Dictionary Objects

`PyDictObject`

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

PyTypeObject **`PyDict_Type`**

This instance of `PyTypeObject` represents the Python dictionary type. This is the same object as `dict` in the Python layer.

int **`PyDict_Check`**(PyObject *p)

Return true if *p* is a dict object or an instance of a subtype of the dict type.

int **`PyDict_CheckExact`**(PyObject *p)

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

PyObject* **`PyDict_New`**()

Return value: New reference.

Return a new empty dictionary, or *NULL* on failure.

PyObject* **`PyDictProxy_New`**(PyObject *mapping)

Return value: New reference.

Return a `types.MappingProxyType` object for a mapping which enforces read-only behavior. This is normally used to create a view to prevent modification of the dictionary for non-dynamic class types.

void **`PyDict_Clear`**(PyObject *p)

Empty an existing dictionary of all key-value pairs.

int **`PyDict_Contains`**(PyObject *p, PyObject *key)

Determine if dictionary *p* contains *key*. If an item in *p* is matches *key*, return `1`, otherwise return `0`. On error, return `-1`. This is equivalent to the Python expression `key in p`.

PyObject* **`PyDict_Copy`**(PyObject *p)

Return value: New reference.

Return a new dictionary that contains the same key-value pairs as *p*.

int **`PyDict_SetItem`**(PyObject *p, PyObject *key, PyObject *val)

Insert *value* into the dictionary *p* with a key of *key*. *key* must be hashable; if it isn't, `TypeError` will be raised. Return `0` on success or `-1` on failure.

int **`PyDict_SetItemString`**(PyObject *p, const char *key, PyObject *val)

Insert *value* into the dictionary *p* using *key* as a key. *key* should be a `char*`. The key object is created using `PyUnicode_FromString(key)`. Return `0` on success or `-1` on failure.

int **PyDict_DelItem**(PyObject *p, PyObject *key)

Remove the entry in dictionary *p* with key *key*. *key* must be hashable; if it isn't, `TypeError` is raised. Return `0` on success or `-1` on failure.

int **PyDict_DelItemString**(PyObject *p, const char *key)

Remove the entry in dictionary *p* which has a key specified by the string *key*. Return `0` on success or `-1` on failure.

PyObject* **PyDict_GetItem**(PyObject *p, PyObject *key)

Return value: Borrowed reference.

Return the object from dictionary *p* which has a key *key*. Return *NULL* if the key *key* is not present, but *without* setting an exception.

PyObject* **PyDict_GetItemWithError**(PyObject *p, PyObject *key)

Variant of `PyDict_GetItem()` that does not suppress exceptions. Return *NULL* **with** an exception set if an exception occurred. Return *NULL* **without** an exception set if the key wasn't present.

PyObject* **PyDict_GetItemString**(PyObject *p, const char *key)

Return value: Borrowed reference.

This is the same as `PyDict_GetItem()`, but *key* is specified as a `char*`, rather than a `PyObject*`.

PyObject* **PyDict_SetDefault**(PyObject *p, PyObject *key, PyObject *default)

Return value: Borrowed reference.

This is the same as the Python-level `dict.setdefault()`. If present, it returns the value corresponding to *key* from the dictionary *p*. If the key is not in the dict, it is inserted with value *defaultobj* and *defaultobj* is returned. This function evaluates the hash function of *key* only once, instead of evaluating it independently for the lookup and the insertion.

New in version 3.4.

PyObject* **PyDict_Items**(PyObject *p)

Return value: New reference.

Return a `PyListObject` containing all the items from the dictionary.

PyObject* **PyDict_Keys**(PyObject *p)

Return value: New reference.

Return a `PyListObject` containing all the keys from the dictionary.

PyObject* **PyDict_Values**(PyObject *p)

Return value: New reference.

Return a `PyListObject` containing all the values from the dictionary *p*.

Py_ssize_t **PyDict_Size**(PyObject *p)

Return the number of items in the dictionary. This is equivalent to `len(p)` on a dictionary.

int **PyDict_Next**(PyObject *p, Py_ssize_t *ppos, PyObject **pkey, PyObject **pvalue)

Iterate over all key-value pairs in the dictionary *p*. The `Py_ssize_t` referred to by *ppos* must be initialized to `0` prior to the first call to this function to start the iteration; the function returns true for each pair in the dictionary, and false once all pairs have been reported. The parameters *pkey* and *pvalue* should either point to `PyObject*` variables that will be filled in with each key and value, respectively, or may be *NULL*. Any references returned through them are borrowed. *ppos* should not be altered during iteration. Its value represents offsets within the internal dictionary structure, and since the structure is sparse, the offsets are not consecutive.

For example:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value)) {
    /* do something interesting with the values... */
    ...
}
```

The dictionary *p* should not be mutated during iteration. It is safe to modify the values of the keys as you iterate over the dictionary, but only so long as the set of keys does not change. For example:

```
PyObject *key, *value;
Py_ssize_t pos = 0;

while (PyDict_Next(self->dict, &pos, &key, &value)) {
    long i = PyLong_AsLong(value);
    if (i == -1 && PyErr_Occurred()) {
        return -1;
    }
    PyObject *o = PyLong_FromLong(i + 1);
    if (o == NULL)
        return -1;
    if (PyDict_SetItem(self->dict, key, o) < 0) {
        Py_DECREF(o);
```

```
        return -1;
    }
    Py_DECREF(o);
}
```

int **PyDict_Merge**(PyObject *a, PyObject *b, int *override*)

Iterate over mapping object *b* adding key-value pairs to dictionary *a*. *b* may be a dictionary, or any object supporting `PyMapping_Keys()` and `PyObject_GetItem()`. If *override* is true, existing pairs in *a* will be replaced if a matching key is found in *b*, otherwise pairs will only be added if there is not a matching key in *a*. Return `0` on success or `-1` if an exception was raised.

int **PyDict_Update**(PyObject *a, PyObject *b)

This is the same as `PyDict_Merge(a, b, 1)` in C, and is similar to `a.update(b)` in Python except that `PyDict_Update()` doesn't fall back to the iterating over a sequence of key value pairs if the second argument has no "keys" attribute. Return `0` on success or `-1` if an exception was raised.

int **PyDict_MergeFromSeq2**(PyObject *a, PyObject *seq2, int *override*)

Update or merge into dictionary *a*, from the key-value pairs in *seq2*. *seq2* must be an iterable object producing iterable objects of length 2, viewed as key-value pairs. In case of duplicate keys, the last wins if *override* is true, else the first wins. Return `0` on success or `-1` if an exception was raised. Equivalent Python (except for the return value):

```
def PyDict_MergeFromSeq2(a, seq2, override):
    for key, value in seq2:
        if override or key not in a:
            a[key] = value
```

int **PyDict_ClearFreeList**()

Clear the free list. Return the total number of freed items.

New in version 3.3.