Bytes Objects

These functions raise `TypeError` when expecting a bytes parameter and are called with a non-bytes parameter.

`PyBytesObject`

This subtype of `PyObject` represents a Python bytes object.

PyTypeObject **`PyBytes_Type`**

This instance of `PyTypeObject` represents the Python bytes type; it is the same object as `bytes` in the Python layer.

int **`PyBytes_Check`**(PyObject **o*)

Return true if the object *o* is a bytes object or an instance of a subtype of the bytes type.

int **`PyBytes_CheckExact`**(PyObject **o*)

Return true if the object *o* is a bytes object, but not an instance of a subtype of the bytes type.

PyObject* **`PyBytes_FromString`**(const char **v*)

Return a new bytes object with a copy of the string *v* as value on success, and *NULL* on failure. The parameter *v* must not be *NULL*; it will not be checked.

PyObject* **`PyBytes_FromStringAndSize`**(const char **v*, Py_ssize_t *len*)

Return a new bytes object with a copy of the string *v* as value and length *len* on success, and *NULL* on failure. If *v* is *NULL*, the contents of the bytes object are uninitialized.

PyObject* **`PyBytes_FromFormat`**(const char **format*, ...)

Take a C `printf()`-style *format* string and a variable number of arguments, calculate the size of the resulting Python bytes object and return a bytes object with the values formatted into it. The variable arguments must be C types and must correspond exactly to the format characters in the *format* string. The following format characters are allowed:

Format Characters	Type	Comment
%%	*n/a*	The literal % character.
%c	int	A single byte, represented as a C int.
%d	int	Exactly equivalent to `printf("%d")`.

Format Characters	Type	Comment
`%u`	unsigned int	Exactly equivalent to `printf("%u")`.
`%ld`	long	Exactly equivalent to `printf("%ld")`.
`%lu`	unsigned long	Exactly equivalent to `printf("%lu")`.
`%zd`	Py_ssize_t	Exactly equivalent to `printf("%zd")`.
`%zu`	size_t	Exactly equivalent to `printf("%zu")`.
`%i`	int	Exactly equivalent to `printf("%i")`.
`%x`	int	Exactly equivalent to `printf("%x")`.
`%s`	char*	A null-terminated C character array.
`%p`	void*	The hex representation of a C pointer. Mostly equivalent to `printf("%p")` except that it is guaranteed to start with the literal `0x` regardless of what the platform's `printf` yields.

An unrecognized format character causes all the rest of the format string to be copied as-is to the result object, and any extra arguments discarded.

PyObject* **PyBytes_FromFormatV**(const char **format*, va_list *vargs*)

Identical to `PyBytes_FromFormat()` except that it takes exactly two arguments.

PyObject* **PyBytes_FromObject**(PyObject **o*)

Return the bytes representation of object *o* that implements the buffer protocol.

Py_ssize_t **PyBytes_Size**(PyObject **o*)

Return the length of the bytes in bytes object *o*.

Py_ssize_t **PyBytes_GET_SIZE**(PyObject **o*)

Macro form of `PyBytes_Size()` but without error checking.

char* **PyBytes_AsString**(PyObject **o*)

Return a pointer to the contents of *o*. The pointer refers to the internal buffer of *o*, which consists of `len(o) + 1` bytes. The last byte in the buffer is always null, regardless of whether there are any other null bytes. The data must not be modified in any way, unless the object was just created using `PyBytes_FromStringAndSize(NULL, size)`. It must not be deallocated. If *o* is not a bytes object at all, `PyBytes_AsString()` returns *NULL* and raises `TypeError`.

char* **PyBytes_AS_STRING**(PyObject **string*)

Macro form of `PyBytes_AsString()` but without error checking.

int **PyBytes_AsStringAndSize**(PyObject **obj*, char ***buffer*, Py_ssize_t **length*)

Return the null-terminated contents of the object *obj* through the output variables *buffer* and *length*.

If *length* is *NULL*, the bytes object may not contain embedded null bytes; if it does, the function returns `-1` and a `ValueError` is raised.

The buffer refers to an internal buffer of *obj*, which includes an additional null byte at the end (not counted in *length*). The data must not be modified in any way, unless the object was just created using `PyBytes_FromStringAndSize(NULL, size)`. It must not be deallocated. If *obj* is not a bytes object at all, `PyBytes_AsStringAndSize()` returns `-1` and raises `TypeError`.

Changed in version 3.5: Previously, `TypeError` was raised when embedded null bytes were encountered in the bytes object.

void **PyBytes_Concat**(PyObject ***bytes*, PyObject **newpart*)

Create a new bytes object in **bytes* containing the contents of *newpart* appended to *bytes*; the caller will own the new reference. The reference to the old value of *bytes* will be stolen. If the new object cannot be created, the old reference to *bytes* will still be discarded and the value of **bytes* will be set to *NULL*; the appropriate exception will be set.

void **PyBytes_ConcatAndDel**(PyObject ***bytes*, PyObject **newpart*)

Create a new bytes object in **bytes* containing the contents of *newpart* appended to *bytes*. This version decrements the reference count of *newpart*.

int **_PyBytes_Resize**(PyObject ***bytes*, Py_ssize_t *newsize*)

A way to resize a bytes object even though it is “immutable”. Only use this to build up a brand new bytes object; don’t use this if the bytes may already be known in other parts of the code. It is an error to call this function if the refcount on the input bytes object is not one. Pass the address of an existing bytes ob-

ject as an lvalue (it may be written into), and the new size desired. On success, **bytes* holds the resized bytes object and `0` is returned; the address in **bytes* may differ from its input value. If the reallocation fails, the original bytes object at **bytes* is deallocated, **bytes* is set to *NULL*, `MemoryError` is set, and `-1` is returned.