## Contents

1 **The Python part**  
2 **Templating with LibreOffice**  
   2.1 Use control structures  
   2.2 Define variables  
   2.3 Data Dictionary  
   2.4 Insert variables  
   2.5 Use format functions  
   2.6 Example documents  
3 **Contributing to py3o.template**  
   3.1 Tests  
   3.2 Code style  
4 **Source code documentation**  
   4.1 Templating  
   4.2 Data extraction  

Python Module Index
Contents:
Here is an example program that you can find in our source code. It shows how you can use a templated odt and create a final odt with your dataset from python code:

```python
from py3o.template import Template

t = Template("py3o_example_template.odt", "py3o_example_output.odt")

t.set_image_path('staticimage.logo', 'images/new_logo.png')

class Item(object):
    pass

items = list()

item1 = Item()
item1.val1 = 'Item1 Value1'
item1.val2 = 'Item1 Value2'
item1.val3 = 'Item1 Value3'
item1.Currency = 'EUR'
item1.Amount = '12345.35'
item1.InvoiceRef = '#1234'
items.append(item1)

for i in xrange(1000):
    item = Item()
    item.val1 = 'Item%s Value1' % i
    item.val2 = 'Item%s Value2' % i
    item.val3 = 'Item%s Value3' % i
    item.Currency = 'EUR'
    item.Amount = '6666.77'
    item.InvoiceRef = 'Reference #%04d' % i
    items.append(item)

document = Item()
document.total = '9999999999999.999'

data = dict(items=items, document=document)
t.render(data)
```
Templating with LibreOffice

If you have read the Python code above you have seen that we pushed a dictionary to our template.render() method. We must now declare the attributes you want to use from those variables in LibreOffice.

**Use control structures**

At the moment “for” and “if” controls are available.

In our example python code we have a dataset that contains a list of items. This list itself is named “items” and we want to iterate on all the items.

We should add a for loop using an hyperlink or an input field.

**Hyperlink method**

Every control structure must be added to you document using a specially formatted hyperlink:

```
link = py3o://for="item in items"
text = for="item in items"
```

Here is an example setup:
It is especially important to have the link value equivalent to the text value as in the example above.

Once you save your hyperlink, your py3o:// URL will become URL escaped which is fine.

Every control structure must be closed by a corresponding closing tag. In our case we must insert a “/for” hyperlink:

```
link = py3o://for
text = /for
```

Defined in the user interface as such:

![Hyperlink interface](image)

**Input field method**

Every control structure must be added to your document using an input field:

```
reference = py3o://for="item in items"
name = for="item in items"
```

Here is an example setup:
Every control structure must be closed by a corresponding closing tag. In our case we must insert a "/for" input field:

```plaintext
reference = py3o://for
name = /for
```

Defined in the user interface as such:
Define variables

This is done by creating user fields (CTRL-F2) with specific names. The naming scheme is important because it permits differentiate real user fields, which have their own purpose we won’t discuss in this document, from the ones we define in our templates.

Since we are inside a for loop that defines a variable names “items” we want to create a user variable in LibreOffice that is named like this:

```
py3o.item.Amount
```

The “Amount” is not something we invent. This is because the item variable is an object coming from your python code. And we defined the Amount attribute back then.

In LibreOffice, user fields can be defined by pressing CTRL-F2 then choosing variables and user-fields:
You must enter a value in name and value then press the green arrow to the right.

the “py3o.” prefix is mandatory. Without this prefix the templating system will not be able to find your variables.

The value (in our screenshot: Invoice.Reference) is only some sugar that helps read the template in OpenOffice.

You should take care to pick a nice and meaningfull value so that your end-users know what they will get just by looking at the document without being forced to open the variable definition.

Data Dictionary

If you are a developer and want to provide some kind of raw document for your users, it is a good idea to create all the relevent user variables yourself. This is what we call in our jargon creating the data dictionary.

This is especially important because the variable names (eg: py3o.variable.attribute) are linked to your code. And remember that your users do not have access to the code.

You should put them in a position where they can easily pick from a list instead of being forced to ask you what are the available variables.

