+++ title = "Add support for Explore queries" +++

Add support for Explore queries

This guide explains how to improve support for [Explore]({{< relref "../../explore/_index.md" >}}) in an existing data source plugin.

This guide assumes that you're already familiar with how to [Build a data source plugin]({{< relref "/tutorials/build-a-data-source-plugin.md" >}}).

With Explore, users can make ad-hoc queries without the use of a dashboard. This is useful when users want to troubleshoot or to learn more about the data.

Your data source already supports Explore by default, and will use the existing query editor for the data source. If you want to offer extended Explore functionality for your data source however, you can define a Explore-specific query editor.

Add a query editor for Explore

The query editor for Explore is similar to the query editor for the data source itself. In fact, you'll probably reuse the same components for both query editors.

1. Create a file `ExploreQueryEditor.tsx` in the `src` directory of your plugin, with the following content:

```
import React from 'react';

import { QueryEditorProps } from '@grafana/data';
import { QueryField } from '@grafana/ui';
import { DataSource } from './DataSource';
import { MyQuery, MyDataSourceOptions } from './types';

export type Props = QueryEditorProps<DataSource, MyQuery,
MyDataSourceOptions>;

export default (props: Props) => {
  return <h2>My query editor</h2>;
};
```

2. Configure the plugin to use the `ExploreQueryEditor`.

```
import ExploreQueryEditor from './ExploreQueryEditor';
```

```
export const plugin = new DataSourcePlugin<DataSource, MyQuery,
MyDataSourceOptions>(DataSource)
  .setConfigEditor(ConfigEditor)
  .setQueryEditor(QueryEditor)
  .setExploreQueryField(ExploreQueryEditor);
```

3. Add a `QueryField` to `ExploreQueryEditor`.

```
import { QueryField } from '@grafana/ui';
```

```
export default (props: Props) => {
  const { query } = props;

  const onQueryChange = (value: string, override?: boolean) => {
    const { query, onChange, onRunQuery } = props;

    if (onChange) {
      // Update the query whenever the query field changes.
      onChange({ ...query, queryText: value });

      // Run the query on Enter.
      if (override && onRunQuery) {
        onRunQuery();
      }
    }
  };

  return (
    <QueryField
      portalOrigin="mock-origin"
      onChange={onQueryChange}
      onRunQuery={props.onRunQuery}
      onBlur={props.onBlur}
      query={query.queryText || ''}
      placeholder="Enter a query"
    />
  );
};
```

Selecting preferred visualisation

Explore should by default select a reasonable visualization for your data so users do not have to tweak and play with the visualizations and just focus on querying. This usually works fairly well and Explore can figure out whether the returned data is time series data or logs or something else.

If this does not work for you or you want to show some data in a specific visualization, add a hint to your returned data frame using the `preferredVisualisationType` meta attribute.

You can construct a data frame with specific metadata:

```
const firstResult = new MutableDataFrame({
    fields: [...],
    meta: {
        preferredVisualisationType: 'logs',
    },
});
```

For possible options, refer to PreferredVisualisationType.