Documentation

search

Search

- *rocket_launch*

  Get started

  - Installation
    add
  - Fundamentals
    add
  - First steps
    add
- *code*

  Develop

  - Concepts
    add
  - API reference
    remove
    - PAGE ELEMENTS

    - Write and magic
      add
    - Text elements
      add
    - Data elements
      add
    - Chart elements
      add
    - Input widgets
      add
    - Media elements
      add
    - Layouts and containers
      add
    - Chat elements
      add
    - Status elements
      add
    - Third-party components*open_in_new*
    - APPLICATION LOGIC

    - Navigation and pages
      add
    - Execution flow
      add
    - Caching and state
      remove
      - st.cache_data
      - st.cache_resource
      - st.session_state

        - st.query_params
        - st.experimental_get_query_params*delete*
        - st.experimental_set_query_params*delete*
      - Connections and secrets
        add
      - Custom components
        add
      - Utilities
        add
      - Configuration
        add
      - TOOLS

      - App testing
        add
      - Command line
        add
    - Tutorials
      add
    - Quick reference
      add
- *web_asset*

  Deploy

    - Concepts
      add
    - Streamlit Community Cloud
      add
    - Snowflake
    - Other platforms
      add
- *school*

  Knowledge base

    - FAQ
    - Installing dependencies
    - Deployment issues

- Home/
- Develop/
- API reference/
- Caching and state/
- st.session_state

Session State

Session State is a way to share variables between reruns, for each user session. In addition to the ability to store and persist state, Streamlit also exposes the ability to manipulate state using Callbacks. Session state also persists across apps inside a multipage app.

Check out this Session State basics tutorial video by Streamlit Developer Advocate Dr. Marisa Smith to get started:

Initialize values in Session State

The Session State API follows a field-based API, which is very similar to Python dictionaries:

```
# Initialization if 'key' not in st.session_state: st.session_state['key'] = 'value' # Session State
also supports attribute based syntax if 'key' not in st.session_state: st.session_state.key = 'value'
```

Reads and updates

Read the value of an item in Session State and display it by passing to `st.write` :

```
# Read st.write(st.session_state.key) # Outputs: value
```

Update an item in Session State by assigning it a value:

```
st.session_state.key = 'value2' # Attribute API st.session_state['key'] = 'value2' # Dictionary like
API
```

Curious about what is in Session State? Use `st.write` or magic:

```
st.write(st.session_state) # With magic: st.session_state
```

Streamlit throws a handy exception if an uninitialized variable is accessed:

```
st.write(st.session_state['value']) # Throws an exception!
```
state-uninitialized-exception

Delete items

Delete items in Session State using the syntax to delete items in any Python dictionary:

```
# Delete a single key-value pair del st.session_state[key] # Delete all the items in Session state for
key in st.session_state.keys(): del st.session_state[key]
```

Session State can also be cleared by going to Settings → Clear Cache, followed by Rerunning the app.

state-clear-cache

Session State and Widget State association

Every widget with a key is automatically added to Session State:

```
st.text_input("Your name", key="name") # This exists now: st.session_state.name
```

Use Callbacks to update Session State

A callback is a python function which gets called when an input widget changes.

Order of execution: When updating Session state in response to **events**, a callback function gets executed first, and then the app is executed from top to bottom.

Callbacks can be used with widgets using the parameters `on_change` (or `on_click`), `args`, and `kwargs`:

Parameters

- **on_change** or **on_click** - The function name to be used as a callback
- **args** (*tuple*) - List of arguments to be passed to the callback function
- **kwargs** (*dict*) - Named arguments to be passed to the callback function

Widgets which support the `on_change` event:

- `st.checkbox`
- `st.color_picker`
- `st.date_input`
- `st.data_editor`
- `st.file_uploader`
- `st.multiselect`
- `st.number_input`
- `st.radio`
- `st.select_slider`
- `st.selectbox`
- `st.slider`
- `st.text_area`
- `st.text_input`
- `st.time_input`
- `st.toggle`

Widgets which support the `on_click` event:

- `st.button`
- `st.download_button`
- `st.form_submit_button`

To add a callback, define a callback function **above** the widget declaration and pass it to the widget via the `on_change` (or `on_click`) parameter.

