PEP 514 -- Python registration in the Windows registry

PEP:	514
Title:	Python registration in the Windows registry
Author:	Steve Dower <steve.dower at python.org>
BDFL-Delegate:	Paul Moore <p.f.moore at gmail.com>
Status:	Active
Type:	Informational
Created:	02-Feb-2016
Post-History:	02-Feb-2016, 01-Mar-2016, 18-Jul-2016
Resolution:	https://mail.python.org/pipermail/python-dev/2016-July/145697.html

---

Contents

Abstract

This PEP defines a schema for the Python registry key to allow third-party installers to register their installation, and to allow tools and applications detect and correctly display all Python environments on a user's machine. No implementation changes to Python are proposed with this PEP.

Python environments are not required to be registered unless they want to be automatically discoverable by external tools. As this relates to Windo only, these tools are expected to be predominantly GUI applications. However, console applications may also make use of the registered informatio This PEP covers the information that may be made available, but the actual presentation and use of this information is left to the tool designers.

The schema matches the registry values that have been used by the official installer since at least Python 2.5, and the resolution behaviour matche the behaviour of the official Python releases. Some backwards compatibility rules are provided to ensure tools can correctly detect versions of CPyt that do not register full information.

Motivation

When installed on Windows, the official Python installer creates a registry key for discovery and detection by other applications. This allows tools su as installers or IDEs to automatically detect and display a user's Python installations. For example, the [PEP 397](#) py.exe launcher and editors such PyCharm and Visual Studio already make use of this information.

Third-party installers, such as those used by distributions, typically create identical keys for the same purpose. Most tools that use the registry to de Python installations only inspect the keys used by the official installer. As a result, third-party installations that wish to be discoverable will overwrite

By describing a layout for registry keys that allows third-party installations to register themselves uniquely, as well as providing tool developers guidance for discovering all available Python installations, these collisions should be prevented. We also take the opportunity to add some well-kno metadata so that more information can be presented to users.

Definitions

A "registry key" is the equivalent of a file-system path into the registry. Each key may contain "subkeys" (keys nested within keys) and "values" (nar and typed attributes attached to a key). These are used on Windows to store settings in much the same way that directories containing configuratio files would work.

`HKEY_CURRENT_USER` is the root of settings for the currently logged-in user, and this user can generally read and write all settings under this root.

`HKEY_LOCAL_MACHINE` is the root of settings for all users. Generally, any user can read these settings but only administrators can modify them. It is typical for values under `HKEY_CURRENT_USER` to take precedence over those in `HKEY_LOCAL_MACHINE`.

On 64-bit Windows, `HKEY_LOCAL_MACHINE\Software\Wow6432Node` is a special key that 32-bit processes transparently read and write to rather th accessing the `Software` key directly.

Further documentation regarding registry redirection on Windows is available from the MSDN Library [1].

Structure

We consider there to be a single collection of Python environments on a machine, where the collection may be different for each user of the machir There are three potential registry locations where the collection may be stored based on the installation options of each environment:

```
HKEY_CURRENT_USER\Software\Python\<Company>\<Tag>
HKEY_LOCAL_MACHINE\Software\Python\<Company>\<Tag>
HKEY_LOCAL_MACHINE\Software\Wow6432Node\Python\<Company>\<Tag>
```

Official Python releases use `PythonCore` for Company, and the value of `sys.winver` for Tag. The Company `PyLauncher` is reserved. Other registered environments may use any values for Company and Tag. Recommendations are made later in this document.

Company-Tag pairs are case-insensitive, and uniquely identify each environment. Depending on the purpose and intended use of a tool, there are t suggested approaches for resolving conflicts between Company-Tag pairs.

Tools that list every installed environment may choose to include those even where the Company-Tag pairs match. They should ensure users can easily identify whether the registration was per-user or per-machine, and which registration has the higher priority.

Tools that aim to select a single installed environment from all registered environments based on the Company-Tag pair, such as the `py.exe` launch should always select the environment registered in `HKEY_CURRENT_USER` when than the matching one in `HKEY_LOCAL_MACHINE`.

Conflicts between `HKEY_LOCAL_MACHINE\Software\Python` and `HKEY_LOCAL_MACHINE\Software\Wow6432Node\Python` should only occur when both 64-bit and 32-bit versions of an interpreter have the same Tag. In this case, the tool should select whichever is more appropriate for its use.

If a tool is able to determine from the provided information (or lack thereof) that it cannot use a registered environment, there is no obligation to pres it to users.

