# pyxml2pdf Documentation

**Björn Ludwig**

**Sep 19, 2021**

# Contents:

Convert XML input to PDF table. Since we forked the upstream this project has generalized quite a bit on the generation of a multipage PDF file containing a table with subtables each containing a subset of the xml tags based on the texts of some of their children tags.

For the *pyxml2pdf* homepage go to GitHub.

*pyxml2pdf* is written in Python 3 and currently requires Python 3.6 or later.

# CHAPTER 1

# API reference

## 1.1 properties

The applied set of parameters for interpreting the XML input and output formatting

To change these setting just take any of the provided variables and overwrite it with your desired value in *input/custom_properties.py*. I.e. to change the *pagesize* to something a bit smaller than ISO A5 put the following into *input/custom_properties.py*:

```
pagesize = (178, 134)
```

pyxml2pdf.input.properties.**columns = [Column(label='name', tag=['name_tag'], width=30), Co**
This is of what and how to include into output's columns

The desired column headings in the output are supposed to be specified as 'label' and the XML tags containing the content to be displayed in the corresponding rows of the output's column are supposed to be specified as 'tag'. By default, the content of each tag is transferred to one cell. If several tags are to be merged within one cell, nested lists can be used here. All tags listed here, which finally shall be displayed in the columns of one and the same row, must each belong to one parent tag *rows_xmltag*. The column widths are specified with 'width' in mm.

pyxml2pdf.input.properties.**filter_xmltag = 'filter_tag'**
The XML tag used to check for filter criteria in the respective rows

pyxml2pdf.input.properties.**font = Font(normal='LiberationSans-Regular.ttf', italic='Liberat**
Fonts for the output

Files with the respective file names are expected to be found inside the subfolder *pyxml2pdf/styles/fonts/*.

pyxml2pdf.input.properties.**fontsize = FontSize(normal=6.5, table_heading=12, column_heading**
Set the font size for the table.

This is the font sizes used throughout the output table, where *normal* applies to all text except table and column headings.

pyxml2pdf.input.properties.**identifier_xmltag = ['name_tag', 'info_tag', 'filter_tag']**
The XML tag, which will be used to identify the row for error message printing

This is used in case an instance of *rows_xmltag* in the input XML file does match any filter criteria and thus would not be included in the output. This results in an error message telling which element is not printed where *identifier_xmltag* is used to inform about which element is affected. The error message starts with something close to:

```
XML row identified by <IDENTIFIER_XMLTAG's CONTENT> would not be printed,
because it does not contain a valid combination of criteria.
```

pyxml2pdf.input.properties.**pagesize = (90.1, 84.3)**
The page size of the output Pdf in mm

pyxml2pdf.input.properties.**rows_xmltag = 'row_tag'**
The XML tag, which will be represented by one row in the table

pyxml2pdf.input.properties.**sort_xmltag = 'name_tag'**
The XML tag, which will be used to sort the tables' rows

If possible the tags' contents will be sorted as dates otherwise they will be sorted alphanumerically, each in ascending order.

pyxml2pdf.input.properties.**subtable_settings = (SubtableSetting(label='filter 1', include=**
The subtables' headings and filter criteria to choose the XML's content to display

There will be one subtable for each set of a label and include filter-criteria, as long as at least one element is found for the subtable. To *match* the criteria, the element's *filter_xmltag*'s content needs to contain at least one of the list elements of each of the nested lists, given here. The *filter_xmltag*'s content will be compared against the given include filters, where for comma-separated elements of one list a boolean OR is used and a boolean AND for the separate lists.

## 1.2 core

### 1.2.1 rows

This module contains the base class to create table rows

Specifically it contains a class *XMLCell* for unified styled cells and a class *XMLRow* for xml extracted data.

**class** pyxml2pdf.core.rows.**XMLCell**(*text: str*)
This class represents the text of type reportlab.platypus.Paragraph in a cell

It inherits from `reportlab.platypus.Paragraph` and ensures the unified styling of all table cells.

`reportlab.platypus.Paragraph` is solely used with one certain style, which is supposed to be set as a class attribute during runtime.

**Parameters** **text** (*str*) – the text to write into row

**style**
The one for all stylesheet to style all cells.

**Type** StyleSheet1

**class** pyxml2pdf.core.rows.**XMLRow**(*element*)
A wrapper class for `xml.etree.ElementTree.Element`

`xml.etree.ElementTree.Element` is augmented with the table row representation and the attributes and methods to manipulate everything according to the final tables needs. A *XMLRow* can only be initialized with an object of type `xml.etree.ElementTree.Element`.

