35.3. `_winreg` – Windows registry access¶

Platforms: Windows

Note

The `_winreg` module has been renamed to `winreg` in Python 3.0. The *2to3* tool will automatically adapt imports when converting your sources to 3.0.

New in version 2.0.

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

This module exposes a very low-level interface to the Windows registry; it is expected that in the future a new `winreg` module will be created offering a higher-level interface to the registry API.

This module offers the following functions:

`_winreg.CloseKey`(*hkey*)¶

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

Note that if *hkey* is not closed using this method (or via `handle.Close()`), it is closed when the *hkey* object is destroyed by Python.

`_winreg.ConnectRegistry`(*computer_name*, *key*)¶

Establishes a connection to a predefined registry handle on another computer, and returns a *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, a `WindowsError` exception is raised.

`_winreg.CreateKey`(*key*, *sub_key*)¶

Creates or opens the specified key, returning a *handle object*

key is an already open key, or one of the predefined `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, a `WindowsError` exception is raised.

`_winreg.DeleteKey`(*key*, *sub_key*)¶

Deletes the specified key.

key is an already open key, or any one of the predefined `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, a `WindowsError` exception is raised.

`_winreg.DeleteValue`(*key*, *value*)¶

Removes a named value from a registry key.

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

value is a string that identifies the value to remove.

`_winreg.EnumKey`(*key*, *index*)¶

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

key is an already open key, or any one of the predefined `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 a `WindowsError` exception is raised, indicating, no more values are available.

`_winreg.EnumValue`(*key*, *index*)¶

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

key is an already open key, or any one of the predefined `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 a `WindowsError` 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

`_winreg.ExpandEnvironmentStrings`(*unicode*)¶

Expands environment strings %NAME% in unicode string like const:REG_EXPAND_SZ:

```
>>> ExpandEnvironmentStrings(u"%windir%")
u"C:\\Windows"
```

New in version 2.6.

`_winreg.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.

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.

`_winreg.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 an already open key, or any of the predefined `HKEY_*` constants.

sub_key is a string that identifies the sub_key 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 than permissions - see the Win32 documentation for more details.

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

The Win32 documentation implies *key* must be in the `HKEY_USER` or `HKEY_LOCAL_MACHINE` tree. This may or may not be true.

`_winreg.OpenKey`(*key*, *sub_key*[, *res=0*][, *sam=KEY_READ*])¶

Opens the specified key, returning a *handle object*

key is an already open key, or any one of the predefined `HKEY_*` constants.

sub_key is a string that identifies the sub_key to open.

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

sam is an integer that specifies an access mask that describes the desired security access for the key. Default is `KEY_READ`

The result is a new handle to the specified key.

If the function fails, `WindowsError` is raised.

`_winreg.OpenKeyEx`()¶

The functionality of `OpenKeyEx()` is provided via `OpenKey()`, by the use of default arguments.

`_winreg.QueryInfoKey`(*key*)¶

Returns information about a key, as a tuple.

key is an already open key, or one of the predefined `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`	A long integer giving when the key was last modified (if available) as 100's of nanoseconds since Jan 1, 1600.

`_winreg.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.

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.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.

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.

`_winreg.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.

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()`, `ReplaceKey()` or `RestoreKey()` methods.

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 Win32 documentation for more details.

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

`_winreg.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.

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.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.

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

type is an integer that specifies the type of the data. This should be one of the following constants defined in this module:

Constant	Meaning
`REG_BINARY`	Binary data in any form.
`REG_DWORD`	A 32-bit number.
`REG_DWORD_LITTLE_ENDIAN`	A 32-bit number in little-endian format.
`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_RESOURCE_LIST`	A device-driver resource list.
`REG_SZ`	A null-terminated string.

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

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.

35.3.1. 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 `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 `__nonzero__()` - thus

```
if handle:
   print "Yes"
```

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

The object also support comparison semantics, so handle objects will compare true if they both reference the same underlying Windows handle value.

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 `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 (or long on 64 bit Windows) 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.

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

The HKEY object implements `__enter__()` and `__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.

New in version 2.6.

Table Of Contents