python-docx Documentation

Release 1.1.2

Steve Canny

Jan 18, 2025

CONTENTS

Release v1.1.2 (*Installation*)

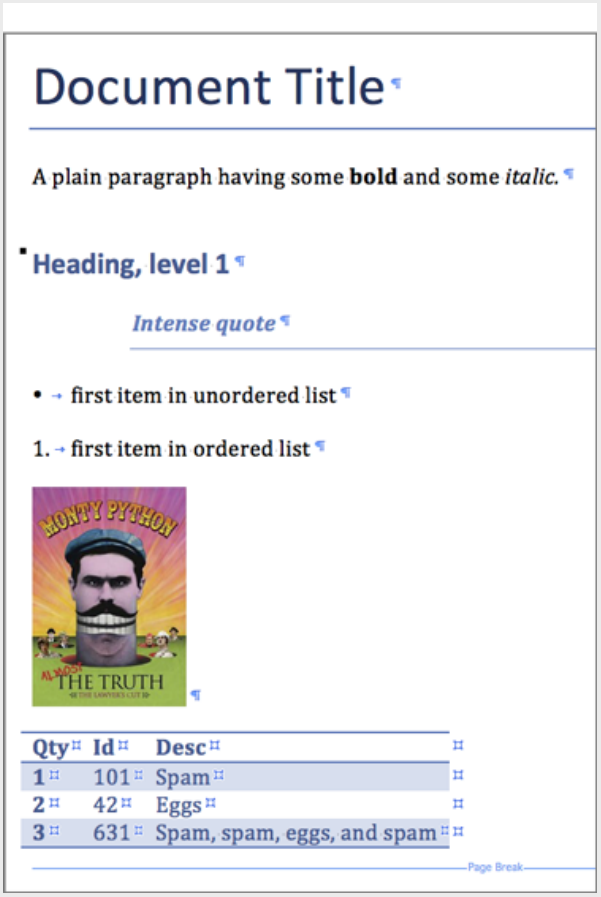
python-docx is a Python library for creating and updating Microsoft Word (.docx) files.

CHAPTER

ONE

WHAT IT CAN DO

Here's an example of what `python-docx` can do:




```
from docx import Document
from docx.shared import Inches

document = Document()

document.add_heading('Document Title', 0)

p = document.add_paragraph('A plain␣
↪paragraph having some ')
p.add_run('bold').bold = True
p.add_run(' and some ')
p.add_run('italic.').italic = True

document.add_heading('Heading, level 1',␣
↪level=1)
document.add_paragraph('Intense quote',␣
↪style='Intense Quote')

document.add_paragraph(
    'first item in unordered list', style=
↪'List Bullet'
)
document.add_paragraph(
    'first item in ordered list', style=
↪'List Number'
)

document.add_picture('monty-truth.png',␣
↪width=Inches(1.25))

records = (
```

CHAPTER

TWO

USER GUIDE

2.1 Installing

Note

python-docx versions 0.3.0 and later are not API-compatible with prior versions.

`python-docx` is hosted on PyPI, so installation is relatively simple, and just depends on what installation utilities you have installed.

`python-docx` may be installed with `pip` if you have it available:

```
pip install python-docx
```

`python-docx` can also be installed using `easy_install`, although this is discouraged:

```
easy_install python-docx
```

If neither `pip` nor `easy_install` is available, it can be installed manually by downloading the distribution from PyPI, unpacking the tarball, and running `setup.py`:

```
tar xvzf python-docx-{version}.tar.gz
cd python-docx-{version}
python setup.py install
```

`python-docx` depends on the `lxml` package. Both `pip` and `easy_install` will take care of satisfying those dependencies for you, but if you use this last method you will need to install those yourself.

2.1.1 Dependencies

- Python 2.6, 2.7, 3.3, or 3.4
- lxml >= 2.3.2

2.2 Quickstart

Getting started with `python-docx` is easy. Let's walk through the basics.

2.2.1 Opening a document

First thing you'll need is a document to work on. The easiest way is this:

```
from docx import Document

document = Document()
```

This opens up a blank document based on the default "template", pretty much what you get when you start a new document in Word using the built-in defaults. You can open and work on an existing Word document using `python-docx`, but we'll keep things simple for the moment.

2.2.2 Adding a paragraph

Paragraphs are fundamental in Word. They're used for body text, but also for headings and list items like bullets.

Here's the simplest way to add one:

```
paragraph = document.add_paragraph('Lorem ipsum dolor sit amet.')
```

This method returns a reference to a paragraph, newly added paragraph at the end of the document. The new paragraph reference is assigned to `paragraph` in this case, but I'll be leaving that out in the following examples unless I have a need for it. In your code, often times you won't be doing anything with the item after you've added it, so there's not a lot of sense in keep a reference to it hanging around.

It's also possible to use one paragraph as a "cursor" and insert a new paragraph directly above it:

```
prior_paragraph = paragraph.insert_paragraph_before('Lorem ipsum')
```

This allows a paragraph to be inserted in the middle of a document, something that's often important when modifying an existing document rather than generating one from scratch.

2.2.3 Adding a heading

In anything but the shortest document, body text is divided into sections, each of which starts with a heading. Here's how to add one:

```
document.add_heading('The REAL meaning of the universe')
```

By default, this adds a top-level heading, what appears in Word as 'Heading 1'. When you want a heading for a subsection, just specify the level you want as an integer between 1 and 9:

```
document.add_heading('The role of dolphins', level=2)
```

If you specify a level of 0, a "Title" paragraph is added. This can be handy to start a relatively short document that doesn't have a separate title page.

2.2.4 Adding a page break

Every once in a while you want the text that comes next to go on a separate page, even if the one you're on isn't full. A "hard" page break gets this done:

```
document.add_page_break()
```

If you find yourself using this very often, it's probably a sign you could benefit by better understanding paragraph styles. One paragraph style property you can set is to break a page immediately before each paragraph having that style. So you might set your headings of a certain level to always start a new page. More on styles later. They turn out to be critically important for really getting the most out of Word.

2.2.5 Adding a table

One frequently encounters content that lends itself to tabular presentation, lined up in neat rows and columns. Word does a pretty good job at this. Here's how to add a table:

```
table = document.add_table(rows=2, cols=2)
```

Tables have several properties and methods you'll need in order to populate them. Accessing individual cells is probably a good place to start. As a baseline, you can always access a cell by its row and column indicies:

```
cell = table.cell(0, 1)
```

This gives you the right-hand cell in the top row of the table we just created. Note that row and column indicies are zero-based, just like in list access.

Once you have a cell, you can put something in it:

```
cell.text = 'parrot, possibly dead'
```

Frequently it's easier to access a row of cells at a time, for example when populating a table of variable length from a data source. The `.rows` property of a table provides access to individual rows, each of which has a `.cells` property. The `.cells` property on both `Row` and `Column` supports indexed access, like a list:

```
row = table.rows[1]
row.cells[0].text = 'Foo bar to you.'
row.cells[1].text = 'And a hearty foo bar to you too sir!'
```

The `.rows` and `.columns` collections on a table are iterable, so you can use them directly in a `for` loop. Same with the `.cells` sequences on a row or column:

```
for row in table.rows:
    for cell in row.cells:
        print(cell.text)
```

If you want a count of the rows or columns in the table, just use `len()` on the sequence:

```
row_count = len(table.rows)
col_count = len(table.columns)
```

You can also add rows to a table incrementally like so:

```
row = table.add_row()
```

This can be very handy for the variable length table scenario we mentioned above:

```
# get table data -------------
items = (
    (7, '1024', 'Plush kittens'),
    (3, '2042', 'Furbees'),
    (1, '1288', 'French Poodle Collars, Deluxe'),
)

# add table ------------------
table = document.add_table(1, 3)

# populate header row --------
```

(continues on next page)

(continued from previous page)

```
heading_cells = table.rows[0].cells
heading_cells[0].text = 'Qty'
heading_cells[1].text = 'SKU'
heading_cells[2].text = 'Description'

# add a data row for each item
for item in items:
    cells = table.add_row().cells
    cells[0].text = str(item.qty)
    cells[1].text = item.sku
    cells[2].text = item.desc
```

The same works for columns, although I've yet to see a use case for it.

Word has a set of pre-formatted table styles you can pick from its table style gallery. You can apply one of those to the table like this:

```
table.style = 'LightShading-Accent1'
```

The style name is formed by removing all the spaces from the table style name. You can find the table style name by hovering your mouse over its thumbnail in Word's table style gallery.

2.2.6 Adding a picture

Word lets you place an image in a document using the `Insert > Photo > Picture from file...` menu item. Here's how to do it in `python-docx`:

```
document.add_picture('image-filename.png')
```

This example uses a path, which loads the image file from the local filesystem. You can also use a *file-like object*, essentially any object that acts like an open file. This might be handy if you're retrieving your image from a database or over a network and don't want to get the filesystem involved.

Image size

By default, the added image appears at *native* size. This is often bigger than you want. Native size is calculated as `pixels / dpi`. So a 300x300 pixel image having 300 dpi resolution appears in a one inch square. The problem is most images don't contain a dpi property and it defaults to 72 dpi. This would make the same image appear 4.167 inches on a side, somewhere around half the page.

To get the image the size you want, you can specify either its width or height in convenient units, like inches or centimeters:

```
from docx.shared import Inches

document.add_picture('image-filename.png', width=Inches(1.0))
```

You're free to specify both width and height, but usually you wouldn't want to. If you specify only one, `python-docx` uses it to calculate the properly scaled value of the other. This way the *aspect ratio* is preserved and your picture doesn't look stretched.

The `Inches` and `Cm` classes are provided to let you specify measurements in handy units. Internally, `python-docx` uses English Metric Units, 914400 to the inch. So if you forget and just put something like `width=2` you'll get an extremely small image :). You'll need to import them from the `docx.shared` sub-package. You can use them in arithmetic just like they were an integer, which in fact they are. So an expression like `width = Inches(3) / thing_count` works just fine.

2.2.7 Applying a paragraph style

If you don't know what a Word paragraph style is you should definitely check it out. Basically it allows you to apply a whole set of formatting options to a paragraph at once. It's a lot like CSS styles if you know what those are.

You can apply a paragraph style right when you create a paragraph:

```
document.add_paragraph('Lorem ipsum dolor sit amet.', style='ListBullet')
```

This particular style causes the paragraph to appear as a bullet, a very handy thing. You can also apply a style afterward. These two lines are equivalent to the one above:

```
paragraph = document.add_paragraph('Lorem ipsum dolor sit amet.')
paragraph.style = 'List Bullet'
```

The style is specified using its style name, 'List Bullet' in this example. Generally, the style name is exactly as it appears in the Word user interface (UI).

2.2.8 Applying bold and italic

In order to understand how bold and italic work, you need to understand a little about what goes on inside a paragraph. The short version is this:

1. A paragraph holds all the *block-level* formatting, like indentation, line height, tabs, and so forth.
2. Character-level formatting, such as bold and italic, are applied at the *run* level. All content within a paragraph must be within a run, but there can be more than one. So a paragraph with a bold word in the middle would need three runs, a normal one, a bold one containing the word, and another normal one for the text after.

When you add a paragraph by providing text to the `.add_paragraph()` method, it gets put into a single run. You can add more using the `.add_run()` method on the paragraph:

```
paragraph = document.add_paragraph('Lorem ipsum ')
paragraph.add_run('dolor sit amet.')
```

This produces a paragraph that looks just like one created from a single string. It's not apparent where paragraph text is broken into runs unless you look at the XML. Note the trailing space at the end of the first string. You need to be explicit about where spaces appear at the beginning and end of a run. They're not automatically inserted between runs. Expect to be caught by that one a few times :).

Run objects have both a `.bold` and `.italic` property that allows you to set their value for a run:

```
paragraph = document.add_paragraph('Lorem ipsum ')
run = paragraph.add_run('dolor')
run.bold = True
paragraph.add_run(' sit amet.')
```

which produces text that looks like this: 'Lorem ipsum **dolor** sit amet.'

Note that you can set bold or italic right on the result of `.add_run()` if you don't need it for anything else:

```
paragraph.add_run('dolor').bold = True

# is equivalent to:

run = paragraph.add_run('dolor')
run.bold = True
```

(continues on next page)

(continued from previous page)

```
# except you don't have a reference to `run` afterward
```

It's not necessary to provide text to the `.add_paragraph()` method. This can make your code simpler if you're building the paragraph up from runs anyway:

```
paragraph = document.add_paragraph()
paragraph.add_run('Lorem ipsum ')
paragraph.add_run('dolor').bold = True
paragraph.add_run(' sit amet.')
```

2.2.9 Applying a character style

In addition to paragraph styles, which specify a group of paragraph-level settings, Word has *character styles* which specify a group of run-level settings. In general you can think of a character style as specifying a font, including its typeface, size, color, bold, italic, etc.

