Set of script leveraging piecash Documentation

Release 0.0.3

sdementen

November 13, 2016

Contents

Release 0.0.3

Date November 13, 2016

Authors sdementen

Project page https://github.com/sdementen/gnucash-utilities

CHAPTER 1

What’s new

1.1 In development

- system to add python report to gnucash

Contents:

CHAPTER 2

Documentation

This project provides a suite of scripts to work on GnuCash files stored in SQL (sqlite3 and Postgres, not tested in MySQL).

2.1 Report creation (Linux and Windows, python >=3.5)

2.1.1 Installation & use

You first need to install the gnucash-utilities with:

```
$ pip install gnucash-utilities
```

Once installed, you can add python reports to gnucash by adding python files of the form 'report_name-of-report.py' to your $HOME/.gnucash folder.

Everytime a python report is added or the signature of the report function is modified (change of report metadata, addition/change/removal of an option), you should run the gc_report script:

```
For windows
$ gc_report

For linux
$ gc_report.py
```

This script generates the scheme wrapper around the python report (it has the same name as the python report file but with a .scm extension) and register the report in the $HOME/.gnucash/config.user file.

2.1.2 A simple report

The simplest report has the form of

```
from piecash_utilities.report import report, execute_report


@report(
    title="My simplest report",
    name="piecash-simple-report",
    menu_tip="This simple report ever",
    options_default_section="general",
)
```

```
def generate_report(
        book_url,
):
    return "<html><body>Hello world from python !</body></html>"


if __name__ == '__main__':
    execute_report(generate_report)
```

The core reporting logic is defined in the function ‘generate_report’ that:

```
1. is decorated with the 'report' decorator
2. takes one argument 'book_url' which is the book URL
3. takes optional arguments representing the report options
4. returns a string with html. This html is what gnucash will display as the result of the report exe
```

Warning: The report system provided by the gnucash-utilities has currently no way to identify the book that is running in gnucash (this can be fixed if a guile function is able to return the gnucash URI of the currently opened book). Hence, it uses a hack. It will look in the registry (for windows) or dconf (for linux) to find the last opened file and uses this a the “active gnucash book” (ie the ‘book_url’ argument of the ‘generate_report’ function).
This hack will fail a.o. if you work with multiple gnucash book at the same time.

2.1.3 A report with options

If you want to define options for your report, you can do it with type annotations as in

```
from piecash_utilities.report import report, RangeOption, DateOption, StringOption, execute_report


@report(
    title="My simplest report with parameters",
    name="piecash-simple-report-parameters",
    menu_tip="A simple report with parameters",
    options_default_section="general",
)
def generate_report(
        book_url,
        a_number: RangeOption(
            section="main",
            sort_tag="a",
            documentation_string="This is a number",
            default_value=3),
        a_str: StringOption(
            section="main",
            sort_tag="c",
            documentation_string="This is a string",
            default_value="with a default value"),
        a_date: DateOption(
            section="main",
            sort_tag="d",
            documentation_string="This is a date",
            default_value="(lambda () (cons 'absolute (cons (current-time) 0)))"),
        another_number: RangeOption(
            section="main",
            sort_tag="b",
            documentation_string="This is a number",
            default_value=3)
):
    return """<html>
    <body>
        Hello world from python !<br>
        Parameters received:<br>
        <ul>
        <li>a_number = {a_number}</li>
        <li>a_str = {a_str}</li>
        <li>a_date = {a_date}</li>
        <li>another_number = {another_number}</li>
        </ul>
    </body>
    </html>""".format(
        a_str=a_str,
        another_number=another_number,
        a_date=a_date,
        a_number=a_number,
    )


if __name__ == '__main__':
    execute_report(generate_report)
```

Each option is an additional argument to the 'generate_report' function with its type defined through python type annotations.

Options currently supported are:

- date with DateOption
- float with RangeOption
- str with StringOption

2.1.4 A report that access the book

Most of the report will want to access the gnucash book. You can use piecash to open the book thanks to the 'book_url' argument that the 'generate_report' function gets automatically as illustrated in the following example

```
import piecash

from piecash_utilities.report import report, execute_report


@report(
    title="My simplest report with a book",
    name="piecash-simple-report-book",
    menu_tip="A simple report that opens a book",
    options_default_section="general",
)
def generate_report(
        book_url,
):
    with piecash.open_book(book_url, readonly=True, open_if_lock=True) as book:
        return """<html>
        <body>
            Hello world from python !<br>
            Book : {book_url}<br>
            List of accounts : {accounts}
        </body>
        </html>""".format(
            book_url=book_url,
            accounts=[acc.fullname for acc in book.accounts],
        )


if __name__ == '__main__':
    execute_report(generate_report)
```

2.1.5 A full fledged example with jinja2 to generate the html

You can use the command 'gc_create_report name-of-report' (under windows) or 'gc_create_report.py name-of-report' (under linux) to create a set of files 'report_name-of-report.py' and 'report_name-of-report.html' that use the jinja2 templating logic to generate the report. For any moderately complex report, this is the suggested approach.

You can also generate a sample file automatically by executing:

```
For windows
$ gc_report_create name-of-report

For linux
$ gc_report_create.py name-of-report
```

2.1.6 Testing your report from the command line

You can test a report by just running the 'report_name-of-report.py' python file and piping the options to it as:

```
$ cat inputs | python report_name-of-report.py
```

with inputs being a file like

```
a_number|3
a_str|with a default value
a_date|1479026587
another_number|3
```

The inputs should be in line with the options required by the report.

2.1.7 How does it work ?

The python report mechanism works as following:

- At report creation:
    1. user creates a report by writing a python script as $HOME/.gnucash/report_name.py
    2. users launches the gc_report command that:
        (a) generates a scheme wrapper as $HOME/.gnucash/report_name.scm
        (b) adds the report to the file $HOME/.gnucash/config.user to have it loaded at each start of gnucash
- At runtime:

1. gnucash starts, loads $HOME/.gnucash/config.user and registers the report declared in the .scm files
2. user launches a python report
3. the scheme wrapper is called and:
   (a) it starts a python subprocess "python report_name.py"
   (b) it retrieves and serialises each report option in the format "option_name|option_value" and pipes it to the standard input of the python subprocess
   (c) the python subprocesses:
      i. deserialises the options => option arguments
      ii. retrieves the "last open gnucash book" => book_url argument
      iii. calls the generate_report function with the arguments which returns an HTML string
      iv. prints the HTML stringto the standard output
   (d) it retrieves the standard output of the python subprocess as the HTML output of the report

The complete api documentation (apidoc) :

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*