**Parameters** **element** (*xml.etree.ElementTree.Element*) – the element to build the instance from

**_cell_styler**
alias of *XMLCell*

**_concatenate_tags_content**(*cell_tags: List[str], separator: str = ' - '*) → str
Form one string from the texts of a set of XML tags's to fill a cell

Form a string of the content for all desired XML tags by concatenating them together with a separator. This is especially necessary, since `reportlab.platypus.Paragraph` cannot handle *None's as texts but handles as well the concatenation of XML tags' content, if 'cell_tags* has more than one element. So we ensure the result to be at least an empty string.

**Parameters**

- **cell_tags** – list of all tags for which the descriptive texts is wanted, even if it is just one
- **separator** – the separator in between the concatenated texts

**Returns** concatenated, separated texts of all tags for the current cell

**_init_criteria**()
Initialize the list of criteria from the according xml tag's content

**_init_full_row**() → List[pyxml2pdf.core.rows.XMLCell]
Initialize the single table row containing all information from the XML input

Extract interesting information from specified row tag's subtags and connect them into a nicely formatted row of a table.

**Returns** the columns of any table representation

**Return type** List[*XMLCell*]

**_table_builder = <pyxml2pdf.tables.builder.TableBuilder object>**

**_table_style = <pyxml2pdf.styles.table_styles.XMLTableStyle object>**

**criteria**
Return the event's criteria

**Returns** a list of the event's criteria

**Return type** Set[str]

**get_full_row**(*subtable_title: str = None*) → reportlab.platypus.tables.Table
Return a table row with all the row's information

This ensures, that in subclasses we can override this function and after handing over the full information, the reduced version with a reference to the subtable containing the full version can be created via a decorator.

See `Event` for an example implementation of this pattern.

**Parameters** **subtable_title** – the title of the subtable in which the row will be integrated

**Returns** a table row with all the event's information

**get_table_row**(*subtable_title: str*) → reportlab.platypus.tables.Table
Return the table row representation of the XML tag

This is the API of *XMLRow* for getting the table row representation of the event. It allows for reacting to the distribution of the XML tags content by creating a shorter version referencing the main subtable. See *get_full_row()* for details.

**Parameters** **subtable_title** (*str*) – the title of the subtable in which the row will be integrated

**Returns** a table row representation of the XML tag's content

**Return type** Table

**identifier**
Return the identifier of the event

**Returns** identifier

**Return type** str

### 1.2.2 events

A wrapper *pyxml2pdf.core.events.Event* for xml extracted data

**class** pyxml2pdf.core.events.**Event**(*element*)
A specialisation of XMLRow onto events from an ACB event program

**Parameters** **element** (*xml.etree.ElementTree.Element*) – the element to build the instance from

**_build_description**(*link: str = ''*) → str
Build the description for the event

This covers all cases with empty texts in some of the according children tags and the full as well as the reduced version with just the reference to the subtable where the full version can be found. Since the title of the event is mandatory, and the beginning of the description is always filled by the same tags' texts those are not received as parameter but directly retrieved from the xml data.

**Parameters** **link** (*str*) – a link to more details like the trainer url or the subtable

**Returns** the full description including url if provided

**Return type** str

**_build_type**() → str
Build the type for the event

This assembles the type of the event from the different kinds.

**Returns** the entry in the type column of the event

**Return type** str

**_init_date**()
Create a properly formatted string containing the identifier of the event

**_init_full_row**() → List[pyxml2pdf.core.rows.XMLCell]
Initialize the single table row containing all information of the event

Extract interesting information from events children tags and connect them into a nicely formatted row of a table.

**Returns** the common starting columns of any table representation

**Return type** List[*XMLCell*]

**_init_reduced_row**(*subtable_title*)
Initializes the reduced version of the event

Create a table row in proper format but just containing a brief description of the event and a reference to the fully described event at another place, namely the subtable with the given title.

**Parameters** **subtable_title** (*str*) – title of the subtable which contains the full event

> **Warning:** Do not call this function directly since it is automatically called right after *get_full_row()* is invoked.

**static _parse_prerequisites**(*personal: str*, *material: str*, *financial: str*, *offers: str*) → str
Determine all prerequisites and assemble a string accordingly.