Insert variables

Once you have setup variables and defined some optional control structures you can start inserting variables inside the document.

The best way it to use the menu:
Insert > Field > Other

or just press:

CTRL-F2

then choose User fields in the field type selection, then choose your desired variable in the second column and then finally click insert at the bottom:

This operation will insert your user field near your cursor. This field will be replaced at template.render() time by the real value coming from the dataset (see above python code)

**Insert placeholder images**

py3o.template can replace images on-the-fly. To add an image field, add a regular image as a placeholder, open its properties and prefix its name with “py3o.staticimage.”; the rest of the image name is then its identifier:
The Python code has to call set_image_path or set_image_data to let py3o know about the image; check our example code:

```python
from py3o.template import Template
t = Template("py3o_example_template.odt", "py3o_example_output.odt")
t.set_image_path('staticimage.logo', 'images/new_logo.png')
```

**Insert images from the data dictionary**

Images can also be injected into the template from the data dictionary. This method should be used in case you have to deal with multiple objects in a for loop, each with their own image. Insert an empty frame as a placeholder (Insert > Frame). Then open its properties and call the py3o.image function in the Name field.
**data (required)** the variable name for the image in the data dictionary.

**mime_type (required)** the image’s file type.

**height (optional)** the desired height for the image with the unit of measure (e.g. ‘1.3cm’).

**width (optional)** The desired width for the image with the unit of measure (e.g. ‘2.55cm’).

**isb64 (optional)** Whether the image data should be interpreted as base64-encoded bytes instead of raw bytes.

**keep_ratio (optional)** Whether the aspect ratio of the image should be kept. If you use keep_ratio=True (which is the default), you should use either the option ‘height’ or ‘width’ (using both doesn’t make sense in this case). In case you give neither ‘height’ nor ‘width’, the image is scaled as to fit into the placeholder frame.

Possible unit of measures: **cm, mm, in, pt, pc, px** and **em** (the OpenDocument format uses the unit of measures defined in the section §5.9.13 of the XSL specifications).

### Use format functions

**Warning:** Format functions are considered to be deprecated. They are meant to be replaced by py3o.types and native ODF formatting capabilities.

Some functions can be called from inside the template in order to format the data. To use a format function, insert a hyperlink as you would to start a loop or condition block:

```
Target: py3o://function="format_function_name(data, format_arguments)"
Text: function="format_function_name(data, format_arguments)"
```

or an input field:
Currency Formatting

```python
format_currency(number, currency, format=None, locale='en_US_POSIX',
currency_digits=True, format_type='standard',
decimal_quantization=True)
```

Relies on babel.numbers.format_currency.


Changes we provide here: * Make the 2nd argument (currency) optional. When not displaying the
  currency symbol, no need to provide a currency.

Their parameter docstring has been copied below.

**number**  the number to format
**currency:** the currency code, optional unless displaying the currency
**format**  the format string to use
**locale**  locale identifier
**currency_digits**  use the currency’s natural number of decimal digits
**format_type**  the currency format type to use
**decimal_quantization**  Truncate and round high-precision numbers to the format pattern. Defaults to True.

Examples used in tests:

```python
function="format_currency(0, format='#')" -> 0
function="format_currency(1, 'USD')" -> $1.00
function="format_currency(42.42, 'EUR')" -> 42.42
function="format_currency(123456789.4242, 'EUR')" -> €123,456,789.42
function="format_currency(123456789.4242, 'EUR', locale='fr_FR')" -> 123 456 789,42 €
```

Date Formatting

```python
format_datetime(date_obj, format)
```

**date_obj**  One of: datetime.date object, datetime.datetime object, ISO formatted string (‘%Y-%m-%d’ or ‘%Y-%m-%d %H:%M:%S’).

**format**  (string) How the date should be formatted. We use babel to format; see <http://babel.pocoo.org/en/latest/dates.html#pattern-syntax>. Optional; when left as is, the default format is: * ‘%Y-%m-%d’ for datetime.date objects. * ‘%Y-%m-%d %H:%M:%S’ for datetime.datetime objects.