Except as discussed in the section on backwards compatibility, Company and Tag values are considered opaque to tools, and no information about interpreter should be inferred from the text. However, some tools may display the Company and Tag values to users, so ideally the Tag will be able help users identify the associated environment.

Python environments are not required to register themselves unless they want to be automatically discoverable by external tools.

Backwards Compatibility

Python 3.4 and earlier did not distinguish between 32-bit and 64-bit builds in `sys.winver`. As a result, it is not possible to have valid side-by-side installations of both 32-bit and 64-bit interpreters under this scheme since it would result in duplicate Tags.

To ensure backwards compatibility, applications should treat environments listed under the following two registry keys as distinct, even when the Ta matches:

```
HKEY_LOCAL_MACHINE\Software\Python\PythonCore\<Tag>
HKEY_LOCAL_MACHINE\Software\Wow6432Node\Python\PythonCore\<Tag>
```

Environments listed under `HKEY_CURRENT_USER` may be treated as distinct from both of the above keys, potentially resulting in three environments discovered using the same Tag. Alternatively, a tool may determine whether the per-user environment is 64-bit or 32-bit and give it priority over the machine environment, resulting in a maximum of two discovered environments.

It is not possible to detect side-by-side installations of both 64-bit and 32-bit versions of Python prior to 3.5 when they have been installed for the current user. Python 3.5 and later always uses different Tags for 64-bit and 32-bit versions.

The following section describe user-visible information that may be registered. For Python 3.5 and earlier, none of this information is available, but alternative defaults are specified for the `PythonCore` key.

Environments registered under other Company names have no backward compatibility requirements and must use distinct Tags to support side-by-installations. Tools consuming these registrations are not required to disambiguate tags other than by preferring the user's setting.

Company

The Company part of the key is intended to group related environments and to ensure that Tags are namespaced appropriately. The key name sho be alphanumeric without spaces and likely to be unique. For example, a trademarked name (preferred), a hostname, or as a last resort, a UUID wo be appropriate:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp
HKEY_CURRENT_USER\Software\Python\www.example.com
HKEY_CURRENT_USER\Software\Python\6C465E66-5A8C-4942-9E6A-D29159480C60
```

The company name `PyLauncher` is reserved for the [PEP 397](#) launcher (`py.exe`). It does not follow this convention and should be ignored by tools.

If a string value named `DisplayName` exists, it should be used to identify the environment manufacturer/developer/destributor to users. Otherwise, the name of the key should be used. (For `PythonCore`, the default display name is "Python Software Foundation".)

If a string value named `SupportUrl` exists, it may be displayed or otherwise used to direct users to a web site related to the environment. (ForPythonCore, the default support URL is "http://www.python.org/".)

A complete example may look like:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp
    (Default) = (value not set)
    DisplayName = "Example Corp"
    SupportUrl = "http://www.example.com"
```

Tag

The Tag part of the key is intended to uniquely identify an environment within those provided by a single company. The key name should be alphanumeric without spaces and stable across installations. For example, the Python language version, a UUID or a partial/complete hash would be appropriate, while a Tag based on the install directory or some aspect of the current machine may not. For example:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp\examplepy
HKEY_CURRENT_USER\Software\Python\ExampleCorp\3.6
HKEY_CURRENT_USER\Software\Python\ExampleCorp\6C465E66
```

It is expected that some tools will require users to type the Tag into a command line, and that the Company may be optional provided the Tag is unique across all Python installations. Short, human-readable and easy to type Tags are recommended, and if possible, select a value likely to be unique across all other Companies.

If a string value named `DisplayName` exists, it should be used to identify the environment to users. Otherwise, the name of the key should be used. (For `PythonCore`, the default is "Python " followed by the Tag.)

If a string value named `SupportUrl` exists, it may be displayed or otherwise used to direct users to a web site related to the environment. (ForPythonCore, the default is "http://www.python.org/".)

If a string value named `Version` exists, it should be used to identify the version of the environment. This is independent from the version of Python implemented by the environment. (For `PythonCore`, the default is the first three characters of the Tag.)

If a string value named `SysVersion` exists, it must be in `x.y` or `x.y.z` format matching the version returned by `sys.version_info` in the interprete omitted, the Python version is unknown. (For `PythonCore`, the default is the first three characters of the Tag.)

If a string value named `SysArchitecture` exists, it must match the first element of the tuple returned by `platform.architecture()`. Typically, thi will be "32bit" or "64bit". If omitted, the architecture is unknown. (ForPythonCore, the architecture is "32bit" when registered underHKEY_LOCAL_MACHINE\Software\Wow6432Node\Python *or* anywhere on a 32-bit operating system, "64bit" when registered underHKEY_LOCAL_MACHINE\Software\Python on a 64-bit machine, and unknown when registered under `HKEY_CURRENT_USER`.)

Note that each of these values is recommended, but optional. OmittingSysVersion or `SysArchitecture` may prevent some tools from correctly supporting the environment. A complete example may look like this:

