PyPerFin

v0.1

Generated by Doxygen 1.8.1.1

Sun May 19 2013 18:23:30

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Class Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 CreateTables.py Namespace Reference

CreateTables is a API wrapper library to support list based table entries.

4.1.1 Detailed Description

CreateTables is a API wrapper library to support list based table entries. It provides a wrapper utility library over TableFactory.py for easily converting lists and arryas into tables.

4.2 misc_utils.py Namespace Reference

This module contains miscellaneous utility functions.

4.2.1 Detailed Description

This module contains miscellaneous utility functions. Utility functions defined here are:

watermark_pdf

add_footer_pdf

merge_pdf

find_curr_month

comma_sep

split_month

get_input

save_proj

load_proj

Watermark the pdf

4.3 PyPerFin_app.py Namespace Reference

This is the application file for PyPerFin tool.

4.3.1 Detailed Description

This is the application file for PyPerFin tool. Application , argument parsing and report generation wrappers are defined here

4.4 PyPerFin_classes.py Namespace Reference

Fundamental classes for the PyPerFin tool.

4.4.1 Detailed Description

Fundamental classes for the PyPerFin tool. This module contains all the base classes and their corresponding methods needed for the storage and maintenance of data under various categories

4.5 TableFactory.py Namespace Reference

TableFactory is a high-level frontend to several table generators.

4.5.1 Detailed Description

TableFactory is a high-level frontend to several table generators. It provides a common API for creating HTML, PDF, or spreadsheet tables from common Python data sources. For example, this is a working example on my development system:

This creates a row with two columns:

rowmaker = RowSpec(ColumnSpec('customer', 'Customer'), ColumnSpec('invamt', 'Invoice Amount')) Fetch 10 invoices from our database and convert them to TableRow objects

lines = rowmaker.makeall(session.query(Invoice).limit(10))

Make a PDF out of those lines:

table1 = PDFTable('Invoice amounts by customer', headers=rowmaker) open('invoicetable.pdf', 'wb').write(table1.render(lines))

Want to make a spreadsheet from the same data? The API is identical:

table2 = SpreadsheetTable('Invoice amounts by customer', headers=rowmaker)

open('invoicetable.xls', 'wb').write(table2.render(lines))

Inside a Pyramid view callable and want to create an HTML table that can be rendered in a template? It's exactly like the first two examples:

table3 = HTMLTable('Invoice amounts by customer', headers=rowmaker)

return {'tablecontents': table3.render(lines)}

Chapter 5

Class Documentation

5.1 TableFactory.Cell Class Reference

Public Member Functions

- def __init__
- def __repr__

Public Attributes

- **value**
- **style**

5.1.1 Detailed Description

```
Cell objects represent a single table cell
```

Definition at line 118 of file TableFactory.py.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 def TableFactory.Cell.__init__ (*self, value, style* = None)

```
'value' is the displayed value of the cell. 'properties' is
a dict of cell styles that each table generator may interpret
as appropriate.
```

Definition at line 121 of file TableFactory.py.

```
00121
00122     def __init__(self, value, style=None):
00123         """'value' is the displayed value of the cell. 'properties' is
00124         a dict of cell styles that each table generator may interpret
00125         as appropriate."""
00126
00127         self.value = value
00128         if style is None:
00129             self.style = StyleAttributes()
00130         else:
00131             self.style = style
```

5.1.3 Member Function Documentation

5.1.3.1 def TableFactory.Cell.__repr__ (*self*)

```
Human-readable Cell representation
```

Definition at line 132 of file TableFactory.py.

```
00132
00133     def __repr__(self):
00134         """Human-readable Cell representation"""
00135         return '<Cell(%s)>' % self.value
00136
```

The documentation for this class was generated from the following file:

- TableFactory.py

5.2 TableFactory.ColumnSpec Class Reference

Public Member Functions

- def __init__
- def __repr__

Public Attributes

- **attributes**
- **title**
- **style**

5.2.1 Detailed Description

```
A ColumnSpec describes the source of values for a particular
column, as well as the properties of each of its cells
```

Definition at line 154 of file TableFactory.py.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 def TableFactory.ColumnSpec.__init__ (*self, attribute, title* = None, *properties*)

```
'attribute' is the name of the attribute or dictionary key
that will be pulled from a row object to find a cell's
value. If 'attribute' is a tuple, each of its elements will be
resolved in turn, recursively. For example, an attribute tuple
of ('foo', 'bar', 'baz') might resolve to:

>>> rowobject['foo'].bar['baz']

If this ColumnSpec is printed as part of a table header it
will be captioned with 'title', which defaults to the value of
'attribute'. Any properties will be applied to cells created
by this ColumnSpec.
```

Definition at line 158 of file TableFactory.py.

```
00158
00159     def __init__(self, attribute, title=None, **properties):
00160         """'attribute' is the name of the attribute or dictionary key
00161         that will be pulled from a row object to find a cell's
00162         value. If 'attribute' is a tuple, each of its elements will be
00163         resolved in turn, recursively. For example, an attribute tuple
00164         of ('foo', 'bar', 'baz') might resolve to:
00165
00166         >>> rowobject['foo'].bar['baz']
00167
00168         If this ColumnSpec is printed as part of a table header it
00169         will be captioned with 'title', which defaults to the value of
00170         'attribute'. Any properties will be applied to cells created
00171         by this ColumnSpec."""
00172
00173         if isinstance(attribute, tuple):
00174             self.attributes = attribute
00175         else:
00176             self.attributes = (attribute,)
00177         if title:
00178             self.title = title
00179         else:
00180             self.title = attribute
00181         self.style = StyleAttributes(**properties)
```

5.2.3 Member Function Documentation

5.2.3.1 def TableFactory.ColumnSpec.__repr__(*self*)

Human-readable ColumnSpec representation

Definition at line 182 of file TableFactory.py.