Examples used in tests:

```python
function="format_datetime('2015-08-02', format='dd/MM/YYYY')" -> 02/08/2015
function="format_datetime('2015-10-15')"
```

2.5. Use format functions
Example documents

You can find several example templates (ODT and ODS) in our source tree.

Here is a screenshot to show you some control structures (for and if) in action. As you can see you can use these control structures even inside tables:
Contributing to py3o.template

Tests

Tests are easy to run; they rely on tox:

```
$ pip install tox
$ tox # Run all tests
$ tox -e py3 # Run Python 3 tests only
```

Tests reside in `py3o/tests`. To add one:

- Add a test ODT file into `py3o/tests/templates`.
- Add XML expected output into `py3o/tests/templates`. To get it, unzip a generated ODT file and take its `content.xml`.
- Add a `test_*` method into `py3o/tests/test_template.py` that compares these (there are many existing tests that do this).

Code style

We let black (<https://pypi.org/project/black/>) take care of everything here.

Follow its install guide to get it (requires Python 3.6). Alternatively, the <https://hub.docker.com/r/houzefaabbap/ python3-black/> Docker image contains Python 3.6 with black installed inside.
Source code documentation

## Templating

class py3o.template.main.Template(template, outfile, ignore_undefined_variables=False, escape_false=False)

The default template to be used to output ODF content.

apply_variable_type_in_cells(content_trees, namespaces)

Replace default `string` type by a function call.

static convert_py3o_to_python_ast(expressions)

Convert py3o expressions to parsable Python code.

The py3o expressions can be extracted from a Template object using the 
get_all_user_python_expression() method. The result can then be parsed by 
Py3oConverctor in order to obtain the data structure expected by the template.

Parameters expressions(list) – A list of strings that represent expressions in the tem-

plate.

Returns The expressions in the form of Python code that can be parsed by AST.

Return type str

static find_image_frames(content_trees, namespaces)

find all frames that must be converted to images

get_all_user_python_expression()

Public method to get all python expression

get_user_instructions()

Public method to help report engine to find all instructions This method will be removed in a future version. Please use Py3oTemplate.get_all_user_python_expression().

get_user_variables()

a public method to help report engines to introspect a template and find what data it needs and how it will
be used returns a list of user variable names without the leading `py3o.` This method will be removed in a future version. Please use Py3oTemplate.get_all_user_python_expression().

handle_draw_frame(frame, py3o_base)

remove a draw:frame content and inject a draw:image with py:attrs="__py3o_image(image_name)" this 
__py3o_image method will be injected inside the template dictionary to be called back from inside the template
handle_link(link, py3o_base, closing_link)
transform a py3o link into a proper Genshi statement rebase a py3o link at a proper place in the tree to be ready for Genshi replacement

render(data)
render the OpenDocument with the user data
@param data: the input stream of user data. This should be a dictionary mapping, keys being the values accessible to your report. @type data: dictionary

render_flow(data)
render the OpenDocument with the user data
@param data: the input stream of user data. This should be a dictionary mapping, keys being the values accessible to your report. @type data: dictionary

render_tree(data)
prepare the flows without saving to file this method has been decoupled from render_flow to allow better unit testing

set_image_data(identifier, data, mime_type=None)
Set data for an image mentioned in the template.
@param identifier: Identifier of the image; refer to the image in the template by setting “py3o.staticimage.[identifier]” as the name of that image. @type identifier: string
@param data: Contents of the image. @type data: binary

set_image_path(identifier, path)
Set data for an image mentioned in the template.
@param identifier: Identifier of the image; refer to the image in the template by setting “py3o.[identifier]” as the name of that image. @type identifier: string
@param path: Image path on the file system @type path: string

static validate_link(link, py3o_base)
this method will ensure a link is valid or raise a TemplateException

Parameters
link(1xml.Element) – a link node found in the tree

Returns
nothing

Raises
TemplateException

exception py3o.template.main.TemplateException(message)
some client code is used to catching ValueErrors, let’s keep the old codebase happy

class py3o.template.main.TextTemplate(template, outfile, encoding='utf-8', ignore_undefined_variables=False)
A specific template that can be used to output textual content.

