

Criação de relatórios com Stoqlib Reporting

Async Open Source
<http://www.async.com.br>

Dezembro de 2004

1 Introdução

Esse documento cobre a instalação e uso do pacote para geração de relatórios, Stoqlib Reporting. Para um melhor entendimento do mesmo é sugerido que o leitor possua conhecimentos básicos na linguagem de programação Python.

A maioria dos exemplos contidos neste documento tem como objetivo mostrar como determinada operação é feita e não tem a mínima intenção de mostrar casos de uso à qual ela se aplica. Isso cabe ao leitor, usuário do pacote Stoqlib Reporting. Exemplos são disponibilizados no diretório `examples/` para ajudar nesta tomada de decisão. Também disponibilizamos junto à distribuição um manual de referência, você pode encontrá-lo no diretório `docs/api/`.

1.1 O que é?

O módulo Reporting tem como objetivo facilitar o trabalho de criação de relatórios utilizando o **ReportLab**. Contêm rotinas para criação de diferentes tipos de tabela, formatação de texto, estilos de páginas e muito mais.

1.2 Instalação

Os requerimentos para a instalação são:

- Python 2.2 ¹
- eGenix mxDateTime 2.0 ²
- ReportLab 1.19 ³

Desde que se tenha instalado os itens acima e o arquivo `stoqlib-reporting-x.y.tar.gz` já esteja em sua máquina, o processo de instalação é simples e rápido. Basicamente, tudo o que deve ser feito é a descompressão do *tarball*, como em:

```
tar -zxvf stoqlib-reporting-x.y.tar.gz
```

sendo `x.y` a versão do pacote.

E em seguida, a execução do script `setup.py`, encontrado no diretório criado pelo comando acima. Para uma instalação padrão, o comando a ser entrado é:

```
./setup.py install
```

¹<http://www.python.org/download/>

²<http://www.egenix.com/files/python/eGenix-mx-Extensions.html#Download-mxBASE>

³<http://www.reportlab.com/downloads.html>

Isso instalará o pacote no diretório padrão de pacotes do interpretador Python (geralmente `/usr/lib/pythonX.Y/site-packages` em sistemas Linux, `C:/PythonX.Y/site-packages` em sistemas Windows).

O script de instalação permite você especificar o diretório destino de instalação através da flag `--prefix`, como em:

```
./setup.py install --prefix=~/.python/
```

Com a execução deste comando, a hierarquia de diretórios de pacotes do Python é seguida, isto é, a hierarquia `lib/pythonX.Y/site-packages` é mantida, então o diretório construído por este processo de instalação seria, no nosso caso, `python/lib/pythonX.Y/site-packages/`.

Se você optar por esse modo de instalação, você também precisará definir o diretório destino de instalação na variável ambiente `PYTHONPATH` para que o interpretador possa saber onde encontrar o pacote recém instalado.

Geralmente, o que deve ser feito é:

```
export PYTHONPATH=$PYTHONPATH: ~/.python/lib/pythonX.Y/site-packages
```

Note que usamos `$PYTHONPATH` antes de especificar o nosso diretório, isso é para poder manter a configuração atual. Note também que precisamos especificar o diretório completo onde os pacotes podem ser encontrados.

Essa configuração é temporária e será descartada quando o sistema for reiniciado; então para fazer com que esta configuração se torne permanente, é recomendado que o trecho mostrado acima seja colocado no arquivo `.bash_profile` do seu diretório `HOME`.

Após a execução do script de instalação, você pode testar se tudo está funcionando abrindo o interpretador Python e importando o módulo recém instalado, desta forma:

```
from stoqlib import reporting
```

Se nenhuma mensagem aparecer, a instalação foi executada com sucesso e você pode prosseguir com a leitura deste documento ou então consultar alguns exemplos no diretório *examples*.

Caso haja algum erro ao importar o módulo, leia novamente esta seção, refaça as instruções com mais atenção e tente novamente importar o módulo. Se for realmente um erro, você pode reportá-lo em bugs.async.com.br.