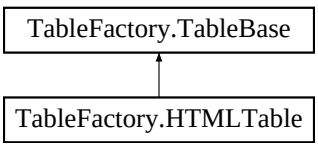
```
00182
00183     def __repr__(self):
00184         """Human-readable ColumnSpec representation"""
00185         return '<ColumnSpec(%s)>' % self.title
00186
```

The documentation for this class was generated from the following file:

- TableFactory.py

5.3 TableFactory.HTMLTable Class Reference

Inheritance diagram for TableFactory.HTMLTable:



Public Member Functions

- def render

Static Public Attributes

- dictionary **cssdefs**

Additional Inherited Members

5.3.1 Detailed Description

```
Table generator that yields an HTML representation of the
data. Note that this class yields *only* the table itself and not
an entire HTML document.

The CSS classes are compatible with jQuery's tablesorter plugin
<http://tablesorter.com/docs/>. With this combination, all
generated tables can be sorted in a client's browser just by
clicking on the column headers.

When a rowset is made of multiple TableRow objects, all rows after
the first are additionally assigned the 'childrow' CSS class. This
adds compatibility with the "Children Rows" mod to tablesorter
<http://www.pengoworks.com/workshop/jquery/tablesorter/tablesorter.htm>,
which groups child rows with their parent rows when sorting.

For example, the following lines in a page's <head> section will
enable all of those client-side options:

    <script type="text/javascript" src="/javascript/jquery-1.5.min.js"></script>
    <script type="text/javascript" src="/javascript/jquery.tablesorter.min.js"></script>
    <script type="text/javascript" src="/javascript/jquery.tablesorter.mod.js"></script>
    <script type="text/javascript">
    $(document).ready(function()
        {
            $(".reporttable").tablesorter({widgets: ['zebra']});
        }
    );
```

Definition at line 491 of file TableFactory.py.

5.3.2 Member Function Documentation

5.3.2.1 def TableFactory.HTMLTable.render (*self, rowsets*)

```
Return the data as a string of HTML
```

Definition at line 549 of file TableFactory.py.

```
00549
00550     def render(self, rowsets):
00551         """Return the data as a string of HTML"""
00552         lines = []
00553
00554         # Display the title, if given
00555         if self.title:
00556             lines.append('<h2>%s</h2>' % self.title)
00557
00558         # Display the explanation, if given
00559         if self.explanation:
00560             lines.append('<p>%s</p>' % self.explanation)
00561
00562         # Create the table
00563         if self.title:
00564             lines.append('<table summary="%s" class="%s">' % (self.title,
      self.cssdefs['table']))
00565         else:
00566             lines.append('<table class="%s">' % self.cssdefs['table'])
00567
00568         # Generate any header rows
00569         if self.headers:
00570             lines.append('  <thead>')
00571             for headerrow in self.headers:
00572                 lines.append('    <tr>')
00573                 for headercolumn in headerrow:
00574                     span = headercolumn.style.span
00575                     if span > 1:
00576                         lines.append('      <th colspan="%d">%s</th>' % (span,
      headercolumn.title))
```

```
00577                     else:
00578                         lines.append('      <th>%s</th>' % headercolumn.title)
00579                 lines.append('    </tr>')
00580 
00581             lines.append('  </thead>')
00582         lines.append('  <tbody>')
00583 
00584         # Write every line
00585         for rowsetindex, rowset in enumerate(rowsets):
00586             if isinstance(rowset, TableRow):
00587                 rowset = [rowset]
00588             for subrowindex, subrow in enumerate(rowset):
00589                 trclasses = [self.cssdefs['zebra'][rowsetindex % 2]]
00590                 if subrowindex:
00591                     trclasses.append(self.cssdefs['childrow'])
00592                 lines.append('    <tr class="%s">' % ' '.join(trclasses))
00593                 for cell in subrow:
00594                     lines.append('      %s' % self._rendercell(cell)
      )
00595                 lines.append('    </tr>')
00596 
00597         # Finish up
00598         lines.append('  </tbody>')
00599         lines.append('</table>')
00600         return '\n'.join(lines)
00601 
```

5.3.3 Member Data Documentation

5.3.3.1 dictionary TableFactory.HTMLTable.cssdefs [static]

Initial value:

```
{
        'bold': 'cell_bold',
        'money': 'cell_money',
        'table': 'reporttable',
        'childrow': 'expand-child',
        'zebra': ('odd', 'even'),
        }
```

Definition at line 522 of file TableFactory.py.

The documentation for this class was generated from the following file:

- TableFactory.py

5.4 PyPerFin_classes.INCOME Class Reference

INCOME.

Public Member Functions

- def __init__

  The constructor.
- def get_total

  get_total

Public Attributes

- **name**
- **salary**
- **dividend**

- **interest**
- **share_trxn**
- **bonus**
- **total_val**

5.4.1 Detailed Description

INCOME.

Defines the list of items from where you earn your money. Used the class "INCOME_CATEGORY"

Definition at line 78 of file PyPerFin_classes.py.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 def PyPerFin_classes.INCOME.__init__ (*self, name*)

The constructor.