**Parameters**

- **material** (*str*) – material prerequisite xml text
- **personal** (*str*) – personal prerequisite xml text
- **financial** (*str*) – financial prerequisite xml text
- **offers** (*str*) – xml text of what is included in the price

**Returns** the text to insert in prerequisite column the current event

**Return type** str

**static _remove_century**(*four_digit_year*)
Remove the first two digits of the string representing the year

**Parameters** **four_digit_year** (*typing.Match*) – the result of `re.sub()`

**Returns** the last two digits of the string representing the year

**Return type** str

**_table_builder = <pyxml2pdf.tables.builder.TableBuilder object>**

**_table_style = <pyxml2pdf.styles.table_styles.XMLTableStyle object>**

**create_reduced_after_full**()
Decorator to execute *_init_reduced_row()* with *get_full_row()*

**Returns** the return value of *get_full_row()*

**Return type** Table

**get_full_row**(**args*, ***kwargs*)
Exchange a table row with all the event's information against a subtable's title

This ensures, that after handing over the full information, the reduced version with a reference to the subtable containing the full version is created.

---

**Note:** This is ensured by a decorator, which is why the function signature on ReadTheDocs.org is displayed incorrectly. The parameter and return value are as follows...

---

**Parameters** **subtable_title** (*str*) – the title of the subtable in which the row will be integrated

**Returns** a table row with all the event's information

**Return type** Table

**get_table_row**(*subtable_title*)
Return the table row representation of the event

This is the API of *pyxml2pdf.core.events.Event* for getting the table row representation of the event. It makes sure, that on the first call *get_full_row()* is invoked and otherwise pyxml2pdf.core.events.Event._reduced_row is returned.

**Parameters** **subtable_title** (*str*) – the title of the subtable in which the row will be integrated

**Returns** a table row representation of the event

**Return type** Table

**identifier**
Return the identifier of the event

**Returns** identifier

**Return type** str

**responsible**
Return the name of the person being responsible for the event

**Returns** first and last name

**Return type** str

### 1.2.3 initializer

This module contains the class *Initializer* to coordinate the process.

**class** pyxml2pdf.core.initializer.**Initializer**(*input_path: str*, *output_path: str*)
Coordinate the construction of the pdf result

Keep strings together, start the actual parsing and build the PDF

**Parameters**

- **input_path** (*str*) – Path to input XML file
- **output_path** (*str*) – Path to resulting PDF file

**_data = None**
The processed content of the XML file as table rows and columns

### 1.2.4 parser

*pyxml2pdf.core.parser* is the interface between XML input and table

**class** pyxml2pdf.core.parser.**Parser**(*elements: List[reportlab.platypus.flowables.KeepTogether]*)
XML parser to extract all interesting information from XML input

**Parameters** **elements** – cells to populate the Parser

**collect_xml_data**(*events*)
Traverse the parsed xml data and gather collected event data

The collected XML data then is passed to the table_manager and all arranged data is return.

**Parameters** **events** (*List[XMLRow]*) – a list of the items from which the texts shall be extracted into a nicely formatted table

**Returns** list of all table rows containing the relevant event data

**Return type** List[KeepTogether]

### 1.2.5 post_processor

This module contains the class *PostProcessor* to arrange the result pages

**class** pyxml2pdf.core.post_processor.**PostProcessor**(*path*)
Arrange for needed modifications of the result to prepare for printing

This creates an instance of a *pyxml2pdf.core.post_processor* for a multipage PDF file to automate splitting and rotating.

**Parameters** **path** (*str*) – path to the PDF file which shall be processed

**finalize_print_preparation**()
Take the resulting multi page PDF and split into rotated single pages

Taken from pythonlibrary.org in combination with johndcook.com

### 1.2.6 sorter

This module contains the class *Sorter* to sort the resulting table.

**class** pyxml2pdf.core.sorter.**Sorter**(*courses*)
Provides a method to sort from xml extracted data by a tag containing a date

We took effbot.org and adapted the code to our needs of sorting a list of `xml.etree.ElementTree.Element` by the texts of one of their tags containing a string representation of a date.

**Parameters** **courses** (*List[xml.etree.ElementTree.Element]*) – rows that where extracted from an xml source

**sort_parsed_xml**(*sort_key*)
Sort a list of `xml.etree.ElementTree.Element` by their date

Taken from effbot.org and adapted.

**Parameters** **sort_key** (*str*) – the XML tag which contains the data

## 1.3 tables

### 1.3.1 tables

This module contains a class *XMLTable* to collect the XML's content.

**class** pyxml2pdf.tables.tables.**XMLTable**(*title: str, include_filters: List[List[str]]*)
An *XMLTable* contains a subset of the xml file's content in a Table

Contains all XML tags which match the desired content specified in `content`. Every XML tag listed has at least one subtag from each of the lists in `content`.

