

Aarhus 2015 CONFERENCE



Upcoming PyUNO improvements in LibreOffice 5.1

- ▼ Matthew Francis



Introduction



- ▼ The state of PyUNO as of LibreOffice 5.0
 - ▼ Implemented near the dawn of Python, barely changed since
 - ▼ Very little syntactic sugar
 - ▼ Generally a lot like writing Java
 - ▼ Slower than it should be – especially from a remote process
- ▼ What I hope to achieve with the new changes for 5.1
 - ▼ Make working with UNO in Python feel more Pythonic
 - Less verbose – make use of available Python syntax
 - ▼ Make it faster than before
 - ▼ Personal goal – make PyUNO more appropriate as a base to build automated UI tests on





New features in PyUNO for 5.1



Collection interfaces



▼ Indexed array interfaces

- ▼ `com::sun::star::container::XIndexAccess`

- ▼ `com::sun::star::container::XIndexReplace`

- ▼ `com::sun::star::container::XIndexContainer`

▼ What's changed?

- ▼ UNO objects implementing these interfaces now behave like Python lists



Collection interfaces



- ▼ Whenever you see method calls like the following, there is a simpler way to do it:

▼ <code>count = obj.getCount()</code>	⇒ <code>count = len(obj)</code>
▼ <code>Value = obj.getByIndex(0)</code>	⇒ <code>value = obj[0]</code>
▼ <code>obj.replaceByIndex(0, value)</code>	⇒ <code>obj[0] = value</code>
▼ <code>obj.insertByIndex(0, value)</code>	⇒ <code>obj[0:0] = value</code>
▼ <code>obj.removeByIndex(0)</code>	⇒ <code>del obj[0]</code>

- ▼ Iteration and testing value presence also works:

- ▼ `for value in obj: ...`
- ▼ `if value in obj: ...`
 - ▼ ...but the `if - in` syntax is probably useful only rarely for indexed collections (and not efficient)





▼ Example: iterating over document footnotes – the old way

```
doc = ... # load a text document
count = doc.Footnotes.getCount()
for i in range(count):
    footnote = doc.Footnotes.getByIndex(i)
    print(footnote.String)
```





▼ Example: iterating over document footnotes – the new way

```
doc = ... # load a text document
count = len(doc.Footnotes)
for i in range(len):
    print(doc.Footnotes[i].String)
```

▼ Or even better, if the index isn't important:

```
doc = ... # load a text document
for footnote in doc.Footnotes:
    print(footnote.String)
```





- ▼ Other examples of XIndex*
 - ▼ Text document
 - ▼ Redlines
 - ▼ Endnotes
 - ▼ Spreadsheet
 - ▼ Charts
 - ▼ NamedRanges



Collection interfaces



▼ Associative array interfaces

- ▼ `com::sun::star::container::XNameAccess`

- ▼ `com::sun::star::container::XNameReplace`

- ▼ `com::sun::star::container::XNameContainer`

▼ What's changed?

- ▼ UNO objects implementing these interfaces now behave like Python dicts



Collection interfaces



- ▼ Whenever you see method calls like the following, there is a simpler way to do it:

▼ <code>Value = obj.getByName(key)</code>	⇒ <code>value = obj[key]</code>
▼ <code>if obj.hasByName(key): ...</code>	⇒ <code>if key in obj: ...</code>
▼ <code>obj.replaceByName(key, value)</code>	⇒ <code>obj[key] = value</code>
▼ <code>obj.insertByName(key, value)</code>	⇒ <code>obj[key] = value</code>
▼ <code>obj.removeByName(key)</code>	⇒ <code>del obj[key]</code>

- ▼ Iteration and testing value presence also works – for keys:

- ▼ `for key in obj: ...`
- ▼ `if key in obj: ...`
 - ▼ Different from indexed collections – the `for` and `in` operators tests for keys, not values



Collection interfaces



▼ Example: Navigate elements of a spreadsheet – the old way

```
spr = ... # load a spreadsheet
```

```
sheet = spr.Sheets.getByName('Sheet1')
```

```
range = sheet.NamedRanges.getByName('MyRange')
```



Collection interfaces



▼ Example: Navigate elements of a spreadsheet – the new way

```
spr = ... # load a spreadsheet
```

```
sheet = spr.Sheets['Sheet1']
```

```
range = sheet.NamedRanges['MyRange']
```



