Geraldo Reports Documentation Documentation

Release 0.4-final

Marinho Brandao

February 19, 2016

Contents

Geraldo is a reports engine with its own API, created in Python language to output PDFs and other formats for complex reports.

Its workflow is the following:

1. You declare a Report class with your report structure and definitions;
2. Instantiate your Report class, providing a queryset with a list of objects;
3. Call report's 'generate_by' method, providing the generator class and filename output (can be a filename or file output). It will generate a file in hard disk or in memory;
4. Do what you want with the output: you can return it in an HttpResponse, keep it in a file, etc.

CHAPTER 1

Content

1.1 Installing

Geraldo is still not available as an installer or system package. So, you have two ways to install it:

1.1.1 Option 1. Get the latest official version

1. Get the latest tarball available from:

   https://sourceforge.net/projects/geraldo/files/geraldo/

2. Then you can do the following steps to install it:

Linux

```
tar xvfz Geraldo-latest.tar.gz
cd Geraldo
sudo python setup.py install
```

This will copy Geraldo's packages inside Python's packages path.

Windows

If you are using Windows, Geraldo is fully compatible. You just have to use a tarball decompression tool, like **7-Zip** which you can download from:

http://www.7-zip.org/download.html

Once you have a tool to decompress tarball files, you have to decompress it, open a DOS Prompt window in the folder you decompress into and run the following command:

```
python setup.py install
```

1.1.2 Option 2. Get the latest development version

Our version control is centralized on GitHub Social repository, in the following URL:

```
http://github.com/marinho/geraldo
```

To get the latest development version from repository server, you will need the Git version control system. Then you can run the following command in Shell or MS-DOS Prompty window:

```
git clone git://github.com/marinho/geraldo.git
```

A folder named **geraldo** will be in your current path with all Geraldo files inside.

Then you can go ahead and use the same commands from **Option 1** to install in your system.

1.1.3 Dependencies

ReportLab

This is required, as you cannot use Geraldo without ReportLab installed.

Get it from the following URL:

http://www.reportlab.org/

The tested version is **2.1**.

Python Imaging Library

Geraldo depends on PIL to work with Images manipulation on the canvas. If you aren't going to use them, you can live without PIL.

To get PIL you can go to the following URL:

http://www.pythonware.com/products/pil/

1.2 Tutorial to say Hello

Once you have Geraldo installed and its dependencies resolved (read the Installation document to know about the dependencies on ReportLab and Python Imaging Library), you must keep in mind Geraldo Reports isn't dependent of any framework, neither a web or desktop paradigm, that means Geraldo just make reports and you do the rest by your favorite way.

Supposing we are working without an ORM or a persistency layer, nor an objects-oriented list with objects, the most simple and probably way we are using is a list of dictionaries, like below:

```
family = [
    {'name': 'Leticia', 'age': 29, 'weight': 55.7, 'genre': 'female', 'status': 'parent'},
    {'name': 'Marinho', 'age': 28, 'weight': 76, 'genre': 'male', 'status': 'parent'},
    {'name': 'Tarsila', 'age': 4, 'weight': 16.2, 'genre': 'female', 'status': 'child'},
    {'name': 'Linus', 'age': 0, 'weight': 1.5, 'genre': 'male', 'status': 'child'},
    {'name': 'Mychelle', 'age': 19, 'weight': 50, 'genre': 'female', 'status': 'nephew'},
    {'name': 'Mychell', 'age': 17, 'weight': 55, 'genre': 'male', 'status': 'niece'},
]
```

The dictionaries have common keys, like 'name' and 'age' that we want to show in a report, so, let's do that!

Open a new file with the dictionaries list above and the imports below:

```
from geraldo import Report, ReportBand, DetailBand, SystemField, Label, ObjectValue
from geraldo.utils import cm
```

Actually, we could sum up the imported elements with short descriptions below:

- Report - the main class of a report, that means you must start everything from this

class if you want to make a report.

- ReportBand - each imaginary slice of a report is an instance of a band, if you are

used to HTML, you could see bands like divs.

- DetailBand - is the especialized band class used as detail band of a report. The

detail band is the only one required and everything is "driven" by it. Each object of a given list will have an instance of this band showing in the report.

- SystemField - is the widget used to show things like the current page number, the

current date, the report title, etc.

- Label - is the widget used to show free text in a report.
- ObjectValue - is the widget used to show object values and expressions.
- cm - is the unit representing the metric "centimeter" you can use to define dimensions

and positions on the report.

Well, once we have the basic things clear, we can declare or first report class:

```
class MyFamilyReport(Report):
    class band_detail(DetailBand):
        height = 0.7*cm
        elements = [
            Label(text='Name'),
            ObjectValue(expression='name', left=1.5*cm),
        ]
        borders = {'bottom': True}
```

The class above is enough to show a basic "hello world" example. The attribute 'band_detail' could be either a class or an instance, whatever, but the important is that it must be something based on a ReportBand, or better, a DetailBand.

Now, we want to generate the PDF file of this, of course. But, considering we could generate in TXT or others formats, there is a way to generate PDF files, the **PDFGenerator**, like below:

```
from geraldo.generators import PDFGenerator

my_report = MyFamilyReport(queryset=family)
my_report.generate_by(PDFGenerator, filename='family.pdf')
```

Save and run the python script.

Good. Working. But as you know, this isn't enough to write a good report, so, let's add something to improve our report a little. Change the report class to be like below:

```
class MyFamilyReport(Report):
    title = 'My Family'

    class band_detail(DetailBand):
        height = 0.7*cm
        elements = [
            ObjectValue(expression='name', left=0.5*cm),
            ObjectValue(expression='age', left=5*cm),
            ObjectValue(expression='weight', left=6.5*cm),
        ]
        borders = {'bottom': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
```

```
            SystemField(expression=u'Page %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            Label(text="Name", top=0.8*cm, left=0.5*cm),
            Label(text="Age", top=0.8*cm, left=5*cm),
            Label(text="Weight", top=0.8*cm, left=6.5*cm),
        ]
        borders = {'all': True}
```

Change also the imports to be like this:

```
from geraldo import Report, ReportBand, DetailBand, SystemField, Label, ObjectValue
from geraldo.utils import cm, BAND_WIDTH, TA_CENTER, TA_RIGHT
```

Save and run the python script.

Wow! Now it's better, but, what about the page footer? And totals? Ok... let's go...

Change the class to be like below:

```
class MyFamilyReport(Report):
    title = 'My Family'

    class band_detail(DetailBand):
        height = 0.7*cm
        elements = [
            ObjectValue(expression='name', left=0.5*cm),
            ObjectValue(expression='age', left=5*cm),
            ObjectValue(expression='weight', left=6.5*cm),
        ]
        borders = {'bottom': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            SystemField(expression=u'Page %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            Label(text="Name", top=0.8*cm, left=0.5*cm),
            Label(text="Age", top=0.8*cm, left=5*cm),
            Label(text="Weight", top=0.8*cm, left=6.5*cm),
        ]
        borders = {'all': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Geraldo Reports', top=0.1*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            ]
        borders = {'top': True}

    class band_summary(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Totals:'),
            ObjectValue(expression='avg(age)', left=5*cm),
            ObjectValue(expression='sum(weight)', left=6.5*cm),
```

```
            ]
        borders = {'top': True}
```

Save and run the python script.

Gut.

Now, let's group those people by their genre and status...

```
class MyFamilyReport(Report):
    title = 'My Family'

    class band_detail(DetailBand):
        height = 0.7*cm
        elements = [
            ObjectValue(expression='name', left=0.5*cm),
            ObjectValue(expression='age', left=5*cm),
            ObjectValue(expression='weight', left=6.5*cm),
        ]
        borders = {'bottom': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            SystemField(expression=u'Page %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            Label(text="Name", top=0.8*cm, left=0.5*cm),
            Label(text="Age", top=0.8*cm, left=5*cm),
            Label(text="Weight", top=0.8*cm, left=6.5*cm),
        ]
        borders = {'all': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Geraldo Reports', top=0.1*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            ]
        borders = {'top': True}

    class band_summary(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Totals:'),
            ObjectValue(expression='avg(age)', left=5*cm, style={'fontName': 'Helvetica-Bold'}),
            ObjectValue(expression='sum(weight)', left=6.5*cm, style={'fontName': 'Helvetica-Bold'}),
            ]
        borders = {'top': True}

    groups = [
        ReportGroup(
            attribute_name='genre',
            band_header=DetailBand(
                height=0.6*cm,
                elements=[
                    ObjectValue(expression='genre', style={'fontSize': 12})
```

```
                ]
            ),
            band_footer=ReportBand(
                height = 0.5*cm,
                elements = [
                    ObjectValue(expression='avg(age)', left=5*cm),
                    ObjectValue(expression='sum(weight)', left=6.5*cm),
                    ],
                borders = {'top': True},
            ),
        ),
        ReportGroup(
            attribute_name='status',
            band_header=DetailBand(
                height=0.6*cm,
                elements=[
                    ObjectValue(expression='status', style={'fontSize': 11}, left=0.2*cm)
                ]
            )
        ),
    ]
```

And change also the imports to be like this:

```
from geraldo import Report, ReportBand, DetailBand, SystemField, Label, ObjectValue, ReportGroup
from geraldo.utils import cm, BAND_WIDTH, TA_CENTER, TA_RIGHT
```

Before save, you must sort list by genre and status, otherwise informations will be a little confuse. So, change the data lines to be like below:

```
family = [
    {'name': 'Leticia', 'age': 29, 'weight': 55.7, 'genre': 'female', 'status': 'parent'},
    {'name': 'Marinho', 'age': 28, 'weight': 76, 'genre': 'male', 'status': 'parent'},
    {'name': 'Tarsila', 'age': 4, 'weight': 16.2, 'genre': 'female', 'status': 'child'},
    {'name': 'Linus', 'age': 0, 'weight': 1.5, 'genre': 'male', 'status': 'child'},
    {'name': 'Mychelle', 'age': 19, 'weight': 50, 'genre': 'female', 'status': 'nephew'},
    {'name': 'Mychell', 'age': 17, 'weight': 55, 'genre': 'male', 'status': 'niece'},
]
family.sort(lambda a,b: cmp(a['genre'], b['genre']) or cmp(a['status'], b['status']))
```

Save and run the python script.

Did you like? Well... this is a good start. Read the rest of **docs** and **examples** and good coding!

1.3 Tutorial using Django

Once you have Geraldo installed and its dependencies resolved (read the Installation document to know about the dependencies on ReportLab and Python Imaging Library), you can start opening the **settings.py** from your project to edit.

> **Attention:** this tutorial assumes you are going to use Geraldo with a Django project, but in fact you can just ignore the Django parts and all the rest is exactly the same for Django and non-Django cases.

1.3.1 Step 1. Preparing your project

To install Geraldo in your Django project, you must change the setting **INSTALLED_APPS** in your **settings.py** file and this should be enough.

But let's create a new URL to load a PDF report online.

> **Attention:** Geraldo can work with online or file ways to get a PDF. You are not dependent on a URL or view to generate reports from Geraldo. This is just one of many ways you can do it.

We are going to work with a common example for this kind of solution: a purchasing application.

Let's say you have the following model classes:

```
class Product(models.Model):
    name = models.CharField(max_length=100)

class Customer(models.Model):
    name = models.CharField(max_length=100)

class Purchase(models.Model):
    customer = models.ForeignKey('Customer')
    delivery_address = models.CharField(max_length=70, blank=True)
    date_creation = models.DateField(blank=True, default=date.today())
    date_bill = models.DateField(blank=True, null=True)
    date_delivery = models.DateField(blank=True, null=True)
    taxes = models.DecimalField(max_digits=12, decimal_places=2, blank=True, default=0)
    sub_total_price = models.DecimalField(max_digits=12, decimal_places=2, blank=True, default=0)
    total_price = models.DecimalField(max_digits=12, decimal_places=2, blank=True, default=0)

class PurchaseItem(models.Model):
    purchase = models.ForeignKey('Purchase')
    product = models.ForeignKey('Product')
    unit_price = models.DecimalField(max_digits=12, decimal_places=2)
    quantity = models.DecimalField(max_digits=12, decimal_places=2)
    total_price = models.DecimalField(max_digits=12, decimal_places=2, blank=True, default=0)
```

First you declare a new URL, for example:

```
urlpatterns = patterns('purchases.views',
    url('^purchase-report/$', 'purchase_report'),
)
```

The next step is you have a *view* for the created URL, like this:

```
from django.http import HttpResponse

from reports import ReportPurchase
from geraldo.generators import PDFGenerator
from models import Purchase

def purchase_report(request):
    resp = HttpResponse(mimetype='application/pdf')

    purchases = Purchase.objects.order_by('customer', 'id')
    report = ReportPurchase(queryset=purchases)
    report.generate_by(PDFGenerator, filename=resp)

    return resp
```

You can see you first imported two important things:

```
from reports import ReportPurchase
from geraldo.generators import PDFGenerator
```

Your report class (we will create it in the next step) and the generator class **PDFGenerator** you will need to generate a PDF file. In the future we will have other generators, for other formats of file.

The next thing to note is that you are going to return a PDF format **HttpResponse**:

```
resp = HttpResponse(mimetype='application/pdf')
```

Now you are going to work your report instance:

```
purchases = Purchase.objects.order_by('customer', 'id')
report = ReportPurchase(queryset=purchases)
report.generate_by(PDFGenerator, filename=resp)
```

- The first thing you did is to get all purchases, ordered by 'customer' and 'id' fields;
- Afterwards, you got a **report** instance from your report class **ReportPurchase**, using purchases queryset as report driver queryset;
- And last, you called the 'generate_by' method of the generator class.

Now you have a prepared Django application to use a report you are going to create at the next step!

1.3.2 Declaring the report class

The report class must be an inheritance from **geraldo.Report** class. You can create a new file 'reports.py' to declare your report classes inside.

```
from geraldo import Report

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'
```

To see something more complex, you can set the page size, margins and other report attributes, like below:

```
from geraldo import Report, landscape
from reportlab.lib.pagesizes import A5
from reportlab.lib.units import cm

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'

    page_size = landscape(A5)
    margin_left = 2*cm
    margin_top = 0.5*cm
    margin_right = 0.5*cm
    margin_bottom = 0.5*cm
```

As you can see, we use what we can from the ReportLab libraries, their units, page sizes, stylizing, etc.

A report is driven by **bands**. If you are used to working with other report engines you know what a band is.

A band is a row with elements in the report canvas. Bands in essence are the same but their use varies depending whether you are using a band as the Page Header or Report Summary, for example.

A report can have a band for each one of following attributes:

1.3.3 Detail band

The detail band is the most important band in a report. This is because the detail band is the reason of the existence of every report.

The detail band is used most of the time to show object field values. This is the same for a detailed info page or just a simple list or grid.

The detail band is rendered one time for each object in the queryset attribute. So, if you have 10 objects in the queryset, you will have the detail band rendered 10 times.

Let's change our report to have a detail band, see below:

```
from geraldo import Report, landscape, ReportBand, ObjectValue
from reportlab.lib.pagesizes import A5
from reportlab.lib.units import cm

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'

    page_size = landscape(A5)
    margin_left = 2*cm
    margin_top = 0.5*cm
    margin_right = 0.5*cm
    margin_bottom = 0.5*cm

    class band_detail(ReportBand):
        height = 0.5*cm
        elements=(
            ObjectValue(attribute_name='id', left=0.5*cm),
            ObjectValue(attribute_name='date_creation', left=3*cm,
                get_value=lambda instance: instance.date_creation.strftime('%m/%d/%Y')),
            )
```

The attribute **band_detail** is locally declared as a class, but it could alternatively be set with an external class.