2 Iniciando o uso

Nessa seção mostrarei como relatórios podem ser criados, visualizados e impressos. Essa seção trata exclusivamente da geração básica do relatório, ou seja, tudo o que deve ser feito para que o relatório possa ser gerado com sucesso. Não se preocupe se não entender alguma coisa, seções posteriores existem justamente para explicar o que possa causar confusão à uma primeira vista.

2.1 Herança de ReportTemplate

Para criarmos um relatório, criamos uma classe. Essa classe pode ser vista como o relatório em si, pois é ela quem dita como a página do relatório deve ser, como a tabela será construída, quais colunas uma tabela deve ter, onde as linhas desta tabela podem ser encontradas, e assim por diante.

O pacote Reporting fornece as classes `ReportTemplate` e `BaseReportTemplate`, onde todos os métodos para inserção de elementos no relatório estão definidos. A classe `ReportTemplate` define atributos do relatório como tamanho do cabeçalho, tamanho do rodapé e o método para desenho do mesmo, ao passo que em `BaseReportTemplate` (módulo `template`) estão definidos (e parcialmente implementados) todos os métodos para inserção de elementos que o pacote oferece. A classe `ReportTemplate` herda de `BaseReportTemplate`, logo, sua classe deve herdar de `ReportTemplate`.

```

from stoqlib.reporting import ReportTemplate

class MyReportClass(ReportTemplate):
    def __init__(self, filename):
        ReportTemplate.__init__(self, filename, "report_name")

```

Repare que tanto o construtor da classe quanto o construtor de `ReportTemplate` recebem um nome de arquivo, ele é passado pela função `build_report` e define onde o relatório vai ser desenhado. `ReportTemplate` também requer um parâmetro obrigatório chamado `report_name`, que, basicamente, define o nome do relatório. O nome do relatório é informado no rodapé da página, junto com a data e o número da página por padrão.

2.2 Tipos de páginas e Margens

Podemos definir o tamanho de papel à utilizar no relatório através do parâmetro `pagesize` de `ReportTemplate`. Alguns tamanho são: A0, A1, A2, A3, A4, A5 e A6. Para uma listagem completa de todos os tipos/tamanhos de papéis suportados, consulte `reportlab.lib.pagesizes`. Caso o parâmetro `pagesize` não receba um argumento, o tamanho padrão, A4, é utilizado. É possível também utilizar o parâmetro `landscape` para especificar se o relatório deve ser gerado no formato "paisagem". O padrão é o formato "retrato" (`landscape=0`).

Quanto às margens, elas podem ser ajustadas através dos parâmetros `topMargin`, `rightMargin`, `leftMargin` e `bottomMargin`.

2.3 Construindo, Visualizando e Imprimindo

Uma vez que se tenha a classe criada, a construção do relatório é feita por meio da função `build_report`:

```

from stoqlib.reporting import build_report

reportfilename = build_report(ReportClass)

```

Como você pode ver, a função `build_report` recebe a classe do relatório e retorna o nome do arquivo onde o relatório foi construído. Como dito anteriormente, esse mesmo nome de arquivo é passado para o construtor de sua classe e, de lá, para o construtor de `ReportTemplate`. Tendo o relatório pronto e salvo no arquivo retornado, você pode visualizá-lo através da função `print_preview` ou imprimí-lo através da função `print_file`.

A função `print_file` aceita os argumentos `printer` e `extra_opts`, que são, respectivamente, o nome da impressora à ser utilizada e opções extras que devem ser consideradas na construção do comando de impressão. Esses argumentos são opcionais e, caso não sejam especificados, impressora e comandos padrões serão utilizados.

3 Parágrafos e estilos de parágrafos

Parágrafos podem ser inseridos em seu relatório pelo método `add_paragraph`. O uso de estilos é possível e o pacote `Reporting` possui vários deles definidos em `stoqlib.reporting.default_style`.