Definition at line 80 of file PyPerFin_classes.py.

```
00080
00081     def __init__(self, name):
00082         self.name = name
00083         self.salary = INCOME_CATEGORY('salary')
00084         self.dividend = INCOME_CATEGORY('dividend')
00085         self.interest = INCOME_CATEGORY('interest')
00086         self.share_trxn = INCOME_CATEGORY('share_trxn'
    )
00087         self.bonus = INCOME_CATEGORY('bonus')
00088         self.total_val = 0.0
```

5.4.3 Member Function Documentation

5.4.3.1 def PyPerFin_classes.INCOME.get_total (*self*)

get_total

Parameters

self	The object pointer.

Definition at line 91 of file PyPerFin_classes.py.

```
00091
00092     def get_total(self):
00093         self.total_val = self.salary.total_val + \
00094                          self.dividend.total_val + \
00095                          self.interest.total_val + \
00096                          self.share_trxn.total_val + \
00097                          self.bonus.total_val
00098
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.5 PyPerFin_classes.INCOME_CATEGORY Class Reference

INCOME_CATEGORY.

Public Member Functions

- def __init__

  The constructor.
- def add_incm

  add_incm

Public Attributes

- **name**
- **val**
- **src_of_incm**
- **dt_of_incm**
- **total_val**

5.5.1 Detailed Description

INCOME_CATEGORY.

Defines the basic underlying class for incomes

Definition at line 53 of file PyPerFin_classes.py.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 def PyPerFin_classes.INCOME_CATEGORY.__init__ (*self, name*)

The constructor.

Definition at line 56 of file PyPerFin_classes.py.

```
00056
00057     def __init__(self, name):
00058         self.name = name
00059         self.val = []
00060         self.src_of_incm = []
00061         self.dt_of_incm = []
00062         self.total_val = 0.0
```

5.5.3 Member Function Documentation

5.5.3.1 def PyPerFin_classes.INCOME_CATEGORY.add_incm (*self, val, src_of_incm, dt_of_incm*)

add_incm

Parameters

self	The object pointer.
val	Amount to be added to the category
src_of_incm	Name of the source of the incomes
dt_of_incm	Date of the income

Definition at line 68 of file PyPerFin_classes.py.

```
00068
00069     def add_incm (self, val, src_of_incm, dt_of_incm):
00070         self.val.append(val)
00071         self.src_of_incm.append(src_of_incm)
```

```
00072         self.dt_of_incm.append(dt_of_incm)
00073         self.total_val = self.total_val + val
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.6 PyPerFin_classes.INV_TYPE_CATEGORY Class Reference

INV_TYPE_CATEGORY.

Public Member Functions

- def __init__

  The constructor.
- def get_total_bal

  get_total_bal

Public Attributes

- **name**
- **total_bal**
- **no_items**
- **item1**
- **item2**
- **item3**
- **item4**
- **item5**
- **item6**
- **item7**
- **item8**
- **item9**
- **item10**

5.6.1 Detailed Description

INV_TYPE_CATEGORY.

Defines the list of items from where you invest your money. Used the class "INV_TYPE_SUB_CATEGORY"

Definition at line 149 of file PyPerFin_classes.py.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 def PyPerFin_classes.INV_TYPE_CATEGORY.__init__ (*self, name*)

The constructor.

Definition at line 152 of file PyPerFin_classes.py.

```
00152
00153     def __init__(self, name):
00154         self.name = name
00155         self.total_bal = 0.0
00156         self.no_items = 0
00157         self.item1 = INV_TYPE_SUB_CATEGORY('item1')
00158         self.item2 = INV_TYPE_SUB_CATEGORY('item2')
00159         self.item3 = INV_TYPE_SUB_CATEGORY('item3')
00160         self.item4 = INV_TYPE_SUB_CATEGORY('item4')
00161         self.item5 = INV_TYPE_SUB_CATEGORY('item5')
00162         self.item6 = INV_TYPE_SUB_CATEGORY('item6')
00163         self.item7 = INV_TYPE_SUB_CATEGORY('item7')
00164         self.item8 = INV_TYPE_SUB_CATEGORY('item8')
00165         self.item9 = INV_TYPE_SUB_CATEGORY('item9')
00166         self.item10 = INV_TYPE_SUB_CATEGORY('item10'
      )
```

5.6.3 Member Function Documentation

5.6.3.1 def PyPerFin_classes.INV_TYPE_CATEGORY.get_total_bal (*self*)

get_total_bal

Parameters

self	The object pointer.

Definition at line 169 of file PyPerFin_classes.py.

```
00169
00170     def get_total_bal(self):
00171         self.total_bal = 0.0
00172         self.total_bal = self.total_bal + self.item1.val +
      self.item2.val + self.item3.val + self.item4.val + self.item5.val +
      self.item6.val + self.item7.val + self.item8.val + self.item9.val + self.item10.val
00173
00174
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.7 PyPerFin_classes.INV_TYPE_SUB_CATEGORY Class Reference

INV_TYPE_SUB_CATEGORY.

Public Member Functions

- def __init__

  The constructor.
- def add_notes

  add_notes

Public Attributes

- **name**
- **val**
- **notes**

5.7.1 Detailed Description

INV_TYPE_SUB_CATEGORY.

Defines the basic underlying object for Investment

Definition at line 131 of file PyPerFin_classes.py.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 def PyPerFin_classes.INV_TYPE_SUB_CATEGORY.__init__ (*self, name*)

The constructor.

Definition at line 134 of file PyPerFin_classes.py.