The important thing here is that the band has the attribute **height**, fixed with a defined height in centimeters.

The second thing to observe is the **elements** list, with 2 widgets representing 2 model class fields: **id** and **date_creation**. The last one is customized with the **get_value** attribute, for date field formatting.

1.3.4 Page Header and Page Footer bands

Page header and Page footer bands are designed to appear on the header and footer, respectively, on every page in the report.

In most reports, a page header is used to show the report title and its page number, page count and maybe the columns header if you are creating a list or grid report.

In the same way, a page footer band is used to show footer information, like the print date/time, software name, company's mark, etc.

Add this code block to the end of the report class to have both bands:

```
class band_page_header(ReportBand):
    height = 1.3*cm
    elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
```

```
            Label(text="ID", top=0.8*cm, left=0.5*cm),
            Label(text=u"Creation Date", top=0.8*cm, left=3*cm),
            SystemField(expression=u'Page %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            ]
    borders = {'bottom': True}

class band_page_footer(ReportBand):
    height = 0.5*cm
    elements = [
            Label(text='Geraldo Reports', top=0.1*cm),
            SystemField(expression=u'Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
            ]
    borders = {'top': True}
```

Now you have to change the top lines of the file with imports to import some new elements:

```
from geraldo import Report, landscape, ReportBand, ObjectValue, SystemField,\
        BAND_WIDTH, Label

from reportlab.lib.pagesizes import A5
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_RIGHT, TA_CENTER
```

The new elements for you now are:

- **SystemField** - shows a system field, like current date/time, page number, page count, report title, report author, etc.
- **Label** - show a free text
- **BAND_WIDTH** - sets the width automatically to its parent band width;
- **TA_RIGHT** and **TA_CENTER** - they set the alignment of a widget

Other good use of Page Header and Page Footer bands is to compose or inherit reports layout. Use your imagination!

1.3.5 Grouping

The next thing now is to group our purchases by a field. Let's choose the field **customer**, ok?

Geraldo allows you to group a report at many levels. This means you can group the report objects by 1, 2 or however many fields you want and have a header and footer to show their aggregation information and other features.

But for now, just add the following code to the end of **reports.py**:

```
groups = [
        ReportGroup(attribute_name = 'customer',
            band_header = ReportBand(
                height = 0.7*cm,
                elements = [
                    ObjectValue(attribute_name='customer', left=0, top=0.1*cm, width=20*cm,
                        get_value=lambda instance: 'Customer: ' + (instance.customer.name),
                        style={'fontName': 'Helvetica-Bold', 'fontSize': 12})
                    ],
                borders = {'bottom': True},
                )
            ),
        ]
```

In the same way as you did before, you have to change the top imports to have the new elements:

```
from geraldo import Report, landscape, ReportBand, ObjectValue, SystemField,\
        BAND_WIDTH, Label, ReportGroup
```

The two important things now are:

- **attribute_name** - this attribute in ReportGroup defines which field we are going to use to group the objects. You must order the queryset beforehand by this attribute to have it working properly. Here you can use a field name, free attribute, property or method that you have in queryset objects;
- **band_header** - this band is for you to show things above the group. You can also use **band_footer** if you wish.

1.3.6 SubReports

SubReport is the way you have to show object children data in a report. As you know, a **purchase** has **many items**, and they are available in attribute **purchase.purchaseitem_set.all()** os something like that.

Geraldo allows you to have however many subreports you want, and they will appear in the same sequence that you place them in the class declaration.

Just add the following block of code to the end of **reports.py**:

```
subreports = [
        SubReport(
            queryset_string = '%(object)s.purchaseitem_set.all()',
            detail_band = ReportBand(
                height=0.5*cm,
                elements=[
                    ObjectValue(attribute_name='product', top=0, left=1*cm),
                    ObjectValue(attribute_name='unit_price', top=0, left=5*cm),
                    ]
                ),
            ),
        ]
```

And now you have to change the top imports line:

```
from geraldo import Report, landscape, ReportBand, ObjectValue, SystemField,\
        BAND_WIDTH, Label, ReportGroup, SubReport
```

The new and most interesting thing here is this:

```
queryset_string = '%(object)s.purchaseitem_set.all()',
```

This is the attribute that you use to join the detail object to the subreport. It is a **string** because you can do a relationship to get objects using different ways.

So, you just use the macro **'%(object)s'** to set the current object!

1.4 Tutorial using Web2Py

Once you have Geraldo installed and its dependencies resolved (read the Installation document to know about the dependencies on ReportLab and Python Imaging Library), you can start using it from **Web2Py** without any need to adjust settings.

1.4.1 Step 1. Preparing your project

Let's create a new URL to load a PDF report online.

We are going to work with a common example for this kind of solution: a purchasing application.

Let's say you have the following model:

```
db.define_table(product,
            Field('name', length=100))

db.define_table(customer,
            Field('name', length=100))

db.define_table(purchase,
            Field('customer', db.customer),
            Field('delivery_address', length=70),
            Field('date_creation', 'datetime', default=request.now),
            Field('date_bill', 'date'),
            Field('date_delivery', 'date'),
            Field('taxes', 'float', default=0.00),
            Field('sub_total_price', 'float', default=0.00),
            Field('total_price', 'float', default=0.00)
            )

db.define_table(purchase_item,
            Field('purchase', db.purchase),
            Field('product', db.product),
            Field('unit_price', 'float')
            Field('quantity', 'float')
            Field('total_price', 'float', default=0.00)
            )
```

Add a new function to your Controller, for example:

```
def purchase_report():
    from reports import ReportPurchase
    from geraldo.generators import PDFGenerator
    import gluon.contenttype
    import StringIO

    resp = StringIO.StringIO()

    purchases = db(db.purchase.id > 0).select(orderby=db.purchase.customer|db.purchase.id)
    report = ReportPurchase(queryset=purchases)
    report.generate_by(PDFGenerator, filename=resp)

    resp.seek(0)
    response.headers['Content-Type'] = gluon.contenttype.contenttype('.pdf')
    filename = "%s_Purchases.pdf" % (request.env.server_name)
    response.headers['Content-disposition'] = "attachment; filename=\"%s\"" % filename
    return resp.read()
```

You can see you first imported two important things:

```
from reports import ReportPurchase
from geraldo.generators import PDFGenerator
```

Your report class (we will create it in the next step) and the generator class **PDFGenerator** you will need to generate a PDF file.

The next thing to note is that you instantiate a StringIO object to use to return the PDF:

```
import StringIO
resp = StringIO.StringIO()
```

Now you are going to work your report instance:

```
purchases = db(db.purchase.id > 0).select(orderby=db.purchase.customer|db.purchase.id)
report = ReportPurchase(queryset=purchases)
report.generate_by(PDFGenerator, filename=resp)
```

Finally you prepare the response:

```
resp.seek(0)
response.headers['Content-Type'] = gluon.contenttype.contenttype('.pdf')
filename = "%s_Purchases.pdf" % (request.env.server_name)
response.headers['Content-disposition'] = "attachment; filename=\"%s\"" % filename
return resp.read()
```

- The first thing you did is to get all purchases, ordered by 'customer' and 'id' fields;
- Afterwards, you got a **report** instance from your report class **ReportPurchase**, using purchases queryset as report driver queryset;
- Next, you called the 'generate_by' method of the generator class.
- And last, you prepared the StringIO object to be delivered to the browser.

Now you have a prepared Web2Py application to use a report you are going to create at the next step!

1.4.2 Declaring the report class

The report class must be an inheritance from **geraldo.Report** class. You can create a new file 'reports.py' to declare your report classes inside.

```
from geraldo import Report

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'
```

To see something more complex, you can set the page size, margins and other report attributes, like below:

```
from geraldo import Report, landscape
from reportlab.lib.pagesizes import A5
from reportlab.lib.units import cm

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'

    page_size = landscape(A5)
    margin_left = 2*cm
    margin_top = 0.5*cm
    margin_right = 0.5*cm
    margin_bottom = 0.5*cm
```

As you can see, we use what we can from the ReportLab libraries, their units, page sizes, stylizing, etc.

A report is driven by **bands**. If you are used to working with other report engines you know what a band is.

A band is a row with elements in the report canvas. Bands in essence are the same but their use varies depending whether you are using a band as the Page Header or Report Summary, for example.

A report can have a band for each one of following attributes:

1.4.3 Detail band

The detail band is the most important band in a report. This is because the detail band is the reason of the existence of every report.

The detail band is used most of the time to show object field values. This is the same for a detailed info page or just a simple list or grid.

The detail band is rendered one time for each object in the queryset attribute. So, if you have 10 objects in the queryset, you will have the detail band rendered 10 times.

Let's change our report to have a detail band, see below:

```
from geraldo import Report, landscape, ReportBand, ObjectValue
from reportlab.lib.pagesizes import A5
from reportlab.lib.units import cm

class ReportPurchase(Report):
    title = 'Purchases list'
    author = 'John Smith Corporation'

    page_size = landscape(A5)
    margin_left = 2*cm
    margin_top = 0.5*cm
    margin_right = 0.5*cm
    margin_bottom = 0.5*cm

    class band_detail(ReportBand):
        height = 0.5*cm
        elements=(
            ObjectValue(attribute_name='id', left=0.5*cm),
            ObjectValue(attribute_name='date_creation', left=3*cm,
                get_value=lambda instance: instance.date_creation.strftime('%m/%d/%Y')),
            )
```

The attribute **band_detail** is locally declared as a class, but it could alternatively be set with an external class.

The important thing here is that the band has the attribute **height**, fixed with a defined height in centimeters.

The second thing to observe is the **elements** list, with 2 widgets representing 2 model class fields: **id** and **date_creation**. The last one is customized with the **get_value** attribute, for date field formatting.

1.5 Basic Reference

1.5.1 Report

class `geraldo.base.Report`

Path: **geraldo.Report**

This is the report main class. Every report must inherit or be an instance of this class. It supports some bands and report definitions.

You can also override some methods to customize some things.

Data source

- **queryset** - Default: None
- **print_if_empty** - Default: False

Report properties

- **title** - Default: '';
- **author** - Default: '';
- **subject** - Default: ''; (you can use it as the report description)
- **keywords** - Default: '';
- **additional_fonts** - Default: {}

  New on 0.4. This is a dictionary where keys have to be the font name and the value have to be the font file path (full path) to support additional true-type fonts inside the PDF. Once you do it, you can use the fone name used on styles as you do with any other default font.

Report page dimensions

- **first_page_number** - Default: 1
- **page_size** - Default: A4
- **margin_top** - Default: 1*cm
- **margin_bottom** - Default: 1*cm
- **margin_left** - Default: 1*cm
- **margin_right** - Default: 1*cm

Report bands

A report band must be a class inherited from **ReportBand** or an instance of **ReportBand** or a class inherited from **ReportBand**

- **band_begin** - Default: None
- **band_summary** - Default: None
- **band_page_header** - Default: None
- **band_page_footer** - Default: None
- **band_detail** - Default: None

Report composition

- **groups** - Default: None
- **subreports** - Default: []

Look & feel

- **default_font_color** - Default: black
- **default_stroke_color** - Default: black
- **default_fill_color** - Default: black
- **default_style** - Default: None

Events system

- **before_print** - Default: None

  Is called by **do_before_print** method, before render the report. Expect arguments **report** and **generator**.

- **before_generate** - Default: None

  Is called by **do_before_generate** method, after render report and before to generate the output files. Expect arguments **report** and **generator**.

- **after_print** - Default: None

  Is called by **do_after_print** method, after generate output files of the report. Expect arguments **report** and **generator**.

- **on_new_page** - Default: None

  Is called by **do_on_new_page** method, when appending new pages when rendering bands. Expect arguments **report**, **page**, **page_number** and **generator**.

  Important: if you are setting events as methods, you must name them as "do_" plus event name (i.e. do_before_print). Otherwise, if you are setting them as attributes (setting when creating objects or setting them dinamically) you must name them without "do_", just their names (i.e. before_print).

Caching related attributes and methods

Take a look on **cache** documentation to see better explanations.

The attributes **cache_status**, **cache_backend** and **cache_file_root** all have a 'real' default None, but they assume values defined on their respective general settings on cache.

- **cache_status** - Default: **geraldo.cache.CACHE_DISABLED**

  Can be setted to 3 different values:

  - geraldo.cache.CACHE_DISABLED
  - geraldo.cache.CACHE_BY_QUERYSET
  - geraldo.cache.CACHE_BY_RENDER

- **cache_backend** - Default: 'geraldo.cache.FileCacheBacked'

  A string path to class used to set/get reports to cache.

- **cache_prefix** - Default: report class module + report class name

  Set the prefix for hash key used to identify generated reports from this class.

- **cache_file_root** - Default: '/tmp/'

  The directory path where cached files are stored.

- **get_cache_relevant_attributes()**

  If you want to set manually what attributes you want to make relevante on cache hash key generation, declare this method to returns a list of the attributes names.

Methods

- **format_date(date, expression)**
- **get_objects_list()**
- **generate_by(generator_class, *args, **kwargs)**

This is the method used to generate a report to file. Report object is just an abstraction of how report must be like. When generating it to a real file, with real list of objects, you must generate using an **generator** - a classe from **geraldo.generators** package.

Example:

```
>>> report = MyReport(queryset=['Rio','London','Beijing'])
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename='test.pdf')
```

- **generate_under_process_by(generator_class, *args, **kwargs)**

  Do the same **generate_by** doest, but uses multiprocessing Process instance to run it. This means it will use a new process to generate the file(s) and will use better the available resources of multi-core servers. In most of cases, also helps to avoid memory consumming.

  But there are servers configured with WSGI with no support to I/O when under separated process and this won't work properly.

  As all of us know, Python processing is by far a better way to work than threading, so, this helps to solve it.

- **find_by_name(name, many=False)**

  Find an object with given name in the children (and children of children and so on).

  If you set **many** to **True**, if many objects with name are found, all of them are returned, instead, an exception **geraldo.exceptions.ManyObjectsFound** is raised. The same is valid when none are found, but the exception **geraldo.exceptions.ObjectNotFound** is raised.

- **find_by_type(typ)**

  Find a list of objects that are instance of class given. As same as **find_by_name**, children of children and so on are found too.

1.5.2 SubReport

class `geraldo.base.`**`SubReport`**

Path: **geraldo.SubReport**

Class to be used for subreport objects. It doesn't need to be inherited.

Attributes

- **queryset_string** - must be a string to create a Python-compatible queryset.

  Examples:

  - '%(object)s.user_permissions.all()'
  - '%(object)s.groups.all()'
  - 'Message.objects.filter(user=%(object)s)'
  - 'Message.objects.filter(user__id=%(object)s.id)'

- **visible** - Default: True

  Set to False if you want to make it not visible.

Report bands

A report band must be a class inherited from **ReportBand** or an instance of **ReportBand** or a class inherited from **ReportBand**

- **name** - Default: None
- **band_detail** - Default: None
- **band_header** - Default: None
- **band_footer** - Default: None

Methods

- **format_date(date, expression)**
- **get_objects_list()**

1.5.3 ReportBand

class `geraldo.base.`**`ReportBand`**

Path: **geraldo.ReportBand**

A band is a horizontal area in the report. It can be used to print things on the top, on summary, on page header, on page footer or one time per object from queryset.