Like paragraph styles, a character style must already be defined in the document you open with the `Document()` call (*see Understanding Styles*).

A character style can be specified when adding a new run:

```
paragraph = document.add_paragraph('Normal text, ')
paragraph.add_run('text with emphasis.', 'Emphasis')
```

You can also apply a style to a run after it is created. This code produces the same result as the lines above:

```
paragraph = document.add_paragraph('Normal text, ')
run = paragraph.add_run('text with emphasis.')
run.style = 'Emphasis'
```

As with a paragraph style, the style name is as it appears in the Word UI.

2.3 Working with Documents

`python-docx` allows you to create new documents as well as make changes to existing ones. Actually, it only lets you make changes to existing documents; it's just that if you start with a document that doesn't have any content, it might feel at first like you're creating one from scratch.

This characteristic is a powerful one. A lot of how a document looks is determined by the parts that are left when you delete all the content. Things like styles and page headers and footers are contained separately from the main content, allowing you to place a good deal of customization in your starting document that then appears in the document you produce.

Let's walk through the steps to create a document one example at a time, starting with two of the main things you can do with a document, open it and save it.

2.3.1 Opening a document

The simplest way to get started is to open a new document without specifying a file to open:

```
from docx import Document
```

(continues on next page)

(continued from previous page)

```
document = Document()
document.save('test.docx')
```

This creates a new document from the built-in default template and saves it unchanged to a file named 'test.docx'. The so-called "default template" is actually just a Word file having no content, stored with the installed `python-docx` package. It's roughly the same as you get by picking the *Word Document* template after selecting Word's **File > New from Template...** menu item.

2.3.2 REALLY opening a document

If you want more control over the final document, or if you want to change an existing document, you need to open one with a filename:

```
document = Document('existing-document-file.docx')
document.save('new-file-name.docx')
```

Things to note:

- You can open any Word 2007 or later file this way (.doc files from Word 2003 and earlier won't work). While you might not be able to manipulate all the contents yet, whatever is already in there will load and save just fine. The feature set is still being built out, so you can't add or change things like headers or footnotes yet, but if the document has them `python-docx` is polite enough to leave them alone and smart enough to save them without actually understanding what they are.
- If you use the same filename to open and save the file, `python-docx` will obediently overwrite the original file without a peep. You'll want to make sure that's what you intend.

2.3.3 Opening a 'file-like' document

`python-docx` can open a document from a so-called *file-like* object. It can also save to a file-like object. This can be handy when you want to get the source or target document over a network connection or from a database and don't want to (or aren't allowed to) interact with the file system. In practice this means you can pass an open file or StringIO/BytesIO stream object to open or save a document like so:

```
f = open('foobar.docx', 'rb')
document = Document(f)
f.close()

# or

with open('foobar.docx', 'rb') as f:
    source_stream = StringIO(f.read())
document = Document(source_stream)
source_stream.close()
...
target_stream = StringIO()
document.save(target_stream)
```

The `'rb'` file open mode parameter isn't required on all operating systems. It defaults to `'r'` which is enough sometimes, but the 'b' (selecting binary mode) is required on Windows and at least some versions of Linux to allow Zipfile to open the file.

Okay, so you've got a document open and are pretty sure you can save it somewhere later. Next step is to get some content in there ...

2.4 Working with Tables

Word provides sophisticated capabilities to create tables. As usual, this power comes with additional conceptual complexity.

This complexity becomes most apparent when *reading* tables, in particular from documents drawn from the wild where there is limited or no prior knowledge as to what the tables might contain or how they might be structured.

These are some of the important concepts you'll need to understand.

2.4.1 Concept: Simple (uniform) tables

```
+---+---+---+
| a | b | c |
+---+---+---+
| d | e | f |
+---+---+---+
| g | h | i |
+---+---+---+
```

The basic concept of a table is intuitive enough. You have *rows* and *columns*, and at each (row, column) position is a different *cell*. It can be described as a *grid* or a *matrix*. Let's call this concept a *uniform table*. A relational database table and a Pandas dataframe are both examples of a uniform table.

The following invariants apply to uniform tables:

- Each row has the same number of cells, one for each column.
- Each column has the same number of cells, one for each row.

2.4.2 Complication 1: Merged Cells

```
+---+---+---+   +---+---+---+
|   a   | b |   |   | b | c |
+---+---+---+   + a +---+---+
| c | d | e |   |   | d | e |
+---+---+---+   +---+---+---+
| f | g | h |   | f | g | h |
+---+---+---+   +---+---+---+
```

While very suitable for data processing, a uniform table lacks expressive power desireable for tables intended for a human reader.

Perhaps the most important characteristic a uniform table lacks is *merged cells*. It is very common to want to group multiple cells into one, for example to form a column-group heading or provide the same value for a sequence of cells rather than repeat it for each cell. These make a rendered table more *readable* by reducing the cognitive load on the human reader and make certain relationships explicit that might easily be missed otherwise.

Unfortunately, accommodating merged cells breaks both the invariants of a uniform table:

- Each row can have a different number of cells.
- Each column can have a different number of cells.

This challenges reading table contents programatically. One might naturally want to read the table into a uniform matrix data structure like a 3 x 3 "2D array" (list of lists perhaps), but this is not directly possible when the table is not known to be uniform.

2.4.3 Concept: The layout grid

```
+ - + - + - +
|   |   |   |
+ - + - + - +
|   |   |   |
+ - + - + - +
|   |   |   |
+ - + - + - +
```

In Word, each table has a *layout grid*.

- The layout grid is *uniform*. There is a layout position for every (layout-row, layout-column) pair.
- The layout grid itself is not visible. However it is represented and referenced by certain elements and attributes within the table XML
- Each table cell is located at a layout-grid position; i.e. the top-left corner of each cell is the top-left corner of a layout-grid cell.
- Each table cell occupies one or more whole layout-grid cells. A merged cell will occupy multiple layout-grid cells. No table cell can occupy a partial layout-grid cell.
- Another way of saying this is that every vertical boundary (left and right) of a cell aligns with a layout-grid vertical boundary, likewise for horizontal boundaries. But not all layout-grid boundaries need be occupied by a cell boundary of the table.

2.4.4 Complication 2: Omitted Cells

```
    +---+---+   +---+---+---+
    | a | b |   | a | b | c |
+---+---+---+   +---+---+---+
| c | d |           | d |
+---+---+       +---+---+---+
    | e |       | e | f | g |
    +---+       +---+---+---+
```

Word is unusual in that it allows cells to be omitted from the beginning or end (but not the middle) of a row. A typical practical example is a table with both a row of column headings and a column of row headings, but no top-left cell (position 0, 0), such as this XOR truth table.

```
    +---+---+
    | T | F |
+---+---+---+
| T | F | T |
+---+---+---+
| F | T | F |
+---+---+---+
```

In *python-docx*, omitted cells in a *_Row* object are represented by the `.grid_cols_before` and `.grid_cols_after` properties. In the example above, for the first row, `.grid_cols_before` would equal `1` and `.grid_cols_after` would equal `0`.

Note that omitted cells are not just "empty" cells. They represent layout-grid positions that are unoccupied by a cell and they cannot be represented by a *_Cell* object. This distinction becomes important when trying to produce a uniform representation (e.g. a 2D array) for an arbitrary Word table.

2.4.5 Concept: *python-docx* approximates uniform tables by default

To accurately represent an arbitrary table would require a complex graph data structure. Navigating this data structure would be at least as complex as navigating the *python-docx* object graph for a table. When extracting content from a collection of arbitrary Word files, such as for indexing the document, it is common to choose a simpler data structure and *approximate* the table in that structure.

Reflecting on how a relational table or dataframe represents tabular information, a straightforward approximation would simply repeat merged-cell values for each layout-grid cell occupied by the merged cell:

```
+---+---+---+      +---+---+---+
|   a   | b |  ->  | a | a | b |
+---+---+---+      +---+---+---+
|   | d | e |  ->  | c | d | e |
+ c +---+---+      +---+---+---+
|   | f | g |  ->  | c | f | g |
+---+---+---+      +---+---+---+
```

This is what `_Row.cells` does by default. Conceptually:

```
>>> [tuple(c.text for c in r.cells) for r in table.rows]
[
  (a, a, b),
  (c, d, e),
  (c, f, g),
]
```

Note this only produces a uniform "matrix" of cells when there are no omitted cells. Dealing with omitted cells requires a more sophisticated approach when maintaining column integrity is required:

```
#     +---+---+
#     | a | b |
# +---+---+---+
# | c | d |
# +---+---+
#     | e |
#     +---+

def iter_row_cell_texts(row: _Row) -> Iterator[str]:
    for _ in range(row.grid_cols_before):
        yield ""
    for c in row.cells:
        yield c.text
    for _ in range(row.grid_cols_after):
        yield ""

>>> [tuple(iter_row_cell_texts(r)) for r in table.rows]
[
  ("",  "a", "b"),
  ("c", "d", ""),
  ("",  "e", ""),
]
```

2.4.6 Complication 3: Tables are Recursive

Further complicating table processing is their recursive nature. In Word, as in HTML, a table cell can itself include one or more tables.

These can be detected using `_Cell.tables` or `_Cell.iter_inner_content()`. The latter preserves the document order of the table with respect to paragraphs also in the cell.

2.5 Working with Text

To work effectively with text, it's important to first understand a little about block-level elements like paragraphs and inline-level objects like runs.

2.5.1 Block-level vs. inline text objects

The paragraph is the primary block-level object in Word.

A block-level item flows the text it contains between its left and right edges, adding an additional line each time the text extends beyond its right boundary. For a paragraph, the boundaries are generally the page margins, but they can also be column boundaries if the page is laid out in columns, or cell boundaries if the paragraph occurs inside a table cell.

A table is also a block-level object.

An inline object is a portion of the content that occurs inside a block-level item. An example would be a word that appears in bold or a sentence in all-caps. The most common inline object is a *run*. All content within a block container is inside of an inline object. Typically, a paragraph contains one or more runs, each of which contain some part of the paragraph's text.

The attributes of a block-level item specify its placement on the page, such items as indentation and space before and after a paragraph. The attributes of an inline item generally specify the font in which the content appears, things like typeface, font size, bold, and italic.

2.5.2 Paragraph properties

A paragraph has a variety of properties that specify its placement within its container (typically a page) and the way it divides its content into separate lines.

In general, it's best to define a *paragraph style* collecting these attributes into a meaningful group and apply the appropriate style to each paragraph, rather than repeatedly apply those properties directly to each paragraph. This is analogous to how Cascading Style Sheets (CSS) work with HTML. All the paragraph properties described here can be set using a style as well as applied directly to a paragraph.

The formatting properties of a paragraph are accessed using the *ParagraphFormat* object available using the paragraph's *paragraph_format* property.

Horizontal alignment (justification)

Also known as *justification*, the horizontal alignment of a paragraph can be set to left, centered, right, or fully justified (aligned on both the left and right sides) using values from the enumeration *WD_PARAGRAPH_ALIGNMENT*:

```
>>> from docx.enum.text import WD_ALIGN_PARAGRAPH
>>> document = Document()
>>> paragraph = document.add_paragraph()
>>> paragraph_format = paragraph.paragraph_format

>>> paragraph_format.alignment
None  # indicating alignment is inherited from the style hierarchy
```

(continues on next page)

(continued from previous page)

```
>>> paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
>>> paragraph_format.alignment
CENTER (1)
```

Indentation

Indentation is the horizontal space between a paragraph and edge of its container, typically the page margin. A paragraph can be indented separately on the left and right side. The first line can also have a different indentation than the rest of the paragraph. A first line indented further than the rest of the paragraph has *first line indent*. A first line indented less has a *hanging indent*.

