Curses C API

- Internal data

`curses` exposes a small C interface for extension modules. Consumers must include the header file `py_curses.h` (which is not included by default by `Python.h`) and `import_curses` must be invoked, usually as part of the module initialisation function, to populate `PyCurses_API`.

> [!WARNING] Neither the C API nor the pure Python `curses` module are compatible with subinterpreters.

> Import the curses C API. The macro does not need a semi-colon to be called.
>
> On success, populate the `PyCurses_API` pointer.
>
> On failure, set `PyCurses_API` to NULL and set an exception. The caller must check if an error occurred via `PyErr_Occurred`:
>
> ```
> import_curses();  // semi-colon is optional but recommended
> if (PyErr_Occurred()) { /* cleanup */ }
> ```

> Dynamically allocated object containing the curses C API. This variable is only available once `import_curses` succeeds.
>
> `PyCurses_API[0]` corresponds to `PyCursesWindow_Type`.
>
> `PyCurses_API[1]`, `PyCurses_API[2]`, and `PyCurses_API[3]` are pointers to predicate functions of type `int (*)(void)`.
>
> When called, these predicates return whether `curses.setupterm`, `curses.initscr`, and `curses.start_color` have been called respectively.

> See also the convenience macros `PyCursesSetupTermCalled`, `PyCursesInitialised`, and `PyCursesInitialisedColor`.
>
> > [!NOTE] The number of entries in this structure is subject to changes. Consider using `PyCurses_API_pointers` to check if new fields are available or not.

> The number of accessible fields (`4`) in `PyCurses_API`. This number is incremented whenever new fields are added.

> The `heap type <heap-types>` corresponding to `curses.window`.

> Return true if *op* is a `curses.window` instance, false otherwise.

The following macros are convenience macros expanding into C statements. In particular, they can only be used as `macro;` or `macro`, but not `macro()` or `macro();`.

> Macro checking if `curses.setupterm` has been called.
>
> The macro expansion is roughly equivalent to:
>
> ```
> {
>     typedef int (*predicate_t)(void);
>     predicate_t was_setupterm_called = (predicate_t)PyCurses_API[1];
>     if (!was_setupterm_called()) {
>         return NULL;
>     }
> }
> ```

> Macro checking if `curses.initscr` has been called.

The macro expansion is roughly equivalent to:

```
{
    typedef int (*predicate_t)(void);
    predicate_t was_initscr_called = (predicate_t)PyCurses_API[2];
    if (!was_initscr_called()) {
        return NULL;
    }
}
```

> Macro checking if `curses.start_color` has been called.
>
> The macro expansion is roughly equivalent to:
>
> ```
> {
>     typedef int (*predicate_t)(void);
>     predicate_t was_start_color_called = (predicate_t)PyCurses_API[3];
>     if (!was_start_color_called()) {
>         return NULL;
>     }
> }
> ```

Internal data

The following objects are exposed by the C API but should be considered internal-only.

> Name of the curses capsule to pass to `PyCapsule_Import`.
>
> Internal usage only. Use `import_curses` instead.

□□□ #2
□ □□□ □□□ 2026-10-31 16:50:40 UTC
□ □□□ □□□ 2026-10-31 16:50:41 UTC