Attributes

- **name** - Default: None
- **height** - Default: 1*cm
- **width** - Default: None
- **visible** - Default: True

  Set to False if you want to make it not visible.
- **borders** - Default: {'top': None, 'right': None, 'bottom': None, 'left': None, 'all': None}

  Borders values can be of three types:

  – Boolean (True/False) - just set if there is a border or not

  – Integer - set there is a border and what is its stroke width. **New on 0.4.**

  – Graphic (instance of Rect, Line, RoundRect, etc.) - set a complex graphic to put along the border
- **elements** - Default: []
- **child_bands** - Default: []
- **force_new_page** - Default: False
- **default_style** - Default: None
- **margin_top** - Default: 0
- **margin_bottom** - Default: 0
- **auto_expanded_height** - Default: False

  Use 'auto_expanded_height' to make flexible bands to fit their heights to their elements.

1.5.4 DetailBand

class geraldo.base.**DetailBand**

Path: **geraldo.DetailBand**

An extension of ReportBand. The only difference is that this class supports more attributes, specific to detail bands.

Attributes

- **name** - Default: None
- **margin_left** - Default: 0
- **margin_right** - Default: 0
- **display_inline** - Default: False

  When you use **display_inline** with **True** value and **width** with a valid value, the generator will try to align the detail band instances in the same way that HTML does with inline displaying of left floating elements: it will keep each detail band to the right of the last one if there is width available.

  This is useful for label reports.

1.5.5 ReportGroup

class geraldo.base.**ReportGroup**

Path: **geraldo.ReportGroup**

This a report grouper class. A report can be grouped to multiple levels by attribute values.

Attributes

- **name** - Default: None
- **attribute_name** - Default: None

Report bands

A report band must be a class inherited from **ReportBand** or an instance of **ReportBand** or a class inherited from **ReportBand**

- **band_header** - Default: None
- **band_footer** - Default: None

1.5.6 ManyElements

class geraldo.base.**ManyElements**

Path: **geraldo.ManyElements**

New on 0.4.

This class is a factory that makes many elements using pre-defined arguments. It is useful to make Cross-Reference Tables, because this is the situation when you almost never know how many columns you ned.

Attributes

- **element_class** - Default: None

  The element class you are going to used to create the instances. i.e: Label, ObjectValue, etc.

- **count** - Default: None

How many elements do you want to create?

- **start_left** - Default: 0

  The left value for the forst element (because it will be incremented with width values for each one.

- **start_top** - Default: 0

  The same of 'start_left', but for top/height relation;

- **visible** - Default: None

  Make the ManyElements not visible and their elements will be also.

The attributes above are just for the class control to make the elements.

But this class supports other attributes, that will be passed to elements it will create. These attributes must be passed when initializing the class, as common arguments like everyone else. Example: you can use ManyElements like below:

```
>>> ATTRIBUTES_LIST = ('date_start', 'date_end', 'salary', 'hours')
>>> band_elements = [
...     ObjectValue(attribute_name='job'),
...     ManyElements(element_class=ObjectValue, count=4, start_left=3*cm,
...         width=1*cm, attribute_name=ATTRIBUTES_LIST),
... ]
```

As you can see, there are 3 special attributes (element_class, count and start_left), there is also a simple value for 'width' attribute, and 'attribute_name' receives a list.

ManyElements is smart enough to know that the attribute had a simple value or a list and use the list in the same sequence of creation (if the list is shorter, the latest item will be used), otherwise that value will be copied to all the created elements.

The attribute 'left' will be increased, starting from 'start_left' plus 'width' for each column.

Result: when rendering, Geraldo will create 4 new elements from the factory.

Methods

- **get_elements(cross_cols=None)**

  This is the method that creates the elements and returns them.

1.6 Widgets Reference

Widgets are report elements to show text values on report canvas.

All widget classes can be initialized with their attributes as class arguments.

1.6.1 Widget

class `geraldo.widgets.`**`Widget`**

Path: **geraldo.widgets.Widget**

Base class for reports widgets, you should use it only to inherit and make your own widgets, otherwise you shouldn't use it directly.

Attributes

- **name** - Default: None

- **height** - Default: 0.5*cm
- **width** - Default: 5*cm
- **left** - Default: 0
- **top** - Default: 0
- **visible** - Default: True

  Set to **False** if you want to make it not visible.

- **borders** - Default: {'top': None, 'right': None, 'bottom': None, 'left': None, 'all': None}

  Borders values can be of three types:

  - Boolean (True/False) - just set if there is a border or not
  - Integer - set there is a border and what is its stroke width. **New on 0.4.**
  - Graphic (instance of Rect, Line, RoundRect, etc.) - set a complex graphic to put along the border

- **style** - Default: {}

  This is a dictionary that uses the powerful ReportLab **ParagraphStyle** class to set style settings for this widget.

  Default keys are:

  - **fontName** - Default: 'Times-Roman',
  - **fontSize** - Default: 10,
  - **leading** - Default: 12,
  - **leftIndent** - Default: 0,
  - **rightIndent** - Default: 0,
  - **firstLineIndent** - Default: 0,
  - **alignment** - Default: TA_LEFT,
  - **spaceBefore** - Default: 0,
  - **spaceAfter** - Default: 0,
  - **bulletFontName** - Default: 'Times-Roman',
  - **bulletFontSize** - Default: 10,
  - **bulletIndent** - Default: 0,
  - **textColor** - Default: black,
  - **backColor** - Default: None,
  - **wordWrap** - Default: None,
  - **allowWidows** - Default: 1,
  - **allowOrphans** - Default: 0,

- **get_value** - Default: None.

  This is the way you can easily customize the widget output. Be careful with this attribute, because each widget has its own set of arguments.

- **truncate_overflow** - Default: False

When "True", truncate the widget to use only its defined **height** and **widget** attributes, with no word wrap. This means that those attributes are required.

Events system

- **before_print** - Default: None

  Is called by **do_before_print** method, before generate the widget output. Expect arguments **widget** and **generator**.

- **after_print** - Default: None

  Is called by **do_after_print** method, after generate the widget output. Expect arguments **widget** and **generator**.

Rendering attributes

They are read-only attributes you can use at render time.

- **instance** - current object being rendered
- **generator** - generator instance
- **report** - report instance this element is in
- **band** - band this element is in
- **page** - current page

1.6.2 Label

class `geraldo.widgets.`**`Label`**

Path: **geraldo.Label**

A label is just a simple text.

Example of use

```
>>> Label(text='Taxes we have paid', left=1*cm, top=0.5*cm, width=10*cm, height=0.5*cm)
```

Attributes

- **height** - Default: 0.5*cm
- **width** - Default: 5*cm
- **left** - Default: 0
- **top** - Default: 0
- **visible** - Default: True
- **style** - Default: {}

get_value must have a 'text' argument.

1.6.3 ObjectValue

class `geraldo.widgets.`**`ObjectValue`**

Path: **geraldo.ObjectValue**

This shows the value from a method, field or property from objects in the queryset.

You can provide an action to show the object value or an aggregation function on it.

You can also use the 'display_format' attribute to set a friendly string formatting, with a mask or additional text.

'get_value' lambda must have an 'instance' argument.

Example of use

```
>>> ObjectValue(attribute_name='name', left=1*cm, top=0.5*cm, width=10*cm, height=0.5*cm)
```

Attributes

- **height** - Default: 0.5*cm
- **width** - Default: 5*cm
- **left** - Default: 0
- **top** - Default: 0
- **visible** - Default: True
- **style** - Default: {}
- **attribute_name** - Required

  You can provide here the object attribute name you want to show in this widget. You can provide a field, common attribute, property or just a method, since it has no required argument to provide.

  You can use paths of callable or properties instead of attribute name in this attribute.

  Examples:

  attribute_name = 'name.upper' attribute_name = 'customer.name' attribute_name = 'customer.type.name.lower'

- **action** - Default: geraldo.FIELD_ACTION_VALUE

  The action is where you say what Geraldo will do with the value of this field. The default action is just to show its value and nothing more.

  But if you are using an ObjectValue in a summary or footer band of report, group or subreport, you will probably have a need for aggregation functions. This is why this attribute exists.

  Field actions are all available in the **geraldo** package to import from. The choices you can use in **action** attributes are:

  - geraldo.FIELD_ACTION_VALUE
  - geraldo.FIELD_ACTION_COUNT
  - geraldo.FIELD_ACTION_AVG
  - geraldo.FIELD_ACTION_MIN
  - geraldo.FIELD_ACTION_MAX
  - geraldo.FIELD_ACTION_SUM
  - geraldo.FIELD_ACTION_DISTINCT_COUNT

- **display_format** - Default: '%s'

  Use simple string formatting on the field value. You could otherwise use **get_value** if you want something more powerful.

- **get_text** - Default: None.

  This 'lambda' attribute is seemed to **get_value** because it works after value getting. It's important to keep aware there are two moments from get a value from instance until to render it on the output generation:

  - 1. First the value is getted (and keeps being an unformatted value)
  - 2. After this, the value is formatted (and transformed to unicode string)

- **stores_text_in_cache** - Default: True

  This is a boolean attribute you can set to **False** if you to force the text getting to run twice: on renderizing and on generating. If you keep it as **True**, it will store the text on the first getting and use the stored text on next time(s) it be requested.

1.6.4 SystemField

class `geraldo.widgets.`**`SystemField`**

Path: **geraldo.SystemField**

This shows system information, like the report title, current date/time, page number, page count, etc.

'get_value' must have 'expression' and 'fields' arguments.

Example of use

```
>>> ObjectValue(expression='%(report_title)s', left=1*cm, top=0.5*cm, width=10*cm, height=0.5*cm)
```

Attributes

- **height** - Default: 0.5*cm
- **width** - Default: 5*cm
- **left** - Default: 0
- **top** - Default: 0
- **visible** - Default: True
- **style** - Default: {}
- **expression** - Default: '%(report_title)s'

  This is a simple string you can write with basic macros to show system field values.

  The available macros are:

  - %(report_title)s
  - %(page_number)s
  - %(first_page_number)s
  - %(last_page_number)s
  - %(page_count)s
  - %(report_author)s
  - %(now:FORMAT)s - in replace of 'FORMAT' you can use date formatting standard template filter date formatting. But if you are using your own **Report.format_date** method, this will use it.

    Examples:

* '%(now:%Y)' - shows the current year
* '%(now:%m/%d/%Y)' - shows something like '01/23/2009'

Note: this is changed now. Before this function used Django template filter 'date' to format datetime, but once we were going to use two standards, this wouldn't be nice. If you want to use Django's template filter, you can override the method **Report.format_date(date, expression)** and use it.

1.7 Graphics Reference

Graphics elements are the way you have to draw images and shapes on the band canvas.

They can be useful also to make customized borders in bands.

1.7.1 Graphic

class `geraldo.graphics.`**`Graphic`**

Path: **geraldo.graphics.Graphic**

This is the basic graphic class. You should use it only when inheriting to create your own graphic class, never use it directly.

Its rect area is based on **left-width-top-height** dimensions.

Attributes

- **name** - Default: None
- **visible** - Default: True

  Set to **False** if you want to make it not visible.
- **stroke** - Default: True
- **stroke_color** - Default: reportlab.lib.colors.black
- **stroke_width** - Default: 1
- **fill** - Default: False
- **fill_color** - Default: reportlab.lib.colors.black

Rendering attributes

They are read-only attributes you can use at render time.

- **instance** - current object being rendered
- **generator** - generator instance
- **report** - report instance this element is in
- **band** - band this element is in
- **page** - current page

Events system

- **before_print** - Default: None

  Is called by **do_before_print** method, before generate the graphic output. Expect arguments **graphic** and **generator**.

- **after_print** - Default: None

  Is called by **do_after_print** method, after generate the graphic output. Expect arguments **graphic** and **generator**.

Methods

- **set_rect(**kwargs)**

  Used to generate a hard rectangle when rendering pages. Override it to set your own rule if you need.

1.7.2 Fixed

class geraldo.graphics.**Fixed**

Path: **geraldo.graphics.Fixed**

In the same way as Graphic class, you should use this just to inherit and create your own graphic. This is a graphic with rect with **left-right-top-bottom** dimensioning.

1.7.3 Rect

class geraldo.graphics.**Rect**

Path: **geraldo.Rect**

A rectangle with square borders on the canvas.

Example of use

```
>>> Rect(left=1*cm, top=0.5*cm, width=10*cm, height=0.5*cm, fill=True, fill_color=yellow)
```

Attributes

- **left** - Required
- **width** - Required
- **top** - Required
- **height** - Required

1.7.4 RoundRect

class geraldo.graphics.**RoundRect**

Path: **geraldo.RoundRect**

A rectangle with rounded borders on the canvas.

Example of use

```
>>> RoundRect(left=1*cm, top=0.5*cm, width=10*cm, height=0.5*cm, stroke=True, stroke_width=3, stroke_
```

Attributes

- **left** - Required
- **width** - Required
- **top** - Required
- **height** - Required

- **radius** - Required

  Inform the radius number for the round corners.

1.7.5 Line

class geraldo.graphics.**Line**

Path: **geraldo.Line**

Example of use

```
>>> Line(left=1*cm, top=0.5*cm, right=10*cm, bottom=0.5*cm)
```

Attributes

- **left** - Required
- **width** - Required
- **right** - Required
- **bottom** - Required

1.7.6 Circle

class geraldo.graphics.**Circle**

Path: **geraldo.Circle**

Example of use

```
>>> Circle(left_center=6*cm, top_center=3.5*cm, radius=3*cm)
```

Attributes

- **left_center** - Required
- **top_center** - Required
- **radius** - Required

1.7.7 Arc

class geraldo.graphics.**Arc**

Path: **geraldo.Arc**

Example of use

```
>>> Arc(left=1*cm, top=0.5*cm, right=10*cm, bottom=0.5*cm, extent=50, start_angle=5)
```

Attributes

- **left** - Required
- **width** - Required
- **right** - Required
- **bottom** - Required
- **start_angle** - Default: 0

- **extent** - Default: 90

1.7.8 Ellipse

class `geraldo.graphics.`**`Ellipse`**

Path: **geraldo.Ellipse**

Example of use

```
>>> Ellipse(left=1*cm, top=0.5*cm, right=10*cm, bottom=0.5*cm)
```

Attributes

- **left** - Required
- **width** - Required
- **right** - Required
- **bottom** - Required

1.7.9 Image

class `geraldo.graphics.`**`Image`**

Path: **geraldo.Image**

Example of use

```
>>> Image(left=1*cm, top=0.5*cm, right=10*cm, bottom=0.5*cm, filename='path/to/file.jpg')
```

Attributes

- **left** - Required
- **width** - Required
- **top** - Required
- **height** - Required
- **filename** - Required

  You can provide a filename path or a Python Imaging Library Image instance. If the latter, then you have to have this library installed.

- **get_image** - Default: None

  You should provide a function or lambda object to this attribute when you want to work with Images or Charts based on object values or logic.

1.8 Barcodes Reference

New on 0.4

The BarCode element is just one, and it is used to show barcodes of many types on the report.

This element is inherited from **geraldo.graphics.Graphic**

1.8.1 BarCode

class `geraldo.barcodes.`**`BarCode`**

Path: **geraldo.barcodes.BarCode**

This is the once class you can use to show bar codes. It supports the following bar code types, supplied by **ReportLab** barcodes library:

- Codabar
- Code11
- Code128
- EAN13
- EAN8
- Extended39
- Extended93
- FIM
- I2of5
- MSI
- POSTNET
- Standard39
- Standard93
- USPS_4State

Attributes

- **name** - Default: None
- **visible** - Default: True

  Set to **False** if you want to make it not visible.

- **height** - Default: 1.5*cm
- **width** - Default: 0.03*cm

  This attribute **does not set the barcode width**, but the **bar width**, what means it is the width of the minimum bar of a barcode. You should set values like 0.02*cm or something like that (i.e. never 5*cm).