Indentation is specified using a *Length* value, such as *Inches*, *Pt*, or *Cm*. Negative values are valid and cause the paragraph to overlap the margin by the specified amount. A value of `None` indicates the indentation value is inherited from the style hierarchy. Assigning `None` to an indentation property removes any directly-applied indentation setting and restores inheritance from the style hierarchy:

```
>>> from docx.shared import Inches
>>> paragraph = document.add_paragraph()
>>> paragraph_format = paragraph.paragraph_format

>>> paragraph_format.left_indent
None  # indicating indentation is inherited from the style hierarchy
>>> paragraph_format.left_indent = Inches(0.5)
>>> paragraph_format.left_indent
457200
>>> paragraph_format.left_indent.inches
0.5
```

Right-side indent works in a similar way:

```
>>> from docx.shared import Pt
>>> paragraph_format.right_indent
None
>>> paragraph_format.right_indent = Pt(24)
>>> paragraph_format.right_indent
304800
>>> paragraph_format.right_indent.pt
24.0
```

First-line indent is specified using the *first_line_indent* property and is interpreted relative to the left indent. A negative value indicates a hanging indent:

```
>>> paragraph_format.first_line_indent
None
>>> paragraph_format.first_line_indent = Inches(-0.25)
>>> paragraph_format.first_line_indent
-228600
>>> paragraph_format.first_line_indent.inches
-0.25
```

Tab stops

A tab stop determines the rendering of a tab character in the text of a paragraph. In particular, it specifies the position where the text following the tab character will start, how it will be aligned to that position, and an optional leader character that will fill the horizontal space spanned by the tab.

The tab stops for a paragraph or style are contained in a *TabStops* object accessed using the *tab_stops* property on *ParagraphFormat*:

```
>>> tab_stops = paragraph_format.tab_stops
>>> tab_stops
<docx.text.tabstops.TabStops object at 0x106b802d8>
```

A new tab stop is added using the *add_tab_stop()* method:

```
>>> tab_stop = tab_stops.add_tab_stop(Inches(1.5))
>>> tab_stop.position
1371600
>>> tab_stop.position.inches
1.5
```

Alignment defaults to left, but may be specified by providing a member of the *WD_TAB_ALIGNMENT* enumeration. The leader character defaults to spaces, but may be specified by providing a member of the *WD_TAB_LEADER* enumeration:

```
>>> from docx.enum.text import WD_TAB_ALIGNMENT, WD_TAB_LEADER
>>> tab_stop = tab_stops.add_tab_stop(Inches(1.5), WD_TAB_ALIGNMENT.RIGHT, WD_TAB_LEADER.
↪DOTS)
>>> print(tab_stop.alignment)
RIGHT (2)
>>> print(tab_stop.leader)
DOTS (1)
```

Existing tab stops are accessed using sequence semantics on *TabStops*:

```
>>> tab_stops[0]
<docx.text.tabstops.TabStop object at 0x1105427e8>
```

More details are available in the *TabStops* and *TabStop* API documentation

Paragraph spacing

The *space_before* and *space_after* properties control the spacing between subsequent paragraphs, controlling the spacing before and after a paragraph, respectively. Inter-paragraph spacing is *collapsed* during page layout, meaning the spacing between two paragraphs is the maximum of the *space_after* for the first paragraph and the *space_before* of the second paragraph. Paragraph spacing is specified as a *Length* value, often using *Pt*:

```
>>> paragraph_format.space_before, paragraph_format.space_after
(None, None)  # inherited by default

>>> paragraph_format.space_before = Pt(18)
>>> paragraph_format.space_before.pt
18.0

>>> paragraph_format.space_after = Pt(12)
```

(continues on next page)

(continued from previous page)

```
>>> paragraph_format.space_after.pt
12.0
```

Line spacing

Line spacing is the distance between subsequent baselines in the lines of a paragraph. Line spacing can be specified either as an absolute distance or relative to the line height (essentially the point size of the font used). A typical absolute measure would be 18 points. A typical relative measure would be double-spaced (2.0 line heights). The default line spacing is single-spaced (1.0 line heights).

Line spacing is controlled by the interaction of the *line_spacing* and *line_spacing_rule* properties. *line_spacing* is either a *Length* value, a (small-ish) `float`, or None. A *Length* value indicates an absolute distance. A `float` indicates a number of line heights. `None` indicates line spacing is inherited. *line_spacing_rule* is a member of the *WD_LINE_SPACING* enumeration or `None`:

```
>>> from docx.shared import Length
>>> paragraph_format.line_spacing
None
>>> paragraph_format.line_spacing_rule
None

>>> paragraph_format.line_spacing = Pt(18)
>>> isinstance(paragraph_format.line_spacing, Length)
True
>>> paragraph_format.line_spacing.pt
18.0
>>> paragraph_format.line_spacing_rule
EXACTLY (4)

>>> paragraph_format.line_spacing = 1.75
>>> paragraph_format.line_spacing
1.75
>>> paragraph_format.line_spacing_rule
MULTIPLE (5)
```

Pagination properties

Four paragraph properties, *keep_together*, *keep_with_next*, *page_break_before*, and *widow_control* control aspects of how the paragraph behaves near page boundaries.

keep_together causes the entire paragraph to appear on the same page, issuing a page break before the paragraph if it would otherwise be broken across two pages.

keep_with_next keeps a paragraph on the same page as the subsequent paragraph. This can be used, for example, to keep a section heading on the same page as the first paragraph of the section.

page_break_before causes a paragraph to be placed at the top of a new page. This could be used on a chapter heading to ensure chapters start on a new page.

widow_control breaks a page to avoid placing the first or last line of the paragraph on a separate page from the rest of the paragraph.

All four of these properties are *tri-state*, meaning they can take the value `True`, `False`, or `None`. `None` indicates the property value is inherited from the style hierarchy. `True` means "on" and `False` means "off":

```
>>> paragraph_format.keep_together
None  # all four inherit by default
>>> paragraph_format.keep_with_next = True
>>> paragraph_format.keep_with_next
True
>>> paragraph_format.page_break_before = False
>>> paragraph_format.page_break_before
False
```

2.5.3 Apply character formatting

Character formatting is applied at the Run level. Examples include font typeface and size, bold, italic, and underline.

A *Run* object has a read-only *font* property providing access to a *Font* object. A run's *Font* object provides properties for getting and setting the character formatting for that run.

Several examples are provided here. For a complete set of the available properties, see the *Font* API documentation.

The font for a run can be accessed like this:

```
>>> from docx import Document
>>> document = Document()
>>> run = document.add_paragraph().add_run()
>>> font = run.font
```

Typeface and size are set like this:

```
>>> from docx.shared import Pt
>>> font.name = 'Calibri'
>>> font.size = Pt(12)
```

Many font properties are *tri-state*, meaning they can take the values `True`, `False`, and `None`. `True` means the property is "on", `False` means it is "off". Conceptually, the `None` value means "inherit". A run exists in the style inheritance hierarchy and by default inherits its character formatting from that hierarchy. Any character formatting directly applied using the *Font* object overrides the inherited values.

Bold and italic are tri-state properties, as are all-caps, strikethrough, superscript, and many others. See the *Font* API documentation for a full list:

```
>>> font.bold, font.italic
(None, None)
>>> font.italic = True
>>> font.italic
True
>>> font.italic = False
>>> font.italic
False
>>> font.italic = None
>>> font.italic
None
```

Underline is a bit of a special case. It is a hybrid of a tri-state property and an enumerated value property. `True` means single underline, by far the most common. `False` means no underline, but more often `None` is the right choice if no underlining is wanted. The other forms of underlining, such as double or dashed, are specified with a member of the *WD_UNDERLINE* enumeration:

```
>>> font.underline
None
>>> font.underline = True
>>> # or perhaps
>>> font.underline = WD_UNDERLINE.DOT_DASH
```

Font color

Each *Font* object has a *ColorFormat* object that provides access to its color, accessed via its read-only *color* property.

Apply a specific RGB color to a font:

```
>>> from docx.shared import RGBColor
>>> font.color.rgb = RGBColor(0x42, 0x24, 0xE9)
```

A font can also be set to a theme color by assigning a member of the *MSO_THEME_COLOR_INDEX* enumeration:

```
>>> from docx.enum.dml import MSO_THEME_COLOR
>>> font.color.theme_color = MSO_THEME_COLOR.ACCENT_1
```

A font's color can be restored to its default (inherited) value by assigning `None` to either the *rgb* or *theme_color* attribute of *ColorFormat*:

```
>>> font.color.rgb = None
```

Determining the color of a font begins with determining its color type:

```
>>> font.color.type
RGB (1)
```

The value of the *type* property can be a member of the *MSO_COLOR_TYPE* enumeration or None. *MSO_COLOR_TYPE.RGB* indicates it is an RGB color. *MSO_COLOR_TYPE.THEME* indicates a theme color. *MSO_COLOR_TYPE.AUTO* indicates its value is determined automatically by the application, usually set to black. (This value is relatively rare.) `None` indicates no color is applied and the color is inherited from the style hierarchy; this is the most common case.

When the color type is *MSO_COLOR_TYPE.RGB*, the *rgb* property will be an *RGBColor* value indicating the RGB color:

```
>>> font.color.rgb
RGBColor(0x42, 0x24, 0xe9)
```

When the color type is *MSO_COLOR_TYPE.THEME*, the *theme_color* property will be a member of *MSO_THEME_COLOR_INDEX* indicating the theme color:

```
>>> font.color.theme_color
ACCENT_1 (5)
```

2.6 Working with Sections

Word supports the notion of a *section*, a division of a document having the same page layout settings, such as margins and page orientation. This is how, for example, a document can contain some pages in portrait layout and others in landscape. Each section also defines the headers and footers that apply to the pages of that section.

Most Word documents have only the single section that comes by default and further, most of those have no reason to change the default margins or other page layout. But when you *do* need to change the page layout, you'll need to understand sections to get it done.

2.6.1 Accessing sections

Access to document sections is provided by the `sections` property on the *Document* object:

```
>>> document = Document()
>>> sections = document.sections
>>> sections
<docx.parts.document.Sections object at 0x1deadbeef>
>>> len(sections)
3
>>> section = sections[0]
>>> section
<docx.section.Section object at 0x1deadbeef>
>>> for section in sections:
...     print(section.start_type)
...
NEW_PAGE (2)
EVEN_PAGE (3)
ODD_PAGE (4)
```

It's theoretically possible for a document not to have any explicit sections, although I've yet to see this occur in the wild. If you're accessing an unpredictable population of .docx files you may want to provide for that possibility using a `len()` check or `try` block to avoid an uncaught `IndexError` exception stopping your program.

2.6.2 Adding a new section

The `Document.add_section()` method allows a new section to be started at the end of the document. Paragraphs and tables added after calling this method will appear in the new section:

```
>>> current_section = document.sections[-1]  # last section in document
>>> current_section.start_type
NEW_PAGE (2)
>>> new_section = document.add_section(WD_SECTION.ODD_PAGE)
>>> new_section.start_type
ODD_PAGE (4)
```

2.6.3 Section properties

The *Section* object has eleven properties that allow page layout settings to be discovered and specified.

Section start type

Section.start_type describes the type of break that precedes the section:

```
>>> section.start_type
NEW_PAGE (2)
>>> section.start_type = WD_SECTION.ODD_PAGE
>>> section.start_type
ODD_PAGE (4)
```

Values of `start_type` are members of the *WD_SECTION_START* enumeration.

Page dimensions and orientation

Three properties on *Section* describe page dimensions and orientation. Together these can be used, for example, to change the orientation of a section from portrait to landscape:

```
>>> section.orientation, section.page_width, section.page_height
(PORTRAIT (0), 7772400, 10058400)  # (Inches(8.5), Inches(11))
>>> new_width, new_height = section.page_height, section.page_width
>>> section.orientation = WD_ORIENT.LANDSCAPE
>>> section.page_width = new_width
>>> section.page_height = new_height
>>> section.orientation, section.page_width, section.page_height
(LANDSCAPE (1), 10058400, 7772400)
```

Page margins

Seven properties on *Section* together specify the various edge spacings that determine where text appears on the page:

```
>>> from docx.shared import Inches
>>> section.left_margin, section.right_margin
(1143000, 1143000)  # (Inches(1.25), Inches(1.25))
>>> section.top_margin, section.bottom_margin
(914400, 914400)  # (Inches(1), Inches(1))
>>> section.gutter
0
>>> section.header_distance, section.footer_distance
(457200, 457200)  # (Inches(0.5), Inches(0.5))
>>> section.left_margin = Inches(1.5)
>>> section.right_margin = Inches(1)
>>> section.left_margin, section.right_margin
(1371600, 914400)
```

2.7 Working with Headers and Footers

Word supports *page headers* and *page footers*. A page header is text that appears in the top margin area of each page, separated from the main body of text, and usually conveying context information, such as the document title, author, creation date, or the page number. The page headers in a document are the same from page to page, with only small differences in content, such as a changing section title or page number. A page header is also known as a *running head*.

A *page footer* is analogous in every way to a page header except that it appears at the bottom of a page. It should not be confused with a footnote, which is not uniform between pages. For brevity's sake, the term *header* is often used here to refer to what may be either a header or footer object, trusting the reader to understand its applicability to both object types.

