Codec registry and support functions

- [Codec lookup API]
- [Registry API for Unicode encoding error handlers]
- [Codec utility variables]

> Register a new codec search function.
>
> As a side effect, this tries to load the `!encodings` package, if not yet done, to make sure that it is always first in the list of search functions.

> Unregister a codec search function and clear the registry's cache. If the search function is not registered, do nothing. Return 0 on success. Raise an exception and return -1 on error.
>
> 3.10

> Return `1` or `0` depending on whether there is a registered codec for the given *encoding*. This function always succeeds.

> Generic codec based encoding API.
>
> *object* is passed through the encoder function found for the given *encoding* using the error handling method defined by *errors*. *errors* may be `NULL` to use the default method defined for the codec. Raises a `LookupError` if no encoder can be found.

> Generic codec based decoding API.
>
> *object* is passed through the decoder function found for the given *encoding* using the error handling method defined by *errors*. *errors* may be `NULL` to use the default method defined for the codec. Raises a `LookupError` if no decoder can be found.

Codec lookup API

In the following functions, the *encoding* string is looked up converted to all lower-case characters, which makes encodings looked up through this mechanism effectively case-insensitive. If no codec is found, a `LookupError` is set and `NULL` returned.

> Get an encoder function for the given *encoding*.

> Get a decoder function for the given *encoding*.

> Get an `~codecs.IncrementalEncoder` object for the given *encoding*.

> Get an `~codecs.IncrementalDecoder` object for the given *encoding*.

> Get a `~codecs.StreamReader` factory function for the given *encoding*.

> Get a `~codecs.StreamWriter` factory function for the given *encoding*.

Registry API for Unicode encoding error handlers

> Register the error handling callback function *error* under the given *name*. This callback function will be called by a codec when it encounters unencodable characters/undecodable bytes and *name* is specified as the error parameter in the call to the encode/decode function.
>
> The callback gets a single argument, an instance of `UnicodeEncodeError`, `UnicodeDecodeError` or `UnicodeTranslateError` that holds information about the problematic sequence of characters or bytes and their offset in the original string (see `unicodeexceptions` for functions to extract this information). The callback must either raise the given exception, or return a two-item tuple containing the replacement for the problematic sequence, and an integer giving the offset in the original string at which encoding/decoding should be resumed.
>
> Return `0` on success, `-1` on error.

> Lookup the error handling callback function registered under *name*. As a special case `NULL` can be passed, in which case the error handling callback for "strict" will be returned.

> Raise *exc* as an exception.

> Ignore the unicode error, skipping the faulty input.

> Replace the unicode encode error with `?` or `U+FFFD`.

> Replace the unicode encode error with XML character references.

> Replace the unicode encode error with backslash escapes (`\x`, `\u` and `\U`).

> Replace the unicode encode error with `\N{...}` escapes.
>
> 3.5

Codec utility variables

> A string constant containing the lowercase hexadecimal digits: `"0123456789abcdef"`.
>
> 3.3

□□□ #2
□ □□□ □□□ 2026-10-31 16:50:11 UTC
□ □□□ □□□ 2026-10-31 16:50:13 UTC