- **attribute_name** - Default: None

  As same as **geraldo.widgets.ObjectValue**, this attribute reffers to an attribute, attribute path or method to get the barcode value.

- **checksum** - Default: 0

  Most of barcode types supports you set the number of digits for checksum.

- **routing_attribute** - Default: None

  Useful only for **USPS_4State** barcodes. Works like **attribute_name**, but **routing** attribute of barcode.

Rendering attributes

They are read-only attributes you can use at render time.

- **instance** - current object being rendered
- **generator** - generator instance
- **report** - report instance this element is in
- **band** - band this element is in
- **page** - current page

1.9 Cross-Reference Tables Reference

New on 0.4

According to [Wikipedia](http://en.wikipedia.org/wiki/Cross_reference):

> A cross-reference (noun) is an instance within a document which refers to related or synonymous information elsewhere, usually within the same work. To cross-reference or to cross-refer (verb) is to make such connections. The term "cross-reference" is often abbreviated as x-ref, xref, or, in computer science, XR. Cross-referencing is usually employed to either verify claims made by an author or to link to another piece of work that is of related interest.

Translating to practical life

Cross-reference is when you have much data and want to cross the relation between two fields to get a third value.

Example

If you have an ERP system that supports many stores, selling many products, you can to need a report with "Product Sellings per Store", what means you have a bunch of data and 3 entities to base on: Products, Sellings and Stores, and you will have rows for Products, columns for Stores and the cells with an aggregation value about the Sellings that cross with a Product AND a Store, like the following:

```
          | Store 1 | Store 2 | Store 3 | Average
-----------------------------------------------
Product 1 |   159   |     0   |   130   |    96
Product 1 |    49   |    20   |    30   |    30
Product 1 |   181   |   230   |    27   |   146
-----------------------------------------------
  Totals  |   389   |   250   |   187   |   275
```

As you probably noted, the first column of rows is the list of a coordinage X (the Products), the first row of columnss is the list of a coordinate Y (the Stores) and the cells are the crossed relation between Products and Stores (probably the count of sellings of each Product on each Store). The latest column is the average aggregation of cells of the same row, and the latest row is the average of cells of same column.

1.9.1 CrossReferenceMatrix

class `geraldo.cross_reference.`**`CrossReferenceMatrix`**

Path: **geraldo.CrossReferenceMatrix**

The Solution

As you can realize, this is not an easy job, because you never know how many columns you will have, and there are many ways to collect cross-reference data.

Some RDBMS's offers *out of box* ways to get a cross-reference among three fields in a select, but this is not only limited but is also not a definitive solution, because even if you have the matrix of data, you still have to prepare all elements on the report you are going to print.

The solution on Geraldo is the class 'geraldo.cross_reference.CrossReferenceMatrix'.

To instantiate this class, you must inform the objects list (not necessarily a matrix, because you can just give a Django queryset, or a list of dictionaries) and the names of the X and Y attributes (**row_attribute** and **col_attribute**).

Once you create that instance, you can call many methods from it to get aggregated values, giving the third attribute you want to aggregate, and the filter for row and column directions.

This instance can be used out of Geraldo's functions, like a template, e-mails, text files, whatever, it is not dependent on reports to work.

Methods

- **__init__(objects_list, rows_attribute, cols_attribute)**

  When you make an instance of **CrossReferenceMatrix**, you must inform the objects list and the attributes to cross: rows_attribute (X) and cols_attribute (Y).

- **rows()**

  Returns the values of row attribute in objects list, that means you get the list of rows on the matrix.

- **cols()**

  Returns the values of column attribute on objects list, that means you get the list of columns on the matrix.

- **values(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  Returns the values (a list of them) crossed between a row and a column. If you just let the argument 'row' or 'col' with default value, it will get all values according to the filter you informed (i.e. if you just informed the 'col', you will get all values for that col, in the sequence of the available rows).

  This method is used by all of the others below.

- **max(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'values', but this find the maximum value for that relation and returns it.

- **min(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'max', but returns the minimum value of them.

- **sum(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'max', but returns the sum of found values.

- **avg(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'max', but returns the average of found values.

- **count(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'max', but returns the count of found values.

- **distinct_count(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  As same as method 'count', but returns the count of not redundant values.

- **first(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

  Just returns the first value found for the relation.

- **last(cell, row=RANDOM_ROW_DEFAULT, col=RANDOM_COL_DEFAULT)**

As same as method 'first', but returns the last value.

- **matrix(cell, func='values')**

  Returns a matrix with rows and columns with cross table values (including headers).

1.10 Generators Reference

Generators are classes to generate a report instance to some data format. In future we will have generators for XML, HTML, PS, images and so on.

1.10.1 PDF Generator

class `geraldo.generators.`**`PDFGenerator`**

Path: **geraldo.generators.PDFGenerator**

This generator is the most mature generator we have. It generates PDF files to be viewed on Adobe Acrobat Reader or similar software. It uses the awesome library 'ReportLab' to create them.

Attributes:

- **first_page_number** - Default: 1

  Set a different number to this attribute if you want to start page counting from a customized number.

- **filename** - Default: None

  You have to provide the filename you are creating, unless you provided the attribute 'canvas'

- **canvas** - Default: None

  New in version 0.3.5

  If you already have a canvas instantiated, you can provide it instead of 'filename' to generate this report inside. This is useful to generate many reports in the same PDF.

- **return_canvas** - Default: False

  New in version 0.3.5

  If you set this to **True**, the file will not be saved and the generator will return the canvas to you to use as you want.

- **multiple_canvas** - Default: False

  New in version 0.3.7

  This attribute has default value **True** if you have **pyPDF** library installed on your PYTHONPATH.

  It is useful to store canvas in a temporary directory and combine all in once when finished the generating. This helps to gain on memory consuming.

  You can find more about pyPDF on http://pypi.python.org/pypi/pyPdf/

- **temp_directory** - Default: '/tmp/'

  Regarding to temporary saving files on report processing, this attribute can receive a string with directory path where save those files.

To use PDFGenerator you just do something like this:

```
>>> my_report_instance.generate_by(PDFGenerator, filename='file.pdf')
```

In **filename** argument you can use a file path string or a file output object.

Examples:

```
>>> fp = file('test.pdf', 'w')
>>> my_report_instance.generate_by(PDFGenerator, filename=fp)
>>>
>>> resp = HttpResponse(mimetype='application/pdf')
>>> my_report_instance.generate_by(PDFGenerator, filename=resp)
```

1.10.2 Text Generator

class geraldo.generators.**TextGenerator**

Path: **geraldo.generators.TextGenerator**

This generator is still in the development stage, but it already works well. It can be used to generate text files or to print to matrix printers.

You can use it exactly like you would use PDF generator, but this has some more features. When running **'generate_by'** report method, you can inform the attribute 'filename' like you do in PDF, but you could also do some other things.

Attributes:

- **row_height** - Default: 0.5*cm

  Should be the equivalent height of a row plus the space between rows. This is important to calculate how many rows a page has.

- **character_width** - Default: 0.23*cm

  Should be the equivalent width of a character. This is important to calculate how many columns a page has.

- **to_printer** - Default: True

  Is a boolean variable which you can set to generate a text to matrix printer or not. This controls whether escape characters will be in the output or not.

- **escape_set** - Default: geraldo.generators.text.DEFAULT_ESCAPE_SET

  Is a dictionary with equivalence table to escape codes. As far as we know, escape codes can vary depending on model or printer manufacturer (i.e. Epson, Lexmark, HP, etc.). This attribute is useful to support this. Default is ESC/P2 standard (Epson matrix printers)

- **filename** - Default: None

  Is the file path you can optionally provide to save text to.

- **encode_to** - Default: None

  Here you can provide the coding identifier to force Geraldo to encode the output string in. Example: 'latin-1'

- **manual_escape_codes** - Default: False

  A boolean variable that sets whether escape codes are manually provided or not.

Examples:

Basic:

```
>>> my_report_instance.generate_by(TextGenerator, filename='file.txt')
```

Returning the output:

```
>>> output = my_report_instance.generate_by(TextGenerator)
```

Setting row height and/or column width:

```
>>> output = my_report_instance.generate_by(TextGenerator, row_height=0.7*cm, character_width=0.2*cm)
```

Forcing the generator to encode the output:

```
>>> output = my_report_instance.generate_by(TextGenerator, encode_to='latin-1')
```

Setting to be printed by a printer or not:

```
>>> output = my_report_instance.generate_by(TextGenerator, to_printer=False)
```

Setting a set of escape codes for printer (matrix printers work with escape codes set for special characters or to setup the output printing):

```
>>> from geraldo.generators.text import DEFAULT_ESCAPE_SET
>>> my_escape_set = DEFAULT_ESCAPE_SET.copy()
>>> my_escape_set['condensed'] = chr(15) + chr(17)
>>> output = my_report_instance.generate_by(TextGenerator, to_printer=True, escape_set=my_escape_set)
```

Forcing the output to print out escape codes before/after report or page print:

```
>>> MyTextGenerator(TextGenerator):
...     to_printer = True
...     manual_escape_codes = True
...     escapes_report_start = chr(15) # Sets condensed mode
...     escapes_report_end = chr(18) # Unsets condensed mode
...     escapes_page_start = chr(12) + chr(10) # Form and line feed
...     escapes_page_end = ''
>>> output = my_report_instance.generate_by(MyTextGenerator)
```

1.11 Caching

Geraldo Reports has support to reports caching. This means you can store reports in a directory (or other kind of cache store you want) and Geraldo will find them and return when the same report be generated.

This is why exists the module **geraldo.cache**. There are general settings you can set to enable caching and extend its functios.

1.11.1 Settings

All of the following settings are generic and are used by respective ones in each report class, and there they can be overridden too.

- **DEFAULT_CACHE_STATUS** - Default: **geraldo.cache.CACHE_DISABLED**

  This setting can receive three different types of status:

  - geraldo.cache.CACHE_DISABLED

    Just disables the caching at all.

  - geraldo.cache.CACHE_BY_QUERYSET

    Enable the caching function to use queryset values as the base to generate a hash key and store in the cache. You must use this if the most important thing for you is the data in the report. By the way, this choice is the **fastest** one.

  - geraldo.cache.CACHE_BY_RENDER

    Enable the caching function to use the rendered objects as the base to generate a hash key and store in the cache. You must use this if you want to consider style and feel when storing to cache. This is not the fastest but probably the most **reliable**.

- **CACHE_BACKEND** - Default: 'geraldo.cache.FileCacheBackend'

  Just inform the full string path to a backend class.

- **CACHE_FILE_ROOT** - Default: '/tmp/'

  Just inform (if you are using the **FileCacheBackend** backend) the path of directory where you want to store the cache files.

1.11.2 Classes

- **FileCacheBackend**

  This is the default (and the only one Geraldo supplies) cache backend. It stores files in a directory on the file system (hard disk).

  You can extend this class if you want to tun up the file system caching.

- **BaseCacheBackend**

  If you want to extend cache to a different kind of cache store (i.e. memcache, database or someone else), you have to make a class inheriting from this one.

  Basically you must set three methods:

  - **get(hash_key)**
  - **set(hash_key, content)**
  - **exsts(hash_key)**

1.12 Utilities Reference

1.12.1 landscape

geraldo.utils.**landscape**()

Path: **geraldo.landscape**

Page sizes are tuples with 2 nodes: first with **width** and second width **height**.

landscape is a helper function to invert this tuple.

1.12.2 BAND_WIDTH

geraldo.utils.**BAND_WIDTH**

Path: **geraldo.BAND_WIDTH**

Once you use this constant in a widget or graphic width, it will assume its band width automatically.

1.12.3 memoize

geraldo.utils.**memoize**()

Path: **geraldo.utils.memoize**

This is a decorator used internally by some functions to store their result values like a cache. When the same function is called again with same arguments values, the stored value is returned, instead of run its code again.

You can use it for your own code, just wrapping your function under the decorator.

Example of use:

```
>>> from geraldo.utils import memoize
>>> @memoize
... def calc_sum(val1, val2):
...     return val1 + val2
```

1.12.4 run_under_process

geraldo.utils.**run_under_process**()

Path: **geraldo.utils.run_under_process**

This is a decorator supplied by Geraldo to help developers to run functions under a new process instead of the main thread.

Geraldo uses it in method **Report.generate_under_process_by**, to run the report generation under other process, but you can use it on your own code, if you don't want/can't use that method.

Example of use:

```
>>> from geraldo.generators import PDFGenerator
>>> from geraldo.utils import run_under_process
>>> @run_under_process
... def generate_by_report(report, filename):
...     report.generate_by(PDFGenerator, filename=filename)
```

1.12.5 DISABLE_MULTIPROCESSING

geraldo.utils.**DISABLE_MULTIPROCESSING**

Path: **geraldo.DISABLE_MULTIPROCESSING**

A short way to disable all use of multiprocessing on Geraldo, at all. This is useful when you are debugging your reports.

1.13 Examples

The following examples are the same as you can find in the 'tests' package, from source code.

1.13.1 A Simple Report

This is just a simple report, that uses the Geraldo API to format a report with the begin, summary, page header, page footer and detail bands:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import Permission

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH

class PermissionsReport(Report):
    title = 'Permissions list'

    class band_begin(ReportBand):
        height = 1*cm
        elements = [
            Label(text='Look those permissions please', top=0.1*cm,
                left=8*cm),
        ]

    class band_summary(ReportBand):
        height = 0.7*cm
        elements = [
            Label(text="That's all", top=0.1*cm, left=0),
            ObjectValue(attribute_name='name', top=0.1*cm, left=3*cm,
                action=FIELD_ACTION_COUNT,
                display_format='%s permissions found'),
        ]
        borders = {'all': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm,
                left=0, width=BAND_WIDTH, style={'fontName': 'Helvetica-Bold',
                'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Name", top=0.8*cm, left=3*cm),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
```

```
        height = 0.5*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0),
            ObjectValue(attribute_name='name', top=0, left=3*cm, width=7*cm),
        ]
```

Below you can see how to instantiate it from a permissions list

```
>>> report = PermissionsReport(queryset=Permission.objects.order_by('id'))
```

Generating PDF...

```
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/simple-report.pdf'))
```

Generating Text...