2.7.1 Accessing the header for a section

Headers and footers are linked to a *section*; this allows each section to have a distinct header and/or footer. For example, a landscape section might have a wider header than a portrait section.

Each section object has a `.header` property providing access to a *_Header* object for that section:

```
>>> document = Document()
>>> section = document.sections[0]
>>> header = section.header
```

(continues on next page)

(continued from previous page)

```
>>> header
<docx.section._Header object at 0x...>
```

A *_Header* object is *always* present on `Section.header`, even when no header is defined for that section. The presence of an actual header definition is indicated by `_Header.is_linked_to_previous`:

```
>>> header.is_linked_to_previous
True
```

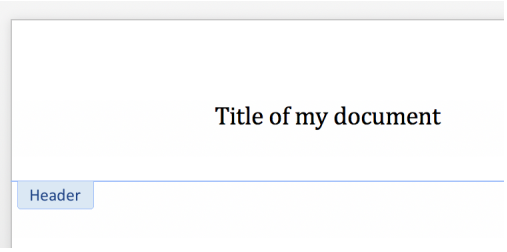
A value of `True` indicates the *_Header* object contains no header definition and the section will display the same header as the previous section. This "inheritance" behavior is recursive, such that a "linked" header actually gets its definition from the first prior section having a header definition. This "linked" state is indicated as *"Same as previous"* in the Word UI.

A new document does not have a header (on the single section it contains) and so `.is_linked_to_previous` is `True` in that case. Note this case may be a bit counterintuitive in that there *is no previous section header* to link to. In this "no previous header" case, no header is displayed.

2.7.2 Adding a header (simple case)

A header can be added to a new document simply by editing the content of the *_Header* object. A *_Header* object is a "story" container and its content is edited just like a *Document* object. Note that like a new document, a new header already contains a single (empty) paragraph:

```
>>> paragraph = header.paragraphs[0]
>>> paragraph.text = "Title of my document"
```



Note also that the act of adding content (or even just accessing `header.paragraphs`) added a header definition and changed the state of `.is_linked_to_previous`:

```
>>> header.is_linked_to_previous
False
```

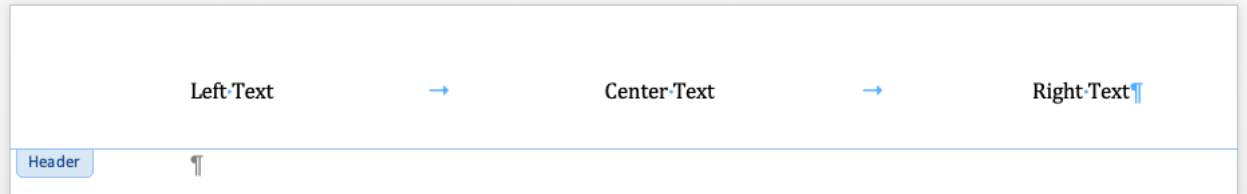
2.7.3 Adding "zoned" header content

A header with multiple "zones" is often accomplished using carefully placed tab stops.

The required tab-stops for a center and right-aligned "zone" are part of the `Header` and `Footer` styles in Word. If you're using a custom template rather than the *python-docx* default, it probably makes sense to define that style in your template.

Inserted tab characters (`"\t"`) are used to separate left, center, and right-aligned header content:

```
>>> paragraph = header.paragraphs[0]
>>> paragraph.text = "Left Text\tCenter Text\tRight Text"
>>> paragraph.style = document.styles["Header"]
```



The `Header` style is automatically applied to a new header, so the third line just above (applying the `Header` style) is unnecessary in this case, but included here to illustrate the general case.

2.7.4 Removing a header

An unwanted header can be removed by assigning `True` to its `.is_linked_to_previous` attribute:

```
>>> header.is_linked_to_previous = True
>>> header.is_linked_to_previous
True
```

The content for a header is irreversably deleted when `True` is assigned to `.is_linked_to_previous`.

2.7.5 Understanding headers in a multi-section document

The "just start editing" approach works fine for the simple case, but to make sense of header behaviors in a multi-section document, a few simple concepts will be helpful. Here they are in a nutshell:

1. Each section can have its own header definition (but doesn't have to).
2. A section that lacks a header definition inherits the header of the section before it. The `_Header.is_linked_to_previous` property simply reflects the presence of a header definition, `False` when a definition is present and `True` when not.
3. Lacking a header definition is the default state. A new document has no defined header and neither does a newly-inserted section. `.is_linked_to_previous` reports `True` in both those cases.
4. The content of a `_Header` object is its own content if it has a header definition. If not, its content is that of the first prior section that *does* have a header definition. If no sections have a header definition, a new one is added on the first section and all other sections inherit that one. This adding of a header definition happens the first time header content is accessed, perhaps by referencing `header.paragraphs`.

2.7.6 Adding a header definition (general case)

An explicit header definition can be given to a section that lacks one by assigning `False` to its `.is_linked_to_previous` property:

```
>>> header.is_linked_to_previous
True
>>> header.is_linked_to_previous = False
>>> header.is_linked_to_previous
False
```

The newly added header definition contains a single empty paragraph. Note that leaving the header this way is occasionally useful as it effectively "turns-off" a header for that section and those after it until the next section with a defined header.

Assigning `False` to `.is_linked_to_previous` on a header that already has a header definition does nothing.

Inherited content is automatically located

Editing the content of a header edits the content of the *source* header, taking into account any "inheritance". So for example, if the section 2 header inherits from section 1 and you edit the section 2 header, you actually change the contents of the section 1 header. A new header definition is not added for section 2 unless you first explicitly assign `False` to its `.is_linked_to_previous` property.

2.8 API basics

The API for `python-docx` is designed to make doing simple things simple, while allowing more complex results to be achieved with a modest and incremental investment of understanding.

It's possible to create a basic document using only a single object, the `docx.api.Document` object returned when opening a file. The methods on `docx.api.Document` allow *block-level* objects to be added to the end of the document. Block-level objects include paragraphs, inline pictures, and tables. Headings, bullets, and numbered lists are simply paragraphs with a particular style applied.

In this way, a document can be "written" from top to bottom, roughly like a person would if they knew exactly what they wanted to say This basic use case, where content is always added to the end of the document, is expected to account for perhaps 80% of actual use cases, so it's a priority to make it as simple as possible without compromising the power of the overall API.

2.8.1 Inline objects

Each block-level method on `docx.api.Document`, such as `add_paragraph()`, returns the block-level object created. Often the reference is unneeded; but when inline objects must be created individually, you'll need the block-item reference to do it.

… add example here as API solidifies …

2.9 Understanding Styles

Grasshopper:
"Master, why doesn't my paragraph appear with the style I specified?"

Master:
"You have come to the right page Grasshopper; read on …"

2.9.1 What is a style in Word?

Documents communicate better when like elements are formatted consistently. To achieve that consistency, professional document designers develop a *style sheet* which defines the document element types and specifies how each should be formatted. For example, perhaps body paragraphs are to be set in 9 pt Times Roman with a line height of 11 pt, justified flush left, ragged right. When these specifications are applied to each of the elements of the document, a consistent and polished look is achieved.

A style in Word is such a set of specifications that may be applied, all at once, to a document element. Word has paragraph styles, character styles, table styles, and numbering definitions. These are applied to a paragraph, a span of text, a table, and a list, respectively.

Experienced programmers will recognize styles as a level of indirection. The great thing about those is it allows you to define something once, then apply that definition many times. This saves the work of defining the same thing over an over; but more importantly it allows you to change the definition and have that change reflected in all the places you have applied it.

2.9.2 Why doesn't the style I applied show up?

This is likely to show up quite a bit until I can add some fancier features to work around it, so here it is up top.

1. When you're working in Word, there are all these styles you can apply to things, pretty good looking ones that look all the better because you don't have to make them yourself. Most folks never look further than the built-in styles.
2. Although those styles show up in the UI, they're not actually in the document you're creating, at least not until you use it for the first time. That's kind of a good thing. They take up room and there's a lot of them. The file would get a little bloated if it contained all the style definitions you could use but haven't.
3. If you apply a style using `python-docx` that's not defined in your file (in the styles.xml part if you're curious), Word just ignores it. It doesn't complain, it just doesn't change how things are formatted. I'm sure there's a good reason for this. But it can present as a bit of a puzzle if you don't understand how Word works that way.
4. When you use a style, Word adds it to the file. Once there, it stays. I imagine there's a way to get rid of it, but you have to work at it. If you apply a style, delete the content you applied it to, and then save the document; the style definition stays in the saved file.

All this adds up to the following: If you want to use a style in a document you create with `python-docx`, the document you start with must contain the style definition. Otherwise it just won't work. It won't raise an exception, it just won't work.

If you use the "default" template document, it contains the styles listed below, most of the ones you're likely to want if you're not designing your own. If you're using your own starting document, you need to use each of the styles you want at least once in it. You don't have to keep the content, but you need to apply the style to something at least once before saving the document. Creating a one-word paragraph, applying five styles to it in succession and then deleting the paragraph works fine. That's how I got the ones below into the default template :).

2.9.3 Glossary

style definition
: A `<w:style>` element in the styles part of a document that explicitly defines the attributes of a style.

defined style
: A style that is explicitly defined in a document. Contrast with *latent style*.

built-in style
: One of the set of 276 pre-set styles built into Word, such as "Heading 1". A built-in style can be either defined or latent. A built-in style that is not yet defined is known as a *latent style*. Both defined and latent built-in styles may appear as options in Word's style panel and style gallery.

custom style
: Also known as a *user defined style*, any style defined in a Word document that is not a built-in style. Note that a custom style cannot be a latent style.

latent style
: A built-in style having no definition in a particular document is known as a *latent style* in that document. A latent style can appear as an option in the Word UI depending on the settings in the `LatentStyles` object for the document.

recommended style list
: A list of styles that appears in the styles toolbox or panel when "Recommended" is selected from the "List:" dropdown box.

Style Gallery
: The selection of example styles that appear in the ribbon of the Word UI and which may be applied by clicking on one of them.

2.9.4 Identifying a style

A style has three identifying properties, *name*, *style_id*, and *type*.

Each style's `name` property is its stable, unique identifier for access purposes.

A style's `style_id` is used internally to key a content object such as a paragraph to its style. However this value is generated automatically by Word and is not guaranteed to be stable across saves. In general, the style id is formed simply by removing spaces from the *localized* style name, however there are exceptions. Users of `python-docx` should generally avoid using the style id unless they are confident with the internals involved.

A style's `type` is set at creation time and cannot be changed.

2.9.5 Built-in styles

Word comes with almost 300 so-called *built-in* styles like *Normal*, *Heading 1*, and *List Bullet*. Style definitions are stored in the *styles.xml* part of a .docx package, but built-in style definitions are stored in the Word application itself and are not written to *styles.xml* until they are actually used. This is a sensible strategy because they take up considerable room and would be largely redundant and useless overhead in every .docx file otherwise.

The fact that built-in styles are not written to the .docx package until used gives rise to the need for *latent style* definitions, explained below.

2.9.6 Style Behavior

In addition to collecting a set of formatting properties, a style has five properties that specify its *behavior*. This behavior is relatively simple, basically amounting to when and where the style appears in the Word or LibreOffice UI.

The key notion to understanding style behavior is the recommended list. In the style pane in Word, the user can select which list of styles they want to see. One of these is named *Recommended* and is known as the *recommended list*. All five behavior properties affect some aspect of the style's appearance in this list and in the style gallery.

In brief, a style appears in the recommended list if its `hidden` property is `False` (the default). If a style is not hidden and its `quick_style` property is `True`, it also appears in the style gallery. If a hidden style's `unhide_when_used` property is `True`, its hidden property is set `False` the first time it is used. Styles in the style lists and style gallery are sorted in `priority` order, then alphabetically for styles of the same priority. If a style's `locked` property is `True` and formatting restrictions are turned on for the document, the style will not appear in any list or the style gallery and cannot be applied to content.

2.9.7 Latent styles

The need to specify the UI behavior of built-in styles not defined in *styles.xml* gives rise to the need for *latent style* definitions. A latent style definition is basically a stub style definition that has at most the five behavior attributes in addition to the style name. Additional space is saved by defining defaults for each of the behavior attributes, so only those that differ from the default need be defined and styles that match all defaults need no latent style definition.

Latent style definitions are specified using the *w:latentStyles* and *w:lsdException* elements appearing in *styles.xml*.

A latent style definition is only required for a built-in style because only a built-in style can appear in the UI without a style definition in *styles.xml*.

