`!winreg` --- Windows registry access

- Functions
- Constants
  - HKEY* Constants
  - Access Rights
    - 64-bit Specific
  - Value Types
- Registry Handle Objects

winreg

Source code: `PC/winreg.c`

These functions expose the Windows registry API to Python. Instead of using an integer as the registry handle, a `handle object <handle-object>` is used to ensure that the handles are closed correctly, even if the programmer neglects to explicitly close them.

Windows.

3.3 Several functions in this module used to raise a `WindowsError`, which is now an alias of `OSError`.

Functions

This module offers the following functions:

CloseKey(hkey)

Closes a previously opened registry key. The *hkey* argument specifies a previously opened key.

> [!NOTE] If *hkey* is not closed using this method (or via `hkey.Close() <PyHKEY.Close>`), it is closed when the *hkey* object is destroyed by Python.

ConnectRegistry(computer_name, key)

Establishes a connection to a predefined registry handle on another computer, and returns a `handle object <handle-object>`.

computer_name is the name of the remote computer, of the form `r"\\computername"`. If `None`, the local computer is used.

key is the predefined handle to connect to.

The return value is the handle of the opened key. If the function fails, an `OSError` exception is raised.

winreg.ConnectRegistry computer_name,key winreg.ConnectRegistry

3.3 See `above <exception-changed>`.

CreateKey(key, sub_key)

Creates or opens the specified key, returning a `handle object <handle-object>`.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that names the key this method opens or creates.

If *key* is one of the predefined keys, *sub_key* may be `None`. In that case, the handle returned is the same key handle passed in to the function.

If the key already exists, this function opens the existing key.

The return value is the handle of the opened key. If the function fails, an `OSError` exception is raised.

winreg.CreateKey key,sub_key,access winreg.CreateKey

winreg.OpenKey/result key winreg.CreateKey

3.3 See `above <exception-changed>`.

CreateKeyEx(key, sub_key, reserved=0, access=KEY_WRITE)

Creates or opens the specified key, returning a `handle object <handle-object>`.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that names the key this method opens or creates.

reserved is a reserved integer, and must be zero. The default is zero.

access is an integer that specifies an access mask that describes the desired security access for the key. Default is `KEY_WRITE`. See `Access Rights <access-rights>` for other allowed values.

If *key* is one of the predefined keys, *sub_key* may be `None`. In that case, the handle returned is the same key handle passed in to the function.

If the key already exists, this function opens the existing key.

The return value is the handle of the opened key. If the function fails, an `OSError` exception is raised.

winreg.CreateKey key,sub_key,access winreg.CreateKeyEx

winreg.OpenKey/result key winreg.CreateKeyEx

3.2

3.3 See `above <exception-changed>`.

DeleteKey(key, sub_key)

Deletes the specified key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that must be a subkey of the key identified by the *key* parameter. This value must not be `None`, and the key may not have subkeys.

This method can not delete keys with subkeys.

If the method succeeds, the entire key, including all of its values, is removed. If the method fails, an `OSError` exception is raised.

winreg.DeleteKey key,sub_key,access winreg.DeleteKey

3.3 See `above <exception-changed>`.

DeleteKeyEx(key, sub_key, access=KEY_WOW64_64KEY, reserved=0)

Deletes the specified key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that must be a subkey of the key identified by the *key* parameter. This value must not be `None`, and the key may not have subkeys.

reserved is a reserved integer, and must be zero. The default is zero.

access is an integer that specifies an access mask that describes the desired security access for the key. Default is `KEY_WOW64_64KEY`. On 32-bit Windows, the WOW64 constants are ignored. See `Access Rights <access-rights>` for other allowed values.

This method can not delete keys with subkeys.

If the method succeeds, the entire key, including all of its values, is removed. If the method fails, an `OSError` exception is raised.

On unsupported Windows versions, `NotImplementedError` is raised.

winreg.DeleteKey key,sub_key,access winreg.DeleteKeyEx

3.2

3.3 See `above <exception-changed>`.

DeleteTree(key, sub_key=None)

Deletes the specified key and all its subkeys and values recursively.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that names the subkey to delete. If `None`, deletes all subkeys and values of the specified key.

This function deletes a key and all its descendants. If *sub_key* is `None`, all subkeys and values of the specified key are deleted.

winreg.DeleteTree key,sub_key winreg.DeleteTree

3.15

DeleteValue(key, value)

Removes a named value from a registry key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

value is a string that identifies the value to remove.

winreg.DeleteValue key,value winreg.DeleteValue

EnumKey(key, index)

Enumerates subkeys of an open registry key, returning a string.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

index is an integer that identifies the index of the key to retrieve.

