Documentation

search

Search

        - 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.cache_resource

star

Tip

This page only contains information on the `st.cache_resource` API. For a deeper dive into caching and how to use it, check out Caching.

st.cache_resource

Streamlit Version Version 1.41.0

Decorator to cache functions that return global resources (e.g. database connections, ML models).

Cached objects are shared across all users, sessions, and reruns. They must be thread-safe because they can be accessed from multiple threads concurrently. If thread safety is an issue, consider using `st.session_state` to store resources per session instead.

You can clear a function's cache with `func.clear()` or clear the entire cache with `st.cache_resource.clear()`.

A function's arguments must be hashable to cache it. If you have an unhashable argument (like a database connection) or an argument you want to exclude from caching, use an underscore prefix in the argument name. In this case, Streamlit will return a cached value when all other arguments match a previous function call. Alternatively, you can declare custom hashing functions with `hash_funcs`.

To cache data, use `st.cache_data` instead. Learn more about caching at https://docs.streamlit.io/develop/concepts/architecture/caching.

Function signature[source]

st.cache_resource(func, *, ttl, max_entries, show_spinner, validate, experimental_allow_widgets, hash_funcs=None)

Parameters	
func (callable)	The function that creates the cached resource. Streamlit hashes the function's source code.
ttl (float, timedelta, str, or None)	The maximum time to keep an entry in the cache. Can be one of: • `None` if cache entries should never expire (default). • A number specifying the time in seconds. • A string specifying the time in a format supported by [Pandas's Timedelta constructor](), e.g. `"1d"`, `"1.5 days"`, or `"1h23s"`. • A `timedelta` object from [Python's built-in datetime library](), e.g. `timedelta(days=1)`.
max_entries (int or None)	The maximum number of entries to keep in the cache, or None for an unbounded cache. When a new entry is added to a full cache, the oldest cached entry will be removed. Defaults to None.
show_spinner (bool or str)	Enable the spinner. Default is True to show a spinner when there is a "cache miss" and the cached resource is being created. If string, value of show_spinner param will be used for spinner text.
validate (callable or None)	An optional validation function for cached data. `validate` is called each time the cached value is accessed. It receives the cached value as its only parameter and it must return a boolean. If `validate` returns False, the current cached value is discarded, and the decorated function is called to compute a new value. This is useful e.g. to check the health of database connections.
experimental_allow_widgets (bool)	*delete* The cached widget replay functionality was removed in 1.38. Please remove the `experimental_allow_widgets` parameter from your caching decorators. This parameter will be removed in a future version. Allow widgets to be used in the cached function. Defaults to False.

Function signature[source]

st.cache_resource(func, *, ttl, max_entries, show_spinner, validate, experimental_allow_widgets, hash_funcs=None)

hash_funcs (dict or None)	Mapping of types or fully qualified names to hash functions. This is used to override the behavior of the hasher inside Streamlit's caching mechanism: when the hasher encounters an object, it will first check to see if its type matches a key in this dict and, if so, will use the provided function to generate a hash for it. See below for an example of how this can be used.

Example

```
import streamlit as st

@st.cache_resource
def get_database_session(url):
    # Create a database session object that points to the URL.
    return session

s1 = get_database_session(SESSION_URL_1)
# Actually executes the function, since this is the first time it was
# encountered.

s2 = get_database_session(SESSION_URL_1)
# Does not execute the function. Instead, returns its previously computed
# value. This means that now the connection object in s1 is the same as in s2.

s3 = get_database_session(SESSION_URL_2)
# This is a different URL, so the function executes.
```

By default, all parameters to a cache_resource function must be hashable. Any parameter whose name begins with _ will not be hashed. You can use this as an "escape hatch" for parameters that are not hashable:

```
import streamlit as st

@st.cache_resource
def get_database_session(_sessionmaker, url):
    # Create a database connection object that points to the URL.
    return connection

s1 = get_database_session(create_sessionmaker(), DATA_URL_1)
# Actually executes the function, since this is the first time it was
# encountered.

s2 = get_database_session(create_sessionmaker(), DATA_URL_1)
# Does not execute the function. Instead, returns its previously computed
# value - even though the _sessionmaker parameter was different
# in both calls.
```

A cache_resource function's cache can be procedurally cleared:

```
import streamlit as st

@st.cache_resource
def get_database_session(_sessionmaker, url):
    # Create a database connection object that points to the URL.
    return connection

fetch_and_clean_data.clear(_sessionmaker, "https://streamlit.io/")
# Clear the cached entry for the arguments provided.

get_database_session.clear()
# Clear all cached entries for this function.
```

To override the default hashing behavior, pass a custom hash function. You can do that by mapping a type (e.g. `Person`) to a hash function (`str`) like this:

```
import streamlit as st
from pydantic import BaseModel

class Person(BaseModel):
    name: str

@st.cache_resource(hash_funcs={Person: str})
def get_person_name(person: Person):
    return person.name
```

Alternatively, you can map the type's fully-qualified name (e.g. `"__main__.Person"`) to the hash function instead:

```
import streamlit as st
from pydantic import BaseModel

class Person(BaseModel):
    name: str

@st.cache_resource(hash_funcs={"__main__.Person": str})
def get_person_name(person: Person):
    return person.name
```

st.cache_resource.clear

Streamlit Version Version 1.41.0

Clear all cache_resource caches.