Forms and Callbacks

Widgets inside a form can have their values be accessed and set via the Session State API. `st.form_submit_button` can have a callback associated with it. The callback gets executed upon clicking on the submit button. For example:

```
def form_callback(): st.write(st.session_state.my_slider) st.write(st.session_state.my_checkbox) with
st.form(key='my_form'): slider_input = st.slider('My slider', 0, 10, 5, key='my_slider')
checkbox_input = st.checkbox('Yes or No', key='my_checkbox') submit_button =
st.form_submit_button(label='Submit', on_click=form_callback)
```

Serializable Session State

Serialization refers to the process of converting an object or data structure into a format that can be persisted and shared, and allowing you to recover the data's original structure. Python's built-in pickle module serializes Python objects to a byte stream ("pickling") and deserializes the stream into an object ("unpickling").

By default, Streamlit's Session State allows you to persist any Python object for the duration of the session, irrespective of the object's pickle-serializability. This property lets you store Python primitives such as integers, floating-point numbers, complex numbers and booleans, dataframes, and even lambdas returned by functions. However, some execution environments may require serializing all data in Session State, so it may be useful to detect incompatibility during development, or when the execution environment will stop supporting it in the future.

To that end, Streamlit provides a `runner.enforceSerializableSessionState` configuration option that, when set to `true`, only allows pickle-serializable objects in Session State. To enable the option, either create a global or project config file with the following or use it as a command-line flag:

```
# .streamlit/config.toml [runner] enforceSerializableSessionState = true
```

By "*pickle-serializable*", we mean calling `pickle.dumps(obj)` should not raise a `PicklingError` exception. When the config option is enabled, adding unserializable data to session state should result in an exception. E.g.,

```
import streamlit as st def unserializable_data(): return lambda x: x #👇 results in an exception when
enforceSerializableSessionState is on st.session_state.unserializable = unserializable_data()
```

UnserializableSessionStateError

priority_high

Warning

When `runner.enforceSerializableSessionState` is set to `true`, Session State implicitly uses the `pickle` module, which is known to be insecure. Ensure all data saved and retrieved from Session State is trusted because it is possible to construct malicious pickle data that will execute arbitrary code during unpickling. Never load data that could have come from an untrusted source in an unsafe mode or that could have been tampered with. **Only load data you trust**.

Caveats and limitations

- Only the `st.form_submit_button` has a callback in forms. Other widgets inside a form are not allowed to have callbacks.

- `on_change` and `on_click` events are only supported on input type widgets.

- Modifying the value of a widget via the Session state API, after instantiating it, is not allowed and will raise a `StreamlitAPIException`. For example:

  ```
  slider = st.slider( label='My Slider', min_value=1, max_value=10, value=5, key='my_slider')
  st.session_state.my_slider = 7 # Throws an exception!
  ```
  state-modified-instantiated-exception

- Setting the widget state via the Session State API and using the `value` parameter in the widget declaration is not recommended, and will throw a warning on the first run. For example:

  ```
  st.session_state.my_slider = 7 slider = st.slider( label='Choose a Value', min_value=1,
  max_value=10, value=5, key='my_slider')
  ```
  state-value-api-exception

- Setting the state of button-like widgets: `st.button`, `st.download_button`, and `st.file_uploader` via the Session State API is not allowed. Such type of widgets are by default *False* and have ephemeral *True* states which are only valid for a single run. For example:

```
if 'my_button' not in st.session_state: st.session_state.my_button = True st.button('My button', key='my_button') # Throws an exception!
```

state-button-exception

Previous: st.experimental_singletonNext: st.query_params

forum

Still have questions?

Our forums are full of helpful information and Streamlit experts.