```
 HKEY_CURRENT_USER\Software\Python\ExampleCorp\examplepy
     (Default) = (value not set)
     DisplayName = "Example Py Distro 3"
     SupportUrl = "http://www.example.com/distro-3"
     Version = "3.0.12345.0"
     SysVersion = "3.6.0"
     SysArchitecture = "64bit"
```

Beneath the environment key, an `InstallPath` key must be created. This key is always named `InstallPath`, and the default value must match `sys.prefix`:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp\3.6\InstallPath
    (Default) = "C:\ExampleCorpPy36"
```

If a string value named `ExecutablePath` exists, it must be the full path to the `python.exe` (or equivalent) executable. If omitted, the environment is executable. (For `PythonCore`, the default is the `python.exe` file in the directory referenced by the `(Default)` value.)

If a string value named `ExecutableArguments` exists, tools should use the value as the first arguments when executing `ExecutablePath`. Tools m add other arguments following these, and will reasonably expect standard Python command line options to be available.

If a string value named `WindowedExecutablePath` exists, it must be a path to the `pythonw.exe` (or equivalent) executable. If omitted, the default is value of `ExecutablePath`, and if that is omitted the environment is not executable. (For `PythonCore`, the default is the `pythonw.exe` file in the directory referenced by the `(Default)` value.)

If a string value named `WindowedExecutableArguments` exists, tools should use the value as the first arguments when executing `WindowedExecutablePath`. Tools may add other arguments following these, and will reasonably expect standard Python command line options to be available.

A complete example may look like:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp\examplepy\InstallPath
    (Default) = "C:\ExampleDistro30"
    ExecutablePath = "C:\ExampleDistro30\ex_python.exe"
    ExecutableArguments = "--arg1"
    WindowedExecutablePath = "C:\ExampleDistro30\ex_pythonw.exe"
    WindowedExecutableArguments = "--arg1"
```

Help

Beneath the environment key, a `Help` key may be created. This key is always named `Help` if present and has no default value.

Each subkey of `Help` specifies a documentation file, tool, or URL associated with the environment. The subkey may have any name, and the defau
value is a string appropriate for passing to `os.startfile` or equivalent.

If a string value named `DisplayName` exists, it should be used to identify the help file to users. Otherwise, the key name should be used.

A complete example may look like:

```
HKEY_CURRENT_USER\Software\Python\ExampleCorp\6C465E66\Help
    Python\
        (Default) = "C:\ExampleDistro30\python36.chm"
        DisplayName = "Python Documentation"
    Extras\
        (Default) = "http://www.example.com/tutorial"
        DisplayName = "Example Distro Online Tutorial"
```

Other Keys

All other subkeys under a Company-Tag pair are available for private use.

Official CPython releases have traditionally used certain keys in this space to determine the location of the Python standard library and other install
modules. This behaviour is retained primarily for backward compatibility. However, as the code that reads these values is embedded into the
interpreter, third-party distributions may be affected by values written into `PythonCore` if using an unmodified interpreter.

Sample Code

This sample code enumerates the registry and displays the available Company-Tag pairs that could be used to launch an environment and the targ

```python
# Display most-preferred environments.
# Assumes a 64-bit operating system
# Does not correctly handle PythonCore compatibility

import winreg

def enum_keys(key):
    i = 0
    while True:
        try:
            yield winreg.EnumKey(key, i)
        except OSError:
            break
        i += 1

def get_value(key, value_name):
    try:
        return winreg.QueryValue(key, value_name)
    except FileNotFoundError:
        return None

seen = set()
for hive, key, flags in [
    (winreg.HKEY_CURRENT_USER, r'Software\Python', 0),
    (winreg.HKEY_LOCAL_MACHINE, r'Software\Python', winreg.KEY_WOW64_64KEY),
    (winreg.HKEY_LOCAL_MACHINE, r'Software\Python', winreg.KEY_WOW64_32KEY),
```