Para inserção de um simples parágrafo, utilizando a formatação padrão, você faz:

```

class ReportClass(ReportTemplate):
    ...
    self.add_paragraph("Um simples parágrafo")

```

Para usar estilos do módulo *default_style* usa-se o parâmetro `style`, como mostrado no exemplo abaixo:

```
self.add_paragraph("Título alinhado ao centro", style="Title-AlignCenter")
```

É interessante notar que o método *add_paragraph*, para facilitar o trabalho, somente aceita **nomes** de estilos já definidos na instância `STYLE_SHEET` de `ReportLab` (que seriam os estilos padrões definidos pelo `ReportLab` e os definidos no módulo `default_style`). Você pode criar um módulo próprio com os estilos que você precisa, para isto você teria uma ou várias instâncias do tipo `ParagraphStyle` e as adicionaria ao `STYLE_SHEET` através do método `add`, um bom exemplo de como fazer isso pode ser visto no código-exemplo `examples/contract_example.py`.

`Reportlab` também permite que você utilize certas tags *HTML* como `` e `` para delimitar textos que devem ser escritos em negrito, `<i>` e `</i>` para textos em itálico e `<u>`-`</u>` para sublinhado (em algumas instalações do `ReportLab` esta tag pode não funcionar, consulte documentação relevante para maiores informações).

4 Tabelas

Em `Stoqlib Reporting` existem 4 tipos de tabelas: Tabelas Simples, Tabelas Relatório, Tabelas Coluna e Tabelas Objeto. Cada uma destas será discutida nas subseções a seguir.

4.1 Tabelas Simples

Esse é o tipo de tabela mais simples e é utilizado em situações onde precisamos somente exibir linhas no formato Campo-Valor (nesse tipo de tabela a coluna campo sempre é destacada). Geralmente, situações como esta ocorrem quando precisamos exibir informações de cadastro e, em alguns momentos, quando precisamos inserir parágrafos em sequência, como é o caso do exemplo *examples/contract_example*. Partes relevantes deste exemplo são mostradas abaixo:

```
def add_info_table(self):
    rows = [{"Concedentes:", ""},
            ["Endereço:", ""],
            ["Estagiário:", ""],
            ["Instituição de Ensino:", ""],
            ["Endereço:", ""],
            ["Nível:", ""],
            ["Curso:", ""]]
    self.add_data_table(rows)
```

É interessante observar o atributo `rows`, ele contém uma lista de listas. Cada lista interna representa uma linha e cada item dela uma coluna. Esse conceito é utilizado em todas as funções para inserção de tabela.

Neste código temos que todas as colunas **pares** das linhas serão destacadas, é obrigatório para isto que cada linha contenha mais de uma coluna (isto justifica o por que da string vazia de cada lista interna). No exemplo *contract_example*, preferi deixar a segunda coluna de cada linha vazia, para permitir ao usuário do relatório preenchê-la "manualmente", mas pode ser desejo do usuário fazer com que, dado as informações, o relatório já seja preenchido com os dados, ficando somente os campos de assinaturas à serem preenchidos. Neste caso, uma classe com os devidos dados pode ser passada ao método, e, dentro dele, preencher os devidos campos da lista.

Tabelas simples são simples e não nos permite ir muito longe quando o assunto é formatação, nas próximas seções será discutido tabelas que permitem borda, estilo zebreado e linhas extra.