It works as the ODT or ODS templates, minus the fact that is does not support images.

render(data)
Render the template with the provided data.

Parameters data – a dictionary containing your data (preferably a iterators): return: Nothing

py3o.template.main.detect_keep_boundary(start, end, namespaces)
a helper to inspect a link and see if we should keep the link boundary

py3o.template.main.format_amount(amount, format='%f')
Replace the thousands separator from '.' to ','

Chapter 4. Source code documentation
**py3o.template.main.format_currency (**args, **kwargs)**
Format the specified amount according to a format string & a currency.

Relies on babel.numbers.format_currency.


Changes we provide here: * Make the 2nd argument (currency) optional. When not displaying the currency symbol, no need to provide a currency.

Their parameter docstring has been copied below.

Parameters

- **number** – the number to format
- **currency** – the currency code, optional unless displaying the currency
- **format** – the format string to use
- **locale** – locale identifier
- **currency_digits** – use the currency’s natural number of decimal digits
- **format_type** – the currency format type to use
- **decimal_quantization** – Truncate and round high-precision numbers to the format pattern. Defaults to True.

Return type  String.

**py3o.template.main.format_date (date, format='%Y-%m-%d')**
Format the date according to format string.

Parameters  date – One of: datetime.date object, datetime.datetime object, ISO formatted string (‘%Y-%m-%d’ or ‘%Y-%m-%d %H:%M:%S’).

**py3o.template.main.format_datetime (date_obj, format=None, locale=None)**
Format the specified date / date-time according to a format string.

Parameters  date – One of: datetime.date object, datetime.datetime object, ISO formatted string (‘%Y-%m-%d’ or ‘%Y-%m-%d %H:%M:%S’).

Parameters  format – How the date should be formatted. We use babel to format; see <http://babel.pocoo.org/en/latest/dates.html#pattern-syntax>. Optional; when left as is, the default format is: * ‘YYYY-MM-dd’ for datetime.date objects. * ‘YYYY-MM-dd HH:mm:ss’ for datetime.datetime objects.

: type format: String.

Parameters  locale (String.) – Locale identifier used during babel formatting. Optional.

 Return type  String.

**py3o.template.main.format_locale (amount, format, locale, grouping=True)**
format the given amount using the format and a locale example: formatLocale(10000.33, ‘%.02f’, ‘fr_FR.UTF-8’) will give you: “10 000,33”

**py3o.template.main.format_multiline (value)**
Allow line breaks in input data with a format function. Escape and replace code originally by tonthon tonthon.

**py3o.template.main.get_all_python_expression (content_trees, namespaces)**
Return all the python expressions found in the whole document

4.1. Templating
**Data extraction**

**AST Conversion**

```python
class py3o.template.helpers.Py3oConvertor
    Provide the data extraction functionality.

    bind_target (iterable, target, context, iterated=True)
        Helper function to the For node. This function fill the context according to the iterable and
        target and return a new_context to pass through the for body

        The new context should contain the for loop declared variable as main key so our children can update
        their content without knowing where they come from.

        Example: python_code = 'for i in list' context = {
            'i': Py3oArray([]), '__py3o_module__': Py3oModule({'list': Py3oArray([])})},
        }
```
In the above example, the two Py3oArray are the same instance. So if we later modif\y the context[‘i’] Py3oArray,

we also modify the context[‘\_\_py3o\_\_module\_\_’][‘list’] one.

**static set\_last\_item** (py3o\_obj, inst)

**Helper function that take a Py3oObject and set the first leaf found** with inst.

This should not be called with a leaf directly.

**visit** (node, local\_context=None)

Call the node-class specific visit function, and propagate the context

**visit\_attribute** (node, local\_context)

Visit our children and return a Py3oDummy equivalent Example:

```
.i.\_egg\_foo -> Py3oDummy{
    ‘i’: Py3oName({ ‘egg’: Py3oName({‘foo’: Py3oName()})
```
}

**visit\_call** (node, local\_context)

Visit a function call.

