plotly

Star 23,447

Dash Python > ***Persisting User Preferences & Control Values***

Persistence

Released September 2019 with Dash 1.3

Sometimes you want to save things the user has done in your app beyond the lifetime of the individual affected components. Perhaps you let your users choose their language, or edit the headers of a table, and they want to see these same settings every time they load the app. Or perhaps you have a tabbed app, so a form disappears when you switch to a different tab, and you want the same settings when the user comes back to that tab, but you want to reset them if the page is reloaded. There are ways to do this with `dcc.Store` components, but the Dash `persistence` feature dramatically simplifies these cases.

Persistence is supported by `dash-table` and all the form-like components in `dash-core-components`. If you are a component author, it's easy to add it to your component - see the last section below.

Components that support persistence have three new props:

- **`persistence`** (boolean | string | number; optional): Any truthy value will enable persistence for this component. Persistence is keyed to the combination of component `id` and this `persistence` value, so if this value changes you will see different persisted settings. For example, perhaps you have dropdowns for country and city, and you know that your users want to see the same country pre-filled as the last time they used your app - you can just set `persistence=True` on the country dropdown. But for the *city* they would like to see the same one as the last time they chose that *country* - just set `persistence=country_name` (where `country_name` is the value of the chosen country) on the city dropdown and we'll save one preferred city for each country.
- **`persistence_type`** ('local', 'session', or 'memory'; default 'local'): Where persisted user changes will be stored:
    - memory: only kept in memory, reset on page refresh. This is useful for example if you have a tabbed app, that deletes the component when a different tab is active, and you want changes persisted as you switch tabs but not after reloading the app.
    - local: uses `window.localStorage`. This is the default, and keeps the data indefinitely within that browser on that computer.
    - session: uses `window.sessionStorage`. Like `'local'` the data is kept when you reload the page, but cleared when you close the browser or open the app in a new browser tab.
- **`persistence_props`** (list of strings; optional): These are the props whose values will persist. Typically this defaults to all user-editable props, which in many cases is just one (ie `'value'`). `dash-table` has many props users can edit, and all of them except `'data'` are included here by default. Sometimes only a *part* of a prop is saved; this happens with table headers, which are only part of the `columns` prop. The corresponding `persistence_props` value is `'columns.name'`.

In the following example, notice that a callback that depends on persisted values receives those persisted values, even if the layout initially provided something else. Also the city is persisted in `localStorage` so will be saved indefinitely, but the neighborhood - which remembers a different value for each city - resets if you open the page in a new tab.

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

CITIES = ['Boston', 'London', 'Montreal']
NEIGHBORHOODS = {
    'Boston': ['Back Bay', 'Fenway', 'Jamaica Plain'],
    'London': ['Canary Wharf', 'Hackney', 'Kensington'],
    'Montreal': ['Le Plateau', 'Mile End', 'Rosemont']
```

```
}

app = Dash()

app.layout = html.Div([
    'Choose a city:',
    dcc.Dropdown(CITIES, 'Montreal', id='persisted-city', persistence=True),
    html.Br(),

    'correlated persistence - choose a neighborhood:',
    html.Div(dcc.Dropdown(id='neighborhood'), id='neighborhood-container'),
    html.Br(),
    html.Div(id='persisted-choices')
])


@callback(
    Output('neighborhood-container', 'children'),
    Input('persisted-city', 'value')
)
def set_neighborhood(city):
    neighborhoods = NEIGHBORHOODS[city]
    return dcc.Dropdown(neighborhoods, neighborhoods[0], id='neighborhood',
        persistence_type='session',
        persistence=city
    )


@callback(
    Output('persisted-choices', 'children'),
    Input('persisted-city', 'value'), Input('neighborhood', 'value')
)
def set_out(city, neighborhood):
    return f'You chose: {neighborhood}, {city}'


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

Choose a city:

Montreal

correlated persistence - choose a neighborhood:

Le Plateau

You chose: Le Plateau, Montreal

Explicitly clearing saved data

Persistence continues (subject to `persistence_type`) as long as the component `id`, the `persistence` value, and the prop provided by the server when creating or updating the *entire* component has the same value as it had before the user changed it. But if you have a callback whose `Output` is the specific persisted prop itself, that takes precedence over any saved value. This lets you reset user edits using `PreventUpdate` or `dash.no_update` until you detect the reset condition.

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

INITIAL = '1+1=2'

app = Dash()

app.layout = html.Div([
    "Remember this important info:",
    html.Br(),
    dcc.Input(id='important-info', value=INITIAL, persistence=True),
    html.Button("Forget it!", id='clear-info')
])
```

```
@callback(
    Output('important-info', 'value'),
    Input('clear-info', 'n_clicks')
)
def clear_persistence(n):
    return INITIAL if n else no_update


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

Remember this important info:

1+1=2 FORGET IT!

For component developers

Supporting persistence in your own components is easy. Just add the three props: `persistence`, `persistence_type`, and `persistence_props`, and set the appropriate defaults for `persistence_type` (normally `'local'`) and `persistence_props` (normally all of them, unless there are some that the user *can* change but normally wouldn't want to save). `persistence` should have no default, so that each created component must specifically opt in to persistence. Check out the **PR in dash-core-components** where we added persistence to those components.

The only case where there's actual code involved is persisting a partial prop. An example of this is in `dash-table`, where the `columns` prop is an array of objects, with just one property of each item (`columns[i].name` for all `i`) editable by the user. Here is the **PR that added it**. The entry in `persistence_props` is `columns.name`, which we call a "nested prop ID". It must have the form `'<propName>.<piece>'` where `propName` is the prop that contains this info, and `piece` may or may not map to the exact substructure being stored but is meaningful to the user. In principle, one prop could have several nested pieces; that's allowed, though it hasn't been used (yet?) in any official Plotly components. If you make use of this feature, let us know in the **community forum**!

Partially-persisted props also need to define a *class property* (not a React prop) `persistenceTransforms`, as an object:

```
{
  [propName]: {
    [piece]: {
       extract: propValue => valueToStore,
       apply: (storedValue, propValue) => newPropValue
     }
  }
}
```

- `extract` turns a prop value into a reduced value to store.
- `apply` puts an extracted value back into the prop. Make sure this creates a new object rather than mutating `propValue`, and that if there are multiple `piece` entries for one `propName`, their `apply` functions commute - which should not be an issue if they extract and apply non-intersecting parts of the full prop.

You only need to define these for the props that need them. It's important that `extract` pulls out *only* the relevant pieces of the prop, because persistence is only maintained if the extracted value of the prop before applying persistence is the same as it was before the user's changes.

It is also possible to define and use `persistenceTransforms` on non-nested props, i.e. `propName`, in order to apply some transformation to your persisted prop. An example of this is used in DatePickerSingle and DatePickerRange to strip the time portion of persisted `date`, `start_date`, and `end_date` props. You can check out the **PR in dash-core-components**.