+++ title = “Custom panel option editors” +++

Custom panel option editors

The Grafana plugin platform comes with a range of editors that allow your users to customize a panel. The standard editors cover the most common types of options, such as text input and boolean switches. If you don’t find the editor you’re looking for, you can build your own. In this guide, you’ll learn how to build your own panel option editor.

The simplest editor is a React component that accepts two props: `value` and `onChange`. `value` contains the current value of the option, and `onChange` updates it.

The editor in the example below lets the user toggle a boolean value by clicking a button:

SimpleEditor.tsx

```
import React from 'react';
import { Button } from '@grafana/ui';
import { StandardEditorProps } from '@grafana/data';

export const SimpleEditor: React.FC<StandardEditorProps<boolean>> = ({ value, onChange }) =>
  return <Button onClick={() => onChange(!value)}>{value ? 'Disable' : 'Enable'}</Button>;
};
```

To use a custom panel option editor, use the `addCustomEditor` on the `OptionsUIBuilder` object in your `module.ts` file. Configure the editor to use by setting the `editor` property to the `SimpleEditor` component.

module.ts

```
export const plugin = new PanelPlugin<SimpleOptions>(SimplePanel).setPanelOptions((builder)
  return builder.addCustomEditor({
    id: 'label',
    path: 'label',
    name: 'Label',
    editor: SimpleEditor,
  });
});
```

Add settings to your panel option editor

If you’re using your custom editor to configure multiple options, you might want to be able to customize it. Add settings to your editor by setting the second template variable of `StandardEditorProps` to an interface that contains the settings you want to be able to configure.

You can access the editor settings through the `item` prop. Here's an example of an editor that populates a drop-down with a range of numbers. The range is defined by the `from` and `to` properties in the `Settings` interface.

SimpleEditor.tsx

```tsx
interface Settings {
  from: number;
  to: number;
}

export const SimpleEditor: React.FC<StandardEditorProps<number, Settings>> = ({ item, value,
  const options: Array<SelectableValue<number>> = [];

  // Default values
  const from = item.settings?.from ?? 1;
  const to = item.settings?.to ?? 10;

  for (let i = from; i <= to; i++) {
    options.push({
      label: i.toString(),
      value: i,
    });
  }

  return <Select options={options} value={value} onChange={(selectableValue) => onChange(sel
};
```

You can now configure the editor for each option, by configuring the `settings` property in the call to `addCustomEditor`.

```tsx
export const plugin = new PanelPlugin<SimpleOptions>(SimplePanel).setPanelOptions((builder)
  return builder.addCustomEditor({
    id: 'index',
    path: 'index',
    name: 'Index',
    editor: SimpleEditor,
    settings: {
      from: 1,
      to: 10,
    },
  });
});
```

Use query results in your panel option editor

Option editors can access the results from the last query. This lets you update your editor dynamically, based on the data returned by the data source.

> **Note:** This feature was introduced in 7.0.3. Anyone using an older version of Grafana will see an error when using your plugin.

The editor context is available through the `context` prop. The data frames returned by the data source are available under `context.data`.

SimpleEditor.tsx

```
export const SimpleEditor: React.FC<StandardEditorProps<string>> = ({ item, value, onChange,
  const options: SelectableValue<string>[] = [];

  if (context.data) {
    const frames = context.data;

    for (let i = 0; i < frames.length; i++) {
      options.push({
        label: frames[i].name,
        value: frames[i].name,
      });
    }
  }

  return <Select options={options} value={value} onChange={(selectableValue) => onChange(sel
};
```

Have you built a custom editor that you think would be useful to other plugin developers? Consider contributing it as a standard editor!