```
>>> from geraldo.generators import TextGenerator
>>> report.generate_by(TextGenerator, filename=os.path.join(cur_dir, 'output/simple-report.txt'), to_
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/simple-report.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/simple-report.txt

1.13.2 An Intermediary Report

This is just a report with intermediary features: aggregation functions, customized widget values and some stylized elements:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY, TA_RIGHT
from reportlab.lib.colors import navy, yellow, red

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, FIELD_ACTION_AVG, FIELD_ACTION_MIN,\
    FIELD_ACTION_MAX, FIELD_ACTION_SUM, FIELD_ACTION_DISTINCT_COUNT, BAND_WIDTH,\
    RoundRect, Line
from geraldo.base import EmptyQueryset

class UsersReport(Report):
    title = 'System users'

    class band_begin(ReportBand):
        height = 1*cm
        elements = [
            Label(text="This is a <b>users</b> report.<br/>This time <i>our idea</i> is to show"+\
                       "some <font color=red>other features</font> of this reports engine", top=0.1*c
                  left=0.1*cm, width=BAND_WIDTH),
        ]
        borders = {'all': True}
```

```
    class band_summary(ReportBand):
        height = 3.2*cm
        elements = [
            Label(text="Users count:", top=0.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='username', top=0.1*cm, left=4*cm,\
                action=FIELD_ACTION_COUNT, display_format='%s permissions found'),

            Label(text="Users ids average:", top=0.6*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=0.6*cm, left=4*cm, action=FIELD_ACTION_AVG),

            Label(text="Users ids minimum:", top=1.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=1.1*cm, left=4*cm, action=FIELD_ACTION_MIN),

            Label(text="Users ids maximum:", top=1.6*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=1.6*cm, left=4*cm, action=FIELD_ACTION_MAX),

            Label(text="Users ids sum:", top=2.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=2.1*cm, left=4*cm, action=FIELD_ACTION_SUM),

            Label(text="Users first name distinct:", top=2.6*cm, left=0.2*cm),
            ObjectValue(attribute_name='first_name', top=2.6*cm, left=4*cm, action=FIELD_ACTION_DISTI
        ]
        borders = {'all': RoundRect(radius=5, fill_color=yellow, fill=True)}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER,
                    'textColor': navy}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Username", top=0.8*cm, left=3*cm),
            Label(text="First name", top=0.8*cm, left=8*cm),
            Label(text="Last name", top=0.8*cm, left=13*cm),
            Label(text="Staff", top=0.8*cm, left=18*cm),
        ]
        borders = {'bottom': Line(stroke_color=navy)}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm,
                right=0),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': Line(stroke_color=red, stroke_width=3)}

    class band_detail(ReportBand):
        height = 0.7*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0),
            ObjectValue(attribute_name='username', top=0, left=3*cm, display_format='<font size=14 na
            ObjectValue(attribute_name='first_name', top=0, left=8*cm),
            ObjectValue(attribute_name='last_name', top=0, left=13*cm),
            ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
                get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
        ]
```

Below you can see how to instantiate it from a permissions list

```
>>> queryset = User.objects.order_by('id')
>>> report = UsersReport(queryset=queryset)
```

Generating PDF...

```
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/users-report.pdf'))
```

Generating Text...

```
>>> from geraldo.generators import TextGenerator
>>> report.generate_by(TextGenerator, filename=os.path.join(cur_dir, 'output/users-report.txt'))
```

Generating PDF with half height page

```
>>> report.page_size = (A4[0], A4[1] / 2)
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/users-report-half-height
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/users-report.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/users-report-half-height.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/users-report.txt

1.13.3 A report with all kinds of graphic elements

This is just a report with every graphic element:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.colors import navy, yellow, red, purple, orange,\
    green, white, blue
from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY

from geraldo import Report, ReportBand, RoundRect, Rect, Line, Circle,\
    Arc, Ellipse, SystemField, Label, BAND_WIDTH, Image

class GraphicsReport(Report):
    title = 'Graphics demonstration'
    print_if_empty = True

    class band_begin(ReportBand):
        height = 15*cm
        elements = [
            RoundRect(left=0.2*cm, top=0.5*cm, width=3*cm, height=2*cm,
                radius=10, stroke_color=purple),
            Rect(left=4*cm, top=1.0*cm, width=3*cm, height=2*cm,
                fill=True, stroke=False, fill_color=orange),
            Line(left=8*cm, top=3*cm, right=9*cm, bottom=0),
            Line(left=9*cm, top=0, right=10*cm, bottom=3*cm),
            Line(left=8.5*cm, top=3*cm, right=9*cm, bottom=6*cm),
            Line(left=9*cm, top=6*cm, right=10*cm, bottom=3*cm),
            Circle(left_center=5*cm, top_center=5*cm, radius=1*cm, fill_color=yellow,
```

```
                fill=True),
            Arc(left=1*cm, top=3.0*cm, right=4*cm, bottom=5*cm,
                start_angle=150, extent=100),
            Ellipse(left=1*cm, top=6.0*cm, right=4.5*cm, bottom=8*cm,
                fill_color=blue, fill=True, stroke_width=3),
            Image(left=10*cm, top=6*cm, width=4*cm, height=5.12*cm,
                filename=os.path.join(cur_dir, 'photo.jpg')),
            Image(left=13*cm, top=6*cm,
                filename=os.path.join(cur_dir, 'photo.jpg')),
            #Poligon(), # --> uses drawPath
            Label(text="""<b>William Shakespeare</b> (baptised 26 April 1564 - 23 April 1616)[a] was
                left=12*cm, top=1*cm, width=6*cm, height=4*cm,
                style={'wordWrap': True, 'borderWidth': 1,
                    'borderColor': green, 'borderPadding': 4,
                    'borderRadius': 2, 'alignment': TA_JUSTIFY}),
        ]

    class band_page_header(ReportBand):
        height = 1.4*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0,
                width=BAND_WIDTH, style={'fontName': 'Helvetica-Bold',
                    'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0, width=1*cm,
                style={'borderWidth': 1, 'borderColor': green,
                    'borderPadding': 1, 'borderRadius': 2}),
            Label(text="Name", top=0.8*cm, left=3*cm,
                style={'backColor': red, 'textColor': white,
                'fontName': 'Helvetica'}),
        ]
        borders = {'bottom': True}
```

Generating PDF...

```
>>> report = GraphicsReport()
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/graphics-report.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/graphics-report.pdf

1.13.4 Child Bands

This example uses child bands. All bands can have child bands attached. Child bands are bands that are attached to a parent band and will print below them:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY
from reportlab.lib.colors import navy, yellow, red

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
```

```
    FIELD_ACTION_COUNT, FIELD_ACTION_AVG, FIELD_ACTION_MIN,\
    FIELD_ACTION_MAX, FIELD_ACTION_SUM, FIELD_ACTION_DISTINCT_COUNT, BAND_WIDTH,\
    RoundRect, Line

class ChildBandsReport(Report):
    title = 'Child bands demonstration'

    class band_summary(ReportBand):
        height = 0.8*cm
        elements = [
            Label(text="Users count:", top=0.1*cm, left=0),
            ObjectValue(attribute_name='username', top=0.1*cm, left=4*cm,\
                action=FIELD_ACTION_COUNT, display_format='%s permissions found'),
        ]
        child_bands = [
            ReportBand(
                height = 0.6*cm,
                elements = [
                    Label(text="Users ids average:", top=0.6*cm, left=0),
                    ObjectValue(attribute_name='id', top=0.6*cm, left=4*cm, action=FIELD_ACTION_AVG),
                ]),
            ReportBand(
                visible = False,
                height = 0.6*cm,
                elements = [
                    Label(text="Users ids minimum:", top=1.1*cm, left=0),
                    ObjectValue(attribute_name='id', top=1.1*cm, left=4*cm, action=FIELD_ACTION_MIN),
                ]),
            ReportBand(
                height = 0.6*cm,
                elements = [
                    Label(text="Users ids maximum:", top=1.6*cm, left=0),
                    ObjectValue(attribute_name='id', top=1.6*cm, left=4*cm, action=FIELD_ACTION_MAX),
                ],
                borders = {'bottom': True}),
            ReportBand(
                height = 0.6*cm,
                elements = [
                    Label(text="Users ids sum:", top=2.1*cm, left=0),
                    ObjectValue(attribute_name='id', top=2.1*cm, left=4*cm, action=FIELD_ACTION_SUM),
                ]),
            ReportBand(
                visible = False,
                height = 0.6*cm,
                elements = [
                    Label(text="Users first name distinct:", top=2.6*cm, left=0),
                    ObjectValue(attribute_name='first_name', top=2.6*cm, left=4*cm, action=FIELD_ACTI
                ]),
        ]
        borders = {'all': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Username", top=0.8*cm, left=3*cm),
```

```
            Label(text="First name", top=0.8*cm, left=8*cm),
            Label(text="Last name", top=0.8*cm, left=13*cm),
            Label(text="Staff", top=0.8*cm, left=18*cm),
        ]
        borders = {'bottom': Line(stroke_color=red, stroke_width=3)}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm),
        ]
        borders = {'top': Line(stroke_color=navy)}

    class band_detail(ReportBand):
        height = 0.7*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0),
            ObjectValue(attribute_name='username', top=0, left=3*cm),
            ObjectValue(attribute_name='first_name', top=0, left=8*cm),
            ObjectValue(attribute_name='last_name', top=0, left=13*cm),
            ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
                get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
        ]
```

Generating PDF...

```
>>> report = ChildBandsReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
>>> report.queryset = User.objects.order_by('id')
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/child-bands-report.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/child-bands-report.pdf

1.13.5 Grouping Bands

This example uses grouping bands. We must have support for multiple groups in a report.

Grouping is something like this:

- Country
    - City
    - Soccer team

As you can see above we have 2 groupings: by country and by city (under country).

In this test we work with users as following:

- If staff
    - If superuser
    - User

See below:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))
```

```
from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY
from reportlab.lib.colors import navy, yellow, red

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, FIELD_ACTION_SUM, BAND_WIDTH, Line, ReportGroup

class GrouppingReport(Report):
    title = 'Groupping demonstration'

    class band_summary(ReportBand):
        height = 0.8*cm
        elements = [
            Label(text="Users count:", top=0.1*cm, left=0),
            ObjectValue(attribute_name='id', top=0.1*cm, left=4*cm,\
                action=FIELD_ACTION_COUNT, display_format='%s users found'),
        ]
        borders = {'all': True}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Username", top=0.8*cm, left=3*cm),
            Label(text="First name", top=0.8*cm, left=8*cm),
            Label(text="Superuser", top=0.8*cm, left=13*cm),
            Label(text="Staff", top=0.8*cm, left=18*cm),
        ]
        borders = {'bottom': Line(stroke_color=red, stroke_width=3)}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm),
        ]
        borders = {'top': Line(stroke_color=navy)}

    class band_detail(ReportBand):
        height = 0.7*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=1*cm),
            ObjectValue(attribute_name='username', top=0, left=3*cm),
            ObjectValue(attribute_name='first_name', top=0, left=8*cm),
            ObjectValue(attribute_name='is_superuser', top=0, left=13*cm,
                get_value=lambda instance: instance.is_superuser and 'Yes' or 'No'),
            ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
                get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
        ]

    groups = [
        ReportGroup(attribute_name='is_superuser',
            band_header=ReportBand(
                height=0.7*cm,
```

```
                elements=[
                    ObjectValue(attribute_name='is_superuser', left=0, top=0.1*cm,
                        get_value=lambda instance: 'Superuser: ' + (instance.is_superuser and 'Yes' o
                        style={'fontName': 'Helvetica-Bold', 'fontSize': 12})
                ],
                borders={'bottom': True},
            ),
            band_footer=ReportBand(
                height=0.7*cm,
                elements=[
                    ObjectValue(attribute_name='id', action=FIELD_ACTION_COUNT,
                        display_format='%s superusers', left=0*cm, top=0.1*cm),
                    ObjectValue(attribute_name='id', action=FIELD_ACTION_SUM,
                        display_format='%s is the sum of IDs above', left=4*cm, top=0.1*cm),
                ],
                borders={'top': True},
            ),
        ),
        ReportGroup(attribute_name='is_staff',
            band_header=ReportBand(
                height=0.7*cm,
                elements=[
                    ObjectValue(attribute_name='is_staff', left=0.5*cm, top=0.1*cm,
                        get_value=lambda instance: 'Staff: ' + (instance.is_staff and 'Yes' or 'No'))
                ],
                borders={'bottom': True},
            ),
            band_footer=ReportBand(
                height=0.7*cm,
                elements=[
                    ObjectValue(attribute_name='id', action=FIELD_ACTION_COUNT,
                        display_format='%s staffs', left=0.5*cm, top=0.1*cm)
                ],
                borders={'top': True},
            ),
        ),
    ]
```

Generating PDF...

```
>>> queryset = User.objects.order_by('is_superuser','is_staff','id')
>>> report = GrouppingReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/groupping-report.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/groupping-report.pdf

1.13.6 SubReports

This is they best way to have a master/detail structure in a report. Its logic is: you have a detail band that shows object details and subreports to show ManyToMany or ManyRelated objects, like children.

This is different from ChildBand. SubReport is to show object child items in the report, not as band childs:

```
>>> import os
>>> cur_dir = os.path.dirname(os.path.abspath(__file__))
```

```
>>> from django.contrib.auth.models import User, Permission

>>> from reportlab.lib.pagesizes import A4
>>> from reportlab.lib.units import cm
>>> from reportlab.lib.enums import TA_CENTER, TA_RIGHT
>>> from reportlab.lib.colors import navy, red

>>> from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
...     FIELD_ACTION_COUNT, FIELD_ACTION_SUM, BAND_WIDTH, Line, ReportGroup,\
...     SubReport

>>> class MasterReport(Report):
...     title = 'Subreports demonstration'
...
...     class band_summary(ReportBand):
...         height = 0.8*cm
...         elements = [
...             Label(text="Users count:", top=0.1*cm, left=0),
...             ObjectValue(attribute_name='id', top=0.1*cm, left=4*cm,\
...                 action=FIELD_ACTION_COUNT, display_format='%s users found'),
...         ]
...         borders = {'top': Line(stroke_color=red, stroke_width=3)}
...
...     class band_page_header(ReportBand):
...         height = 0.8*cm
...         elements = [
...             SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
...                 style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
...             SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
...                 width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
...         ]
...         borders = {'bottom': Line(stroke_color=red, stroke_width=3)}
...
...     class band_page_footer(ReportBand):
...         height = 0.5*cm
...         elements = [
...             Label(text='Created with Geraldo Reports', top=0.1*cm),
...             SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
...                 width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
...         ]
...         borders = {'top': Line(stroke_color=navy)}
...
...     class band_detail(ReportBand):
...         height = 2*cm
...         elements = [
...             Label(text="Username", top=0, left=0, style={'fontName': 'Helvetica-Bold', 'fontSize
...             Label(text="Full name", top=1*cm, left=0.2*cm, style={'fontName': 'Helvetica-Bold'}),
...             Label(text="Superuser", top=1.5*cm, left=0.2*cm, style={'fontName': 'Helvetica-Bold'
...             ObjectValue(attribute_name='username', top=0, left=4*cm, style={'fontName': 'Helvetic
...             ObjectValue(attribute_name='get_full_name', top=1*cm, left=4*cm, style={'fontName':
...             ObjectValue(attribute_name='is_superuser', top=1.5*cm, left=4*cm, style={'fontName':
...         ]
...         borders = {'bottom': Line(stroke_color=navy)}
...
...     subreports = [
...         SubReport(
...             queryset_string = '%(object)s.user_permissions.all()',
...             band_header = ReportBand(
```

```
...                         height=0.5*cm,
...                         elements=[
...                             Label(text='ID', top=0, left=0.2*cm, style={'fontName': 'Helvetica-Bold'
...                             Label(text='Name', top=0, left=4*cm, style={'fontName': 'Helvetica-Bold'
...                         ],
...                         borders={'top': True, 'left': True, 'right': True},
...                     ),
...                 band_detail = ReportBand(
...                         height=0.5*cm,
...                         elements=[
...                             ObjectValue(attribute_name='id', top=0, left=0.2*cm),
...                             ObjectValue(attribute_name='name', top=0, left=4*cm),
...                         ],
...                         borders={'left': True, 'right': True},
...                     ),
...                 band_footer = ReportBand(
...                         height=0.5*cm,
...                         elements=[
...                             ObjectValue(attribute_name='id', left=4*cm,\
...                                 action=FIELD_ACTION_COUNT, display_format='%s permissions found',
...                                 style={'fontName': 'Helvetica-Bold'}),
...                         ],
...                         borders={'bottom': True, 'left': True, 'right': True},
...                     ),
...             ),
...     ]
```