```
00134
00135     def __init__(self, name):
00136         self.name = name
00137         self.val = 0.0
00138         self.notes = ""
```

5.7.3 Member Function Documentation

5.7.3.1 def PyPerFin_classes.INV_TYPE_SUB_CATEGORY.add_notes (*self, note*)

add_notes

Parameters

self	The object pointer.

Definition at line 141 of file PyPerFin_classes.py.

```
00141
00142     def add_notes(self,note):
00143         self.notes = note
00144
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.8 PyPerFin_classes.INVESTMENT_CATEGORY Class Reference

INVESTMENT_CATEGORY.

Public Member Functions

- def __init__

  The constructor.
- def get_total_inv

  get_total_inv

Public Attributes

- **name**
- **total_val**
- **typ1**
- **typ2**
- **typ3**
- **typ4**
- **typ5**
- **typ6**
- **typ7**
- **typ8**
- **typ9**
- **typ10**
- **typ11**
- **typ12**
- **typ13**
- **typ14**
- **typ15**

5.8.1 Detailed Description

INVESTMENT_CATEGORY.

Defines the list of items where you invest your money. Used the class "INV_TYPE_CATEGORY"

Definition at line 179 of file PyPerFin_classes.py.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 def PyPerFin_classes.INVESTMENT_CATEGORY.__init__ (*self*, *name*)

The constructor.

Definition at line 182 of file PyPerFin_classes.py.

```
00182
00183     def __init__(self, name):
00184         self.name = name
00185         self.total_val = 0.0
00186         self.typ1 = INV_TYPE_CATEGORY('typ1')
00187         self.typ2 = INV_TYPE_CATEGORY('typ2')
00188         self.typ3 = INV_TYPE_CATEGORY('typ3')
00189         self.typ4 = INV_TYPE_CATEGORY('typ4')
00190         self.typ5 = INV_TYPE_CATEGORY('typ5')
00191         self.typ6 = INV_TYPE_CATEGORY('typ6')
00192         self.typ7 = INV_TYPE_CATEGORY('typ7')
00193         self.typ8 = INV_TYPE_CATEGORY('typ8')
00194         self.typ9 = INV_TYPE_CATEGORY('typ9')
00195         self.typ10 = INV_TYPE_CATEGORY('typ10')
00196         self.typ11 = INV_TYPE_CATEGORY('typ11')
00197         self.typ12 = INV_TYPE_CATEGORY('typ12')
00198         self.typ13 = INV_TYPE_CATEGORY('typ13')
00199         self.typ14 = INV_TYPE_CATEGORY('typ14')
00200         self.typ15 = INV_TYPE_CATEGORY('typ15')
```

5.8.3 Member Function Documentation

5.8.3.1 def PyPerFin_classes.INVESTMENT_CATEGORY.get_total_inv (*self*)

get_total_inv

Parameters

self	The object pointer.

Definition at line 203 of file PyPerFin_classes.py.

```
00203
00204     def get_total_inv(self):
00205         self.total_val = self.typ1.total_bal + self.typ2.total_bal +
      self.typ3.total_bal + self.typ4.total_bal + self.typ5.total_bal +
      self.typ6.total_bal + self.typ7.total_bal + self.typ8.total_bal + self.typ9.total_bal +
      self.typ10.total_bal \
00206         + self.typ11.total_bal  + self.typ12.total_bal + self.typ13.total_bal +
       self.typ14.total_bal + self.typ15.total_bal
00207
00208
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.9 PyPerFin_classes.MONTH Class Reference

MONTH_VIEW.

Public Member Functions

- def __init__

  The constructor.

Public Attributes

- **name**
- **dest_dir**
- **ttl_incm**
- **ttl_spend**
- **ttl_investment**

5.9.1 Detailed Description

MONTH_VIEW.

Defines the list of spends, incomes, investments in a month Used the class "INCOME", "INVESTMENT_CATEGORY" and "SPEND_ITEMS"

Definition at line 213 of file PyPerFin_classes.py.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 def PyPerFin_classes.MONTH.__init__ (*self, name*)

The constructor.

Definition at line 216 of file PyPerFin_classes.py.

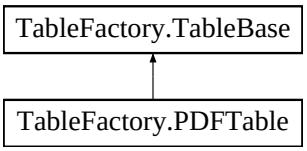
```
00216
00217     def __init__(self, name):
00218         self.name = name
00219         self.dest_dir = '~/Documents/'
00220         self.ttl_incm = INCOME('ttl_incm')
00221         self.ttl_spend = SPEND_ITEMS('ttl_spend')
00222         self.ttl_investment = INVESTMENT_CATEGORY
      ('ttl_investment')
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.10 TableFactory.PDFTable Class Reference

Inheritance diagram for TableFactory.PDFTable:



