordpress/plugins` package.
2. Import the SlotFill we want from the `@wordpress/editor'` package.
3. Define a component to render our changes. Our changes/additions will be wrapped in the SlotFill component we imported.
4. Register the plugin.

Here is an example using the `PluginPostStatusInfo` slotFill:

```
import { registerPlugin } from '@wordpress/plugins';
import { PluginPostStatusInfo } from '@wordpress/editor';
```

```
const PluginPostStatusInfoTest = () => (
    <PluginPostStatusInfo>
        <p>Post Status Info SlotFill</p>
    </PluginPostStatusInfo>
);

registerPlugin( 'post-status-info-test', { render: PluginPostStatusInfoTest } );
```

Conditionally rendering SlotFill content

With the exception of MainDashboardButton, every available SlotFill is exposed in both the Post Editor and Site Editor and any Fill that is registered will be rendered in both contexts. There are a number of approaches that can be implemented to conditionally render Fills.

Restricting fills to the Post Editor

A fill can be restricted to the Post Editor by checking to see if the current post type object property `viewable` is set to `true`. Any post type not set to `viewable`, does not have an associated edit post screen and is a good indicator that the user is not in the Post Editor. The example below will render its content on the edit post screen for any registered post type.

```
import { registerPlugin } from '@wordpress/plugins';
import {
    PluginDocumentSettingPanel,
    store as editorStore,
} from '@wordpress/editor';
import { store as coreStore } from '@wordpress/core-data';
import { useSelect } from '@wordpress/data';
import { __ } from '@wordpress/i18n';

/**
 * The component to be rendered  as part of the plugin.
 */
const EditPostDocumentSettingPanel = () => {
```

```
    // Retrieve information about the current post type.
    const isViewable = useSelect( ( select ) => {
        const postTypeName = select( editorStore ).getCurrentPostType();
        const postTypeObject = select( coreStore ).getPostType( postTypeName );
        return postTypeObject?.viewable;
    }, [] );

    // If the post type is not viewable, then do not render the plugin.
    if ( ! isViewable ) {
        return null;
    }

    return (
        <PluginDocumentSettingPanel
            name="custom-panel"
            title={ __( 'Post Editor Example' ) }
            className="custom-panel"
        >
            <p>{ __( 'Only appears in the Edit Post screen' ) }</p>
        </PluginDocumentSettingPanel>
    );
};

registerPlugin( 'example-post-edit-only', {
    render: EditPostDocumentSettingPanel,
} );
```

Restricting fills to certain post types.

The following example expands on the example above by creating an allow list of post types where the fill should be rendered. In this case, the fill is only rendered when editing pages.

```
import { registerPlugin } from '@wordpress/plugins';
import {
    PluginDocumentSettingPanel,
    store as editorStore,
} from '@wordpress/editor';
import { store as coreStore } from '@wordpress/core-data';
import { useSelect } from '@wordpress/data';
```

```
import { __, sprintf } from '@wordpress/i18n';

/**
 * The component to be rendered  as part of the plugin.
 */
const RestrictPostTypes = () => {
    // Retrieve information about the current post type.
    const { isViewable, postTypeName } = useSelect( ( select ) => {
        const postType = select( editorStore ).getCurrentPostType();
        const postTypeObject = select( coreStore ).getPostType( postType );
        return {
            isViewable: postTypeObject?.viewable,
            postTypeName: postType,
        };
    }, [] );

    // The list of post types that are allowed to render the plugin.
    const allowedPostTypes = [ 'page' ];

    // If the post type is not viewable or not in the allowed list, do not render the plugin.
    if ( ! isViewable || ! allowedPostTypes.includes( postTypeName ) ) {
        return null;
    }

    return (
        <PluginDocumentSettingPanel
            name="custom-panel"
            title={ __( 'Restrict Post Types Example' ) }
            className="custom-panel"
        >
            <p>
                { sprintf(
                    __(
                        'Only appears on Post Types that are in the allowed list. %s'
                    ),
                    allowedPostTypes.join( ', ' )
                ) }
            </p>
        </PluginDocumentSettingPanel>
```

```
    );
};

registerPlugin( 'example-restrict-post-types', {
    render: RestrictPostTypes,
} );
```

Restricting fills to the Side Editor

To restrict fills to the Site Editor, the reverse logic is true. If the post type object's `viewable` property is set to `true`, then the fill should not be rendered. The example below will render its content on any Site Editor screen.

```
import { registerPlugin } from '@wordpress/plugins';
import {
    PluginDocumentSettingPanel,
    store as editorStore,
} from '@wordpress/editor';
import { store as coreStore } from '@wordpress/core-data';
import { useSelect } from '@wordpress/data';
import { __ } from '@wordpress/i18n';

/**
 * The component to be rendered  as part of the plugin.
 */
const SiteEditorDocumentSettingPanel = () => {
    // Retrieve information about the current post type.
    const isViewable = useSelect( ( select ) => {
        const postTypeName = select( editorStore ).getCurrentPostType();
        const postTypeObject = select( coreStore ).getPostType( postTypeName );

        // A viewable post type is one than can be viewed in the WordPress admin. Internal
ones are not set to viewable.
        return postTypeObject?.viewable;
    }, [] );

    // If the post type is viewable, do not render my fill
    if ( isViewable ) {
        return null;
```

```
    }

    return (
        <PluginDocumentSettingPanel
            name="custom-panel"
            title={ __( 'Site Editor Example' ) }
            className="custom-panel"
        >
            <p>{ __( 'Only appears in the Site Editor' ) }</p>
        </PluginDocumentSettingPanel>
    );
};

registerPlugin( 'example-site-editor', {
    render: SiteEditorDocumentSettingPanel,
} );
```

Restricting fills to certain screens in the Site Editor.

