ooouno

Release 0.2.0

:Barry-Thomas-Paul: Moss

Apr 18, 2022

CONTENTS:

CHAPTER

ONE

OOOUNO

ooouno is a library of all *(more than 4300)* classes, typings and types for the LibreOffice API.

ooouno is for version `7.3` of LibreOffice API.

More about LibreOffice.

CHAPTER

TWO

DOCS

Read the docs here

CHAPTER

THREE

INSTALLATION

3.1 CONDA

ooouno on Anaconda

```
$ conda install -c conda-forge ooouno
```

For LibreOffice <= 7.2 `$ conda install -c conda-forge ooouno==0.1.13`

3.2 PIP

ooouno PyPI

```
$ pip install ooouno
```

For LibreOffice <= 7.2 `pip install ooouno==0.1.13`

CHAPTER

FOUR

USAGE

All class found in LO API are recreated in this library.

For instance:

`from ooo.dyn.style.line_spacing import LineSpacing` is equivalent to
`from com.sun.star.style import LineSpacing`

`from ooo.cssdyn.style import LineSpacing` is equivalent to
`from com.sun.star.style import LineSpacing`

4.1 Namespace

There are four namespaces representing LO API in this library.

4.1.1 ooo.lo

Namespace `ooo.lo` contains all static python classes. The format is
`ooo.lo.<ns>.<snake_case_name>.<PascalCaseName>`

```
from ooo.lo.uno.x_interface import XInterface

def foo(input:str) -> XInterface: ...
```

4.1.2 ooo.dyn

Namespace `ooo.dyn` contains static and dynamic classes depending on class type. The format is `ooo.dyn.<ns>.<snake_case_name>.<PascalCaseName>`

This namespace has dynamic classes that are changed at runtime. For classes that are dynamic are fully or partially replaced by UNO version at runtime.

This allows for typings while in design time (working in IDE) and at runtime UNO classes are created instead.

```
>>> from ooo.dyn.style.line_spacing import LineSpacing as DynLineSpacing
>>> from ooo.lo.style.line_spacing import LineSpacing as LoLineSpacing
>>> dyn_lns = DynLineSpacing(Height=10, Mode=3)
>>> lo_lns = LoLineSpacing(Height=10, Mode=3)
>>> assert dyn_lns.Height == 10
>>> assert dyn_lns.Mode == 3
>>> type(dyn_lns).__name__
'com.sun.star.style.LineSpacing'
>>> type(lo_lns).__name__
'LineSpacing'
```

4.1.3 ooo.csslo

Namespace `ooo.csslo` contains static classes as LO API style imports.
The format is `ooo.csslo.<ns>.<PascalCaseName>`

When importing from `ooo.csslo` all classes in that namespace are also loaded. Under some circumstances this may not be desired. Such as packaging with stickytape.

```
>>> from ooo.lo.style.line_spacing import LineSpacing as LoLineSpacing
>>> from ooo.csslo.style import LineSpacing as CssLineSpacing
>>> LoLineSpacing is CssLineSpacing
True
>>> ls = CssLineSpacing()
>>> type(ls).__name__
'LineSpacing'
```

4.1.4 ooo.cssdyn

Namespace `ooo.cssdyn` contains static and dynamic classes depending on class type as LO API style imports.

When importing from `ooo.cssdyn` all classes in that namespace are also loaded. Under some circumstances this may not be desired. Such as packaging with stickytape.

```
>>> from ooo.dyn.style.line_spacing import LineSpacing as DynLineSpacing
>>> from ooo.cssdyn.style import LineSpacing as CssLineSpacing
>>> DynLineSpacing is CssLineSpacing
True
>>> ls = CssLineSpacing()
>>> type(ls).__name__
'com.sun.star.style.LineSpacing'
```

4.2 Generally speaking

When using ooo as typings then import from `ooo.lo` or `ooo.csslo`.

When using ooo interactivly such as creating structs, enums, singletons, const classes then import from `ooo.dyn` or `ooo.cssdyn`.

4.2.1 Class types

Const Class

Const classes represent a const

Static

Const classes in `ooo.lo` and `ooo.csslo` namespaces are static classes.

Example static:

```
from ooo.lo.awt.device_capability import DeviceCapability
assert DeviceCapability.RASTEROPERATIONS == 1
assert DeviceCapability.GETBITS == 2
```

Const classes in `ooo.lo` and `ooo.csslo` namespaces are the same classes.

Example:

```
from ooo.lo.awt.device_capability import DeviceCapability as LoDeviceCapability
from ooo.csslo.awt import DeviceCapability as CssDeviceCapability
from ooo.dyn.awt.device_capability import DeviceCapability as DynDeviceCapability
same = LoDeviceCapability is CssDeviceCapability
assert same == True
same = CssDeviceCapability is DynDeviceCapability
assert same == False
```

Dynamic

Const classes in `ooo.dyn` and `ooo.cssdyn` namespaces are dynamic classes and are changed during runtime.