4.2 Tabelas Relatório

Tabela Relatório é o tipo de tabela mais básico do pacote para uso *real* em relatórios, esse tipo de tabela trabalha igual às tabelas simples, isto é, você precisa somente ter uma lista de listas para a criação das linhas. Você também pode ter uma lista extra para criação de um cabeçalho (definido através do parâmetro `header`). A formatação padrão de Tabelas Relatório incluem borda e estilo zebrado nas linhas.

```
from stoqlib.reporting import print_preview, build_report, ReportTemplate
from stoqlib.reporting.common import read_file, safe_int

class VehiclesProductionReport(ReportTemplate):
    def __init__(self, filename, **args):
        report_name = 'Vehicles Production Report'
        ReportTemplate.__init__(self, filename, report_name, timestamp=1,
                                leftMargin=30, topMargin=20,
                                rightMargin=30, do_header=0)

        rows = self.get_rows()
        self.run_report(rows)

    def run_report(self, production_list):
        header = ['Id', 'Description', 'Type', 'Control', 'Start M.',
                  'End M.', 'Total', 'Difference', 'Price',
                  'Total Price']
        self.add_report_table(production_list, header=header)
        self.add_paragraph('%d itens listed' % len(production_list),
                            style='Normal-AlignRight')

    def get_rows(self):
        production = []
        csv_file = 'csv/vehicles_production.txt'
        for data in read_file(csv_file):
            id, description, type, measurement, start_measurement,
            end_measurement, total, difference, price, total_value = data[:10]

            columns = [id, description, type, measurement, start_measurement,
                       end_measurement, total, difference, price,
                       total_value]

            production.append(columns)

        production.sort()
        return production

report_filename = build_report(VehiclesProductionReport)
print_preview(report_filename)
```

Conforme você pode ver no exemplo acima, o método `run_report` recebe uma lista de linhas (criada por uma chamada prévia ao método `get_rows()`), cria o cabeçalho (`header`) da tabela, cria a tabela e insere um parágrafo resumo.

Tabelas Relatório também permitem o uso de linhas extra, tudo o que deve ser feito é a utilização do parâmetro `extra_row`, como em:

```
...
extra_row = ("", "", "Total:", format(total_value))
```

```
self.add_report_table(rows, extra_row=extra_row)
```

É claro, você deve obedecer à ordem das colunas, isto é, a linha extra deve conter o mesmo número de colunas que as outras linhas da tabela, embora nem todas precisem estar preenchidas, como é mostrado no trecho acima.

Você também pode retirar tanto o estilo zebrado quanto a borda da tabela através dos parâmetros `highlight` e `style`, respectivamente:

```
from stoqlib.reporting.tables import HIGHLIGHT_NEVER, TABLE_LINE_BLANK

self.add_report_table(self.get_rows(), style=TABLE_LINE_BLANK,
                      highlight=HIGHLIGHT_NEVER)
```

4.3 Tabelas Coluna

4.3.1 Introdução

Nos tipos de tabelas apresentados anteriormente, precisávamos passar para a função somente as linhas da tabela (uma lista de listas, como explicado anteriormente). Nesse caso, colunas são criadas automaticamente (cada item de uma lista interna representa uma coluna, cada lista interna, por sua vez, representa uma linha) e, quando precisam ser nomeadas, uma linha adicional, chamada de cabeçalho, é criada.

Tabelas Coluna requerem que colunas sejam criadas e inseridas em uma lista, com cada elemento da lista representando uma instância do tipo de coluna a ser criado. Isso é utilizado para permitir formatação de colunas, fornecendo suporte à, por exemplo, "truncamento" de textos em colunas e alinhamento diferente para colunas de uma mesma linha.

4.3.2 Criando colunas

Como dito anteriormente, uma lista de colunas deve ser criada. Cada item da lista será uma instância de *TableColumn* (um tipo de coluna), isso nos permitirá definir estilos próprios para cada coluna e, com isso, conseguir uma boa formatação da tabela.

```
from stoqlib.reporting.tables import TableColumn as TC
...

def get_cols(self):
    return [TC("Cód.", width=80),
            TC("Descrição", width=300),
            TC("Valor", width=100)]
..
```

Os parâmetros para a classe *TableColumn* (no trecho acima referenciada simplesmente como TC) são explicados abaixo:

- *name*: Equivale ao nome da coluna, isto é, a *string* que será inserida no cabeçalho da tabela para representar a coluna.
- *width*: Comprimento da coluna
- *truncate*: Definido como True para permitir que colunas sejam truncadas, caso muito útil em colunas que "guardam" textos.

