plotly

Star 23,446

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. **Sign up for early access now.**

Callback Error Handlers

New in Dash 2.18

Dash's default error handler displays unhandled exceptions that occur in a Dash app's callbacks in Dash Dev Tools in the app UI when debug mode is enabled (`debug=True`). When debug mode is disabled, no error is displayed in the app UI and exceptions that occur are displayed in the terminal.

You can override the default behavior and create error handlers to:

- Inform the app user about errors in the app. For example, using a notification.
- Forward details about errors to another system. For example, sending detailed information on the app exception via email.

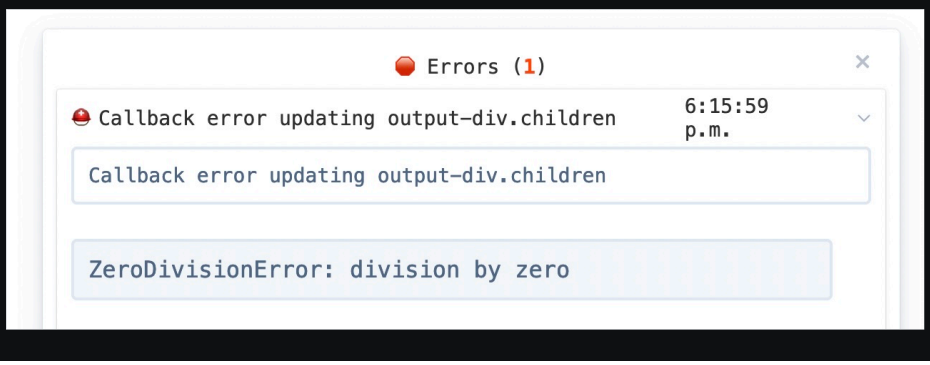
Callback error handlers in Dash apps can apply globally to all callbacks in an app or can be added to specific callbacks.

Basic Example

Here's an example of a Dash app that will raise an exception when the input is 0.

```
from dash import Dash, html, dcc, Input, Output, callback

app = Dash()

app.layout = html.Div([
    dcc.Input(id='input-number', type='number', value=1),
    html.Div(id='output-div')
])

@callback(
    Output('output-div', 'children'),
    Input('input-number', 'value')
)
def update_output(value):
    result = 10 / value
    return f'The result is {result}'

if __name__ == '__main__':
    app.run(debug=True)
```

In debug mode, you'll see this in Dash Dev Tools:



And with `debug=False`, you'll see something like this in the terminal:

```
ZeroDivisionError: division by zero
```

To add a custom error handler to this app:

1. Write an error handler function that receives one argument `err`. This function contains the logic for what you want to happen when an exception occurs. In the following code snippet, we just print some additional text along with that `err` argument to demonstrate that error handling is now happening within the custom error handler function. In later examples, we'll look at updating the app UI when an exception occurs.

   ```
   ...
   def custom_error_handler(err):
   # This function defines what we want to happen when an exception occurs
   # For now, we just print the exception to the terminal with additional text
       print(f"The app raised the following exception: {err}")
   ```

2. Pass the error handler function to the `Dash` app instance or a specific callback using the `on_error` parameter.

   ```
   ...
   app = Dash(on_error=custom_error_handler)
   ...
   ```

If you run this example and change the input value to 0, you'll see this in the terminal:

```
The app raised the following exception: division by zero
```

Notes on this example

- When the exception occurs, `err` is passed into the error handler and printed in the terminal.
- The `custom_error_handler` doesn't have a return statement, meaning it returns `None`. When an exception happens in a callback that uses a callback error handler, and that error handler returns `None`, the callback's outputs are not updated. You can also return values from the error handler to update the callback's outputs. See the **Updating the Callback Outputs** section below for more details.

Error Handlers for Specific Callbacks

In the previous example, we have one global callback error handler. You can also add error handlers at a callback level.

For example, here we have no global error handler, but an error handler on two of three callbacks in the app:

```
from dash import Dash, Input, Output, callback
...
def update_graph_error_handler(err):
    # This function defines what we want to happen when an exception occurs

def update_text_error_handler(err):
    # This function defines what we want to happen when an exception occurs

app = Dash()
...
@callback(
    Output('main-graph', 'figure'),
    Input('dropdown-input', 'value'),
    on_error=update_graph_error_handler
)
def update_main_graph(value):
...
@callback(
    Output('html-div', 'children'),
    Input('checklist-input', 'value'),
    on_error=update_text_error_handler
)
```

```
def update_text(value):
...
@callback(
    Output('graph-by-country', 'figure'),
    Input('slider-input', 'value')
)
def update_graph_by_country(value):
...
```

Callback-specific error handlers can be used alongside a global callback error handler. In this example, the `global_error_handler` applies to the callbacks in the app that do not have their own error handlers. In this example, that's the `update_graph_by_country` callback.

```
from dash import Dash, Input, Output, callback
...
def update_graph_error_handler(err):
    # Callback error handler logic

def update_text_error_handler(err):
    # Callback error handler logic

def global_error_handler(err):
    # Global error handler logic

app = Dash(on_error=global_error_handler)
...
@callback(
    Output('main-graph', 'figure'),
    Input('dropdown-input', 'value'),
    on_error=update_graph_error_handler
)
def update_main_graph(value)
...
@callback(
    Output('html-div', 'children'),
    Input('checklist-input', 'value'),
    on_error=update_text_error_handler
)
def update_text(value)
...
@callback(
    Output('graph-by-country', 'figure'),
    Input('slider-input', 'value')
)
def update_graph_by_country(value)
...
```

Updating the Callback Outputs

The error handlers in the previous examples have no return statements. This means that the callbacks they run for won't update outputs when an exception happens. You can update the outputs by returning values from the callback error handler.