**visit\_expr** (node, local\_context)

An Expr is the way to express the will of printing a variable in a Py3oTemplate. So here we must update the context to map all attribute access.

We only handle attribute access and simple name (i.\_\_foo or i)

**visit\_for** (node, local\_context)

Update the context so our children have access to the newly declared variable.

**visit\_module** (node, local\_context)

The main node, should be alone. Here we initialize the context and loop for all our children

**visit\_name** (node, local\_context)

Simply return Py3oDummy equivalent

**visit\_str** (node, local\_context)

Do nothing

### Data structure

This file contains all the data structures used by Py3oConvertor See the docstring of Py3oConvertor.__call__() for further information

**class py3o.template.data\_struct\_Py3oArray**

A class representing an iterable value in the data structure. The attribute direct\_access will tell if this class should be considered

as a list of dict or a list of values.

**render** (data)

This function will render the datastruct according to the user’s data

**class py3o.template.data\_struct\_Py3oBuiltin**

This class holds information about builtins
**classmethod from_name** *(name=None)*

Return the Py3oObject subclass for the given built-in name. Return None if the name does not correspond to a builtin.

**Parameters**
- `name` – A Py3oObject instance that represent a name/attribute path

**Returns**
- A Py3oObject subclass or None

```python
class py3o.template.data_struct.Py3oCall(name, dict)
```

This class holds information of function call. ‘name’ holds the name of function as a Py3oName. The keys are the arguments as:

- numeric keys are positional arguments ordered ascendently
- string keys are keywords arguments

```python
class py3o.template.data_struct.Py3oContainer(values)
```

Represent a container defined in the template. This container can be:
- A literal list, tuple, set or dict definition
- A tuple of variables that are the target of an unpack assignment

```python
def get_tuple()
```

Return the container’s values in a tuple

```python
class py3o.template.data_struct.Py3oDummy
```

This class holds temporary dict, or unused attribute such as counters from enumerate()

```python
class py3o.template.data_struct.Py3oEnumerate(name, dict)
```

Represent an enumerate call

```python
class py3o.template.data_struct.Py3oName
```

This class holds information of variables. Keys are attributes and values the type of this attribute

(a another Py3o class or a simple value)

i.e.: `i.egg` -> Py3oName({‘i’: Py3oName({‘egg’: Py3oName(())})})

```python
render(data)
```

This function will render the datastruct according to the user’s data

```python
class py3o.template.data_struct.Py3oObject
```

Base class to be inherited.

```python
get_key()
```

Return the first key

```python
get_size()
```

Return the max depth of the object

```python
get_tuple()
```

Return the value of the Py3oObject as a tuple. As a default behavior, the object returns None.

```python
rget(other)
```

Get the value for the path described by the other Py3oObject.

Recursively checks that the values in other can be found in self.

The method returns the values of self and other at the point where the search stopped. If other is a leaf, the search stops successfully. The method returns True, the value that corresponds to the path described by other, and the leaf in question. If other cannot be found in self, the search stops unsuccessfully. The method returns False, the value that corresponds to the deepest point reached in self, and the rest of the path.