2.9.8 Style inheritance

A style can inherit properties from another style, somewhat similarly to how Cascading Style Sheets (CSS) works. Inheritance is specified using the `base_style` attribute. By basing one style on another, an inheritance hierarchy of arbitrary depth can be formed. A style having no base style inherits properties from the document defaults.

2.9.9 Paragraph styles in default template

- Normal
- Body Text
- Body Text 2
- Body Text 3
- Caption
- Heading 1
- Heading 2
- Heading 3
- Heading 4
- Heading 5
- Heading 6
- Heading 7
- Heading 8
- Heading 9
- Intense Quote
- List
- List 2
- List 3
- List Bullet
- List Bullet 2
- List Bullet 3
- List Continue
- List Continue 2
- List Continue 3
- List Number
- List Number 2
- List Number 3
- List Paragraph
- Macro Text
- No Spacing
- Quote
- Subtitle
- TOCHeading
- Title

2.9.10 Character styles in default template

- Body Text Char
- Body Text 2 Char
- Body Text 3 Char
- Book Title
- Default Paragraph Font
- Emphasis
- Heading 1 Char
- Heading 2 Char
- Heading 3 Char
- Heading 4 Char
- Heading 5 Char
- Heading 6 Char
- Heading 7 Char
- Heading 8 Char
- Heading 9 Char
- Intense Emphasis
- Intense Quote Char
- Intense Reference
- Macro Text Char
- Quote Char
- Strong
- Subtitle Char
- Subtle Emphasis
- Subtle Reference
- Title Char

2.9.11 Table styles in default template

- Table Normal
- Colorful Grid
- Colorful Grid Accent 1
- Colorful Grid Accent 2
- Colorful Grid Accent 3
- Colorful Grid Accent 4
- Colorful Grid Accent 5
- Colorful Grid Accent 6

- Colorful List
- Colorful List Accent 1
- Colorful List Accent 2
- Colorful List Accent 3
- Colorful List Accent 4
- Colorful List Accent 5
- Colorful List Accent 6
- Colorful Shading
- Colorful Shading Accent 1
- Colorful Shading Accent 2
- Colorful Shading Accent 3
- Colorful Shading Accent 4
- Colorful Shading Accent 5
- Colorful Shading Accent 6
- Dark List
- Dark List Accent 1
- Dark List Accent 2
- Dark List Accent 3
- Dark List Accent 4
- Dark List Accent 5
- Dark List Accent 6
- Light Grid
- Light Grid Accent 1
- Light Grid Accent 2
- Light Grid Accent 3
- Light Grid Accent 4
- Light Grid Accent 5
- Light Grid Accent 6
- Light List
- Light List Accent 1
- Light List Accent 2
- Light List Accent 3
- Light List Accent 4
- Light List Accent 5
- Light List Accent 6
- Light Shading

- Light Shading Accent 1
- Light Shading Accent 2
- Light Shading Accent 3
- Light Shading Accent 4
- Light Shading Accent 5
- Light Shading Accent 6
- Medium Grid 1
- Medium Grid 1 Accent 1
- Medium Grid 1 Accent 2
- Medium Grid 1 Accent 3
- Medium Grid 1 Accent 4
- Medium Grid 1 Accent 5
- Medium Grid 1 Accent 6
- Medium Grid 2
- Medium Grid 2 Accent 1
- Medium Grid 2 Accent 2
- Medium Grid 2 Accent 3
- Medium Grid 2 Accent 4
- Medium Grid 2 Accent 5
- Medium Grid 2 Accent 6
- Medium Grid 3
- Medium Grid 3 Accent 1
- Medium Grid 3 Accent 2
- Medium Grid 3 Accent 3
- Medium Grid 3 Accent 4
- Medium Grid 3 Accent 5
- Medium Grid 3 Accent 6
- Medium List 1
- Medium List 1 Accent 1
- Medium List 1 Accent 2
- Medium List 1 Accent 3
- Medium List 1 Accent 4
- Medium List 1 Accent 5
- Medium List 1 Accent 6
- Medium List 2
- Medium List 2 Accent 1

- Medium List 2 Accent 2
- Medium List 2 Accent 3
- Medium List 2 Accent 4
- Medium List 2 Accent 5
- Medium List 2 Accent 6
- Medium Shading 1
- Medium Shading 1 Accent 1
- Medium Shading 1 Accent 2
- Medium Shading 1 Accent 3
- Medium Shading 1 Accent 4
- Medium Shading 1 Accent 5
- Medium Shading 1 Accent 6
- Medium Shading 2
- Medium Shading 2 Accent 1
- Medium Shading 2 Accent 2
- Medium Shading 2 Accent 3
- Medium Shading 2 Accent 4
- Medium Shading 2 Accent 5
- Medium Shading 2 Accent 6
- Table Grid

2.10 Working with Styles

This page uses concepts developed in the prior page without introduction. If a term is unfamiliar, consult the prior page *Understanding Styles* for a definition.

2.10.1 Access a style

Styles are accessed using the *Document.styles* attribute:

```
>>> document = Document()
>>> styles = document.styles
>>> styles
<docx.styles.styles.Styles object at 0x10a7c4f50>
```

The *Styles* object provides dictionary-style access to defined styles by name:

```
>>> styles['Normal']
<docx.styles.style._ParagraphStyle object at <0x10a7c4f6b>
```

Note

Built-in styles are stored in a WordprocessingML file using their English name, e.g. 'Heading 1', even though users working on a localized version of Word will see native language names in the UI, e.g. 'Kop 1'. Because `python-docx` operates on the WordprocessingML file, style lookups must use the English name. A document available on this external site allows you to create a mapping between local language names and English style names: http://www.thedoctools.com/index.php?show=mt_create_style_name_list

User-defined styles, also known as *custom styles*, are not localized and are accessed with the name exactly as it appears in the Word UI.

The *Styles* object is also iterable. By using the identification properties on *BaseStyle*, various subsets of the defined styles can be generated. For example, this code will produce a list of the defined paragraph styles:

```
>>> from docx.enum.style import WD_STYLE_TYPE
>>> styles = document.styles
>>> paragraph_styles = [
...     s for s in styles if s.type == WD_STYLE_TYPE.PARAGRAPH
... ]
>>> for style in paragraph_styles:
...     print(style.name)
...
Normal
Body Text
List Bullet
```

2.10.2 Apply a style

The *Paragraph*, *Run*, and *Table* objects each have a `style` attribute. Assigning a style object to this attribute applies that style:

```
>>> document = Document()
>>> paragraph = document.add_paragraph()
>>> paragraph.style
<docx.styles.style._ParagraphStyle object at <0x11a7c4c50>
>>> paragraph.style.name
'Normal'
>>> paragraph.style = document.styles['Heading 1']
>>> paragraph.style.name
'Heading 1'
```

A style name can also be assigned directly, in which case `python-docx` will do the lookup for you:

```
>>> paragraph.style = 'List Bullet'
>>> paragraph.style
<docx.styles.style._ParagraphStyle object at <0x10a7c4f84>
>>> paragraph.style.name
'List Bullet'
```

A style can also be applied at creation time using either the style object or its name:

```
>>> paragraph = document.add_paragraph(style='Body Text')
>>> paragraph.style.name
```

(continues on next page)

(continued from previous page)

```
'Body Text'
>>> body_text_style = document.styles['Body Text']
>>> paragraph = document.add_paragraph(style=body_text_style)
>>> paragraph.style.name
'Body Text'
```

2.10.3 Add or delete a style

A new style can be added to the document by specifying a unique name and a style type:

```
>>> from docx.enum.style import WD_STYLE_TYPE
>>> styles = document.styles
>>> style = styles.add_style('Citation', WD_STYLE_TYPE.PARAGRAPH)
>>> style.name
'Citation'
>>> style.type
PARAGRAPH (1)
```

Use the `base_style` property to specify a style the new style should inherit formatting settings from:

```
>>> style.base_style
None
>>> style.base_style = styles['Normal']
>>> style.base_style
<docx.styles.style._ParagraphStyle object at 0x10a7a9550>
>>> style.base_style.name
'Normal'
```

A style can be removed from the document simply by calling its *delete()* method:

```
>>> styles = document.styles
>>> len(styles)
10
>>> styles['Citation'].delete()
>>> len(styles)
9
```

Note

The `Style.delete()` method removes the style's definition from the document. It does not affect content in the document to which that style is applied. Content having a style not defined in the document is rendered using the default style for that content object, e.g. 'Normal' in the case of a paragraph.

2.10.4 Define character formatting

Character, paragraph, and table styles can all specify character formatting to be applied to content with that style. All the character formatting that can be applied directly to text can be specified in a style. Examples include font typeface and size, bold, italic, and underline.

Each of these three style types have a `font` attribute providing access to a *Font* object. A style's *Font* object provides properties for getting and setting the character formatting for that style.

Several examples are provided here. For a complete set of the available properties, see the *Font* API documentation.

The font for a style can be accessed like this:

```
>>> from docx import Document
>>> document = Document()
>>> style = document.styles['Normal']
>>> font = style.font
```

Typeface and size are set like this:

```
>>> from docx.shared import Pt
>>> font.name = 'Calibri'
>>> font.size = Pt(12)
```

Many font properties are *tri-state*, meaning they can take the values `True`, `False`, and `None`. `True` means the property is "on", `False` means it is "off". Conceptually, the `None` value means "inherit". Because a style exists in an inheritance hierarchy, it is important to have the ability to specify a property at the right place in the hierarchy, generally as far up the hierarchy as possible. For example, if all headings should be in the Arial typeface, it makes more sense to set that property on the *Heading 1* style and have *Heading 2* inherit from *Heading 1*.

Bold and italic are tri-state properties, as are all-caps, strikethrough, superscript, and many others. See the *Font* API documentation for a full list:

```
>>> font.bold, font.italic
(None, None)
>>> font.italic = True
>>> font.italic
True
>>> font.italic = False
>>> font.italic
False
>>> font.italic = None
>>> font.italic
None
```

Underline is a bit of a special case. It is a hybrid of a tri-state property and an enumerated value property. `True` means single underline, by far the most common. `False` means no underline, but more often `None` is the right choice if no underlining is wanted since it is rare to inherit it from a base style. The other forms of underlining, such as double or dashed, are specified with a member of the *WD_UNDERLINE* enumeration:

```
>>> font.underline
None
>>> font.underline = True
>>> # or perhaps
>>> font.underline = WD_UNDERLINE.DOT_DASH
```

2.10.5 Define paragraph formatting

Both a paragraph style and a table style allow paragraph formatting to be specified. These styles provide access to a *ParagraphFormat* object via their `paragraph_format` property.

Paragraph formatting includes layout behaviors such as justification, indentation, space before and after, page break before, and widow/orphan control. For a complete list of the available properties, consult the API documentation page for the *ParagraphFormat* object.

Here's an example of how you would create a paragraph style having hanging indentation of 1/4 inch, 12 points spacing above, and widow/orphan control:

```
>>> from docx.enum.style import WD_STYLE_TYPE
>>> from docx.shared import Inches, Pt
>>> document = Document()
>>> style = document.styles.add_style('Indent', WD_STYLE_TYPE.PARAGRAPH)
>>> paragraph_format = style.paragraph_format
>>> paragraph_format.left_indent = Inches(0.25)
>>> paragraph_format.first_line_indent = Inches(-0.25)
>>> paragraph_format.space_before = Pt(12)
>>> paragraph_format.widow_control = True
```

2.10.6 Use paragraph-specific style properties

A paragraph style has a `next_paragraph_style` property that specifies the style to be applied to new paragraphs inserted after a paragraph of that style. This is most useful when the style would normally appear only once in a sequence, such as a heading. In that case, the paragraph style can automatically be set back to a body style after completing the heading.

In the most common case (body paragraphs), subsequent paragraphs should receive the same style as the current paragraph. The default handles this case well by applying the same style if a next paragraph style is not specified.

Here's an example of how you would change the next paragraph style of the *Heading 1* style to *Body Text*:

```
>>> from docx import Document
>>> document = Document()
>>> styles = document.styles

>>> styles['Heading 1'].next_paragraph_style = styles['Body Text']
```

The default behavior can be restored by assigning `None` or the style itself:

```
>>> heading_1_style = styles['Heading 1']
>>> heading_1_style.next_paragraph_style.name
'Body Text'

>>> heading_1_style.next_paragraph_style = heading_1_style
>>> heading_1_style.next_paragraph_style.name
'Heading 1'

>>> heading_1_style.next_paragraph_style = None
>>> heading_1_style.next_paragraph_style.name
'Heading 1'
```

2.10.7 Control how a style appears in the Word UI

The properties of a style fall into two categories, *behavioral properties* and *formatting properties*. Its behavioral properties control when and where the style appears in the Word UI. Its formatting properties determine the formatting of content to which the style is applied, such as the size of the font and its paragraph indentation.