Generating PDF...

```
>>> queryset = User.objects.order_by('username')
>>> report = MasterReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/subreports-report.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/subreports-report.pdf

1.13.7 Inheritance

Inheritance is useful when you have two or more reports sharing common things.

Example: two reports of Users, one with some fields and another with more fields and landscape page orientation:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    BAND_WIDTH, landscape

class UsersReport(Report):
    title = 'Users'
```

```
    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Username", top=0.8*cm, left=3*cm),
            Label(text="First name", top=0.8*cm, left=8*cm),
            Label(text="Last name", top=0.8*cm, left=13*cm),
            Label(text="Staff", top=0.8*cm, left=18*cm),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
        height = 0.7*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0),
            ObjectValue(attribute_name='username', top=0, left=3*cm, display_format='<font size=14 na
            ObjectValue(attribute_name='first_name', top=0, left=8*cm),
            ObjectValue(attribute_name='last_name', top=0, left=13*cm),
            ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
                get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
        ]

class LandscapeUsersReport(UsersReport):
    """An inheritance report class"""

    title = 'Detailed list of users'
    page_size = landscape(A4)

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports - Landscape', top=0.1*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    def __init__(self, *args, **kwargs):
        super(LandscapeUsersReport, self).__init__(*args, **kwargs)

        self.band_page_header.elements += [
            Label(text="Date joined", top=0.8*cm, left=21*cm),
            ]

        self.band_detail.elements += [
            ObjectValue(attribute_name='date_joined', top=0, left=21*cm),
            ]
```

Generating PDF...

```
>>> report = LandscapeUsersReport(queryset=queryset)
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/inheritance-landscape-rep
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/inheritance-landscape-report.pdf

1.13.8 Composition

Composition is useful when you have two or more reports sharing common things.

But there is a big difference from Inheritance: while **inheritance** starts a report from the point where another one ends, **composition** is about you making separate things and afterwards composing a report with them.

Example: two reports of Users, one with some fields and another with more fields and landscape page orientation:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    BAND_WIDTH, landscape

class PageHeaderBand(ReportBand):
    height = 1.3*cm
    elements = [
        SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
            style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
        Label(text="ID", top=0.8*cm, left=0),
        Label(text="Username", top=0.8*cm, left=3*cm),
        Label(text="First name", top=0.8*cm, left=8*cm),
        Label(text="Last name", top=0.8*cm, left=13*cm),
        Label(text="Staff", top=0.8*cm, left=18*cm),
    ]
    borders = {'bottom': True}

class PageFooterBand(ReportBand):
    height = 0.5*cm
    elements = [
        Label(text='Created with Geraldo Reports', top=0.1*cm),
        SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
            width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
    ]
    borders = {'top': True}

class DetailBand(ReportBand):
    height = 0.7*cm
    elements = [
        ObjectValue(attribute_name='id', top=0, left=0),
        ObjectValue(attribute_name='username', top=0, left=3*cm, display_format='<font size=14 name=
        ObjectValue(attribute_name='first_name', top=0, left=8*cm),
        ObjectValue(attribute_name='last_name', top=0, left=13*cm),
```

```
        ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
            get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
    ]

class UsersReport(Report):
    """Report 1 class"""

    title = 'Users'

    band_page_header = PageHeaderBand
    band_page_footer = PageFooterBand
    band_detail = DetailBand

class LandscapeUsersReport(Report):
    """Report 2 class"""

    title = 'Users'
    page_size = landscape(A4)

    band_page_header = PageHeaderBand
    band_page_footer = PageFooterBand

    class band_detail(DetailBand):
        elements = DetailBand.elements+[
            ObjectValue(attribute_name='date_joined', top=0, left=21*cm),
        ]

    def __init__(self, *args, **kwargs):
        super(LandscapeUsersReport, self).__init__(*args, **kwargs)

        self.band_page_header.elements += [
            Label(text="Date joined", top=0.8*cm, left=21*cm),
            ]

        self.band_page_footer.elements[0].text += ' - Landscape'
```

Generating PDF...

```
>>> queryset = User.objects.order_by('id')
>>> report = UsersReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/composition-report.pdf'))
```

Now generating the composed landscape report

```
>>> report = LandscapeUsersReport(queryset=queryset)
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/composition-landscape-rep
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/composition-report.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/composition-landscape-report.pdf

1.13.9 Charts with CairoPlot

This test is done with a chart generated by CairoPlot. You can find it here:

http://linil.wordpress.com/2008/09/16/cairoplot-11/

Charts are another important thing in reports. Geraldo is compatible with every charting library if it has a way to render the chart as a common image format, like JPG, PNG, GIF, etc:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

import Image as PILImage
from CairoPlot import pie_plot, bar_plot

from django.contrib.auth.models import User

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY, TA_RIGHT
from reportlab.lib.colors import navy, yellow, red, white

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    SubReport, FIELD_ACTION_COUNT, FIELD_ACTION_AVG, FIELD_ACTION_MIN,\
    FIELD_ACTION_MAX, FIELD_ACTION_SUM, FIELD_ACTION_DISTINCT_COUNT,\
    BAND_WIDTH, Rect, Line, Image

def get_chart_for_user(graphic):
    """Method to get chart"""
    data = [dic['id'] for dic in graphic.instance.user_permissions.values('id')]

    if not data:
        return None

    filename = os.path.join(cur_dir, 'output/user-chart-%d.png'%graphic.instance.id)

    bar_plot(filename, data, 400, 300, border = 20, grid = True, rounded_corners = True)

    return PILImage.open(filename)

class UsersReport(Report):
    title = 'Using chart from CairoPlot'
    borders = {'all': True}
    default_style = {'fontName': 'Helvetica'}

    class band_summary(ReportBand):
        height = 5*cm
        elements = [
            Label(text="Users count:", top=0.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='username', top=0.1*cm, left=4*cm,\
                action=FIELD_ACTION_COUNT, display_format='%s permissions found'),

            Label(text="Users ids average:", top=0.6*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=0.6*cm, left=4*cm, action=FIELD_ACTION_AVG),

            Label(text="Users ids minimum:", top=1.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=1.1*cm, left=4*cm, action=FIELD_ACTION_MIN),

            Label(text="Users ids maximum:", top=1.6*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=1.6*cm, left=4*cm, action=FIELD_ACTION_MAX),

            Label(text="Users ids sum:", top=2.1*cm, left=0.2*cm),
            ObjectValue(attribute_name='id', top=2.1*cm, left=4*cm, action=FIELD_ACTION_SUM),

            Label(text="Users first name distinct:", top=2.6*cm, left=0.2*cm),
```

```
            ObjectValue(attribute_name='first_name', top=2.6*cm, left=4*cm, action=FIELD_ACTION_DISTI

            Image(filename=os.path.join(cur_dir, 'output/cairoplot.png'), left=11*cm, top=0.2*cm)
        ]
        borders = {'top': True}

    class band_page_footer(ReportBand):
        height = 1*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0, left=0.5*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s  ', top=0,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT, 'rightIndent': 0.5*cm}),
        ]

    class band_detail(ReportBand):
        height = 8.5*cm
        force_new_page = True
        elements = [
            # check why BAND_WIDTH doesn't work XXX
            Rect(width=BAND_WIDTH, height=1.6*cm, left=0, top=0, fill_color=yellow, fill=True, _test_
            ObjectValue(attribute_name='username', width=BAND_WIDTH, left=0.4*cm, top=0.4*cm,
                get_value=lambda instance: instance.get_full_name().strip() or instance.username,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 16, 'textColor': navy}),
            Image(left=0.5*cm, top=2*cm,
                get_image=lambda graphic: PILImage.open(os.path.join(cur_dir, '%d.jpg'%graphic.instan

            Label(text='First name:', top=2*cm, left=5*cm),
            ObjectValue(attribute_name='first_name', width=BAND_WIDTH, top=2*cm, left=7*cm),

            Label(text='Last name:', top=2.5*cm, left=5*cm),
            ObjectValue(attribute_name='last_name', width=BAND_WIDTH, top=2.5*cm, left=7*cm),

            Label(text='Username:', top=3*cm, left=5*cm),
            ObjectValue(attribute_name='username', width=BAND_WIDTH, top=3*cm, left=7*cm),

            Image(left=10.5*cm, top=2*cm, get_image=get_chart_for_user),
        ]
        borders = {'bottom': True}
        child_bands = [
            ReportBand(
                default_style={'textColor': white, 'fontName': 'Helvetica-Bold'},
                height = 0.6*cm,
                elements = [
                    Rect(left=0, top=0, width=BAND_WIDTH, height=0.6*cm,
                        fill_color=navy, fill=True, _test_temp=True),
                    Label(text="ID", top=0.1*cm, left=0.5*cm),
                    Label(text="Permission", top=0.1*cm, left=4*cm),
                ],
                borders={'bottom': True}),
            ]

    subreports = [
        SubReport(
            queryset_string = '%(object)s.user_permissions.all()',
            band_detail = ReportBand(
                    default_style={'fontName': 'Helvetica'},
                    height=0.5*cm,
                    elements=[
```

```
                        ObjectValue(attribute_name='id', top=0, left=0.5*cm),
                        ObjectValue(attribute_name='name', top=0, left=4*cm),
                    ],
                    borders={'bottom': True},
                ),
        ),
    ]
```

Instantiating the report...

```
>>> queryset = User.objects.all().order_by('-id')
>>> report = UsersReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
```

Building the chart

```
>>> data = dict([(user['username'], user['id']) for user in report.queryset.values('id', 'username')]
>>> pie_plot(os.path.join(cur_dir, 'output/cairoplot.png'), data, 350, 200)
```

PDF generation

```
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/charts-cairoplot-report.p
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/charts-cairoplot-report.pdf

1.13.10 Charts with MatPlotLib

This test is done with a chart generated by **matplotlib**, a famous charting library for Python Language. You can get it from

http://matplotlib.sourceforge.net

Charts are another important thing in reports. Geraldo is compatible with every charting library if it has a way to render the chart as a common image format, like JPG, PNG, GIF, etc:

```
>>> import os
>>> cur_dir = os.path.dirname(os.path.abspath(__file__))

>>> from django.contrib.auth.models import User

>>> from reportlab.lib.pagesizes import A4
>>> from reportlab.lib.units import cm
>>> from reportlab.lib.enums import TA_CENTER, TA_JUSTIFY, TA_RIGHT
>>> from reportlab.lib.colors import navy, yellow, red

>>> from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
...     FIELD_ACTION_COUNT, FIELD_ACTION_AVG, FIELD_ACTION_MIN,\
...     FIELD_ACTION_MAX, FIELD_ACTION_SUM, FIELD_ACTION_DISTINCT_COUNT, BAND_WIDTH,\
...     RoundRect, Line, Image

>>> class UsersReport(Report):
...     title = 'Using chart from CairoPlot'
...
...     class band_summary(ReportBand):
...         height = 5*cm
...         elements = [
...             Label(text="Users count:", top=0.1*cm, left=0.2*cm),
```

```
...             ObjectValue(attribute_name='username', top=0.1*cm, left=4*cm,\
...                 action=FIELD_ACTION_COUNT, display_format='%s permissions found'),
...
...             Label(text="Users ids average:", top=0.6*cm, left=0.2*cm),
...             ObjectValue(attribute_name='id', top=0.6*cm, left=4*cm, action=FIELD_ACTION_AVG),
...
...             Label(text="Users ids minimum:", top=1.1*cm, left=0.2*cm),
...             ObjectValue(attribute_name='id', top=1.1*cm, left=4*cm, action=FIELD_ACTION_MIN),
...
...             Label(text="Users ids maximum:", top=1.6*cm, left=0.2*cm),
...             ObjectValue(attribute_name='id', top=1.6*cm, left=4*cm, action=FIELD_ACTION_MAX),
...
...             Label(text="Users ids sum:", top=2.1*cm, left=0.2*cm),
...             ObjectValue(attribute_name='id', top=2.1*cm, left=4*cm, action=FIELD_ACTION_SUM),
...
...             Label(text="Users first name distinct:", top=2.6*cm, left=0.2*cm),
...             ObjectValue(attribute_name='first_name', top=2.6*cm, left=4*cm, action=FIELD_ACTION_D
...
...             Image(filename=os.path.join(cur_dir, 'output/matplotlib.png'), left=11*cm, top=0.2*cm
...         ]
...         borders = {'top': True}
...
...     class band_page_header(ReportBand):
...         height = 1.3*cm
...         elements = [
...             SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
...                 style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER,
...                     'textColor': navy}),
...             Label(text="ID", top=0.8*cm, left=0),
...             Label(text="Username", top=0.8*cm, left=3*cm),
...             Label(text="First name", top=0.8*cm, left=8*cm),
...             Label(text="Last name", top=0.8*cm, left=13*cm),
...             Label(text="Staff", top=0.8*cm, left=18*cm),
...         ]
...         borders = {'bottom': Line(stroke_color=navy)}
...
...     class band_page_footer(ReportBand):
...         height = 0.5*cm
...         elements = [
...             Label(text='Created with Geraldo Reports', top=0.1*cm,
...                 right=0),
...             SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
...                 width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
...         ]
...         borders = {'top': True}
...
...     class band_detail(ReportBand):
...         height = 0.7*cm
...         elements = [
...             ObjectValue(attribute_name='id', top=0, left=0),
...             ObjectValue(attribute_name='username', top=0, left=3*cm, display_format='<font size=1
...             ObjectValue(attribute_name='first_name', top=0, left=8*cm),
...             ObjectValue(attribute_name='last_name', top=0, left=13*cm),
...             ObjectValue(attribute_name='is_staff', top=0, left=18*cm,
...                 get_value=lambda instance: instance.is_staff and 'Yes' or 'No'),
...         ]
```

Instantiating the report...

```
>>> queryset = User.objects.order_by('id')
>>> report = UsersReport(queryset=queryset)
```

Building the chart

```
>>> import pylab
>>> from numpy import numarray
>>>
>>> user_ids = report.queryset.filter(id__in=[1,5,7,12]).values('id','username')
>>> labels = [user['username'] for user in user_ids]
>>> data = [user['id'] for user in user_ids]
>>> left = numarray.array(range(len(data))) + 0.5
>>>
>>> tmp = pylab.bar(left, data, width=0.5)
>>> tmp = pylab.yticks(range(15))
>>> tmp = pylab.xticks(left+0.25, labels)
>>> tmp = pylab.title('Users IDs')
>>> pylab.gca().get_xaxis().tick_bottom()
>>> pylab.gca().get_yaxis().tick_left()
>>> pylab.savefig(os.path.join(cur_dir, 'output/matplotlib.png'), dpi=50)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/charts-matplotlib-report
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/charts-matplotlib-report.pdf

1.13.11 Without Django

This example is done without relation to Django querysets:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH

class SimpleListReport(Report):
    title = 'Demonstration without Django'

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Name", top=0.8*cm, left=3*cm),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
```

```
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
        height = 0.5*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0),
            ObjectValue(attribute_name='name', top=0, left=3*cm),
        ]
```

Now the object class

```
>>> class MyObject(object):
...     def __init__(self, **kwargs):
...         for k,v in kwargs.items():
...             setattr(self, k, v)
```

Populating a list of objects

```
>>> objects_list = [
...     MyObject(id=1, name='Rio de Janeiro'),
...     MyObject(id=2, name='New York'),
...     MyObject(id=3, name='Paris'),
...     MyObject(id=4, name='London'),
...     MyObject(id=5, name='Tokyo'),
...     MyObject(id=6, name='Moscow'),
...     MyObject(id=7, name='Beijing'),
...     MyObject(id=8, name='Hamburg'),
...     MyObject(id=9, name='New Delhi'),
...     MyObject(id=10, name='Jakarta'),
... ]
```

Generating PDF...

```
>>> report = SimpleListReport(queryset=objects_list)
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/without-django.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/without-django-half-height.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/without-django.pdf

1.13.12 Inline Detail Band

This report prints a detail band in "inline display", like an HTML tag, with display: inline and float: left.

