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

star

Tip

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

st.cache_data

Streamlit Version Version 1.41.0

Decorator to cache functions that return data (e.g. dataframe transforms, database queries, ML inference).

Cached objects are stored in "pickled" form, which means that the return value of a cached function must be pickleable. Each caller of the cached function gets its own copy of the cached data.

You can clear a function's cache with `func.clear()` or clear the entire cache with `st.cache_data.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 global resources, use `st.cache_resource` instead. Learn more about caching at https://docs.streamlit.io/develop/concepts/architecture/caching.

Function signature[source]

st.cache_data(func=None, *, ttl, max_entries, show_spinner, persist, experimental_allow_widgets, hash_funcs=None)

Parameters

func (callable)	The function to cache. 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)`. Note that `ttl` will be ignored if `persist="disk"` or `persist=True`.
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 data is being created. If string, value of show_spinner param will be used for spinner text.
persist ("disk", bool, or None)	Optional location to persist cached data to. Passing "disk" (or True) will persist the cached data to the local disk. None (or False) will disable persistence. The default is None.
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.
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

Function signature[source]

st.cache_data(func=None, *, ttl, max_entries, show_spinner, persist, experimental_allow_widgets, hash_funcs=None)

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_data
def fetch_and_clean_data(url):
    # Fetch data from URL here, and then clean it up.
    return data

d1 = fetch_and_clean_data(DATA_URL_1)
# Actually executes the function, since this is the first time it was
# encountered.

d2 = fetch_and_clean_data(DATA_URL_1)
# Does not execute the function. Instead, returns its previously computed
# value. This means that now the data in d1 is the same as in d2.

d3 = fetch_and_clean_data(DATA_URL_2)
# This is a different URL, so the function executes.
```

To set the `persist` parameter, use this command as follows:

```
import streamlit as st

@st.cache_data(persist="disk")
def fetch_and_clean_data(url):
    # Fetch data from URL here, and then clean it up.
    return data
```

By default, all parameters to a cached 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_data
def fetch_and_clean_data(_db_connection, num_rows):
    # Fetch data from _db_connection here, and then clean it up.
    return data

connection = make_database_connection()
d1 = fetch_and_clean_data(connection, num_rows=10)
# Actually executes the function, since this is the first time it was
# encountered.

another_connection = make_database_connection()
d2 = fetch_and_clean_data(another_connection, num_rows=10)
# Does not execute the function. Instead, returns its previously computed
# value - even though the _database_connection parameter was different
# in both calls.
```

A cached function's cache can be procedurally cleared:

```
import streamlit as st

@st.cache_data
def fetch_and_clean_data(_db_connection, num_rows):
    # Fetch data from _db_connection here, and then clean it up.
    return data
```

```
fetch_and_clean_data.clear(_db_connection, 50)
# Clear the cached entry for the arguments provided.

fetch_and_clean_data.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. `datetime.datetime`) to a hash function (`lambda dt: dt.isoformat()`) like this:

```
import streamlit as st
import datetime

@st.cache_data(hash_funcs={datetime.datetime: lambda dt: dt.isoformat()})
def convert_to_utc(dt: datetime.datetime):
    return dt.astimezone(datetime.timezone.utc)
```

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

```
import streamlit as st
import datetime

@st.cache_data(hash_funcs={"datetime.datetime": lambda dt: dt.isoformat()})
def convert_to_utc(dt: datetime.datetime):
    return dt.astimezone(datetime.timezone.utc)
```

priority_high

Warning

`st.cache_data` implicitly uses the `pickle` module, which is known to be insecure. Anything your cached function returns is pickled and stored, then unpickled on retrieval. Ensure your cached functions return trusted values 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**.

st.cache_data.clear

Streamlit Version Version 1.41.0

Clear all in-memory and on-disk data caches.

Function signature[source]

st.cache_data.clear()

Example

In the example below, pressing the "Clear All" button will clear memoized values from all functions decorated with `@st.cache_data`.

```
import streamlit as st @st.cache_data def square(x): return x**2 @st.cache_data def cube(x): return x**3 if st.button("Clear All"): # Clear values from *all* all in-memory and on-disk data caches: # i.e. clear values from both square and cube st.cache_data.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:

```
@st.cache_data def get_api_data(): data = api.get(...) st.success("Fetched data from API!") # 👈 Show a success message return data
```

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_data def show_data(): st.header("Data analysis") data = api.get(...) st.success("Fetched data from API!") st.write("Here is a plot of the data:") st.line_chart(data) st.write("And here is the raw data:") st.dataframe(data)
```

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_data(experimental_allow_widgets=True) # 👈 Set the parameter def get_data(): num_rows = st.slider("Number of rows to get") # 👈 Add a slider data = api.get(..., num_rows) return data
```

Streamlit treats the slider like an additional input parameter to the cached function. If you change the slider position, Streamlit will see if it has already cached the function for this slider value. 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: Caching and stateNext: st.cache_resource→
forum

Still have questions?

Our forums are full of helpful information and Streamlit experts.