Function signature[source]

st.cache_resource.clear()

Example

In the example below, pressing the "Clear All" button will clear *all* cache_resource caches. i.e. Clears cached global resources from all functions decorated with `@st.cache_resource`.

```
import streamlit as st from transformers import BertModel @st.cache_resource def
get_database_session(url): # Create a database session object that points to the URL. return session
@st.cache_resource def get_model(model_type): # Create a model of the specified type. return
BertModel.from_pretrained(model_type) if st.button("Clear All"): # Clears all st.cache_resource
caches: st.cache_resource.clear()
```

CachedFunc.clear

Streamlit Version Version 1.41.0

Clear the cached function's associated cache.

If no arguments are passed, Streamlit will clear all values cached for the function. If arguments are passed, Streamlit will clear the cached value for these arguments only.

Function signature[source]

CachedFunc.clear(*args, **kwargs)

Parameters

*args (Any)	Arguments of the cached functions.
**kwargs (Any)	Keyword arguments of the cached function.

Example

```
import streamlit as st
import time

@st.cache_data
def foo(bar):
    time.sleep(2)
    st.write(f"Executed foo({bar}).")
    return bar

if st.button("Clear all cached values for `foo`", on_click=foo.clear):
    foo.clear()

if st.button("Clear the cached value of `foo(1)`"):
    foo.clear(1)

foo(1)
foo(2)
```

Using Streamlit commands in cached functions

Static elements

Since version 1.16.0, cached functions can contain Streamlit commands! For example, you can do this:

```
from transformers import pipeline @st.cache_resource def load_model(): model = pipeline("sentiment-analysis") st.success("Loaded NLP model from Hugging Face!") # 👈 Show a success message return model
```

As we know, Streamlit only runs this function if it hasn't been cached before. On this first run, the `st.success` message will appear in the app. But what happens on subsequent runs? It still shows up! Streamlit realizes that there is an `st.` command inside the cached function, saves it during the first run, and replays it on subsequent runs. Replaying static elements works for both caching decorators.

You can also use this functionality to cache entire parts of your UI:

```
@st.cache_resource def load_model(): st.header("Data analysis") model = torchvision.models.resnet50(weights=ResNet50_Weights.DEFAULT) st.success("Loaded model!") st.write("Turning on evaluation mode...") model.eval() st.write("Here's the model:") return model
```

Input widgets

You can also use [interactive input widgets](#) like `st.slider` or `st.text_input` in cached functions. Widget replay is an experimental feature at the moment. To enable it, you need to set the `experimental_allow_widgets` parameter:

```
@st.cache_resource(experimental_allow_widgets=True) # 👈 Set the parameter def load_model():
pretrained = st.checkbox("Use pre-trained model:") # 👈 Add a checkbox model =
torchvision.models.resnet50(weights=ResNet50_Weights.DEFAULT, pretrained=pretrained) return model
```

Streamlit treats the checkbox like an additional input parameter to the cached function. If you uncheck it, Streamlit will see if it has already cached the function for this checkbox state. If yes, it will return the cached value. If not, it will rerun the function using the new slider value.

Using widgets in cached functions is extremely powerful because it lets you cache entire parts of your app. But it can be dangerous! Since Streamlit treats the widget value as an additional input parameter, it can easily lead to excessive memory usage. Imagine your cached function has five sliders and returns a 100 MB DataFrame. Then we'll add 100 MB to the cache for *every permutation* of these five slider values – even if the sliders do not influence the returned data! These additions can make your cache explode very quickly. Please be aware of this limitation if you use widgets in cached functions. We recommend using this feature only for isolated parts of your UI where the widgets directly influence the cached return value.

priority_high

Warning

Support for widgets in cached functions is currently experimental. We may change or remove it anytime without warning. Please use it with care!

push_pin

Note

Two widgets are currently not supported in cached functions: `st.file_uploader` and `st.camera_input`. We may support them in the future. Feel free to [open a GitHub issue](#) if you need them!

←[Previous: st.cache_data](#)[Next: st.experimental_memo](#)→

forum

Still have questions?

Our [forums](#) are full of helpful information and Streamlit experts.