The function retrieves the name of one subkey each time it is called. It is typically called repeatedly until an `OSError` exception is raised, indicating, no more values are available.

winreg.EnumKey key,index winreg.EnumKey

3.3 See `above <exception-changed>`.

EnumValue(key, index)

Enumerates values of an open registry key, returning a tuple.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

index is an integer that identifies the index of the value to retrieve.

The function retrieves the name of one subkey each time it is called. It is typically called repeatedly, until an `OSError` exception is raised, indicating no more values.

The result is a tuple of 3 items:

Index	Meaning
`0`	A string that identifies the value name
`1`	An object that holds the value data, and whose type depends on the underlying registry type
`2`	An integer that identifies the type of the value data (see table in docs for `SetValueEx`)

winreg.EnumValue key,index winreg.EnumValue

3.3 See `above <exception-changed>`.

single: % (percent); environment variables expansion (Windows)

ExpandEnvironmentStrings(str)

Expands environment variable placeholders `%NAME%` in strings like `REG_EXPAND_SZ`:

```
>>> ExpandEnvironmentStrings('%windir%')
'C:\\Windows'
```

winreg.ExpandEnvironmentStrings str winreg.ExpandEnvironmentStrings

FlushKey(key)

Writes all the attributes of a key to the registry.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

It is not necessary to call `FlushKey` to change a key. Registry changes are flushed to disk by the registry using its lazy flusher. Registry changes are also flushed to disk at system shutdown. Unlike `CloseKey`, the `FlushKey` method returns only when all the data has been written to the registry. An application should only call `FlushKey` if it requires absolute certainty that registry changes are on disk.

> [!NOTE] If you don't know whether a `FlushKey` call is required, it probably isn't.

LoadKey(key, sub_key, file_name)

Creates a subkey under the specified key and stores registration information from a specified file into that subkey.

key is a handle returned by `ConnectRegistry` or one of the constants `HKEY_USERS` or `HKEY_LOCAL_MACHINE`.

sub_key is a string that identifies the subkey to load.

file_name is the name of the file to load registry data from. This file must have been created with the `SaveKey` function. Under the file allocation table (FAT) file system, the filename may not have an extension.

A call to `LoadKey` fails if the calling process does not have the `!SE_RESTORE_PRIVILEGE` privilege. Note that privileges are different from permissions -- see the RegLoadKey documentation for more details.

If *key* is a handle returned by `ConnectRegistry`, then the path specified in *file_name* is relative to the remote computer.

winreg.LoadKey key,sub_key,file_name winreg.LoadKey

OpenKey(key, sub_key, reserved=0, access=KEY_READ) OpenKeyEx(key, sub_key, reserved=0, access=KEY_READ)

Opens the specified key, returning a `handle object <handle-object>`.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that identifies the sub_key to open.

reserved is a reserved integer, and must be zero. The default is zero.

access is an integer that specifies an access mask that describes the desired security access for the key. Default is `KEY_READ`. See `Access Rights <access-rights>` for other allowed values.

The result is a new handle to the specified key.

If the function fails, `OSError` is raised.

winreg.OpenKey key,sub_key,access winreg.OpenKey

winreg.OpenKey/result key winreg.OpenKey

3.2 Allow the use of named arguments.

3.3 See `above <exception-changed>`.

QueryInfoKey(key)

Returns information about a key, as a tuple.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

The result is a tuple of 3 items:

Index	Meaning
`0`	An integer giving the number of sub keys this key has.
`1`	An integer giving the number of values this key has.
`2`	An integer giving when the key was last modified (if available) as 100's of nanoseconds since Jan 1, 1601.

winreg.QueryInfoKey key winreg.QueryInfoKey

QueryValue(key, sub_key)

Retrieves the unnamed value for a key, as a string.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that holds the name of the subkey with which the value is associated. If this parameter is `None` or empty, the function retrieves the value set by the `SetValue` method for the key identified by *key*.

Values in the registry have name, type, and data components. This method retrieves the data for a key's first value that has a `NULL` name. But the underlying API call doesn't return the type, so always use `QueryValueEx` if possible.

winreg.QueryValue key,sub_key,value_name winreg.QueryValue

QueryValueEx(key, value_name)

Retrieves the type and data for a specified value name associated with an open registry key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

value_name is a string indicating the value to query.