Example: `self = Py3oObject({
    ‘a’: Py3oObject({}, ‘b’: Py3oObject({
```
{'c': Py3oObject({})},
}),
)) other = Py3oObject({

  'b': Py3oObject({  'd': Py3oObject({}),
  }},
}),
)) res = (False, Py3oObject({'c': Py3oObject({})}), # is self['b'] Py3oObject({'d': Py3oObject({})}), # is other['b']
)

) if other["b"] was a leaf, res[0] would be True and res[2] the leaf.

**Returns** A triple: - True if the search was successful, False otherwise - The active sub-element of self when the search stopped - The active sub-element of other when the search stopped

**rupdate**(other)

Update recursively the Py3oObject self with the Py3oObject other. Example: self = Py3oObject({
  'a': Py3oObject({}),  'b': Py3oObject({
    'c': Py3oObject({}),
  }},
}),
)) other = Py3oObject({

  'b': Py3oObject({  'd': Py3oObject({}),
  }},
}),
)) res = Py3oObject({
  'a': Py3oObject({}),  'b': Py3oObject({
    'c': Py3oObject({}),  'd': Py3oObject({}),
  }},
}),
})

}

4.2. Data extraction 23
Python Module Index

p
py3o.template.data_struct, 21
py3o.template.helpers, 20
py3o.template.main, 17
Index

A
apply_variable_type_in_cells()
(py3o.template.main.Template method), 17
get_list_transformer() (in module py3o.template.main), 20
get_size() (py3o.template.data_struct.Py3oObject method), 22
get_tuple() (py3o.template.data_struct.Py3oContainer method), 22
get_tuple() (py3o.template.data_struct.Py3oObject method), 22
get_user_instructions() (py3o.template.main.Template method), 17
get_user_variables() (py3o.template.main.Template method), 17
get_var_corresponding_ods_type() (in module py3o.template.main), 20

B
bind_target() (py3o.template.helpers.Py3oConvertor method), 20

C
convert_py3o_to_python_ast()
(py3o.template.main.Template static method), 17

D
detect_keep_boundary() (in module py3o.template.main), 18

E

F
find_image_frames() (py3o.template.main.Template static method), 17
format_amount() (in module py3o.template.main), 18
format_currency() (in module py3o.template.main), 19
format_date() (in module py3o.template.main), 19
format_datetime() (in module py3o.template.main), 19
format_multiline() (in module py3o.template.main), 19
from_name() (py3o.template.data_struct.Py3oBuiltin class method), 21

G
get_all_python_expression() (in module py3o.template.main), 19
get_all_user_python_expression()
(py3o.template.main.Template method), 17
get_image_frames() (in module py3o.template.main), 19
get_instructions() (in module py3o.template.main), 20
get_key() (py3o.template.data_struct.Py3oObject method), 22

H
handle_draw_frame() (py3o.template.main.Template method), 17
handle_link() (py3o.template.main.Template method), 17

M
move_siblings() (in module py3o.template.main), 20

P
py3o.template.data_struct (module), 21
py3o.template.helpers (module), 20
py3o.template.main (module), 17
Py3oArray (class in py3o.template.data_struct), 21
Py3oBuiltin (class in py3o.template.data_struct), 21
Py3oCall (class in py3o.template.data_struct), 22
Py3oContainer (class in py3o.template.data_struct), 22
Py3oConvertor (class in py3o.template.helpers), 20
Py3oDummy (class in py3o.template.data_struct), 22
Py3oEnumerate (class in py3o.template.data_struct), 22
Py3oName (class in py3o.template.data_struct), 22
Py3oObject (class in py3o.template.data_struct), 22

R
render() (py3o.template.data_struct.Py3oArray method), 21
render() (py3o.template.data_struct.Py3oName method), 22
render() (py3o.template.main.Template method), 18
render() (py3o.template.main.TextTemplate method), 18
render_flow() (py3o.template.main.Template method), 18
render_tree() (py3o.template.main.Template method), 18
rget() (py3o.template.data_struct.Py3oObject method), 22
rupdate() (py3o.template.data_struct.Py3oObject method), 23

S
set_image_data() (py3o.template.main.Template method), 18
set_image_path() (py3o.template.main.Template method), 18
set_last_item() (py3o.template.helpers.Py3oConvertor static method), 21

T
Template (class in py3o.template.main), 17
TemplateException, 18
TextTemplate (class in py3o.template.main), 18

V
validate_link() (py3o.template.main.Template static method), 18
visit() (py3o.template.helpers.Py3oConvertor method), 21
visit_attribute() (py3o.template.helpers.Py3oConvertor method), 21
visit_call() (py3o.template.helpers.Py3oConvertor method), 21
visit_expr() (py3o.template.helpers.Py3oConvertor method), 21
visit_for() (py3o.template.helpers.Py3oConvertor method), 21
visit_module() (py3o.template.helpers.Py3oConvertor method), 21
visit_name() (py3o.template.helpers.Py3oConvertor method), 21
visit_str() (py3o.template.helpers.Py3oConvertor method), 21