There are five behavioral properties of a style:

- *hidden*
- *unhide_when_used*
- *priority*
- *quick_style*

- *locked*

See the *Style Behavior* section in *Understanding Styles* for a description of how these behavioral properties interact to determine when and where a style appears in the Word UI.

The `priority` property takes an integer value. The other four style behavior properties are *tri-state*, meaning they can take the value `True` (on), `False` (off), or `None` (inherit).

Display a style in the style gallery

The following code will cause the 'Body Text' paragraph style to appear first in the style gallery:

```
>>> from docx import Document
>>> document = Document()
>>> style = document.styles['Body Text']

>>> style.hidden = False
>>> style.quick_style = True
>>> style.priorty = 1
```

Remove a style from the style gallery

This code will remove the 'Normal' paragraph style from the style gallery, but allow it to remain in the recommended list:

```
>>> style = document.styles['Normal']

>>> style.hidden = False
>>> style.quick_style = False
```

2.10.8 Working with Latent Styles

See the *Built-in styles* and *Latent styles* sections in *Understanding Styles* for a description of how latent styles define the behavioral properties of built-in styles that are not yet defined in the *styles.xml* part of a .docx file.

Access the latent styles in a document

The latent styles in a document are accessed from the styles object:

```
>>> document = Document()
>>> latent_styles = document.styles.latent_styles
```

A *LatentStyles* object supports `len()`, iteration, and dictionary-style access by style name:

```
>>> len(latent_styles)
161

>>> latent_style_names = [ls.name for ls in latent_styles]
>>> latent_style_names
['Normal', 'Heading 1', 'Heading 2', ... 'TOC Heading']

>>> latent_quote = latent_styles['Quote']
>>> latent_quote
<docx.styles.latent.LatentStyle object at 0x10a7c4f50>
>>> latent_quote.priority
29
```

Change latent style defaults

The *LatentStyles* object also provides access to the default behavioral properties for built-in styles in the current document. These defaults provide the value for any undefined attributes of the *_LatentStyle* definitions and to all behavioral properties of built-in styles having no explicit latent style definition. See the API documentation for the *LatentStyles* object for the complete set of available properties:

```
>>> latent_styles.default_to_locked
False
>>> latent_styles.default_to_locked = True
>>> latent_styles.default_to_locked
True
```

Add a latent style definition

A new latent style can be added using the *add_latent_style()* method on *LatentStyles*. This code adds a new latent style for the builtin style 'List Bullet', setting it to appear in the style gallery:

```
>>> latent_style = latent_styles['List Bullet']
KeyError: no latent style with name 'List Bullet'
>>> latent_style = latent_styles.add_latent_style('List Bullet')
>>> latent_style.hidden = False
>>> latent_style.priority = 2
>>> latent_style.quick_style = True
```

Delete a latent style definition

A latent style definition can be deleted by calling its `delete()` method:

```
>>> latent_styles['Light Grid']
<docx.styles.latent.LatentStyle object at 0x10a7c4f50>
>>> latent_styles['Light Grid'].delete()
>>> latent_styles['Light Grid']
KeyError: no latent style with name 'Light Grid'
```

2.11 Understanding pictures and other shapes

Conceptually, Word documents have two *layers*, a *text layer* and a *drawing layer*. In the text layer, text objects are flowed from left to right and from top to bottom, starting a new page when the prior one is filled. In the drawing layer, drawing objects, called *shapes*, are placed at arbitrary positions. These are sometimes referred to as *floating* shapes.

A picture is a shape that can appear in either the text or drawing layer. When it appears in the text layer it is called an *inline shape*, or more specifically, an *inline picture*.

Inline shapes are treated like a big text character (a *character glyph*). The line height is increased to accomodate the shape and the shape is wrapped to a line it will fit on width-wise, just like text. Inserting text in front of it will cause it to move to the right. Often, a picture is placed in a paragraph by itself, but this is not required. It can have text before and after it in the paragraph in which it's placed.

At the time of writing, `python-docx` only supports inline pictures. Floating pictures can be added. If you have an active use case, submit a feature request on the issue tracker. The `Document.add_picture()` method adds a specified picture to the end of the document in a paragraph of its own. However, by digging a little deeper into the API you can place text on either side of the picture in its paragraph, or both.

CHAPTER

THREE

API DOCUMENTATION

3.1 Document objects

The main Document and related objects.

3.1.1 `Document` constructor

docx.**Document**(*docx: str | IO[bytes] | None = None*) → DocumentObject

Return a *Document* object loaded from *docx*, where *docx* can be either a path to a `.docx` file (a string) or a file-like object.

If *docx* is missing or `None`, the built-in default document "template" is loaded.

3.1.2 `Document` objects

class docx.document.**Document**

WordprocessingML (WML) document.

Not intended to be constructed directly. Use *docx.Document()* to open or create a document.

add_heading(*text: str = ''*, *level: int = 1*)

Return a heading paragraph newly added to the end of the document.

The heading paragraph will contain *text* and have its paragraph style determined by *level*. If *level* is 0, the style is set to *Title*. If *level* is 1 (or omitted), *Heading 1* is used. Otherwise the style is set to *Heading {level}*. Raises `ValueError` if *level* is outside the range 0-9.

add_page_break()

Return newly *Paragraph* object containing only a page break.

add_paragraph(*text: str = ''*, *style: str | ParagraphStyle | None = None*) → *Paragraph*

Return paragraph newly added to the end of the document.

The paragraph is populated with *text* and having paragraph style *style*.

text can contain tab (`\t`) characters, which are converted to the appropriate XML form for a tab. *text* can also include newline (`\n`) or carriage return (`\r`) characters, each of which is converted to a line break.

add_picture(*image_path_or_stream: str | IO[bytes]*, *width: int | Length | None = None*, *height: int | Length | None = None*)

Return new picture shape added in its own paragraph at end of the document.

The picture contains the image at *image_path_or_stream*, scaled based on *width* and *height*. If neither width nor height is specified, the picture appears at its native size. If only one is specified, it is used to compute a scaling factor that is then applied to the unspecified dimension, preserving the aspect ratio of

the image. The native size of the picture is calculated using the dots-per-inch (dpi) value specified in the image file, defaulting to 72 dpi if no value is specified, as is often the case.

add_section(*start_type: WD_SECTION_START = WD_SECTION_START.NEW_PAGE*)

Return a *Section* object newly added at the end of the document.

The optional *start_type* argument must be a member of the *WD_SECTION_START* enumeration, and defaults to `WD_SECTION.NEW_PAGE` if not provided.

add_table(*rows: int*, *cols: int*, *style: str | _TableStyle | None = None*)

Add a table having row and column counts of *rows* and *cols* respectively.

style may be a table style object or a table style name. If *style* is `None`, the table inherits the default table style of the document.

property core_properties

A *CoreProperties* object providing Dublin Core properties of document.

property inline_shapes

The *InlineShapes* collection for this document.

An inline shape is a graphical object, such as a picture, contained in a run of text and behaving like a character glyph, being flowed like other text in a paragraph.

iter_inner_content() → Iterator[*Paragraph* | *Table*]

Generate each *Paragraph* or *Table* in this document in document order.

property paragraphs: List[*Paragraph*]

The *Paragraph* instances in the document, in document order.

Note that paragraphs within revision marks such as <w:ins> or <w:del> do not appear in this list.

property part: DocumentPart

The `DocumentPart` object of this document.

save(*path_or_stream: str | IO[bytes]*)

Save this document to *path_or_stream*.

path_or_stream can be either a path to a filesystem location (a string) or a file-like object.

property sections: *Sections*

Sections object providing access to each section in this document.

property settings: *Settings*

A *Settings* object providing access to the document-level settings.

property styles

A *Styles* object providing access to the styles in this document.

property tables: List[*Table*]

All *Table* instances in the document, in document order.

Note that only tables appearing at the top level of the document appear in this list; a table nested inside a table cell does not appear. A table within revision marks such as <w:ins> or <w:del> will also not appear in the list.

3.1.3 CoreProperties objects

Each *Document* object provides access to its *CoreProperties* object via its core_properties attribute. A *CoreProperties* object provides read/write access to the so-called *core properties* for the document. The core properties are author, category, comments, content_status, created, identifier, keywords, language, last_modified_by, last_printed, modified, revision, subject, title, and version.

Each property is one of three types, str, datetime.datetime, or int. String properties are limited in length to 255 characters and return an empty string (‘’) if not set. Date properties are assigned and returned as datetime.datetime objects without timezone, i.e. in UTC. Any timezone conversions are the responsibility of the client. Date properties return None if not set.

python-docx does not automatically set any of the document core properties other than to add a core properties part to a presentation that doesn’t have one (very uncommon). If python-docx adds a core properties part, it contains default values for the title, last_modified_by, revision, and modified properties. Client code should update properties like revision and last_modified_by if that behavior is desired.

class docx.opc.coreprops.**CoreProperties**

author

string – An entity primarily responsible for making the content of the resource.

category

string – A categorization of the content of this package. Example values might include: Resume, Letter, Financial Forecast, Proposal, or Technical Presentation.

comments

string – An account of the content of the resource.

content_status

string – completion status of the document, e.g. ‘draft’

created

datetime – time of intial creation of the document

identifier

string – An unambiguous reference to the resource within a given context, e.g. ISBN.

keywords

string – descriptive words or short phrases likely to be used as search terms for this document

language

string – language the document is written in

last_modified_by

string – name or other identifier (such as email address) of person who last modified the document

last_printed

datetime – time the document was last printed

modified

datetime – time the document was last modified

revision

int – number of this revision, incremented by Word each time the document is saved. Note however python-docx does not automatically increment the revision number when it saves a document.

subject

string – The topic of the content of the resource.

title

string – The name given to the resource.

version

string – free-form version string

3.2 Document Settings objects

class docx.settings.**Settings**

Provides access to document-level settings for a document.

Accessed using the *Document.settings* property.

property element

The lxml element proxied by this object.

property odd_and_even_pages_header_footer: bool

True if this document has distinct odd and even page headers and footers.

Read/write.

3.3 Style-related objects

A style is used to collect a set of formatting properties under a single name and apply those properties to a content object all at once. This promotes formatting consistency throughout a document and across related documents and allows formatting changes to be made globally by changing the definition in the appropriate style.

3.3.1 Styles objects

class docx.styles.styles.**Styles**

Provides access to the styles defined in a document.

Accessed using the *Document.styles* property. Supports `len()`, iteration, and dictionary-style access by style name.

add_style(*name*, *style_type*, *builtin=False*)

Return a newly added style object of *style_type* and identified by *name*.

A builtin style can be defined by passing True for the optional *builtin* argument.

default(*style_type: WD_STYLE_TYPE*)

Return the default style for *style_type* or `None` if no default is defined for that type (not common).

property element

The lxml element proxied by this object.

property latent_styles

A *LatentStyles* object providing access to the default behaviors for latent styles and the collection of *_LatentStyle* objects that define overrides of those defaults for a particular named latent style.

3.3.2 BaseStyle objects

class docx.styles.style.**BaseStyle**

Base class for the various types of style object, paragraph, character, table, and numbering.

These properties and methods are inherited by all style objects.

property builtin

Read-only.

`True` if this style is a built-in style. `False` indicates it is a custom (user-defined) style. Note this value is based on the presence of a *customStyle* attribute in the XML, not on specific knowledge of which styles are built into Word.

delete()

Remove this style definition from the document.

Note that calling this method does not remove or change the style applied to any document content. Content items having the deleted style will be rendered using the default style, as is any content with a style not defined in the document.

property element

The lxml element proxied by this object.

property hidden

`True` if display of this style in the style gallery and list of recommended styles is suppressed.

`False` otherwise. In order to be shown in the style gallery, this value must be `False` and *quick_style* must be `True`.

property locked

Read/write Boolean.

`True` if this style is locked. A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property name

The UI name of this style.

property priority

The integer sort key governing display sequence of this style in the Word UI.

`None` indicates no setting is defined, causing Word to use the default value of 0. Style name is used as a secondary sort key to resolve ordering of styles having the same priority value.

property quick_style

`True` if this style should be displayed in the style gallery when *hidden* is `False`.

Read/write Boolean.

property type

Member of *WD_STYLE_TYPE* corresponding to the type of this style, e.g. `WD_STYLE_TYPE.PARAGRAPH`.

property unhide_when_used

`True` if an application should make this style visible the next time it is applied to content.

False otherwise. Note that `python-docx` does not automatically unhide a style having `True` for this attribute when it is applied to content.