Collection interfaces



- ▼ What if an object supports both `XIndex*` and `XName*` ?
 - ▼ You can access it using both `obj[0]` and `obj['Name']`
 - ▼ However, iterating yields keys rather than values
 - ▼ Like a Python dict
- ▼ Examples:
 - ▼ Text document
 - ▼ TextTables
 - ▼ EmbeddedObjects
 - ▼ GraphicObjects



Collection interfaces



- ▼ Enumerations
 - ▼ `com::sun::star::container::XEnumerationAccess`
 - ▼ `com::sun::star::container::XEnumeration`
- ▼ What's changed
 - ▼ You can iterate over UNO enumerations the Python way



Collection interfaces



- ▼ Whenever you see method calls like the following, there is a quicker way to do it:

```
enm = obj.createEnumeration()  
while enm.hasMoreElements():  
    value = enm.nextElement()  
    ...
```

- ▼ Instead, do:

```
for value in obj:  
    ...
```



Collection interfaces



▼ Example: iterating over document paragraphs – the old way

```
doc = ... # Load a text document
enm = doc.Text.createEnumeration()
while enm.hasMoreElements():
    paragraph = enm.nextElement()
    print(paragraph.String)
```



Collection interfaces



▼ Example: iterating over document paragraphs – the new way

```
doc = ... # Load a text document
for paragraph in doc.Text:
    print(paragraph.String)
```

▼ Or use a Python style explicit iterator:

```
doc = ... # Load a text document
itr = iter(doc.Text)
paragraph = next(itr)
print(paragraph.String)
```

▼ Or flatten the text so it can be accessed by index:

```
doc = ... # Load a text document
paragraphs = list(doc.Text)
print(paragraphs[0].String)
```

- ▼ Obviously this can be inefficient for a large document – but extremely convenient in the context of e.g. a short test when there are only a few paragraphs



Elimination of explicit Any



- ▼ Certain method calls need to be passed an Any with a sequence of a specific type
- ▼ Most commonly this occurs with collection interfaces
- ▼ The syntax to deal with this in PyUNO was obscure and annoying

- ▼ Example: creating a document index

```
doc = ... # Load a text document
index = doc.createInstance("com.sun.star.text.ContentIndex");
uno.invoke(index.LevelParagraphStyles, \
    "replaceByIndex", (0, uno.Any("[]string", ('Caption',))))
```

- ▼ PyUNO can now infer the type required by the collection automatically
- ```
index.LevelParagraphStyles[0] = ('Caption',)
```



# List and iterator arguments



- ▼ Wherever a UNO API expects a sequence, a Python list or iterator can now be passed.
- ▼ This enables the use of list comprehensions and generator expressions for method calls and property assignments.

## ▼ Example: Populate a text table

```
doc = ... # Load a text document
tbl = doc.createInstance('com.sun.star.text.TextTable')
tbl.initialize(10,10)
doc.Text.insertTextContent(doc.CurrentController.ViewCursor, tbl, False)
Assign numbers 0..99 to the cells using a generator expression
tbl.Data = ((y for y in range(10*x,10*x + 10)) for x in range(10))
```



# List and iterator arguments



Untitled 1 - LibreOfficeDev Writer 5.1 [afce2eee1d163d942385c7b33db4901b3c71135d]

File Edit View Insert Format Styles Table Tools Window Help

Default Style Liberation Serif 12

| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|----|----|----|----|----|----|----|----|----|----|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |

Page 1 of 1 100 words, 190 characters Default Style English (UK) 100%



# Tolerant struct initialisation



- ▼ Initialising a UNO struct previously required all members to be set, or none
  - ▼ Example: PropertyValue – frequently, only name and value are needed

```
from com.sun.star.beans import PropertyValue
prop1 = PropertyValue()
prop1.Name = 'foo'
prop1.Value = 'bar'
prop2 = PropertyValue('foo', 0, 'bar', 0)
prop3 = PropertyValue(Name='foo', Handle=0, Value='bar', State=0)
```
  - ▼ This requirement is now relaxed when all arguments are named

```
prop4 = PropertyValue(Name='foo', Value='bar')
```