Each const class in the `ooo.dyn` and `ooo.cssdyn` namespaces have a corrsponding enum class as well. The enum class is always the name of the UNO class with `Enum` appended.

For instance: Class `ooo.cssdyn.DeviceCapability` will have corresponding `ooo.cssdyn.DeviceCapabilityEnum` class containing same values. This is for convenience.

Example dynamic:

```
from ooo.dyn.awt.device_capability import DeviceCapability, DeviceCapabilityEnum
assert DeviceCapability.GETBITS == DeviceCapabilityEnum.GETBITS
assert DeviceCapability.RASTEROPERATIONS == DeviceCapabilityEnum.RASTEROPERATIONS
assert DeviceCapability.GETBITS == DeviceCapabilityEnum.GETBITS.value
```

(continues on next page)

(continued from previous page)

```
assert DeviceCapability.RASTEROPERATIONS == DeviceCapabilityEnum.RASTEROPERATIONS.
↪value
assert DeviceCapability.__module__ == 'uno'
```

Const classes in `ooo.dyn` and `ooo.cssdyn` namespaces are the same classes.

Example:

```
from ooo.dyn.awt.device_capability import DeviceCapability as DynDeviceCapability
from ooo.cssdyn.awt import DeviceCapability as CssDeviceCapability
from ooo.lo.awt.device_capability import DeviceCapability as LoDeviceCapability
same = DynDeviceCapability is CssDeviceCapability
assert same == True
same = DynDeviceCapability is LoDeviceCapability
assert same == False
```

Note: This library is for Libre Office 7.2; However, it is somewhat backwards compatable.

For instance `ooo.cssdyn.style.NumberingType` had the attributes **ARABIC_ZERO3**, **ARABIC_ZERO4**, **ARABIC_ZERO5**, **SZEKELY_ROVAS** added in LO 7.x. When dynamic `NumberingType` or `NumberingTypeEnum` is generated in pre LO 7 environment these attributes will be missing.

See: API NumberingType

Enum Class

Enum classes represent an enumeration

Static

Enum classes in `ooo.lo` and `ooo.csslo` namespaces are static classes.

Example static:

```
from ooo.csslo.awt import FontSlant
assert FontSlant.ITALIC.__module__ == 'ooo.lo.awt.font_slant'
assert FontSlant.NONE.__module__ == 'ooo.lo.awt.font_slant'
e = FontSlant('OBLIQUE')
assert e == FontSlant.OBLIQUE
assert e.value == FontSlant.OBLIQUE.value
assert e.__module__ == 'ooo.lo.awt.font_slant'
e = FontSlant(FontSlant.OBLIQUE)
assert e == FontSlant.OBLIQUE
assert e.__module__ == 'ooo.lo.awt.font_slant'
```

Enum classes in `ooo.lo` and `ooo.csslo` namespaces are the same classes.

Example:

```
from ooo.lo.awt.font_slant import FontSlant as LoFontSlant
from ooo.csslo.awt import FontSlant as CssFontSlant
```

(continues on next page)

(continued from previous page)

```
from ooo.dyn.awt.font_slant import FontSlant as DynFontSlant
same = LoFontSlant is CssFontSlant
assert same == True
same = LoFontSlant is DynFontSlant
assert same == False
```

Dynamic

Enum classes in `ooo.dyn` and `ooo.cssdyn` namespaces are dynamic classes and are changed during runtime. All enumeration values are UNO equivalent.

Example dynamic:

```
from ooo.cssdyn.awt import FontSlant

e = FontSlant('OBLIQUE')
assert e == FontSlant.OBLIQUE
assert e.value == FontSlant.OBLIQUE.value

e = FontSlant(FontSlant.OBLIQUE)
assert e == FontSlant.OBLIQUE
```

At runtime dynamic enum are the same as UNO enum.

```
from ooo.cssdyn.awt import FontSlant
from com.sun.star.awt.FontSlant import ITALIC
same = FontSlant.ITALIC is ITALIC
assert same
```

Enum classes in `ooo.dyn` and `ooo.cssdyn` namespaces are the same classes.

Example:

```
from ooo.dyn.awt.font_slant import FontSlant as DynFontSlant
from ooo.cssdyn.awt import FontSlant as CssFontSlant
from ooo.lo.awt.font_slant import FontSlant as LoFontSlant
same = DynFontSlant is CssFontSlant
assert same == True
same = DynFontSlant is LoFontSlant
assert same == False
```

Exception Class

Exception classes represent a UNO Exception

Static

Exception classes in `ooo.lo` and `ooo.csslo` namespaces are static classes.