It is useful to make labels:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User
```

```
from reportlab.lib.pagesizes import LETTER
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH

class LabelsReport(Report):
    page_size = LETTER
    margin_left = 0.48*cm
    margin_right = 0.48*cm
    margin_top = 1.27*cm
    margin_bottom = 0.5*cm

    class band_detail(ReportBand):
        height = 2.54*cm

        # This is a new attribute to force the band width
        width = 6.67*cm

        # This attribute forces a distance at right side of the band
        margin_right = 0.31*cm

        # With this attribute as True, the band will try to align in
        # the same line
        display_inline = True

        elements = [
            ObjectValue(attribute_name='get_full_name', top=0, left=0, height=0.7*cm,
                get_value=lambda inst: inst.get_full_name() or inst.username,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 13}),
            Label(text='User ID:', top=0.6*cm, left=0),
            ObjectValue(attribute_name='id', top=0.6*cm, left=1.5*cm),
            Label(text='E-mail:', top=1.2*cm, left=0),
            ObjectValue(attribute_name='email', top=1.2*cm, left=1.5*cm),
        ]
```

Generating PDF...

```
>>> report = LabelsReport(queryset=User.objects.order_by('id'))
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/inline-detail-report.pdf
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/inline-detail-report.pdf

1.13.13 Inline SubReports

Inline SubReports can be used to show little boxes in a master/detail report. There are other needs where it can be useful.

This is just the same SubReport test, but with subreport detail band changed to be inline:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from django.contrib.auth.models import User, Permission
```

```
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT
from reportlab.lib.colors import navy, red

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, FIELD_ACTION_SUM, BAND_WIDTH, Line, ReportGroup,\
    SubReport

class MasterReport(Report):
    title = 'Subreports demonstration'

    class band_summary(ReportBand):
        height = 0.8*cm
        elements = [
            Label(text="Users count:", top=0.1*cm, left=0),
            ObjectValue(attribute_name='id', top=0.1*cm, left=4*cm,\
                action=FIELD_ACTION_COUNT, display_format='%s users found'),
        ]
        borders = {'top': Line(stroke_color=red, stroke_width=3)}

    class band_page_header(ReportBand):
        height = 0.8*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'bottom': Line(stroke_color=red, stroke_width=3)}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm),
            SystemField(expression='Printed in %(now:%Y, %b %d)s at %(now:%H:%M)s', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': Line(stroke_color=navy)}

    class band_detail(ReportBand):
        height = 2*cm
        elements = [
            Label(text="Username", top=0, left=0, style={'fontName': 'Helvetica-Bold', 'fontSize': 14
            Label(text="Full name", top=1*cm, left=0.2*cm, style={'fontName': 'Helvetica-Bold'}),
            Label(text="Superuser", top=1.5*cm, left=0.2*cm, style={'fontName': 'Helvetica-Bold'}),
            ObjectValue(attribute_name='username', top=0, left=4*cm, style={'fontName': 'Helvetica',
            ObjectValue(attribute_name='get_full_name', top=1*cm, left=4*cm, style={'fontName': 'Helv
            ObjectValue(attribute_name='is_superuser', top=1.5*cm, left=4*cm, style={'fontName': 'Hel
        ]
        borders = {'bottom': Line(stroke_color=navy)}

    subreports = [
        SubReport(
            queryset_string = '%(object)s.user_permissions.all()',
            band_header = ReportBand(
                    height=0.9*cm,
                    elements=[
```

```
                        Label(text='ID', top=0.2*cm, left=0.2*cm, style={'fontName': 'Helvetica-Bold
                        Label(text='Name', top=0.2*cm, left=4*cm, style={'fontName': 'Helvetica-Bold
                    ],
                ),
            band_detail = ReportBand(
                    height=1.3*cm,

                    # This is a new attribute to force the band width
                    width = 6*cm,

                    # This attribute forces a distance at right and bottom sides of the band
                    margin_right = 0.2*cm,
                    margin_bottom = 0.2*cm,

                    # With this attribute as True, the band will try to align in
                    # the same line
                    display_inline = True,

                    elements=[
                        ObjectValue(attribute_name='id', top=0.1*cm, left=0.2*cm),
                        ObjectValue(attribute_name='name', top=0.7*cm, left=0.2*cm),
                    ],
                    borders={'all': True},
                ),
            band_footer = ReportBand(
                    height=0.9*cm,
                    elements=[
                        ObjectValue(attribute_name='id', top=0.2*cm, left=4*cm,\
                            action=FIELD_ACTION_COUNT, display_format='%s permissions found',
                            style={'fontName': 'Helvetica-Bold'}),
                    ],
                ),
        ),
    ]
```

Generating PDF...

```
>>> queryset = User.objects.order_by('username')
>>> report = MasterReport(queryset=queryset)
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/inline-subreports-report
```

Generating Text...

```
>>> from geraldo.generators import TextGenerator
>>> report.generate_by(TextGenerator, filename=os.path.join(cur_dir,
...     'output/inline-subreports-report.txt'), to_printer=False, encode_to='latin-1')
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/inline-subreports-report.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/inline-subreports-report.txt

1.13.14 Auto expanding band height

This is an example of auto expanding function on bands. This means a band has the height flexible and adapted to its elements. It is useful to print long text data that can be short or long depending on the object:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
     FIELD_ACTION_COUNT, BAND_WIDTH

class SimpleReport(Report):
     title = 'Auto Expanded Bands Report'

     class band_begin(ReportBand):
          height = 1*cm
          elements = [
               Label(text='Look those permissions please', top=0.1*cm,
                    left=8*cm),
          ]

     class band_summary(ReportBand):
          height = 0.7*cm
          elements = [
               Label(text="That's all", top=0.1*cm, left=0),
               ObjectValue(attribute_name='city', top=0.1*cm, left=3*cm,\
                    action=FIELD_ACTION_COUNT, display_format='%s permissions found'),
          ]
          borders = {'all': True}

     class band_page_header(ReportBand):
          height = 1.3*cm
          elements = [
               SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                    style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
               Label(text="ID", top=0.8*cm, left=0),
               Label(text="Name", top=0.8*cm, left=3*cm),
          ]
          borders = {'bottom': True}

     class band_page_footer(ReportBand):
          height = 0.5*cm
          elements = [
               Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
               SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                    width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
          ]
          borders = {'top': True}

     class band_detail(ReportBand):
          height = 0.5*cm
          auto_expand_height = True
          margin_bottom = 0.4*cm
          elements = [
               ObjectValue(attribute_name='city', top=0, left=0),
               ObjectValue(attribute_name='country', top=0, left=5*cm, width=5*cm),
               ObjectValue(attribute_name='about', top=0, left=10*cm, width=8*cm),
          ]
```

Objects list

```
>>> objects_list = [
...     {'city': 'New York', 'country': 'USA', 'about': "New York is the most populous city in the Un
...     {'city': 'London', 'country': 'UK', 'about': "London contains four World Heritage Sites: the
...     {'city': 'Paris', 'country': 'FR', 'about': "An important settlement for more than two mille
...     {'city': 'Moscow', 'country': 'RU', 'about': "A person from Moscow is called a Muscovite in E
... ]
>>>
>>> report = SimpleReport(queryset=objects_list)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/auto-expand-bands-report.
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/auto-expand-bands-report.pdf

1.13.15 Two Reports At Once

Sometimes we need to generate two or more reports in the same file. This example uses the new attributes 'canvas' and 'return_canvas' from PDFGenerator to do that:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from reportlab.lib.enums import TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, SubReport, ReportGroup
from geraldo.base import Element, cm
from geraldo import GeraldoObject, ObjectNotFound, ManyObjectsFound
from geraldo import ObjectValue, SystemField, Label, Line,\
    FIELD_ACTION_COUNT, BAND_WIDTH

class CitiesReport(Report):
    title = 'Cities'

    class band_page_header(ReportBand):
        height = 1.2*cm
        name = 'the-page-header-band'
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="Name", top=0.8*cm, style={'fontName': 'Helvetica-Bold'}),
            Label(text="Country", top=0.8*cm, left=10*cm, style={'fontName': 'Helvetica-Bold'}),
        ]
        borders = {'bottom': True}

    class band_detail(ReportBand):
        height = 0.5*cm
        elements = [
            ObjectValue(attribute_name='city', top=0.1*cm),
            ObjectValue(attribute_name='country', top=0.1*cm, left=10*cm),
        ]

class PeopleReport(Report):
```

```
    title = 'People'

    class band_page_header(ReportBand):
        height = 1.2*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="Name", top=0.8*cm, style={'fontName': 'Helvetica-Bold'}),
        ]
        borders = {'bottom': True}

    class band_detail(ReportBand):
        height = 0.6*cm
        elements = [
            ObjectValue(attribute_name='capitalize'),
        ]
```

Objects list

```
>>> cities_list = [
...     {'city': 'New York', 'country': 'USA'},
...     {'city': 'London', 'country': 'UK'},
...     {'city': 'Paris', 'country': 'FR'},
...     {'city': 'Moscow', 'country': 'RU'},
... ]
```

People list

```
>>> people_list = ['Mary','John','Joseph','Stephen','William','Peter',
...     'Mauri','Jaquez','Francois','Putin','Ivan','Yuri']
```

Report instances

```
>>> report_cities = CitiesReport(queryset=cities_list)
>>> report_people = PeopleReport(queryset=people_list)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
```

Because of a requirement from ReportLab, you must inform the file name when creating the Canvas (in the first report), even if you aren't going to save it really.

```
>>> canvas = report_cities.generate_by(PDFGenerator,
...     filename=os.path.join(cur_dir, 'output/two-reports-at-once.pdf'),
...     return_canvas=True,
...     )
```

Here is the magic

```
>>> report_people.generate_by(PDFGenerator,
...     canvas=canvas,
...     )
```

Text generation

```
>>> from geraldo.generators import TextGenerator
>>>
>>> text = report_cities.generate_by(TextGenerator)
>>> text += report_people.generate_by(TextGenerator)
```

```
>>>
>>> fp = file(os.path.join(cur_dir, 'output/two-reports-at-once.txt'), 'w')
>>> fp.write(text)
>>> fp.close()
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/two-reports-at-once.pdf
- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/two-reports-at-once.txt

1.13.16 Generating under new Process

This is exactly the same report we made for "without-Django" test. But this is going to be ran under an independent Process, from multiprocessing library.

Is important to keep aware on:

- multiprocessing works on Python 2.3 or higher - and is a builtin package on 2.6
- you must use 'generate_under_process_by' instead of 'generate_by'
- you can do it without 'generate_under_process_by', using geraldo.utils.run_under_process decorator, if you prefer do it manually
- the method ''generate_under_process_by' is not the best solution ever. You must keep aware on what kind of report generation you are doing, and how server is configure, but most of cases will work well
- the decorator 'run_under_process' will work only if geraldo.utils.DISABLE_MULTIPROCESSING is False

```
    >>> import os
    >>> cur_dir = os.path.dirname(os.path.abspath(__file__))
```

```
    >>> from geraldo.utils import A4, cm, TA_CENTER, TA_RIGHT
```

```
    >>> from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    ...     FIELD_ACTION_COUNT, BAND_WIDTH
```

Report class

```
>>> class SimpleListReport(Report):
...     title = 'Demonstration without Django'
...
...     class band_page_header(ReportBand):
...         height = 1.3*cm
...         elements = [
...             SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
...                 style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
...             Label(text="ID", top=0.8*cm, left=0),
...             Label(text="Name", top=0.8*cm, left=3*cm),
...         ]
...         borders = {'bottom': True}
...
...     class band_page_footer(ReportBand):
...         height = 0.5*cm
...         elements = [
...             Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
...             SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
...                 width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
...         ]
```

```
...         borders = {'top': True}
...
...     class band_detail(ReportBand):
...         height = 0.5*cm
...         elements = [
...             ObjectValue(attribute_name='id', top=0, left=0),
...             ObjectValue(attribute_name='name', top=0, left=3*cm),
...         ]
```

```
>>> class MyObject(object):
...     def __init__(self, **kwargs):
...         for k,v in kwargs.items():
...             setattr(self, k, v)
```

```
>>> objects_list = [
...     MyObject(id=1, name='Rio de Janeiro'),
...     MyObject(id=2, name='New York'),
...     MyObject(id=3, name='Paris'),
...     MyObject(id=4, name='London'),
...     MyObject(id=5, name='Tokyo'),
...     MyObject(id=6, name='Moscow'),
...     MyObject(id=7, name='Beijing'),
...     MyObject(id=8, name='Hamburg'),
...     MyObject(id=9, name='New Delhi'),
...     MyObject(id=10, name='Jakarta'),
... ]
```

```
>>> report = SimpleListReport(queryset=objects_list)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
```

```
>>> report.generate_under_process_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/generated-
```

1.13.17 Using Events System

This example uses events to make customizations and also print messages while is printing:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from geraldo.utils import A4, cm, TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH

def element_before_print(self, generator):
    """Element with before print event - hides if the text is 'New York'"""
    self.visible = self.text != 'New York'

def element_after_print(self, generator):
    """Element with after print event, that works"""
    print 'After print city %s'%self.text

class SimpleListReport(Report):
    title = 'Demonstration without Django'
```

```
    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica-Bold', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Name", top=0.8*cm, left=3*cm),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
        height = 0.5*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0,),
            ObjectValue(attribute_name='name', top=0, left=3*cm,
                # Element with before print event - hides if the text is 'New York'
                before_print=element_before_print,

                #Element with after print event, that works
                after_print=element_after_print,
                )
        ]

    # Report events overring methods

    def do_before_print(self, generator):
        """Report before print event"""
        self.title = '%s - %s objects'%(self.title, len(objects_list))

    def do_before_generate(self, generator):
        """Report before generate event"""
        print 'Starting generating!'

    def do_after_print(self, generator):
        """Report after print event"""
        print 'Printing finished!'

    def do_on_new_page(self, page, page_number, generator):
        """Report on new page event"""
        print 'Printing page # %s!'%page_number
```

Important: if you are setting events as methods, you must name them as "do_" plus event name (i.e. do_before_print). Otherwise, if you are setting them as attributes (setting when creating objects or setting them dinamically) you must name them without "do_", just their names (i.e. before_print).

Executing for a list of objects (list of cities)...

