Frame objects

- Frame locals proxies
- Legacy local variable APIs
- Internal frames

> The C structure of the objects used to describe frame objects.
>
> There are no public members in this structure.
>
> 3.11 The members of this structure were removed from the public C API. Refer to the `What's New entry <pyframeobject-3.11-hiding>` for details.

The `PyEval_GetFrame` and `PyThreadState_GetFrame` functions can be used to get a frame object.

See also `Reflection <reflection>`.

> The type of frame objects. It is the same object as `types.FrameType` in the Python layer.
>
> 3.11
>
> Previously, this type was only available after including `<frameobject.h>`.

> Create a new frame object. This function returns a `strong reference` to the new frame object on success, and returns `NULL` with an exception set on failure.

> Return non-zero if *obj* is a frame object.

3.11

Previously, this function was only available after including `<frameobject.h>`.

Get the *frame* next outer frame.

Return a `strong reference`, or `NULL` if *frame* has no outer frame. This raises no exceptions.

3.9

Get the *frame*'s `~frame.f_builtins` attribute.

Return a `strong reference`. The result cannot be `NULL`.

3.11

Get the *frame* code.

Return a `strong reference`.

The result (frame code) cannot be `NULL`.

3.9

Get the generator, coroutine, or async generator that owns this frame, or `NULL` if this frame is not owned by a generator. Does not raise an exception, even if the return value is `NULL`.

Return a `strong reference`, or `NULL`.

3.11

Get the *frame*'s `~frame.f_globals` attribute.

Return a `strong reference`. The result cannot be `NULL`.

3.11

Get the *frame*'s `~frame.f_lasti` attribute.

Returns -1 if `frame.f_lasti` is `None`.

3.11

Get the variable *name* of *frame*.

- Return a `strong reference` to the variable value on success.
- Raise `NameError` and return `NULL` if the variable does not exist.
- Raise an exception and return `NULL` on error.

name type must be a `str`.

3.12

Similar to `PyFrame_GetVar`, but the variable name is a C string encoded in UTF-8.

3.12

Get the *frame*'s `~frame.f_locals` attribute. If the frame refers to an `optimized scope`, this returns a write-through proxy object that allows modifying the locals. In all other cases (classes, modules, `exec`, `eval`) it returns the mapping representing the frame locals directly (as described for `locals`).

Return a `strong reference`.

3.11

3.13 As part of `667`, return an instance of `PyFrameLocalsProxy_Type`.

> Return the line number that *frame* is currently executing.

Frame locals proxies

3.13

The `~frame.f_locals` attribute on a `frame object <frame-objects>` is an instance of a "frame-locals proxy". The proxy object exposes a write-through view of the underlying locals dictionary for the frame. This ensures that the variables exposed by `f_locals` are always up to date with the live local variables in the frame itself.

See `667` for more information.

> The type of frame `locals` proxy objects.

> Return non-zero if *obj* is a frame `locals` proxy.

Legacy local variable APIs

These APIs are `soft deprecated`. As of Python 3.13, they do nothing. They exist solely for backwards compatibility.

> Prior to Python 3.13, this function would copy the `~frame.f_locals` attribute of *f* to the internal "fast" array of local variables, allowing changes in frame objects to be visible to the interpreter. If *clear* was true, this function would process variables that were unset in the locals dictionary.
>
> 3.13 This function now does nothing.

> Prior to Python 3.13, this function would copy the internal "fast" array of local variables (which is used by the interpreter) to the `~frame.f_locals` attribute of *f*,

> allowing changes in local variables to be visible to frame objects.
>
> 3.13 This function now does nothing.

> Prior to Python 3.13, this function was similar to `PyFrame_FastToLocals`, but would return `0` on success, and `-1` with an exception set on failure.
>
> 3.13 This function now does nothing.

`667`

Internal frames

Unless using `523`, you will not need this.

> The interpreter's internal frame representation.
>
> 3.11

> Return a `strong reference` to the code object for the frame.
>
> 3.12

> Return the byte offset into the last executed instruction.
>
> 3.12

> Return the currently executing line number, or -1 if there is no line number.
>
> 3.12

□□□ #2
□ □□□ □□□ 2026-10-31 16:51:16 UTC
□ □□□ □□□ 2026-10-31 16:51:17 UTC