3.3.3 CharacterStyle objects

class docx.styles.style.**CharacterStyle**

Bases: *BaseStyle*

A character style.

A character style is applied to a *Run* object and primarily provides character- level formatting via the *Font* object in its *font* property.

property base_style

Style object this style inherits from or `None` if this style is not based on another style.

property builtin

Read-only.

`True` if this style is a built-in style. `False` indicates it is a custom (user-defined) style. Note this value is based on the presence of a *customStyle* attribute in the XML, not on specific knowledge of which styles are built into Word.

delete()

Remove this style definition from the document.

Note that calling this method does not remove or change the style applied to any document content. Content items having the deleted style will be rendered using the default style, as is any content with a style not defined in the document.

property font

The *Font* object providing access to the character formatting properties for this style, such as font name and size.

property hidden

`True` if display of this style in the style gallery and list of recommended styles is suppressed.

`False` otherwise. In order to be shown in the style gallery, this value must be `False` and *quick_style* must be `True`.

property locked

Read/write Boolean.

`True` if this style is locked. A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property name

The UI name of this style.

property priority

The integer sort key governing display sequence of this style in the Word UI.

`None` indicates no setting is defined, causing Word to use the default value of 0. Style name is used as a secondary sort key to resolve ordering of styles having the same priority value.

property quick_style

`True` if this style should be displayed in the style gallery when *hidden* is `False`.

Read/write Boolean.

property unhide_when_used

`True` if an application should make this style visible the next time it is applied to content.

False otherwise. Note that `python-docx` does not automatically unhide a style having `True` for this attribute when it is applied to content.

3.3.4 ParagraphStyle objects

class docx.styles.style.**ParagraphStyle**

Bases: *CharacterStyle*

A paragraph style.

A paragraph style provides both character formatting and paragraph formatting such as indentation and line-spacing.

property base_style

Style object this style inherits from or None if this style is not based on another style.

property builtin

Read-only.

True if this style is a built-in style. False indicates it is a custom (user-defined) style. Note this value is based on the presence of a *customStyle* attribute in the XML, not on specific knowledge of which styles are built into Word.

delete()

Remove this style definition from the document.

Note that calling this method does not remove or change the style applied to any document content. Content items having the deleted style will be rendered using the default style, as is any content with a style not defined in the document.

property font

The *Font* object providing access to the character formatting properties for this style, such as font name and size.

property hidden

True if display of this style in the style gallery and list of recommended styles is suppressed.

False otherwise. In order to be shown in the style gallery, this value must be False and *quick_style* must be True.

property locked

Read/write Boolean.

True if this style is locked. A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property name

The UI name of this style.

property next_paragraph_style

ParagraphStyle object representing the style to be applied automatically to a new paragraph inserted after a paragraph of this style.

Returns self if no next paragraph style is defined. Assigning None or *self* removes the setting such that new paragraphs are created using this same style.

property paragraph_format

The *ParagraphFormat* object providing access to the paragraph formatting properties for this style such as indentation.

property priority

The integer sort key governing display sequence of this style in the Word UI.

`None` indicates no setting is defined, causing Word to use the default value of 0. Style name is used as a secondary sort key to resolve ordering of styles having the same priority value.

property quick_style

`True` if this style should be displayed in the style gallery when *hidden* is `False`.

Read/write Boolean.

property unhide_when_used

`True` if an application should make this style visible the next time it is applied to content.

False otherwise. Note that `python-docx` does not automatically unhide a style having `True` for this attribute when it is applied to content.

3.3.5 _TableStyle objects

class docx.styles.style._TableStyle

Bases: *ParagraphStyle*

A table style.

A table style provides character and paragraph formatting for its contents as well as special table formatting properties.

property base_style

Style object this style inherits from or `None` if this style is not based on another style.

property builtin

Read-only.

`True` if this style is a built-in style. `False` indicates it is a custom (user-defined) style. Note this value is based on the presence of a *customStyle* attribute in the XML, not on specific knowledge of which styles are built into Word.

delete()

Remove this style definition from the document.

Note that calling this method does not remove or change the style applied to any document content. Content items having the deleted style will be rendered using the default style, as is any content with a style not defined in the document.

property font

The *Font* object providing access to the character formatting properties for this style, such as font name and size.

property hidden

`True` if display of this style in the style gallery and list of recommended styles is suppressed.

`False` otherwise. In order to be shown in the style gallery, this value must be `False` and *quick_style* must be `True`.

property locked

Read/write Boolean.

`True` if this style is locked. A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property name

The UI name of this style.

property next_paragraph_style

ParagraphStyle object representing the style to be applied automatically to a new paragraph inserted after a paragraph of this style.

Returns self if no next paragraph style is defined. Assigning `None` or *self* removes the setting such that new paragraphs are created using this same style.

property paragraph_format

The *ParagraphFormat* object providing access to the paragraph formatting properties for this style such as indentation.

property priority

The integer sort key governing display sequence of this style in the Word UI.

`None` indicates no setting is defined, causing Word to use the default value of 0. Style name is used as a secondary sort key to resolve ordering of styles having the same priority value.

property quick_style

`True` if this style should be displayed in the style gallery when *hidden* is `False`.

Read/write Boolean.

property unhide_when_used

`True` if an application should make this style visible the next time it is applied to content.

False otherwise. Note that `python-docx` does not automatically unhide a style having `True` for this attribute when it is applied to content.

3.3.6 _NumberingStyle objects

class docx.styles.style.**_NumberingStyle**

A numbering style.

Not yet implemented.

3.3.7 LatentStyles objects

class docx.styles.latent.**LatentStyles**

Provides access to the default behaviors for latent styles in this document and to the collection of *_LatentStyle* objects that define overrides of those defaults for a particular named latent style.

add_latent_style(*name*)

Return a newly added *_LatentStyle* object to override the inherited defaults defined in this latent styles object for the built-in style having *name*.

property default_priority

Integer between 0 and 99 inclusive specifying the default sort order for latent styles in style lists and the style gallery.

`None` if no value is assigned, which causes Word to use the default value 99.

property default_to_hidden

Boolean specifying whether the default behavior for latent styles is to be hidden.

A hidden style does not appear in the recommended list or in the style gallery.

property default_to_locked

Boolean specifying whether the default behavior for latent styles is to be locked.

A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property default_to_quick_style

Boolean specifying whether the default behavior for latent styles is to appear in the style gallery when not hidden.

property default_to_unhide_when_used

Boolean specifying whether the default behavior for latent styles is to be unhidden when first applied to content.

property element

The lxml element proxied by this object.

property load_count

Integer specifying the number of built-in styles to initialize to the defaults specified in this *LatentStyles* object.

`None` if there is no setting in the XML (very uncommon). The default Word 2011 template sets this value to 276, accounting for the built-in styles in Word 2010.

3.3.8 _LatentStyle objects

class docx.styles.latent.**_LatentStyle**

Proxy for an *w:lsdException* element, which specifies display behaviors for a built-in style when no definition for that style is stored yet in the *styles.xml* part.

The values in this element override the defaults specified in the parent *w:latentStyles* element.

delete()

Remove this latent style definition such that the defaults defined in the containing *LatentStyles* object provide the effective value for each of its attributes.

Attempting to access any attributes on this object after calling this method will raise `AttributeError`.

property element

The lxml element proxied by this object.

property hidden

Tri-state value specifying whether this latent style should appear in the recommended list.

`None` indicates the effective value is inherited from the parent `<w:latentStyles>` element.

property locked

Tri-state value specifying whether this latent styles is locked.

A locked style does not appear in the styles panel or the style gallery and cannot be applied to document content. This behavior is only active when formatting protection is turned on for the document (via the Developer menu).

property name

The name of the built-in style this exception applies to.

property priority

The integer sort key for this latent style in the Word UI.

property quick_style

Tri-state value specifying whether this latent style should appear in the Word styles gallery when not hidden.

`None` indicates the effective value should be inherited from the default values in its parent *LatentStyles* object.

property unhide_when_used

Tri-state value specifying whether this style should have its *hidden* attribute set `False` the next time the style is applied to content.

`None` indicates the effective value should be inherited from the default specified by its parent *LatentStyles* object.

3.4 Text-related objects

3.4.1 Paragraph objects

class docx.text.paragraph.**Paragraph**

Proxy object wrapping a <*w:p*> element.

add_run(*text: str | None = None*, *style: str | CharacterStyle | None = None*) → *Run*

Append run containing *text* and having character-style *style*.

text can contain tab (`\t`) characters, which are converted to the appropriate XML form for a tab. *text* can also include newline (`\n`) or carriage return (`\r`) characters, each of which is converted to a line break. When *text* is *None*, the new run is empty.

property alignment: WD_PARAGRAPH_ALIGNMENT | None

A member of the *WD_PARAGRAPH_ALIGNMENT* enumeration specifying the justification setting for this paragraph.

A value of `None` indicates the paragraph has no directly-applied alignment value and will inherit its alignment value from its style hierarchy. Assigning `None` to this property removes any directly-applied alignment value.

clear()

Return this same paragraph after removing all its content.

Paragraph-level formatting, such as style, is preserved.

property contains_page_break: bool

True when one or more rendered page-breaks occur in this paragraph.

property hyperlinks: List[*Hyperlink*]

A *Hyperlink* instance for each hyperlink in this paragraph.

insert_paragraph_before(*text: str | None = None*, *style: str | ParagraphStyle | None = None*) → *Paragraph*

Return a newly created paragraph, inserted directly before this paragraph.

If *text* is supplied, the new paragraph contains that text in a single run. If *style* is provided, that style is assigned to the new paragraph.

iter_inner_content() → Iterator[*Run* | *Hyperlink*]

Generate the runs and hyperlinks in this paragraph, in the order they appear.

The content in a paragraph consists of both runs and hyperlinks. This method allows accessing each of those separately, in document order, for when the precise position of the hyperlink within the paragraph text is important. Note that a hyperlink itself contains runs.

property paragraph_format

The *ParagraphFormat* object providing access to the formatting properties for this paragraph, such as line spacing and indentation.

property rendered_page_breaks: List[*RenderedPageBreak*]

All rendered page-breaks in this paragraph.

Most often an empty list, sometimes contains one page-break, but can contain more than one is rare or contrived cases.

property runs: List[*Run*]

Sequence of *Run* instances corresponding to the <w:r> elements in this paragraph.

property style: *ParagraphStyle* | None

Read/Write.

ParagraphStyle object representing the style assigned to this paragraph. If no explicit style is assigned to this paragraph, its value is the default paragraph style for the document. A paragraph style name can be assigned in lieu of a paragraph style object. Assigning `None` removes any applied style, making its effective value the default paragraph style for the document.

property text: str

The textual content of this paragraph.

The text includes the visible-text portion of any hyperlinks in the paragraph. Tabs and line breaks in the XML are mapped to `\t` and `\n` characters respectively.

Assigning text to this property causes all existing paragraph content to be replaced with a single run containing the assigned text. A `\t` character in the text is mapped to a `<w:tab/>` element and each `\n` or `\r` character is mapped to a line break. Paragraph-level formatting, such as style, is preserved. All run-level formatting, such as bold or italic, is removed.

3.4.2 ParagraphFormat objects

class docx.text.parfmt.**ParagraphFormat**

Provides access to paragraph formatting such as justification, indentation, line spacing, space before and after, and widow/orphan control.

property alignment

A member of the *WD_PARAGRAPH_ALIGNMENT* enumeration specifying the justification setting for this paragraph.

A value of `None` indicates paragraph alignment is inherited from the style hierarchy.

property first_line_indent

Length value specifying the relative difference in indentation for the first line of the paragraph.

A positive value causes the first line to be indented. A negative value produces a hanging indent. `None` indicates first line indentation is inherited from the style hierarchy.

property keep_together

`True` if the paragraph should be kept "in one piece" and not broken across a page boundary when the document is rendered.

`None` indicates its effective value is inherited from the style hierarchy.

property keep_with_next

`True` if the paragraph should be kept on the same page as the subsequent paragraph when the document is rendered.

For example, this property could be used to keep a section heading on the same page as its first paragraph. `None` indicates its effective value is inherited from the style hierarchy.

property left_indent

Length value specifying the space between the left margin and the left side of the paragraph.

`None` indicates the left indent value is inherited from the style hierarchy. Use an *Inches* value object as a convenient way to apply indentation in units of inches.

property line_spacing

`float` or *Length* value specifying the space between baselines in successive lines of the paragraph.