Public Member Functions

- def render

Static Public Attributes

- tuple **rowoddcolor** = colors.Color(.98, .98, .98)
- tuple **gridcolor** = colors.Color(.6, .6, .6)
- tuple **rowevencolor** = colors.Color(.98, .98, .98)
- tuple **headerbackgroundcolor** = colors.Color(.04, 0, .5)
- tuple **tablebasestyle**
- tuple **tableparentstyle**
- tuple **tablerowstyle**
- tuple **tableheaderstyle**
- tuple **titlestyle** = ParagraphStyle(name='Title Style', fontName='Helvetica-Bold', fontSize=16)
- tuple **explanationstyle** = ParagraphStyle(name='Explanation Style', fontName='Helvetica', fontSize=13)
- tuple **headercellstyle** = ParagraphStyle(name='Table Header Style', fontName='Helvetica-Bold', textColor=colors.white, fontSize=6)
- tuple **contentcellstyle** = ParagraphStyle(name='Table Cell Style', fontName='Helvetica', fontSize=6)
- tuple **contentmoneycellstyle** = ParagraphStyle(name='Table Cell Style', fontName='Helvetica', fontSize=6, alignment=TA_RIGHT)

Additional Inherited Members

5.10.1 Detailed Description

```
Table generator that yields a PDF representation of the data
```

Definition at line 286 of file TableFactory.py.

5.10.2 Member Function Documentation

5.10.2.1 def TableFactory.PDFTable.render (*self, rowsets*)

```
Return the data as a binary string holding a PDF
```

Definition at line 344 of file TableFactory.py.

```
00344
00345     def render(self, rowsets):
00346         """Return the data as a binary string holding a PDF"""
00347
00348         # Start by creating the table headers
00349         rowtables = []
00350         if self.headers:
00351             for headerrow in self.headers:
00352                 widths = [headercolumn.style.width for headercolumn in
      headerrow]
00353                 # Let ReportLab calculate the width of the last column
00354                 # so that it occupies the total remaining open space
00355                 widths[-1] = None
00356                 headertable = Table([[Paragraph(headercolumn.title, self.
      headercellstyle)
00357                                       for headercolumn in headerrow]],
00358                                     style=self.tablebasestyle,
00359                                     colWidths=widths)
00360                 headertable.setStyle(self.tablerowstyle)
00361                 headertable.setStyle(self.tableheaderstyle)
00362                 rowtables.append([headertable])
00363
00364         # Then create a table to hold the contents of each line
00365         for rowset in rowsets:
00366             subrowtables = []
00367             if isinstance(rowset, TableRow):
00368                 rowset = [rowset]
00369             for subrow in rowset:
00370                 subrowtable = Table([[self._rendercell(cell) for
      cell in subrow]],
00371                                     style=self.tablebasestyle,
00372                                     colWidths=[cell.style.width for cell in
      subrow])
00373                 subrowtable.setStyle(self.tablerowstyle)
00374                 subrowtables.append([subrowtable])
00375
00376             rowtable = Table(subrowtables, style=self.tablebasestyle
      )
00377             rowtables.append([rowtable])
00378
00379         # Wrap all of those rows into an outer table
00380         parenttable = Table(rowtables, style=self.tablebasestyle,
       repeatRows=1)
00381         parenttable.setStyle(self.tableparentstyle)
00382
00383         # Finally, build the list of elements that the table will
00384         # comprise
00385         components = []
00386         if self.title:
00387             components.append(Paragraph(self.title, self.titlestyle
      ))
00388         if self.explanation:
00389             components.extend([Spacer(1, .2 * inch),
00390                                Paragraph(self.explanation, self.
      explanationstyle)])
00391         components.extend([Spacer(1, .3 * inch), parenttable])
00392
00393         # Compile the whole thing and return the results
00394         stringbuf = StringIO.StringIO()
00395         doc = SimpleDocTemplate(stringbuf,
00396                                 bottomMargin=.5 * inch, topMargin=.5 * inch,
00397                                 rightMargin=.5 * inch, leftMargin=.5 * inch)
00398         doc.build(components)
00399         return stringbuf.getvalue()
00400
```

5.10.3 Member Data Documentation

5.10.3.1 tuple TableFactory.PDFTable.tablebasestyle [static]

Initial value:

```
TableStyle([
            ('TOPPADDING', (0, 0), (-1, -1), 0),
            ('BOTTOMPADDING', (0, 0), (-1, -1), 0),
            ('LEFTPADDING', (0, 0), (-1, -1), 0),
            ('RIGHTPADDING', (0, 0), (-1, -1), 0),
            ('VALIGN', (0, 0), (-1, -1), 'TOP'),
            ('INNERGRID', (0, 0), (-1, -1), 1, gridcolor),
            ])
```

Definition at line 295 of file TableFactory.py.

5.10.3.2 tuple TableFactory.PDFTable.tableheaderstyle [static]

Initial value:

```
TableStyle([
            ('BACKGROUND', (0, 0), (-1, -1), headerbackgroundcolor),
            ])
```

Definition at line 319 of file TableFactory.py.

5.10.3.3 tuple TableFactory.PDFTable.tableparentstyle [static]

Initial value:

```
TableStyle([
            ('ROWBACKGROUNDS', (0, 0), (-1, -1), [rowoddcolor, rowevencolor]),
            ('LINEABOVE', (0, 1), (-1, -2), 1, colors.black),
            ('LINEBELOW', (0, 1), (-1, -2), 1, colors.black),
            ('BOX', (0, 0), (-1, -1), 1, colors.black),
            ])
```

Definition at line 305 of file TableFactory.py.

5.10.3.4 tuple TableFactory.PDFTable.tablerowstyle [static]

Initial value:

```
TableStyle([
            ('LEFTPADDING', (0, 0), (-1, -1), 3),
            ('RIGHTPADDING', (0, 0), (-1, -1), 3),
            ('TOPPADDING', (0, 0), (-1, -1), 2),
            ])
```

Definition at line 313 of file TableFactory.py.