4.3.3 Criando linhas e adicionando tudo na tabela

Com tudo o que foi explicado, podemos criar um exemplo teste:

```
from stoqlib.reporting import print_preview, build_report, ReportTemplate
from stoqlib.reporting.tables import TableColumn as TC, RIGHT

class TableColumnTest(ReportTemplate):
    def __init__(self, filename):
        reportname = "Simples Teste com Tabelas Coluna"
        ReportTemplate.__init__(self, filename, reportname)
        self.add_column_table(self.get_rows(), self.get_cols())

    def get_cols(self):
        return [TC("Código", width=80, align=RIGHT),
                TC("Descrição", width=300),
                TC("Valor (R$)", width=100, align=RIGHT)]

    def get_rows(self):
        return [
            ["001", "Monitor 15'' LG 4723", "432,89"],
            ["002", "Mesa para escritório", "624,53"]
        ]

report_file = build_report(TableColumnTest)
print_preview(report_file)
```

4.4 Tabelas Objeto

O último tipo de tabela é a Tabela Objeto, altamente recomendada em situações onde precisamos gerar relatórios baseado em uma lista de instâncias. Essa lista de instâncias pode ser resultado de uma busca em uma base de dados ou de uma importação de dados.

Tabelas Objeto possui algumas características de Tabelas Coluna, dentre elas a necessidade de ter uma lista de colunas:

```
from stoqlib.reporting.tables import ObjectTableColumn as OTC, RIGHT
[...]

def get_cols(self):
    return [OTC("CÓD.", align=RIGHT, width=90),
            OTC("Descrição", width=300),
            OTC("Valor", width=100)
    ]

[...]
```

Quando citei que tabelas objeto são altamente recomendada em situações onde temos uma lista de instâncias, me referia ao seguinte fato: Tabelas objeto não precisam de lista de linhas. Ao criar a lista de colunas, passamos como segundo parâmetro de OTC o nome de uma função que recebe um objeto (item da lista de instâncias) e retorna um valor (valor do atributo correspondente à coluna), então, reformulando nosso método *get_cols* teríamos:

```
def get_cols(self):
    return [OTC("CÓD.", lambda obj: obj.id, align=RIGHT, width=90),
            OTC("Descrição", lambda obj: obj.desc, width=300),
            OTC("Valor", lambda obj: obj.value, width=100)
    ]
```

No exemplo a seguir, crio uma classe *Client* e em seguida uma lista de instâncias desta classe com a intenção de representar uma lista de instâncias retornadas por uma pesquisa em uma base de dados:

```
from stoqlib.reporting import print_preview, build_report, ReportTemplate
from stoqlib.reporting.tables import ObjectTableColumn as OTC

class Client:
    def __init__(self, id, name):
        self.id, self.name = (id, name)

class ObjectTableColumnTest(ReportTemplate):
    def __init__(self, filename, clients):
        report_name = "Simples teste com ObjectTableColumn"
        ReportTemplate.__init__(self, filename, report_name)
        self.add_title("Relatório de Clientes")
        self.add_object_table(clients, self.get_cols())

    def get_cols(self):
        return [OTC("Cód", lambda o: "%04d" % o.id, width=80, align=RIGHT),
                OTC("Nome", lambda o: o.name, width=400)
                ]

client_list = []
for i in range(20):
    client_list.append(Client(i, "Nome do Cliente #%d" % i))

report_file = build_report(ObjectTableColumn, client_list)
print_preview(report_file)
```

No diretório *examples/* há um exemplo melhor de como utilizar *ObjectTableColumn*. No exemplo, nomeado *purchase_order.py* temos uma lista de clientes armazenada em um arquivo CSV, então importamos essa lista para dentro de uma lista de instâncias e geramos o relatório. É uma boa tática para geração de relatório, mas você ficará realmente surpreso com a facilidade de criação de relatório quando começar a escrever programas que façam a interação entre uma base de dados e *ObjectTableColumn*

5 Títulos, Linhas e Assinaturas

Nenhum pacote para geração de relatórios estaria completo senão fornecesse rotinas para inserção de linhas e assinaturas, por isso *BaseReportTemplate* de *Stoqlib Reporting* fornece os métodos: *add_title*, *add_line* e *add_signatures*.