In the following example, the `output-div` component's `children` property is updated with `f'The result is {result}'` when there is no exception in the callback, and with `f"There was an error processing your input. Error: {err}. Enter another value."` when there is an exception:

```
from dash import Dash, html, dcc, Input, Output, callback

def update_output_error_handler(err):
    return f"There was an error processing your input. Error: {err}. Enter another value."

app = Dash(on_error=update_output_error_handler)

app.layout = html.Div([
    dcc.Input(id='input-number', type='number', value=1),
    html.Div(id='output-div')
])

@callback(
```

```
    Output('output-div', 'children'),
    Input('input-number', 'value')
)
def update_output(value):
    result = 10 / value
    return f'The result is {result}'

if __name__ == '__main__':
    app.run(debug=False)
```

Returning `None` from the error handler

In cases where you have a callback with a single output and want to update it to `None` from the error handler, return a list containing `None`, like this: `[None]`. Ensure that the `Output` on the callback and the return value from the `callback` also use lists:

```
def update_output_error_handler(err):
    # Error handler logic
    return [None]  # Returning `None` within a list

@callback(
    [Output('my-output', 'children')],  # Output within a list
    Input('my-input', 'value'),
    on_error=update_output_error_handler
)
def update_output(input_value):
    return [f'This is the input value: {input_value}']  # Return value within a list
```

Updating Other Component Properties

In cases where you want to update component properties that were not outputs on the callback (for example, to display an error notification elsewhere in the app UI), you can use **dash.set_props**.

`set_props` takes the `id` of the component to update as its first argument, followed by a `dict` of properties to update. Each `dict` key should be a property name, with the key's value being the value to update the component property to.

In the following example, when an exception occurs in the `update_graph` callback, the `custom_error_handler` function updates the `"children"` property of the component with the `id` `"error-output"`. This displays the message "An error occurred", just under the dropdown in the app.

```
from dash import Dash, Input, Output, html, dcc, callback, set_props
import plotly.express as px

import pandas as pd
...
def custom_error_handler(err):
    set_props("error-output", dict(children="An error occurred"))

app = Dash(on_error=custom_error_handler)

app.layout = [
    html.Span(
        [
            dcc.Dropdown(
                id="dropdown-input",
                options=["Asia", "Europe", "Africa", "Americas", "Oceania"],
                value="Asia",
            ),
            html.P(id="error-output"),
        ]
    ),
    dcc.Graph(id="graph-output"),
]

@callback(
    Output("graph-output", "figure"),
    Input("dropdown-input", "value"),
)
def update_graph(value):
```

```
    url = "https://raw.githubusercontent.com/plotly/datasets/master/gapminder20078.csv"
    df = pd.read_csv(url)
    ...
```

Exception Information

You can use information about the exception to display different errors based on the type of exception. Similarly, if you want to forward the error information to an external system, you may want to send additional information about the error.

In the following example, we create an error handler that sends an email containing the exception details.

Here we create a subject for the email using the exception, the `err`.

```
def custom_error_handler(err):
    email_subject = f"Error {str(err)}"
    ...
```

And here, we create a body for the email that contains the traceback information.

```
import traceback
...
def custom_error_handler(err):
    email_subject = f"Error {str(err)}"
    email_body = traceback.format_exc()
    ...
```

Callback Context

Error handler functions also have access to details about the callback context such as which inputs triggered the callback. The following example adds to the previous example that constructed a string with the traceback information. Here, we include details about what triggered the callback and what value was passed into it using `callback_context.triggered`.

```
...
import json
import traceback
...
def custom_error_handler(err):
    callback_context = dash.ctx
    email_subject = f"Error {str(err)}"
    email_body = f"""
    Traceback info: {traceback.format_exc()}\n\n
    Input info: {json.dumps(callback_context.triggered)}
    """
    ...
```

See the callback context section on the **Advanced Callbacks page** for more details about the information that you can access using callback context.

Jupyter Notebooks

When `on_error` is set in a Dash app running in a Jupyter Notebook, exceptions are no longer shown inline in addition to not showing in Dash Dev Tools.

Limitations

- When used with Background or Long Callbacks, the type of exception that occurs and the original stacktrace will not be available inside the error handler. The type will be `dash.exceptions.LongCallbackError`. `err` will provide a formatted stacktrace.

Additional Resources

- **Advanced Callbacks**
- **Dash Dev Tools**
- **Exceptions tutorial in the Python docs**

Dash Python ***> Callback Error Handlers***

Products

Dash

Consulting and Training

Pricing

Enterprise Pricing

About Us

Careers

Resources

Blog

Support

Community Support

Graphing Documentation

Join our mailing list

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

SUBSCRIBE

Copyright © 2025 Plotly. All rights reserved.

Terms of Service Privacy Policy