# Cell ranges



- ▼ A custom behaviour is applied to cell range objects
  - ▼ `com::sun::star::table::XCellRange`
- ▼ This is different to the other changes – the collection interfaces are generic, this is a higher level API
  - ▼ However, it's one that is widely used and could benefit from some syntactic sugar
- ▼ Applies to:
  - ▼ Sheets in Calc spreadsheets
  - ▼ Writer text tables
  - ▼ Subset cell ranges created on these



# Cell ranges



## ▼ Existing syntax

```
cell = cellrange.getCellByPosition(col, row)
rng = cellrange.getCellRangeByPosition(left, top, right, bottom)
rng = cellrange.getCellRangeByName(name)
```

## ▼ New syntax – access like a two dimensional array

```
cell = cellrange[0,0] # Access cell by indices
rng = cellrange[0,1:2] # Access cell range by index,slice
rng = cellrange[1:2,0] # Access cell range by slice,index
rng = cellrange[0:1,2:3] # Access cell range by slices
rng = cellrange['A1:B2'] # Access cell range by descriptor
rng = cellrange['Name'] # Access cell range by name
```



# Cell ranges



- ▼ Note that the indices used are in **Python/C order**
- ▼ These pairs are equivalent:

```
row r, column c
cell = cellrange[r,c]
cell = cellrange.getCellByPosition(c,r)
```

```
rows t to b, columns l to r
rng = cellrange[t:b,l:r]
rng = cellrange.getCellRangeByPosition(l,t,r-1,b-1).
```





# Cell ranges



- ▼ Objects which also implement `com::sun::star::table::XColumnRowRange` support negative indices (from-end indexing) and the below syntax for referencing whole rows and columns
- ▼ Calc spreadsheet sheets and cell ranges created upon these support this interface
- ▼ Writer text tables unfortunately don't

```
rng = cellrange[0] # Access cell range by row index
rng = cellrange[0, :] # Access cell range by row index
rng = cellrange[:, 0] # Access cell range by column index
```



# Import constants by group name



- ▼ Previously, UNO constants had to be imported individually

- ▼ Example

```
from com.sun.star.accessibility.AccessibleRole import MENU_BAR
from com.sun.star.accessibility.AccessibleRole import DIALOG
from com.sun.star.accessibility.AccessibleRole import PUSH_BUTTON
```

- ▼ Constant groups can now be imported as a whole

```
from com.sun.star.accessibility import AccessibleRole
Now you can reference AccessibleRole.MENU_BAR etc.
```



# Object hashability



- ▼ UNO objects should now have stable hash values
  - ▼ This allows them to be safely used as keys for collections

```
s = set()
s[obj] = 1
...
Later, we get the same object from UNO again
This only works if the object has a stable hash
del s[obj]
```
  - ▼ What's that “should” doing there?
    - ▼ Handle with care, don't rely on this if possible
    - ▼ Cases where this is useful should be rare



# Performance improvements



- ▼ Every time a UNO object is passed to PyUNO, we have to perform introspection on it to find out information about its methods and properties
- ▼ In the case of remote (out of process) PyUNO, this means making inter-process calls
- ▼ Inter-process calls are slow, so the fewer the better
- ▼ We can't avoid making at least a few calls
- ▼ Up to LibreOffice 5.0 there was a bug which meant there were up to 50 inter-process calls for each object
  - ▼ Predictably this wasn't very fast
- ▼ Further optimisations made to eliminate unnecessary calls and make some others lazy (only when actually needed, not for every object)
- ▼ Now it's much faster remotely and a little faster locally



# Fallout



- ▼ A major aim of these changes was not to break existing code
- ▼ Successful? Almost
  - ▼ Caused an issue with LibreLogo –  
commit `181a7b27acf29a2728be5a0eb3696796bc7df3da`
  - ▼ Now that some PyUNO objects behave like proper Python collections, they have truth values that depend on whether or not they're empty
  - ▼ The LibreLogo code used a variable that was either 0 or a PyUNO object, and expected the two choices to be always `False` or `True` respectively
    - ▼ Mea culpa – didn't expect that
    - ▼ Unfortunately no easy way to work around





Questions?



# Aarhus 2015 CONFERENCE



Thank you



All text and image content in this document is licensed under the [Creative Commons Attribution-Share Alike 3.0 License](#) (unless otherwise specified). "LibreOffice" and "The Document Foundation" are registered trademarks. Their respective logos and icons are subject to international copyright laws. The use of these therefore is subject to the [trademark policy](#).