The result is a tuple of 2 items:

Index	Meaning
`0`	The value of the registry item.
`1`	An integer giving the registry type for this value (see table in docs for `SetValueEx`)

winreg.QueryValue key,sub_key,value_name winreg.QueryValueEx

SaveKey(key, file_name)

Saves the specified key, and all its subkeys to the specified file.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

file_name is the name of the file to save registry data to. This file cannot already exist. If this filename includes an extension, it cannot be used on file allocation table (FAT) file systems by the `LoadKey` method.

If *key* represents a key on a remote computer, the path described by *file_name* is relative to the remote computer. The caller of this method must possess the **SeBackupPrivilege** security privilege. Note that privileges are different than permissions -- see the Conflicts Between User Rights and Permissions documentation for more details.

This function passes `NULL` for *security_attributes* to the API.

winreg.SaveKey key,file_name winreg.SaveKey

SetValue(key, sub_key, type, value)

Associates a value with a specified key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

sub_key is a string that names the subkey with which the value is associated.

type is an integer that specifies the type of the data. Currently this must be `REG_SZ`, meaning only strings are supported. Use the `SetValueEx` function for support for other data types.

value is a string that specifies the new value.

If the key specified by the *sub_key* parameter does not exist, the SetValue function creates it.

Value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as files with the filenames stored in the configuration registry. This helps the registry perform efficiently.

The key identified by the *key* parameter must have been opened with `KEY_SET_VALUE` access.

winreg.SetValue key,sub_key,type,value winreg.SetValue

SetValueEx(key, value_name, reserved, type, value)

Stores data in the value field of an open registry key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

value_name is a string that names the subkey with which the value is associated.

reserved can be anything -- zero is always passed to the API.

type is an integer that specifies the type of the data. See `Value Types <value-types>` for the available types.

value is a string that specifies the new value.

This method can also set additional value and type information for the specified key. The key identified by the key parameter must have been opened with `KEY_SET_VALUE` access.

To open the key, use the `CreateKey` or `OpenKey` methods.

Value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as files with the filenames stored in the configuration registry. This helps the registry

perform efficiently.

winreg.SetValue key,sub_key,type,value winreg.SetValueEx

DisableReflectionKey(key)

Disables registry reflection for 32-bit processes running on a 64-bit operating system.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

Will generally raise `NotImplementedError` if executed on a 32-bit operating system.

If the key is not on the reflection list, the function succeeds but has no effect. Disabling reflection for a key does not affect reflection of any subkeys.

winreg.DisableReflectionKey key winreg.DisableReflectionKey

EnableReflectionKey(key)

Restores registry reflection for the specified disabled key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

Will generally raise `NotImplementedError` if executed on a 32-bit operating system.

Restoring reflection for a key does not affect reflection of any subkeys.

winreg.EnableReflectionKey key winreg.EnableReflectionKey

QueryReflectionKey(key)

Determines the reflection state for the specified key.

key is an already open key, or one of the predefined `HKEY_* constants <hkey-constants>`.

Returns `True` if reflection is disabled.

Will generally raise `NotImplementedError` if executed on a 32-bit operating system.

winreg.QueryReflectionKey key winreg.QueryReflectionKey

Constants

The following constants are defined for use in many `!winreg` functions.

HKEY* Constants

HKEY_CLASSES_ROOT

Registry entries subordinate to this key define types (or classes) of documents and the properties associated with those types. Shell and COM applications use the information stored under this key.

HKEY_CURRENT_USER

Registry entries subordinate to this key define the preferences of the current user. These preferences include the settings of environment variables, data about program groups, colors, printers, network connections, and application preferences.

HKEY_LOCAL_MACHINE

Registry entries subordinate to this key define the physical state of the computer, including data about the bus type, system memory, and installed hardware and software.

HKEY_USERS

Registry entries subordinate to this key define the default user configuration for new users on the local computer and the user configuration for the current user.

HKEY_PERFORMANCE_DATA

Registry entries subordinate to this key allow you to access performance data. The data is not actually stored in the registry; the registry functions cause the system to collect the data from its source.

HKEY_CURRENT_CONFIG

Contains information about the current hardware profile of the local computer system.

HKEY_DYN_DATA

This key is not used in versions of Windows after 98.

Access Rights

For more information, see Registry Key Security and Access.

KEY_ALL_ACCESS

Combines the STANDARD_RIGHTS_REQUIRED, `KEY_QUERY_VALUE`, `KEY_SET_VALUE`, `KEY_CREATE_SUB_KEY`, `KEY_ENUMERATE_SUB_KEYS`, `KEY_NOTIFY`, and `KEY_CREATE_LINK` access rights.