```python
with winreg.OpenKeyEx(hive, key, access=winreg.KEY_READ | flags) as root_key:

    for company in enum_keys(root_key):

        if company == 'PyLauncher':

            continue


        with winreg.OpenKey(root_key, company) as company_key:

            for tag in enum_keys(company_key):

                if (company, tag) in seen:

                    if company == 'PythonCore':

                        # TODO: Backwards compatibility handling

                        pass

                    continue

                seen.add((company, tag))


                try:

                    with winreg.OpenKey(company_key, tag + r'\InstallPath') as ip_key:

                        exec_path = get_value(ip_key, 'ExecutablePath')

                        exec_args = get_value(ip_key, 'ExecutableArguments')

                        if company == 'PythonCore' and not exec_path:

                            # TODO: Backwards compatibility handling

                            pass

                except OSError:

                    exec_path, exec_args = None, None


                if exec_path:

                    print('{}\\{} - {} {}'.format(company, tag, exec_path, exec_args or ''))
```

```
        else:
            print('{}\\{} - (not executable)'.format(company, tag))
```

This example only scans PythonCore entries for the current user. Where data is missing, the defaults as described earlier in the PEP are substitute

Note that these defaults are only for use under PythonCore; other registrations do not have any default values:

```
# Only lists per-user PythonCore registrations
# Uses fallback values as described in PEP 514


import os
import winreg


def enum_keys(key):
    i = 0
    while True:
        try:
            yield winreg.EnumKey(key, i)
        except OSError:
            break
        i += 1


def get_value(key, value_name):
    try:
        return winreg.QueryValue(key, value_name)
    except FileNotFoundError:
        return None
```

```python
with winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Python\PythonCore") as company_key:
    print('Company:', get_value(company_key, 'DisplayName') or 'Python Software Foundation')
    print('Support:', get_value(company_key, 'SupportUrl') or 'http://www.python.org/')
    print()

    for tag in enum_keys(company_key):
        with winreg.OpenKey(company_key, tag) as tag_key:
            print('PythonCore\\' + tag)
            print('Name:', get_value(tag_key, 'DisplayName') or ('Python ' + tag))
            print('Support:', get_value(tag_key, 'SupportUrl') or 'http://www.python.org/')
            print('Version:', get_value(tag_key, 'Version') or tag[:3])
            print('SysVersion:', get_value(tag_key, 'SysVersion') or tag[:3])
            # Architecture is unknown because we are in HKCU
            # Tools may use alternate approaches to determine architecture when
            # the registration does not specify it.
            print('SysArchitecture:', get_value(tag_key, 'SysArchitecture') or '(unknown)')

        try:
            ip_key = winreg.OpenKey(company_key, tag + '\\InstallPath')
        except FileNotFoundError:
            pass
        else:
            with ip_key:
                ip = get_value(ip_key, None)
                exe = get_value(ip_key, 'ExecutablePath') or os.path.join(ip, 'python.exe')
                exew = get_value(ip_key, 'WindowedExecutablePath') or os.path.join(ip, 'python.exe')
                print('InstallPath:', ip)
```

```
            print('ExecutablePath:', exe)
            print('WindowedExecutablePath:', exew)
        print()
```

This example shows a subset of the registration that will be created by a just-for-me install of 64-bit Python 3.6.0. Other keys may also be created:

```
HKEY_CURRENT_USER\Software\Python\PythonCore
    (Default) = (value not set)
    DisplayName = "Python Software Foundation"
    SupportUrl = "http://www.python.org/"

HKEY_CURRENT_USER\Software\Python\PythonCore\3.6
    (Default) = (value not set)
    DisplayName = "Python 3.6 (64-bit)"
    SupportUrl = "http://www.python.org/"
    Version = "3.6.0"
    SysVersion = "3.6"
    SysArchitecture = "64bit"

HKEY_CURRENT_USER\Software\Python\PythonCore\3.6\Help\Main Python Documentation
    (Default) = "C:\Users\Me\AppData\Local\Programs\Python\Python36\Doc\python360.chm"
    DisplayName = "Python 3.6.0 Documentation"

HKEY_CURRENT_USER\Software\Python\PythonCore\3.6\InstallPath
    (Default) = "C:\Users\Me\AppData\Local\Programs\Python\Python36\"
    ExecutablePath = "C:\Users\Me\AppData\Local\Programs\Python\Python36\python.exe"
    WindowedExecutablePath = "C:\Users\Me\AppData\Local\Programs\Python\Python36\pythonw.exe"
```

References

[1]   Registry Redirector (Windows)

            (https://msdn.microsoft.com/en-

            us/library/windows/desktop/aa384232.aspx)

Copyright