Example static:

```
from ooo.lo.uno.exception import Exception
ex = Exception(Message="I made an error")
assert ex.Message == 'I made an error'
assert type(ex).__name__ == 'Exception'
assert ex.__module__ == 'ooo.lo.uno.exception'
```

Exception classes in `ooo.lo` and `ooo.csslo` namespaces are the same classes.

Example:

```
from ooo.lo.uno.exception import Exception as LoException
from ooo.csslo.uno import Exception as CssException
same = LoException is CssException
assert same == True
```

Dynamic

Exception classes in `ooo.dyn` and `ooo.cssdyn` namespaces are dynamic classes and are changed during runtime.

Example dynamic:

```
from ooo.dyn.uno.runtime_exception import RuntimeException
ex = RuntimeException("I made an error")
assert ex.Message == 'I made an error'
assert type(ex).__name__ == 'com.sun.star.uno.RuntimeException'
assert ex.__module__ == 'uno'
```

Dynamic exceptions can be used in try block to catch UNO exceptions.

```
from ooo.dyn.uno.runtime_exception import RuntimeException

def foo():
    try:
        # some uno operation
        ...
    except RuntimeException as e:
        # handle com.sun.star.uno.RuntimeException errors
        print(e)
```

Dynamic exception can be used to raise UNO exceptions.

```
from ooo.dyn.uno.exception import Exception as UnoException

def foo(x):
    if not x:
        raise UnoException('Expected x to be something?')
    # some amazing stuff
```

(continues on next page)

(continued from previous page)

```
def bar(y):
    if not y:
        ex = UnoException()
        ex.Message='Expected y to be something?'
        raise ex
    # some amazing stuff

def foobar(z):
    if not z:
        raise UnoException
    # some amazing stuff
```

Exception classes in `ooo.dyn` and `ooo.cssdyn` namespaces are the same classes.

Note: Dynamic exception classes return equivalent UNO classes

This means dynamic exceptions are interchangable with UNO classes.

Interface Class

An interface defines how something interacts with its environment. A UNO interface resembles a group of subroutine and function declarations; arguments and return types are specified along with functionality. The interface indicates how an object can be used, but it says nothing about the implementation.

In ooo all interface classes are abstract classes.

Tip: UNO interface names start with the capital letter X.

Interface classes imported from `ooo.lo` and `ooo.csslo` namespaces are static and not changed during runtime.

Interface classes imported from `ooo.dyn` and `ooo.cssdyn` namespaces are dynamic and are changed to UNO clasess at runtime.

Example:

```
from ooo.csslo.uno import XInterface as LoXInterface
from ooo.cssdyn.uno import XInterface as DynXInterface
assert LoXInterface != DynXInterface
assert LoXInterface.__name__ == 'XInterface'
assert LoXInterface.__module__ == 'ooo.lo.uno.x_interface'
assert DynXInterface.__name__ == 'com.sun.star.uno.XInterface'
assert DynXInterface.__module__ == 'uno'
```

Service Class

A service abstractly defines an object by combining interfaces and properties to encapsulate some useful functionality. A UNO interface defines how an object interacts with the outside world. A UNO structure defines a collection of data; and a UNO service combines them together. Like a UNO interface, a UNO service does not specify the implementation. It only specifies how to interact with the object.

A service may include multiple services and interfaces. An interface usually defines a single aspect of a service and therefore is usually smaller in scope.

Example creating service using *uno_helper*.

```
from ooo.helper import uno_helper
from ooo.csslo.ucb import SimpleFileAccess
file_access: SimpleFileAccess = uno_helper.create_uno_service(
    'com.sun.star.ucb.SimpleFileAccess')
s = file_access.getContentType(
    'https://raw.githubusercontent.com/Amourspirit/python-ooouno/main/README.rst')
assert s == 'application/http-content'
```

Service class in ooo are the same accross all namespaces and do not have any dynamic generation.

```
>>> from ooo.lo.ucb.simple_file_access import SimpleFileAccess as LoSimpleFileAccess
>>> from ooo.dyn.ucb.simple_file_access import SimpleFileAccess as DynSimpleFileAccess
>>> from ooo.csslo.ucb import SimpleFileAccess as CssLoSimpleFileAccess
>>> from ooo.cssdyn.ucb import SimpleFileAccessas CssDynSimpleFileAccess
>>> LoSimpleFileAccess is DynSimpleFileAccess
True
>>> LoSimpleFileAccess is CssLoSimpleFileAccess
True
>>> LoSimpleFileAccess is CssDynSimpleFileAccess
True
```

Singleton Class

Singleton classes represent a UNO singleton.

Static

Singleton classes in `ooo.lo` and `ooo.csslo` namespaces are static abstract classes.