This example builds on the example above by providing an allow list to control which screens a fill can be rendered within the Site Editor.

```
import { registerPlugin } from '@wordpress/plugins';
import {
    PluginDocumentSettingPanel,
    store as editorStore,
} from '@wordpress/editor';
import { store as coreStore } from '@wordpress/core-data';
import { useSelect } from '@wordpress/data';
import { __, sprintf } from '@wordpress/i18n';

/**
 * The component to be rendered  as part of the plugin.
 */
const SiteEditorDocumentSettingPanel = () => {
    // Allowed areas in the Site Editor.
```

```
    const allowedSiteEditorScreens = [
        'wp_template', // Templates
        'wp_block', // Patterns
        'wp_template_part', // Template Parts
    ];

    const { isViewable, postType } = useSelect( ( select ) => {
        const postTypeName = select( editorStore ).getCurrentPostType();
        const postTypeObject = select( coreStore ).getPostType( postTypeName );

        return {
            // A viewable post type is one than can be viewed in the WordPress admin. Internal
ones are not set to viewable.
            isViewable: postTypeObject?.viewable,
            postType: postTypeName,
        };
    }, [] );

    // If the post type is viewable, do not render my plugin.
    if ( isViewable || ! allowedSiteEditorScreens.includes( postType ) ) {
        return null;
    }

    return (
        <PluginDocumentSettingPanel
            name="custom-panel"
            title={ __( 'Restricted to Site Editor screens' ) }
            className="custom-panel"
        >
            <p>
                { sprintf(
                    __(
                        'Only appears on Editor Screens that are in the allowed list. %s'
                    ),
                    allowedSiteEditorScreens.join( ', ' )
                ) }
            </p>
        </PluginDocumentSettingPanel>
    );
```

```
};

registerPlugin( 'example-site-editor-only', {
    render: SiteEditorDocumentSettingPanel,
} );
```

How do they work?

SlotFills are created using `createSlotFill`. This creates two components, `Slot` and `Fill` which are then used to create a new component that is exported on the `wp.plugins` global.

Definition of the `PluginPostStatusInfo` SlotFill (see core code)

```
/**
 * Defines as extensibility slot for the Summary panel.
 */
import { createSlotFill, PanelRow } from '@wordpress/components';

export const { Fill, Slot } = createSlotFill( 'PluginPostStatusInfo' );

const PluginPostStatusInfo = ( { children, className } ) => (
    <Fill>
        <PanelRow className={ className }>{ children }</PanelRow>
    </Fill>
);

PluginPostStatusInfo.Slot = Slot;

export default PluginPostStatusInfo;
```

This new Slot is then exposed in the editor. The example below is from core and represents the Summary panel.

As we can see, the `<PluginPostStatusInfo.Slot>` is wrapping all of the items that will appear in the panel.
Any items that have been added via the SlotFill (see the example above), will be included in the `fills` parameter and be displayed in the end of the component.

See core code.

```
export default function PostSummary( { onActionPerformed } ) {
    const { isRemovedPostStatusPanel } = useSelect( ( select ) => {
        // We use isEditorPanelRemoved to hide the panel if it was programmatically removed.
We do
        // not use isEditorPanelEnabled since this panel should not be disabled through the
UI.
        const { isEditorPanelRemoved } = select( editorStore );
        return {
            isRemovedPostStatusPanel: isEditorPanelRemoved( PANEL_NAME ),
        };
    }, [] );

    return (
        <PostPanelSection className="editor-post-summary">
            <PluginPostStatusInfo.Slot>
                { ( fills ) => (
                    <>
                        <Stack direction="column" gap="lg">
                            <PostCardPanel
                                onActionPerformed={ onActionPerformed }
                            />
                            <PostFeaturedImagePanel withPanelBody={ false } />
                            <PostExcerptPanel />
                            <Stack direction="column" gap="xs">
                                <PostContentInformation />
                                <PostLastEditedPanel />
                            </Stack>
                            { ! isRemovedPostStatusPanel && (
                                <Stack direction="column" gap="sm">
                                    <Stack direction="column" gap="xs">
                                        <PostStatusPanel />
                                        <PostSchedulePanel />
                                        <PostURLPanel />
                                        <PostAuthorPanel />
                                        <PostTemplatePanel />
                                        <PostDiscussionPanel />
                                        <PageAttributesPanel />
                                        <PostSyncStatus />
                                        <BlogTitle />
```

```
                                        <PostsPerPage />
                                        <SiteDiscussion />
                                        <PostFormatPanel />
                                        <PostStickyPanel />
                                    </Stack>
                                    <TemplateAreas />
                                    { fills }
                                </Stack>
                            ) }
                        </Stack>
                    </>
                ) }
            </PluginPostStatusInfo.Slot>
        </PostPanelSection>
    );
}
```

Currently available SlotFills and examples

The following SlotFills are available in the `edit-post` or `editor` packages. Please refer to the individual items below for usage and example details:

- MainDashboardButton
- PluginBlockSettingsMenuItem
- PluginDocumentSettingPanel
- PluginMoreMenuItem
- PluginPostPublishPanel
- PluginPostStatusInfo
- PluginPrePublishPanel
- PluginSidebar
- PluginSidebarMoreMenuItem

□□□ #2
□ □□□ □□□ 2026-10-31 18:10:15 UTC
□ □□□ □□□ 2026-10-31 18:10:17 UTC