5.1 *add_title*

O método *add_title* já foi utilizado em exemplos anteriores neste documento e você já deve ter noção de seu funcionamento; simplesmente chame o método passando um texto e você terá um título, como em:

```
self.add_title("Seção 25: Formas de Uso")
```

O método *add_title* permitir ainda a inserção de "notas", para isso o parâmetro *note* pode ser utilizado:

```
self.add_title("Seção 25: Formas de Uso",
               note="Leitura altamente recomendada")
```

5.2 *add_line*

Linhas, às vezes, podem ser uma opção à títulos. Algumas vezes preferimos separar textos sem precisarmos de uma nova seção (que seria representada por um título), para isso temos:

```
...
self.add_paragraph(...)
self.add_line()
self.add_paragraph(...)
...
```

Os parâmetros para o método `add_line()` são:

- `thickness`: espessura da linha;
- `v_margins`: margens verticais antes e após a linha;
- `h_margins`: margens horizontais antes e após a linha.

Para maiores detalhes, consulte a documentação API incluída com a distribuição.

5.3 *add_signatures*

São várias as situações onde um relatório precisa de um campo para assinatura. O método `add_signatures` permite a inserção de assinaturas alinhadas e com múltiplas linhas para o texto de "Nome" (o texto que é inserido embaixo da linha). Em sua forma mais simples, o método `add_signatures` pode ser usado assim:

```
sig = ["Fernando Henrique Gonçalves\nGerente"]
self.add_signatures(sig)
```

Como você pode perceber, as assinaturas são divididas dentro de uma lista, cada item da lista corresponde à uma assinatura, sendo que cada item pode possuir mais de uma linha através do caractere *new line*.

Assinaturas também podem ser alinhadas através do parâmetro `align`. No exemplo abaixo foram colocadas e alteradas partes de um contrato com o objetivo de ilustrar o funcionamento dos métodos apresentados nesta seção:

```
from stoqlib.reporting import print_preview, build_report, ReportTemplate

class SignatureTest(ReportTemplate):
    def __init__(self, filename):
        ReportTemplate.__init__(self, filename, "", do_footer=0)
        title = "<b>TERMO DE COMPROMISSO DE ESTÁGIO</b>"
        self.add_title(title)
        self.add_blank_space(30)
        self.add_paragraph("Corpo do Contrato", style="Title-AlignCenter")
        text = ("Por estarem assim justos e contratados, firmam o presente "
              "instrumento, em duas vias de igual teor, juntamente com 2 "
              "testemunhas.")

        self.add_blank_space(30)
        self.add_paragraph(text)
        self.add_blank_space(20)
        date = "São Paulo, 10 de Dezembro de 2004"
        self.add_paragraph(date)
```

```
self.add_signatures(["Estagiário"])
self.add_signatures(["Testemunha 1", "Testemunha 2"])

report_file = build_report(SignatureTest)
print_preview(report_file)
```

Há também a possibilidade do alinhamento do texto de assinatura, isto é, o texto abaixo da linha, através do parâmetro `text_align`. Sobrescrevendo parte do exemplo anterior, teríamos:

```
from stoqlib.reporting.flowables import RIGHT
...
self.add_signatures(["Estagiário"], text_align=RIGHT)
self.add_signatures(["Testemunha 1", "Testemunha 2"],
                    text_align=RIGHT)
...
```

6 Documentação Relevante

Referências de documentações complementares são o manual Python, disponibilizado em <http://docs.python.org>, o guia do usuário ReportLab, disponível em http://www.reportlab.org/os_documentation.html e a documentação API Stoqlib Reporting, distribuída junto com o pacote. A documentação ReportLab é realmente recomendada para o trabalho com o Stoqlib Reporting e obrigatória para implementação de novas funcionalidades no pacote (como a inserção de novos elementos, por exemplo).