Singleton classes in `ooo.lo` and `ooo.csslo` namespaces are the same classes.

Example:

```
from ooo.lo.beans.the_introspection import theIntrospection as LotheIntrospection
from ooo.csslo.beans import theIntrospection as CsstheIntrospection
same = LotheIntrospection is CsstheIntrospection
assert same == True
```

Dynamic

Singleton classes in ooo.dyn and ooo.cssdyn namespaces are dynamic classes and are changed during runtime.

Example dynamic:

```
from ooo.cssdyn.beans import theIntrospection
singleton = theIntrospection()
assert type(singleton).__name__ == "pyuno"
im_name = "com.sun.star.comp.stoc.Introspection"
assert singleton.getImplementationName() == im_name
```

Singleton classes in ooo.dyn and ooo.cssdyn namespaces are dynamic classes and are changed during runtime

---

Note: Dynamic singleton classes have a dynamic constructor.

This means ooo.cssdyn.beans.theIntrospection is a function at runtime
whereas singleton = theIntrospection() is an instance of a UNO class.

Struct Class

Struct classes represent a UNO structure.

Static

Struct classes in ooo.lo and ooo.csslo namespaces are static classes.

Example static:

```
from ooo.csslo.awt import Rectangle
rect1 = Rectangle(X=100, Y=200, Width=50, Height=1)
assert rect1.X == 100
assert rect1.Height == 1
rect1.Width == 250
assert rect1.Width == 250
rect2 = Rectangle()
rect2.X = 122
rect2.Y = 240
assert rect2.X == 122
assert rect2.Y == 240
```

Struct classes in ooo.lo and ooo.csslo namespaces are the same classes.

Example:

```
from ooo.lo.awt.rectangle import Rectangle as LoRectangle
from ooo.csslo.awt import Rectangle as CssRectangle
from ooo.cssdyn.awt import Rectangle as DynRectangle
same = LoRectangle is CssRectangle
```

(continues on next page)

(continued from previous page)

```
assert same == True
same = LoRectangle is DynRectangle
assert same == False
```

Dynamic

Struct classes in `ooo.dyn` and `ooo.cssdyn` namespaces are dynamic classes and are changed to UNO classes at runtime.

Example dynamic:

```
from ooo.cssdyn.awt import Rectangle
from com.sun.star.awt import Rectangle as URectangle
rect1 = Rectangle(X=100, Y=200, Width=50, Height=1)
assert isinstance(rect1, URectangle)
assert rect1.X == 100
assert rect1.Height == 1
rect1.Width == 250
assert rect1.Width == 250
rect2 = Rectangle()
rect2.X = 122
rect2.Y = 240
assert rect2.X == 122
assert rect2.Y == 240
```

General

Struct classes can accept keyword or positional arguments.

Example keyword and positional:

```
from ooo.cssdyn.awt import Rectangle
rect1 = Rectangle(X=100, Y=200, Width=50, Height=1)
rect2 = Rectangle(100, 200, 50, 1)
assert rect1.X == rect2.X
assert rect1.Y == rect2.Y
assert rect1.Width == rect2.Width
assert rect1.Height == rect2.Height
```

Struct classes can be constructed by another struct of the same type:

Example construct from other struct:

```
from ooo.cssdyn.awt import Rectangle
rect1 = Rectangle(100, 200, 50, 1)
rect2 = Rectangle(rect1)
assert rect1.X == rect2.X
assert rect1.Y == rect2.Y
assert rect1.Width == rect2.Width
assert rect1.Height == rect2.Height
```

Typedef

typedef objects define a type using NewType similar to type alias.

Static type checker will treat the new type as if it were a subclass of the original type. This is useful in helping catch logical errors:

In the following example Color is a NewType of int `Color = typing.NewType('Color', int)`

Example:

```
from ooo.dyn.util.color import Color
c1 = Color(234)
c2 = Color(6)
assert isinstance(c1, int)
result = c1 + c2
assert result == 240
```

Typedef's in ooo are the same accross all namespaces and do not have any dynamic generation.

```
>>> from ooo.lo.util.color import Color as LoColor
>>> from ooo.dyn.util.color import Color as DynColor
>>> from ooo.csslo.util import Color as CssLoColor
>>> from ooo.cssdyn.util import Color as CssDynColor
>>> LoColor is DynColor
True
>>> LoColor is CssLoColor
True
>>> LoColor is CssDynColor
True
```

Note: Checks for **NewType** are enforced only by the static type checker. At runtime the statement Color = NewType('Color', int) will make Color a function that immediately returns whatever parameter you pass it. That means the expression Color(some_value) does not create a new class or introduce any overhead beyond that of a regular function call.

4.2.2 helper

uno_helper

4.2.3 Templates

There is a seperate template project. The project is on GitHub.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search