KEY_WRITE

Combines the STANDARD_RIGHTS_WRITE, `KEY_SET_VALUE`, and `KEY_CREATE_SUB_KEY` access rights.

KEY_READ

Combines the STANDARD_RIGHTS_READ, `KEY_QUERY_VALUE`, `KEY_ENUMERATE_SUB_KEYS`, and `KEY_NOTIFY` values.

KEY_EXECUTE

Equivalent to `KEY_READ`.

KEY_QUERY_VALUE

Required to query the values of a registry key.

KEY_SET_VALUE

Required to create, delete, or set a registry value.

KEY_CREATE_SUB_KEY

Required to create a subkey of a registry key.

KEY_ENUMERATE_SUB_KEYS

Required to enumerate the subkeys of a registry key.

KEY_NOTIFY

Required to request change notifications for a registry key or for subkeys of a registry key.

KEY_CREATE_LINK

Reserved for system use.

64-bit Specific

For more information, see Accessing an Alternate Registry View.

KEY_WOW64_64KEY

Indicates that an application on 64-bit Windows should operate on the 64-bit registry view. On 32-bit Windows, this constant is ignored.

KEY_WOW64_32KEY

Indicates that an application on 64-bit Windows should operate on the 32-bit registry view. On 32-bit Windows, this constant is ignored.

Value Types

For more information, see Registry Value Types.

REG_BINARY

Binary data in any form.

REG_DWORD

32-bit number.

REG_DWORD_LITTLE_ENDIAN

A 32-bit number in little-endian format. Equivalent to `REG_DWORD`.

REG_DWORD_BIG_ENDIAN

A 32-bit number in big-endian format.

REG_EXPAND_SZ

Null-terminated string containing references to environment variables (`%PATH%`).

REG_LINK

A Unicode symbolic link.

REG_MULTI_SZ

A sequence of null-terminated strings, terminated by two null characters. (Python handles this termination automatically.)

REG_NONE

No defined value type.

REG_QWORD

A 64-bit number.

3.6

REG_QWORD_LITTLE_ENDIAN

A 64-bit number in little-endian format. Equivalent to `REG_QWORD`.

3.6

REG_RESOURCE_LIST

A device-driver resource list.

REG_FULL_RESOURCE_DESCRIPTOR

A hardware setting.

REG_RESOURCE_REQUIREMENTS_LIST

A hardware resource list.

REG_SZ

A null-terminated string.

Registry Handle Objects

This object wraps a Windows HKEY object, automatically closing it when the object is destroyed. To guarantee cleanup, you can call either the `~PyHKEY.Close` method on the object, or the `CloseKey` function.

All registry functions in this module return one of these objects.

All registry functions in this module which accept a handle object also accept an integer, however, use of the handle object is encouraged.

Handle objects provide semantics for `~object.__bool__` -- thus :

```
if handle:
    print("Yes")
```

will print `Yes` if the handle is currently valid (has not been closed or detached).

The object also support equality comparison semantics, so handle objects will compare equal if they both reference the same underlying Windows handle value. Closed handle objects (those with a handle value of zero) always compare equal.

Handle objects can be converted to an integer (e.g., using the built-in `int` function), in which case the underlying Windows handle value is returned. You can also use the `~PyHKEY.Detach` method to return the integer handle, and also disconnect the Windows handle from the handle object.

PyHKEY.Close()

Closes the underlying Windows handle.

If the handle is already closed, no error is raised.

PyHKEY.Detach()

Detaches the Windows handle from the handle object.

The result is an integer that holds the value of the handle before it is detached. If the handle is already detached or closed, this will return zero.

After calling this function, the handle is effectively invalidated, but the handle is not closed. You would call this function when you need the underlying Win32 handle to exist beyond the lifetime of the handle object.

winreg.PyHKEY.Detach key winreg.PyHKEY.Detach

PyHKEY.__enter__() PyHKEY.__exit__(*exc_info)

The HKEY object implements `~object.__enter__` and `~object.__exit__` and thus supports the context protocol for the `with` statement:

```
with OpenKey(HKEY_LOCAL_MACHINE, "foo") as key:
    ...  # work with key
```

will automatically close *key* when control leaves the `with` block.

3.15 Handle objects are now compared by their underlying Windows handle value instead of object identity for equality comparisons.

□□□ #2
□ □□□ □□□ 2026-10-31 17:24:08 UTC
□ □□□ □□□ 2026-10-31 17:24:10 UTC