The documentation for this class was generated from the following file:

- TableFactory.py

5.11 TableFactory.RowSpec Class Reference

Public Member Functions

- def __init__
- def __repr__
- def __call__
- def __iter__
- def makeall

Public Attributes

- **columnspecs**

5.11.1 Detailed Description

```
A RowSpec is a list of ColumnSpecs. It has two main uses:

1) When passed to a table generator as the 'headers' argument
(possibly in a list of other RowSpecs), its ColumnSpecs form the
title row of a table.

2) As a callable, it creates TableRow objects from various Python
objects that are passed into it, saving you the trouble of
building them manually. This is the recommended method of creating
TableRows as it's easy and it also guarantees that your column
titles (see #1 above) will match their contents.
```

Definition at line 187 of file TableFactory.py.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 def TableFactory.RowSpec.__init__ (*self, columnspecs*)

```
Store the given list of ColumnSpecs
```

Definition at line 201 of file TableFactory.py.

```
00201
00202     def __init__(self, *columnspecs):
00203         """Store the given list of ColumnSpecs"""
00204         self.columnspecs = columnspecs
```

5.11.3 Member Function Documentation

5.11.3.1 def TableFactory.RowSpec.__call__ (*self, rowobject*)

```
A RowSpec can be used as a factory that can take an object
like a dict or SQLAlchemy row, apply each of the ColumnSpecs
to that object in turn, and return a corresponding TableRow
object.
```

Definition at line 209 of file TableFactory.py.

```
00209
00210     def __call__(self, rowobject):
00211         """A RowSpec can be used as a factory that can take an object
00212         like a dict or SQLAlchemy row, apply each of the ColumnSpecs
00213         to that object in turn, and return a corresponding TableRow
00214         object."""
00215         output = []
00216         for column in self.columnspecs:
00217             value = rowobject
00218             for attribute in column.attributes:
00219                 try:
00220                     value = value[attribute]
00221                 except (KeyError, TypeError):
00222                     value = getattr(value, attribute)
00223             output.append(Cell(value, column.style))
00224         return TableRow(*output)
```

5.11.3.2 def TableFactory.RowSpec.__iter__ (*self*)

Return each of the row's ColumnSpecs in turn

Definition at line 225 of file TableFactory.py.

```
00225
00226     def __iter__(self):
00227         """Return each of the row's ColumnSpecs in turn"""
00228         for columnspec in self.columnspecs:
00229             yield columnspec
```

5.11.3.3 def TableFactory.RowSpec.__repr__ (*self*)

Human-readable RowSpec representation

Definition at line 205 of file TableFactory.py.

```
00205
00206     def __repr__(self):
00207         """Human-readable RowSpec representation"""
00208         return '<RowSpec(%s)>' % unicode(self.columnspecs)
```

5.11.3.4 def TableFactory.RowSpec.makeall (*self, rowobjects*)

Create a list of TableRows from a list of source objects

Definition at line 230 of file TableFactory.py.

```
00230
00231     def makeall(self, rowobjects):
00232         """Create a list of TableRows from a list of source objects"""
00233         return [self(rowobject) for rowobject in rowobjects]
00234
```

The documentation for this class was generated from the following file:

- TableFactory.py

5.12 PyPerFin_classes.SPEND_CATEGORY Class Reference

SPENDS_CATEGORY.

Public Member Functions

- def __init__
  The constructor.
- def add_value
  add_value

Public Attributes

- **name**
- **value**

5.12.1 Detailed Description

SPENDS_CATEGORY.

Defines the basic underlying class for expenses

Definition at line 37 of file PyPerFin_classes.py.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 def PyPerFin_classes.SPEND_CATEGORY.__init__ (*self, name*)

The constructor.

Definition at line 40 of file PyPerFin_classes.py.

```
00040
00041     def __init__(self, name):
00042         self.name = name
00043         self.value = 0.0;
```

5.12.3 Member Function Documentation

5.12.3.1 def PyPerFin_classes.SPEND_CATEGORY.add_value (*self, val*)

add_value

Parameters

self	The object pointer.
val	Amount to be added to the category

Definition at line 47 of file PyPerFin_classes.py.

```
00047
00048     def add_value (self, val):
00049         self.value = self.value + val
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.13 PyPerFin_classes.SPEND_ITEMS Class Reference

SPEND_ITEMS.

Public Member Functions

- def __init__

  The constructor.

- def get_total

  get_total

Public Attributes

- **name**
- **food**
- **fuel**
- **vehicle**
- **commutation**
- **rent**
- **electricity**
- **water_bill**
- **investment**
- **tel_bills**
- **insurance**
- **stationaries**
- **emis**
- **medical**
- **misc**
- **total_val**

5.13.1 Detailed Description

SPEND_ITEMS.

Defines the list of items where you spend your money. Used the class "SPEND_CATEGORY"

Definition at line 103 of file PyPerFin_classes.py.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 def PyPerFin_classes.SPEND_ITEMS.__init__ (*self*, *name*)

The constructor.

Definition at line 105 of file PyPerFin_classes.py.