**Parameters**

- **title** (*str*) – Name of the table
- **include_filters** (*List[List[str]]*) – nested list of criteria to collect in table

**append**(*row: reportlab.platypus.tables.Table*)
Append a row to the end of the table

**Parameters** **row** – a single row that should be appended to the table

**extend**(*rows: List[reportlab.platypus.tables.Table]*)
Append a a list of rows to the end of the table

**Parameters** **rows** – a list of rows that should be appended to the table

**include_filters**
include_filters to match XML contents for including

**Type** List[List[str]]

**rows**
The list of rows as Table objects

**Type** List[Table]

**title**
Name of the table

**Type** str

### 1.3.2 builder

This module contains the class *TableBuilder* which deals with XML tables.

**class** pyxml2pdf.tables.builder.**TableBuilder**
Takes over all tasks for building and working with the tables created

**create_fixedwidth_table**(*cells: List[List[reportlab.platypus.flowables.Flowable]], widths: Union[float, List[float], None] = None, style: Optional[pyxml2pdf.styles.table_styles.XMLTableStyle] = None*) → reportlab.platypus.tables.Table
Create a table with specified column widths

Create a table from specified cells with fixed column widths and a specific style.

**Parameters**

- **cells** (*List[List[Flowable]]*) – cells wrapped by a list representing the columns wrapped by a list representing the lines
- **List[float]]] widths** (*Optional[Union[float,*) – Optional column widths. The default results in reasonable settings based on experience.
- **style** (*Optional[XMLTableStyle]*) – Optional table's style. The default results in reasonable settings based on experience.

**Returns** A table containing specified cells in fixed width, styled columns.

**create_subtables**() → List[pyxml2pdf.tables.tables.XMLTable]
Create subtables for all different kinds of rows

**Returns** a list of all subtables

**Return type** List[*XMLTable*]

**distribute_row**(*row*)
Distribute a row to the subtables according to the related criteria

**Parameters** **row** (XMLRow) – row to distribute

**make_header**(*title: str*) → List[reportlab.platypus.tables.Table]
Build the first two rows of a subtable

Build the first two rows of a subtable with its title and column headings taken from the properties file.

**Parameters** **title** (*str*) – the title of the subtable

**Returns** two line table with title and headings

**Return type** List[Table]

**subtables**
Return all subtables at once

**Type** List[Table]

## 1.4 styles

### 1.4.1 table_styles

This module contains the class *XMLTableStyle* to style the result

**class** pyxml2pdf.styles.table_styles.**XMLTableStyle**
Create a collection of styling information about the table to create

**Beautiful colors are:**

- aliceblue (not with azure)
- azure (not with aliceblue)
- honeydew
- ...

**ALIGN_CENTER = ('ALIGN', (0, 0), (-1, -1), 'CENTER')**

**ALIGN_LEFT = ('ALIGN', (0, 0), (-1, -1), 'LEFT')**

**BACKGROUND = ('BACKGROUND', (0, 0), (-1, -1), Color(.941176,1,.941176,1))**

**BACKGROUND_COLOR = Color(.941176,1,.941176,1)**

**BOX = ('BOX', (0, 0), (-1, -1), 0.25, Color(0,0,0,1))**

**FULL_ROW = ((0, 0), (-1, -1))**

**INNERGRID = ('INNERGRID', (0, 0), (-1, -1), 0.25, Color(0,0,0,1))**

**LEFTPADDING_REDUCE = ('LEFTPADDING', (0, 0), (-1, -1), 3)**

**LINE_COLOR = Color(0,0,0,1)**

**LINE_THICKNESS = 0.25**

**PADDING = 3**

**RIGHTPADDING_REDUCE = ('RIGHTPADDING', (0, 0), (-1, -1), 3)**

**VALIGN_MIDDLE = ('VALIGN', (0, 0), (-1, -1), 'MIDDLE')**

**VALIGN_TOP = ('VALIGN', (0, 0), (-1, -1), 'TOP')**

**_init_font_family**()
Register the desired font with reportlab

This ensures that  and  as cell content work well.

**column_widths**
Return the column widths for the tables in mm.

**Type** List[float]

**custom_styles**
Return the custom styles

**Type** Dict[str, TableStyle]

**table_width**
Return the sum of all column widths in mm.

**Type** float

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## Symbols



## A



## B



## C



## D



## E

## F



## G



## I



## L



## M



## P



## R



## S



## T



## V



## X