pyconstring Documentation

Release 1.0.0

Bor González-Usach

Mar 30, 2018

Contents

CHAPTER 1

Usage

1.1 Object construction

Constructing a connection string from scratch. As you can see, the case of the keys is converted automatically:

```
>>> from pyconstring import ConnectionString
>>> cs = ConnectionString()
>>> cs['user'] = 'manuel'
>>> cs['password'] = '1234'
>>> print cs.get_string()
User=manuel;Password=1234;
```

Note: You can specify your own key formatter by subclassing the ConnectionString class, and overriding the `_format_key` method.

Parsing an already existing string:

```
>>> cs = ConnectionString.from_string('key1=value1;key2=value2;')
>>> cs['key1'] = 'another value'
>>> cs.get_string()
u'Key1=another value;Key2=value2;'
>>> cs['user'] = 'johnny'
>>> print cs.get_string()
Key1=another value;Key2=value2;User=johnny;
```

Note: By default when parsing a string, if the key `Provider` appears more than once, the first entry will be preserved. You can control which keys are not overridable by subclassing and overwriting `_non_overridable_keys`

It can be instanciated from iterable:

```
>>> cs = ConnectionString([('key1', 'value1'), ('key2', 'value2')])
>>> cs['key1']
'value1'
>>> print cs.get_string()
Key1=value1;Key2=value2;
```

Or directly from another dictionary:

```
>>> ConnectionString({'key1': 'val1', 'key2': 'val2'})
<ConnectionString 'Key2=val2;Key1=val1;'>
```

1.2 Object manipulation

The ConnectionString is a subclass of OrderedDict and therefore offers the dict API. Some examples of this:

```
>>> cs = ConnectionString.from_string('key1=value1;key2=value2;')
>>> for key, value in cs.iter():
...   print key, value
...
Key1 value1
Key2 value2
>>> 'key1' in cs
True
>>> del cs['key1']
>>> 'key1' in cs
False
>>> list(cs)
[u'Key2']
>>> cs['key3'] = 'hey'
>>> cs2 = ConnectionString.from_string('hello=world;')
>>> cs == cs2
False
>>> cs == cs
True
```

Check the *API* for more details.

Key translations made easy. For instance, useful to convert from ADODB parameters to ODBC ones:

```
>>> cs['Provider'] = 'some provider'
>>> cs['user id'] = 'chanquete'
>>> cs.translate({'provider': 'driver', 'user id': 'uid'})
>>> print cs.get_string()
Driver=some provider;Uid=chanquete;
```

CHAPTER 2

Rules

There is not a universal syntax for connection strings, therefore this can't work for every single possible syntax. However, there is a set of general rules that keeps the problems at bay, and pyconstring has been implemented following these generic rules. Those are:

- Keys are not case sensitive
- If a key contains an equal sign, it must be doubled in the connection string. For instance the key `key=1` will be serialized as `key==1`.
- Surrounding white spaces around the key or value are ignored, unless the value is quoted.
- Quoting of value is necessary when it contains white spaces or semicolons, or it starts with any quote.
- If the value needs to be quoted, it can be quoted with single or double quotes.
- However, if the value needs to be quoted, and it contains the same type quotes, those have to be doubled. If it contains the other type of quotes, no special handling is needed. `value"45` can be serialized as `'value"45'` or `"value""45"`.
- Normally the last appearance of a key in a connection string takes precedence, but there are some exceptions like `Provider`, with which the first appearance will take precedence and will not be overridden.

CHAPTER 3

API

The ConnectionString class has the same API as standard dictionary, plus the following methods:

class pyconstring.pyconstring.**ConnectionString**(**args*, ***kwargs*)
Bases: collections.OrderedDict

classmethod from_string(*string*)
Creates a new instance and loads the passed string

Parameters **string** (*unicode*) – connection string to be parsed

Return type *ConnectionString*

get_string()

Returns the composed connection string

Return type unicode

translate(*trans*, *strict=True*)
Translates the keys of the store.

Parameters

- **trans** (*dict*) – translation mapping {pre name: post name}
- **strict** (*bool*) – When strict, the existing keys in self that are not in *trans* will be removed. If not strict, they will still exist.

CHAPTER 4

Release Notes

4.1 New in 0.5.0

- Now the ConnectionString class inherits from OrderedDict, and therefore the code has been substantially simplified. The only change in the API is that the class methods `from_iterable` and `from_dict` have been removed. Now you can just instantiate the class passing the iterable or the dict to the main constructor.

Python Module Index

p

Index