```
00105
00106     def __init__(self, name):
00107         self.name = name
00108         self.food = SPEND_CATEGORY('food')
00109         self.fuel = SPEND_CATEGORY('fuel')
00110         self.vehicle = SPEND_CATEGORY('vehicle')
00111         self.commutation = SPEND_CATEGORY('commutation
      ')
00112         self.rent = SPEND_CATEGORY('rent')
00113         self.electricity = SPEND_CATEGORY('electricity
      ')
00114         self.water_bill = SPEND_CATEGORY('water_bill')
00115         self.investment = SPEND_CATEGORY('investment')
00116         self.tel_bills = SPEND_CATEGORY('tel_bills')
00117         self.insurance = SPEND_CATEGORY('insurance')
00118         self.stationaries = SPEND_CATEGORY('
      stationaries')
00119         self.emis = SPEND_CATEGORY('emis')
00120         self.medical = SPEND_CATEGORY('medical')
00121         self.misc = SPEND_CATEGORY('misc')
00122         self.total_val = 0.0
```

5.13.3 Member Function Documentation

5.13.3.1 def PyPerFin_classes.SPEND_ITEMS.get_total (*self*)

get_total

Parameters

self	The object pointer.

Definition at line 125 of file PyPerFin_classes.py.

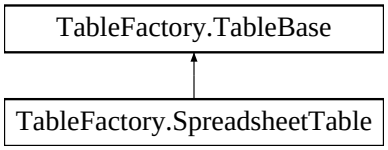
```
00125
00126     def get_total(self):
00127         self.total_val = self.food.value + self.fuel.value +
      self.vehicle.value + self.commutation.value + self.rent.value + self.electricity.value +
      self.water_bill.value + self.investment.value + self.tel_bills.value +
      self.insurance.value + self.stationaries.value + self.emis.value + self.medical.value +
      self.misc.value
```

The documentation for this class was generated from the following file:

- PyPerFin_classes.py

5.14 TableFactory.SpreadsheetTable Class Reference

Inheritance diagram for TableFactory.SpreadsheetTable:



Public Member Functions

- def render

Static Public Attributes

- tuple **headerstyle**
- tuple **explanationstyle** = xlwt.easyxf('font: bold True;')
- dictionary **styletypemap**

Additional Inherited Members

5.14.1 Detailed Description

```
Table generator that yields an Excel spreadsheet representation
of the data. It will have one worksheet named with the given
title
```

Definition at line 401 of file TableFactory.py.

5.14.2 Member Function Documentation

5.14.2.1 def TableFactory.SpreadsheetTable.render (*self*, *rowsets*)

```
Return the data as a binary string holding an Excel spreadsheet
```

Definition at line 451 of file TableFactory.py.

```
00451
00452     def render(self, rowsets):
00453         """Return the data as a binary string holding an Excel spreadsheet"""
00454         book = xlwt.Workbook()
00455         mainsheet = book.add_sheet(self.title or 'Sheet 1')
00456         rownum = 0
00457
00458         if self.explanation:
00459             mainsheet.write(rownum, 0, self.explanation, self.
      explanationstyle)
00460             # Clear the first row's color, or else the headerstyle
00461             # will take over. I have no idea why.
00462             mainsheet.row(0).set_style(self.styletypemap[None][None
      ])
00463             rownum += 2
00464
00465         # Generate any header rows
00466         if self.headers:
00467             for headerrow in self.headers:
00468                 colnum = 0
00469                 for headercolumn in headerrow:
00470                     mainsheet.write(rownum, colnum, headercolumn.title, self.
      headerstyle)
00471                     colnum += headercolumn.style.span
00472                 mainsheet.row(rownum).set_style(self.headerstyle)
00473                 rownum += 1
00474
00475         # Write every line
00476         for rowset in rowsets:
00477             if isinstance(rowset, TableRow):
00478                 rowset = [rowset]
00479             for subrow in rowset:
00480                 colnum = 0
00481                 row = mainsheet.row(rownum)
00482                 for cell in subrow:
00483                     row.write(colnum, cell.value, self._getstyle(cell)
      )
00484                     colnum += cell.style.span
00485                 rownum += 1
00486
00487         stringbuf = StringIO.StringIO()
00488         book.save(stringbuf)
00489         return stringbuf.getvalue()
00490
```

5.14.3 Member Data Documentation

5.14.3.1 tuple TableFactory.SpreadsheetTable.headerstyle [static]

Initial value:

```
xlwt.easyxf('pattern: pattern solid, fore_colour blue;'
                            'font: colour white, bold True;')
```

Definition at line 406 of file TableFactory.py.

5.14.3.2 dictionary TableFactory.SpreadsheetTable.styletypemap [static]

Initial value:

```
{
        None: {None: xlwt.easyxf()},
        datetime.date: {None: xlwt.easyxf(num_format_str='YYYY-MM-DD')},
        datetime.datetime: {None: xlwt.easyxf(num_format_str='YYYY-MM-DD
       HH:MM:SS')},
        }
```

Definition at line 412 of file TableFactory.py.

The documentation for this class was generated from the following file:

- TableFactory.py

5.15 TableFactory.StyleAttributes Class Reference

Public Member Functions

- def __init__
- def __getattr__

Public Attributes

- **properties**

5.15.1 Detailed Description

```
StyleAttribute objects represent the formatting that will be
applied to a cell. Current properties are:

      bold: bool, display a cell in bold

      money: bool, display the cell right-aligned

      width: float, the width of a column in inches

      span: integer, the number of columns the cell should span

      raw: bool, use the cell's contents as-is without escaping
      them

By acting as a thin wrapper around a dict and deferring
calculations until they're needed, we don't do any unnecessary
work or have to worry about values being updated after they're
calculated.
```

Definition at line 83 of file TableFactory.py.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 def TableFactory.StyleAttributes.__init__ (*self, properties*)

```
Save the value of keyword/dict properties
```

Definition at line 103 of file TableFactory.py.