```
>>> class MyObject(object):
...     def __init__(self, **kwargs):
```

```
...         for k,v in kwargs.items():
...             setattr(self, k, v)
```

```
>>> objects_list = [
...     MyObject(id=1, name='Rio de Janeiro'),
...     MyObject(id=2, name='New York'),
...     MyObject(id=3, name='Paris'),
...     MyObject(id=4, name='London'),
...     MyObject(id=5, name='Tokyo'),
...     MyObject(id=6, name='Moscow'),
...     MyObject(id=7, name='Beijing'),
...     MyObject(id=8, name='Hamburg'),
...     MyObject(id=9, name='New Delhi'),
...     MyObject(id=10, name='Jakarta'),
... ]
```

```
>>> report = SimpleListReport(queryset=objects_list)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
```

```
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/generated-with-events.pdf
```

1.13.18 Additional Fonts

This test is about support additional fonts but those supported by default by PDF and ReportLab:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from geraldo.utils import A4, cm, TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH

# Report class

class SimpleListReport(Report):
    title = 'Demonstration without Django'
    additional_fonts = {
        'HandGotDLig': os.path.join(cur_dir, 'handgotl.ttf'),  # full path to font file
        'Footlight MT Light': os.path.join(cur_dir, 'ftltlt.ttf'),
    }
    default_style = {'fontName': 'Footlight MT Light'}

    class band_page_header(ReportBand):
        height = 1.3*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'HandGotDLig', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text="ID", top=0.8*cm, left=0),
            Label(text="Name", top=0.8*cm, left=3*cm),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
```

```
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
        height = 0.5*cm
        elements = [
            ObjectValue(attribute_name='id', top=0, left=0, name='other-field-with-event'),
            ObjectValue(attribute_name='name', top=0, left=3*cm, name='field-with-event'),
        ]

class MyObject(object):
    def __init__(self, **kwargs):
        for k,v in kwargs.items():
            setattr(self, k, v)

objects_list = [
    MyObject(id=1, name='Rio de Janeiro'),
    MyObject(id=2, name='New York'),
    MyObject(id=3, name='Paris'),
    MyObject(id=4, name='London'),
    MyObject(id=5, name='Tokyo'),
    MyObject(id=6, name='Moscow'),
    MyObject(id=7, name='Beijing'),
    MyObject(id=8, name='Hamburg'),
    MyObject(id=9, name='New Delhi'),
    MyObject(id=10, name='Jakarta'),
]

report = SimpleListReport(queryset=objects_list)

# PDF generation

from geraldo.generators import PDFGenerator

report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/additional-fonts.pdf'))
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/additional-fonts.pdf

1.13.19 Showing BarCodes

Bar codes are really important on labels, bills, tickets, etc.

This example demonstrates how to show barcodes of all supported types on a report.

There are many different types of bar codes and we will support those ReportLab already does.

BarCode element

geraldo.barcodes.BarCode is the graphic class that generates the bar codes of all supported types. It is an inheritance from Graphic element.

Supported bar code types:

- Codabar
- Code11
- Code128
- EAN13
- EAN8
- Extended39
- Extended93
- FIM
- I2of5
- MSI
- POSTNET
- Standard39
- Standard93
- USPS_4State

The code:

```
import os
cur_dir = os.path.dirname(os.path.abspath(__file__))

from geraldo.utils import A4, cm, TA_CENTER, TA_RIGHT

from geraldo import Report, ReportBand, Label, ObjectValue, SystemField,\
    FIELD_ACTION_COUNT, BAND_WIDTH
from geraldo.barcodes import BarCode

class SimpleListReport(Report):
    title = 'Demonstration of BarCodes'

    class band_page_header(ReportBand):
        height = 1*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica', 'fontSize': 14, 'alignment': TA_CENTER}),
        ]
        borders = {'bottom': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

    class band_detail(ReportBand):
        height = 15*cm
        elements = [
            ObjectValue(attribute_name='name', style={'fontSize': 12}),
            Label(text='Code:', style={'fontSize': 12}, left=6.5*cm),
```

```
            ObjectValue(attribute_name='code', style={'fontSize': 12}, left=8*cm),

            Label(text='Codabar', top=0.6*cm),
            BarCode(type='Codabar', attribute_name='code', top=1.2*cm, height=1.5*cm),

            Label(text='Code11', top=0.6*cm, left=6*cm),
            BarCode(type='Code11', attribute_name='code', top=1.2*cm, left=6*cm, height=1.5*cm),

            Label(text='Code128', top=0.6*cm, left=12*cm),
            BarCode(type='Code128', attribute_name='code', top=1.2*cm, left=12*cm, height=1.5*cm),

            Label(text='EAN13', top=3.2*cm),
            BarCode(type='EAN13', attribute_name='code', top=3.8*cm, height=1.5*cm),

            Label(text='EAN8', top=3.2*cm, left=6*cm),
            BarCode(type='EAN8', attribute_name='code', top=3.8*cm, left=6*cm, height=1.5*cm),

            Label(text='Extended39', top=3.2*cm, left=12*cm),
            BarCode(type='Extended39', attribute_name='code', top=3.8*cm, left=12*cm, height=1.5*cm),

            Label(text='Extended93', top=5.8*cm),
            BarCode(type='Extended93', attribute_name='code', top=6.4*cm, height=1.5*cm),

            Label(text='USPS FIM (code: "A")', top=5.8*cm, left=6*cm),
            BarCode(type='FIM', attribute_name='code', top=6.4*cm, left=8*cm, height=1.5*cm,
                get_value=lambda inst: 'A'),

            Label(text='I2of5', top=5.8*cm, left=12*cm),
            BarCode(type='I2of5', attribute_name='code', top=6.4*cm, left=12*cm, height=1.5*cm),

            Label(text='MSI', top=8.4*cm),
            BarCode(type='MSI', attribute_name='code', top=9*cm, height=1.5*cm),

            Label(text='POSTNET', top=8.4*cm, left=6*cm),
            BarCode(type='POSTNET', attribute_name='code', top=9*cm, left=6*cm, height=1.5*cm),

            Label(text='Standard39', top=8.4*cm, left=12*cm),
            BarCode(type='Standard39', attribute_name='code', top=9*cm, left=12*cm, height=1.5*cm),

            Label(text='Standard93', top=11*cm),
            BarCode(type='Standard93', attribute_name='code', top=11.6*cm, height=1.5*cm),

            Label(text='USPS_4State / code: 01234567094987654321 / routing:', top=11*cm,
                left=6*cm, width=10*cm),
            ObjectValue(attribute_name='routing', top=11.5*cm, left=14*cm),
            BarCode(type='USPS_4State', attribute_name='code', top=11.6*cm, left=6*cm,
                height=1.5*cm, routing_attribute='routing',
                get_value=lambda inst: '01234567094987654321'),
        ]
        borders = {'bottom': True}
```

Creating a list of objects we are going to list on on the report

```
>>> objects_list = [
...     dict(id=1, name='Arduino Duemilanove', code='123456789', routing='01234567891'),
...     dict(id=2, name='Arduino Diecimila', code='000000000', routing='01234567891'),
...     dict(id=3, name='Robotduino', code='111111111', routing='01234567891'),
... ]
```

Creating an instance of the report

```
>>> report = SimpleListReport(queryset=objects_list)
```

PDF generation

```
>>> from geraldo.generators import PDFGenerator
>>> report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/report-with-barcodes.pdf
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/report-with-barcodes.pdf

1.13.20 Cross-Reference Tables

Cross-reference is the type of a report that shows a matrix of XYZ relations, where X and Y are groupping fields and Z is a field with a summary value of relation between X and Y.

Z field could be just the first/once value of a relation, or could be an aggregation, like COUNT, DISTINCT COUNT, SUM, MAX, MIN, AVERAGE or a CONCATENATION, depending on its data type.

So, just to keep the things clear:

- X = **row_attribute** = is the row groupper attribute
- Y = **col_attribute** = is the column groupper attribute
- Z = **cell** = is the relation between X and Y. It is an aggregation or concatenation

Example:

```
# We have a list of "Cities" in "States" where someones are the "Capitals" and
# others are not. Additionally, there are their respective "Government Types",
# "Populations" and "Areas"

from geraldo import Report, ObjectValue, Label, ReportBand, SystemField,\
    BAND_WIDTH, FIELD_ACTION_COUNT, DetailBand, ManyElements, CROSS_COLS
from geraldo.cross_reference import CrossReferenceMatrix
from geraldo.utils import cm, TA_RIGHT, TA_CENTER

class MyReport(Report):
    title = 'Population per state and their capitais/other cities'

    class band_page_header(ReportBand):
        height = 1.7*cm
        elements = [
            SystemField(expression='%(report_title)s', top=0.1*cm, left=0, width=BAND_WIDTH,
                style={'fontName': 'Helvetica', 'fontSize': 14, 'alignment': TA_CENTER}),
            Label(text='Capital', width=2.5*cm, top=1*cm),
            ManyElements(
                element_class=Label,
                count=CROSS_COLS,
                start_left=4*cm,
                width=2*cm,
                top=1*cm,
                text=CROSS_COLS,
                ),
            Label(text='Average', left=17*cm, width=3.5*cm, top=1*cm),
        ]
        borders = {'bottom': True}
```

```
    class band_detail(DetailBand):
        height=0.6*cm
        elements = [
            ObjectValue(attribute_name='row', width=2.5*cm),
            ManyElements( # Make one ObjectValue for each column and shows the max aggregation
                element_class=ObjectValue,
                count=CROSS_COLS,
                start_left=4*cm,
                width=2*cm,
                attribute_name=CROSS_COLS,
                get_value=lambda self, inst: inst.sum('population', self.attribute_name),
                ),
            ObjectValue( # Calculates the average of this row
                attribute_name='row',
                left=17*cm,
                width=3.5*cm,
                get_value=lambda inst: inst.avg('population'),
                ),
        ]

    class band_summary(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Totals', width=2.5*cm),
            ManyElements(
                element_class=ObjectValue,
                count=CROSS_COLS,
                start_left=4*cm,
                width=2*cm,
                attribute_name=CROSS_COLS,
                get_value=lambda self, inst: inst.matrix.sum('population', col=self.attribute_name),
                ),
            ObjectValue(
                attribute_name='row',
                left=17*cm,
                width=3.5*cm,
                get_value=lambda inst: inst.matrix.avg('population'),
                ),
        ]
        borders = {'top': True}

    class band_page_footer(ReportBand):
        height = 0.5*cm
        elements = [
            Label(text='Created with Geraldo Reports', top=0.1*cm, left=0),
            SystemField(expression='Page # %(page_number)d of %(page_count)d', top=0.1*cm,
                width=BAND_WIDTH, style={'alignment': TA_RIGHT}),
        ]
        borders = {'top': True}

cities = [
    {'city': 'New York City', 'state': 'NY', 'capital': False, 'population': 8363710, 'area': 468.9,
    {'city': 'Albany', 'state': 'NY', 'capital': True, 'population': 95658, 'area': 21.8, 'government
    {'city': 'Austin', 'state': 'TX', 'capital': True, 'population': 757688, 'area': 296.2, 'governme
    {'city': 'Dallas', 'state': 'TX', 'capital': False, 'population': 1279910, 'area': 385.0, 'gover
    {'city': 'Houston', 'state': 'TX', 'capital': False, 'population': 2242193, 'area': 601.7, 'gove
    {'city': 'San Francisco', 'state': 'CA', 'capital': False, 'population': 808976, 'area': 231.92,
    {'city': 'Los Angeles', 'state': 'CA', 'capital': False, 'population': 3833995, 'area': 498.3, '
```

```
    {'city': 'Sacramento', 'state': 'CA', 'capital': True, 'population': 463794, 'area': 99.2, 'gover
    {'city': 'Seattle', 'state': 'WA', 'capital': False, 'population': 602000, 'area': 142.5, 'goverr
]

cross = CrossReferenceMatrix(
    objects_list=cities,
    rows_attribute='capital',
    cols_attribute='state',
)

if __name__ == '__main__':
    # Generating the result report
    report = MyReport(queryset=cross)

    # PDF generation
    import os
    cur_dir = os.path.dirname(os.path.abspath(__file__))
    from geraldo.generators import PDFGenerator
    report.generate_by(PDFGenerator, filename=os.path.join(cur_dir, 'output/cross-reference-table.pdf
```

The Result

- http://geraldo.svn.sourceforge.net/viewvc/geraldo/examples/cross-reference-table.pdf

1.14 Backward Incompatible Changes

This section is to keep users informed about things that have been deprecated in favour of design improvements.

1.14.1 1. Attribute 'detail_band' on class SubReport changed to 'band_detail'

Date: 2009-04-16 **Version:** 0.3.0-alpha-6 **Reason:** class Report already had a 'band_*' signature on this kind of attribute

1.15 Next Steps

Geraldo Reports is an ambitious project. We want to provide much more than we have done so far. See below the main goals we have to do in the future (and maybe you could help us to reach them early)

1.15.1 Engine API

- **Table band** - a band to make it easy to create lists just using tables instead of detail band and lines. Table band must be an alternative to detail band
- Support canvas draw

1.15.2 Generators

- Generate in ODF formats (ODT and ODS)
- Generate in HTML

- Export/import structure to/from an XML format

1.15.3 Third-party tools

- Have a GUI tool to design reports, if possible to be used in a web browser
- Have a way to create and print reports using a server, with no low level coding
- **Django pluggable application to create fast reports** (currently in development)
- Have a preview component for GUI applications (GTK/Qt/wxPython)

1.15.4 Other features

- A better support to images and graphics
- **Map/Reduce generating** - split queryset, generate parts in parallel and combine them after all - on one of both ways: local or distributed.
- **Incremental/asynchronous generating** - just take advantage of common part and update the changed other part, specially useful when the order is datetime-based.
- **Drill down reports** - a link from a report "A" to a report "B"
- Template system

1.16 Authors

Geraldo Reports was created and is maintained by **Marinho Brandão** <marinho at gmail dot com>.

There are others guys who helped to enhance Geraldo to get better, we can feature:

- Miltinho Brandão

  The partner since the first day. Made lots of tests and found many bugs that we could fix them and made Geraldo better.

- Ari Caldeira

  Commited patches, including events system.

- Fran Boon

  Sent improvements on documentation and made tutorial for Web2Py.

CHAPTER 2

Features

Geraldo does most of what any reports engine does. It provides the following features:

- A detail band, the report driver
- Page header, Page footer, Report begin and Report summary bands
- Grouping in multiple levels
- SubReports to print childs
- Multi-formats compatible in output (even if we only have PDF for a while)
- Child bands, to attach helper bands to other ones
- Support Django or any other Python framework/application
- Graphic elements (including images and shapes)
- System fields (to show report title, page numbers, current date/time, etc.)
- Reports composition (you can reuse your bands for other reports)
- Reports inheritance (you can use the same structure for a lot of reports)
- A powerful stylizing support (colors, fonts, sizes, etc.)
- Different page sizes
- Flexibility to be easily customized
- Aggregation functions
- Hiding/showing bands and elements
- Events system

CHAPTER 3

Dependencies

Geraldo is fully dependent on ReportLab, not only because of its PDF generation, but also its units and styles library. We have tested with ReportLab version 2.1

Geraldo is not Django-dependent, but it can work as a Django pluggable application.

CHAPTER 4

A very little demo of how Geraldo works

```
from geraldo import Report, DetailBand, ObjectValue
from geraldo.utils import cm
from geraldo.generators import PDFGenerator

names = ['Mychelle', 'Leticia', 'Tarsila', 'Marta', 'Vera', 'Leni']

class MyReport(Report):
    class band_detail(DetailBand):
        height=0.7*cm
        elements=[
            ObjectValue(attribute_name='capitalize'),
            ]

report = MyReport(queryset=names)
report.generate_by(PDFGenerator, filename='female-names.pdf')
```

This code above will output your report, driven by the queryset 'objects_list', into file 'cars_list.pdf'.

Now, an example in a Django view:

```
from django.http import HttpResponse
from geraldo import Report, DetailBand, ObjectValue
from geraldo.generators import PDFGenerator
from geraldo.utils import cm

class MyReport(Report):
    class band_detail(DetailBand):
        height=0.7*cm
        elements=[
            ObjectValue(attribute_name='username'),
            ]

def my_view(request):
    response = HttpResponse(mimetype='application/pdf')

    objects_list = User.objects.all() # If you are using Django
    report = MyReport(queryset=objects_list)

    report.generate_by(PDFGenerator, filename=response)

    return response
```

As you can see, now returned the PDF output to HttpResponse, instead of a file.

CHAPTER 5

License

Geraldo is under **Lesser Gnu Public License**, take a look on the following URL to read it:

http://www.gnu.org/copyleft/lesser.html

CHAPTER 6

Authors

Geraldo has been created by **Marinho Brandao** a brazilian Django and Python enthusiast.

Home page: **http://marinhobrandao.com/**

E-mail: **marinho at gmail dot com**