```
00103
00104     def __init__(self, **properties):
00105         """Save the value of keyword/dict properties"""
00106         self.properties = properties
```

5.15.3 Member Function Documentation

5.15.3.1 def TableFactory.StyleAttributes.__getattr__ (*self, key*)

```
Return the requested property after applying appropriate
processing to it
```

Definition at line 107 of file TableFactory.py.

```
00107
00108     def __getattr__(self, key):
00109         """Return the requested property after applying appropriate
```

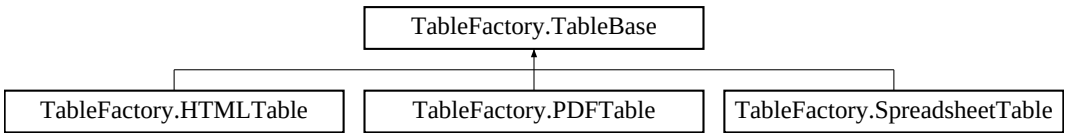
```
00110         processing to it"""
00111         value = self.properties.get(key, None)
00112         if key == 'width' and value is not None:
00113             return value * inch
00114         if key == 'span' and value is None:
00115             return 1
00116         return value
00117
```

The documentation for this class was generated from the following file:

- TableFactory.py

5.16 TableFactory.TableBase Class Reference

Inheritance diagram for TableFactory.TableBase:



Public Member Functions

- def __init__

Public Attributes

- **title**
- **explanation**
- **headers**

Static Public Attributes

- dictionary **castfunctions** = {}

5.16.1 Detailed Description

```
Base class implementing common functionality for all table
classes.
```

Definition at line 235 of file TableFactory.py.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 def TableFactory.TableBase.__init__ (*self*, *title* = None, *explanation* = None, *headers* = None)

```
A rowset is either a TableRow or a collection of
TableRows. 'rowsets' is a collection of rowsets. Passing
multiple rows as a single rowset has two main advantages:

1) The HTMLTable and PDFTable classes use alternating row
colors, and each row in a rowset gets the same color. For
```

```
example, suppose the first row in each rowset contains a list
of detailed columns, and the second row is a note explaining
the first row. By passing them together as single rowsets,
both rows will be colored alike and the colors will alternate
after every other row in the table.

2) The PDFTable class will do its best not to break up rowsets
across page boundaries.

'title' is the table's optional title.

'explanation', if given, will usually be displayed below the
title.

'headers' is a RowSpec or a collection of RowSpecs used to
generate the table's header. If more than one RowSpec is
given, each will be rendered in order as a header row.
```

Definition at line 241 of file TableFactory.py.

```
00241
00242     def __init__(self, title=None, explanation=None, headers=None):
00243         """A rowset is either a TableRow or a collection of
00244         TableRows. 'rowsets' is a collection of rowsets. Passing
00245         multiple rows as a single rowset has two main advantages:
00246
00247         1) The HTMLTable and PDFTable classes use alternating row
00248         colors, and each row in a rowset gets the same color. For
00249         example, suppose the first row in each rowset contains a list
00250         of detailed columns, and the second row is a note explaining
00251         the first row. By passing them together as single rowsets,
00252         both rows will be colored alike and the colors will alternate
00253         after every other row in the table.
00254
00255         2) The PDFTable class will do its best not to break up rowsets
00256         across page boundaries.
00257
00258         'title' is the table's optional title.
00259
00260         'explanation', if given, will usually be displayed below the
00261         title.
00262
00263         'headers' is a RowSpec or a collection of RowSpecs used to
00264         generate the table's header. If more than one RowSpec is
00265         given, each will be rendered in order as a header row.
00266         """
00267         self.title = title
00268         self.explanation = explanation
00269         if isinstance(headers, RowSpec):
00270             self.headers = [headers]
00271         else:
00272             self.headers = headers
```

The documentation for this class was generated from the following file:

- TableFactory.py

5.17 TableFactory.TableRow Class Reference

Public Member Functions

- def __init__
- def __repr__
- def __iter__

Public Attributes

- **cells**

5.17.1 Detailed Description

```
A TableRow is a list of cells
```

Definition at line 137 of file TableFactory.py.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 def TableFactory.TableRow.__init__ (*self, cells*)

```
Store the given list of cells
```

Definition at line 140 of file TableFactory.py.

```
00140
00141     def __init__(self, *cells):
00142         """Store the given list of cells"""
00143         self.cells = cells
```

5.17.3 Member Function Documentation

5.17.3.1 def TableFactory.TableRow.__iter__ (*self*)

```
Return each of the row's cells in turn
```

Definition at line 148 of file TableFactory.py.

```
00148
00149     def __iter__(self):
00150         """Return each of the row's cells in turn"""
00151         for cell in self.cells:
00152             yield cell
00153
```

5.17.3.2 def TableFactory.TableRow.__repr__ (*self*)

```
Human-readable TableRow representation
```

Definition at line 144 of file TableFactory.py.

```
00144
00145     def __repr__(self):
00146         """Human-readable TableRow representation"""
00147         return '<TableRow(%s)>' % unicode(self.cells)
```

The documentation for this class was generated from the following file:

- TableFactory.py

Index