A value of `None` indicates line spacing is inherited from the style hierarchy. A float value, e.g. `2.0` or `1.75`, indicates spacing is applied in multiples of line heights. A *Length* value such as `Pt(12)` indicates spacing is a fixed height. The *Pt* value class is a convenient way to apply line spacing in units of points. Assigning `None` resets line spacing to inherit from the style hierarchy.

property line_spacing_rule

A member of the *WD_LINE_SPACING* enumeration indicating how the value of *line_spacing* should be interpreted.

Assigning any of the *WD_LINE_SPACING* members `SINGLE`, `DOUBLE`, or `ONE_POINT_FIVE` will cause the value of *line_spacing* to be updated to produce the corresponding line spacing.

property page_break_before

`True` if the paragraph should appear at the top of the page following the prior paragraph.

`None` indicates its effective value is inherited from the style hierarchy.

property right_indent

Length value specifying the space between the right margin and the right side of the paragraph.

`None` indicates the right indent value is inherited from the style hierarchy. Use a *Cm* value object as a convenient way to apply indentation in units of centimeters.

property space_after

Length value specifying the spacing to appear between this paragraph and the subsequent paragraph.

`None` indicates this value is inherited from the style hierarchy. *Length* objects provide convenience properties, such as *pt* and *inches*, that allow easy conversion to various length units.

property space_before

Length value specifying the spacing to appear between this paragraph and the prior paragraph.

`None` indicates this value is inherited from the style hierarchy. *Length* objects provide convenience properties, such as *pt* and *cm*, that allow easy conversion to various length units.

tab_stops

TabStops object providing access to the tab stops defined for this paragraph format.

property widow_control

`True` if the first and last lines in the paragraph remain on the same page as the rest of the paragraph when Word repaginates the document.

`None` indicates its effective value is inherited from the style hierarchy.

3.4.3 Hyperlink objects

class docx.text.hyperlink.**Hyperlink**

Proxy object wrapping a <*w:hyperlink*> element.

A hyperlink occurs as a child of a paragraph, at the same level as a Run. A hyperlink itself contains runs, which is where the visible text of the hyperlink is stored.

property address: str

The "URL" of the hyperlink (but not necessarily a web link).

While commonly a web link like "https://google.com" the hyperlink address can take a variety of forms including "internal links" to bookmarked locations within the document. When this hyperlink is an internal "jump" to for example a heading from the table-of-contents (TOC), the address is blank. The bookmark reference (like "_Toc147925734") is stored in the *.fragment* property.

property contains_page_break: bool

True when the text of this hyperlink is broken across page boundaries.

This is not uncommon and can happen for example when the hyperlink text is multiple words and occurs in the last line of a page. Theoretically, a hyperlink can contain more than one page break but that would be extremely uncommon in practice. Still, this value should be understood to mean that "one-or-more" rendered page breaks are present.

property fragment: str

Reference like *#glossary* at end of URL that refers to a sub-resource.

Note that this value does not include the fragment-separator character ("#").

This value is known as a "named anchor" in an HTML context and "anchor" in the MS API, but an "anchor" element (<*a*>) represents a full hyperlink in HTML so we avoid confusion by using the more precise RFC 3986 naming "URI fragment".

These are also used to refer to bookmarks within the same document, in which case the *.address* value with be blank ("") and this property will hold a value like "_Toc147925734".

To reliably get an entire web URL you will need to concatenate this with the *.address* value, separated by "#" when both are present. Consider using the *.url* property for that purpose.

Word sometimes stores a fragment in this property (an XML attribute) and sometimes with the address, depending on how the URL is inserted, so don't depend on this field being empty to indicate no fragment is present.

property runs: list[*Run*]

List of *Run* instances in this hyperlink.

Together these define the visible text of the hyperlink. The text of a hyperlink is typically contained in a single run will be broken into multiple runs if for example part of the hyperlink is bold or the text was changed after the document was saved.

property text: str

String formed by concatenating the text of each run in the hyperlink.

Tabs and line breaks in the XML are mapped to `\t` and `\n` characters respectively. Note that rendered page-breaks can occur within a hyperlink but they are not reflected in this text.

property url: str

Convenience property to get web URLs from hyperlinks that contain them.

This value is the empty string ("") when there is no address portion, so its boolean value can also be used to distinguish external URIs from internal "jump" hyperlinks like those found in a table-of-contents.

Note that this value may also be a link to a file, so if you only want web-urls you'll need to check for a protocol prefix like *https://*.

When both an address and fragment are present, the return value joins the two separated by the fragment-separator hash ("#"). Otherwise this value is the same as that of the *.address* property.

3.4.4 Run objects

class docx.text.run.**Run**

Proxy object wrapping <*w:r*> element.

Several of the properties on Run take a tri-state value, `True`, `False`, or `None`. `True` and `False` correspond to on and off respectively. `None` indicates the property is not specified directly on the run and its effective value is taken from the style hierarchy.

add_break(*break_type: WD_BREAK_TYPE = WD_BREAK_TYPE.LINE*)

Add a break element of *break_type* to this run.

break_type can take the values *WD_BREAK.LINE*, *WD_BREAK.PAGE*, and *WD_BREAK.COLUMN* where *WD_BREAK* is imported from *docx.enum.text*. *break_type* defaults to *WD_BREAK.LINE*.

add_picture(*image_path_or_stream: str | IO[bytes]*, *width: int* | Length | *None = None*, *height: int* | Length | *None = None*) → *InlineShape*

Return `InlineShape` containing image identified by *image_path_or_stream*.

The picture is added to the end of this run.

image_path_or_stream can be a path (a string) or a file-like object containing a binary image.

If neither width nor height is specified, the picture appears at its native size. If only one is specified, it is used to compute a scaling factor that is then applied to the unspecified dimension, preserving the aspect ratio of the image. The native size of the picture is calculated using the dots- per-inch (dpi) value specified in the image file, defaulting to 72 dpi if no value is specified, as is often the case.

add_tab() → None

Add a <`w:tab/`> element at the end of the run, which Word interprets as a tab character.

add_text(*text: str*)

Returns a newly appended `_Text` object (corresponding to a new <`w:t`> child element) to the run, containing *text*.

Compare with the possibly more friendly approach of assigning text to the `Run.text` property.

property bold: bool | None

Read/write tri-state value.

When `True`, causes the text of the run to appear in bold face. When `False`, the text unconditionally appears non-bold. When `None` the bold setting for this run is inherited from the style hierarchy.

clear()

Return reference to this run after removing all its content.

All run formatting is preserved.

property contains_page_break: bool

True when one or more rendered page-breaks occur in this run.

Note that "hard" page-breaks inserted by the author are not included. A hard page-break gives rise to a rendered page-break in the right position so if those were included that page-break would be "double-counted".

It would be very rare for multiple rendered page-breaks to occur in a single run, but it is possible.

property font: *Font*

The *Font* object providing access to the character formatting properties for this run, such as font name and size.

property italic: bool | None

Read/write tri-state value.

When `True`, causes the text of the run to appear in italics. When `False`, the text unconditionally appears non-italic. When `None` the italic setting for this run is inherited from the style hierarchy.

iter_inner_content() → Iterator[str | Drawing | *RenderedPageBreak*]

Generate the content-items in this run in the order they appear.

NOTE: only content-types currently supported by *python-docx* are generated. In this version, that is text and rendered page-breaks. Drawing is included but currently only provides access to its XML element (CT_Drawing) on its *._drawing* attribute. *Drawing* attributes and methods may be expanded in future releases.

There are a number of element-types that can appear inside a run, but most of those (w:br, w:cr, w:noBreakHyphen, w:t, w:tab) have a clear plain-text equivalent. Any contiguous range of such elements is generated as a single *str*. Rendered page-break and drawing elements are generated individually. Any other elements are ignored.

property style: *CharacterStyle*

Read/write.

A *CharacterStyle* object representing the character style applied to this run. The default character style for the document (often *Default Character Font*) is returned if the run has no directly-applied character style. Setting this property to `None` removes any directly-applied character style.

property text: str

String formed by concatenating the text equivalent of each run.

Each <*w:t*> element adds the text characters it contains. A <*w:tab/*> element adds a *t* character. A <*w:cr/*> or <*w:br*> element each add a *n* character. Note that a <*w:br*> element can indicate a page break or column break as well as a line break. Only line-break <*w:br*> elements translate to a *n* character. Others are ignored. All other content child elements, such as <*w:drawing*>, are ignored.

Assigning text to this property has the reverse effect, translating each *t* character to a <*w:tab/*> element and each *n* or *r* character to a <*w:cr/*> element. Any existing run content is replaced. Run formatting is preserved.

property underline: bool | WD_UNDERLINE | None

The underline style for this *Run*.

Value is one of `None`, `True`, `False`, or a member of *WD_UNDERLINE*.

A value of `None` indicates the run has no directly-applied underline value and so will inherit the underline value of its containing paragraph. Assigning `None` to this property removes any directly-applied underline value.

A value of `False` indicates a directly-applied setting of no underline, overriding any inherited value.

A value of `True` indicates single underline.

The values from *WD_UNDERLINE* are used to specify other outline styles such as double, wavy, and dotted.

3.4.5 Font objects

class docx.text.run.**Font**

Proxy object for parent of a *<w:rPr>* element and providing access to character properties such as font name, font size, bold, and subscript.

property all_caps: bool | None

Read/write.

Causes text in this font to appear in capital letters.

property bold: bool | None

Read/write.

Causes text in this font to appear in bold.

property color

A *ColorFormat* object providing a way to get and set the text color for this font.

property complex_script: bool | None

Read/write tri-state value.

When `True`, causes the characters in the run to be treated as complex script regardless of their Unicode values.

property cs_bold: bool | None

Read/write tri-state value.

When `True`, causes the complex script characters in the run to be displayed in bold typeface.

property cs_italic: bool | None

Read/write tri-state value.

When `True`, causes the complex script characters in the run to be displayed in italic typeface.

property double_strike: bool | None

Read/write tri-state value.

When `True`, causes the text in the run to appear with double strikethrough.

property emboss: bool | None

Read/write tri-state value.

When `True`, causes the text in the run to appear as if raised off the page in relief.

property hidden: bool | None

Read/write tri-state value.

When `True`, causes the text in the run to be hidden from display, unless applications settings force hidden text to be shown.

property highlight_color: WD_COLOR_INDEX | None

Color of highlighing applied or `None` if not highlighted.

property imprint: bool | None

Read/write tri-state value.

When `True`, causes the text in the run to appear as if pressed into the page.

property italic: bool | None

Read/write tri-state value.

When `True`, causes the text of the run to appear in italics. `None` indicates the effective value is inherited from the style hierarchy.

property math: bool | None

Read/write tri-state value.

When `True`, specifies this run contains WML that should be handled as though it was Office Open XML Math.

property name: str | None

The typeface name for this *Font*.

Causes the text it controls to appear in the named font, if a matching font is found. `None` indicates the typeface is inherited from the style hierarchy.

property no_proof: bool | None

Read/write tri-state value.

When `True`, specifies that the contents of this run should not report any errors when the document is scanned for spelling and grammar.

property outline: bool | None

Read/write tri-state value.

When `True` causes the characters in the run to appear as if they have an outline, by drawing a one pixel wide border around the inside and outside borders of each character glyph.

property rtl: bool | None

Read/write tri-state value.

When `True` causes the text in the run to have right-to-left characteristics.

property shadow: bool | None

Read/write tri-state value.

When `True` causes the text in the run to appear as if each character has a shadow.

property size: *Length* | None

Font height in English Metric Units (EMU).

`None` indicates the font size should be inherited from the style hierarchy. *Length* is a subclass of `int` having properties for convenient conversion into points or other length units. The *docx.shared.Pt* class allows convenient specification of point values:

```
>>> font.size = Pt(24)
>>> font.size
304800
>>> font.size.pt
24.0
```

property small_caps: bool | None

Read/write tri-state value.

When `True` causes the lowercase characters in the run to appear as capital letters two points smaller than the font size specified for the run.

property snap_to_grid: bool | None

Read/write tri-state value.

When `True` causes the run to use the document grid characters per line settings defined in the docGrid element when laying out the characters in this run.

property spec_vanish: bool | None

Read/write tri-state value.

When `True`, specifies that the given run shall always behave as if it is hidden, even when hidden text is being displayed in the current document. The property has a very narrow, specialized use related to the table of contents. Consult the spec (§17.3.2.36) for more details.

property strike: bool | None

Read/write tri-state value.

When `True` causes the text in the run to appear with a single horizontal line through the center of the line.

property subscript: bool | None

Boolean indicating whether the characters in this *Font* appear as subscript.

`None` indicates the subscript/subscript value is inherited from the style hierarchy.

property superscript: bool | None

Boolean indicating whether the characters in this *Font* appear as superscript.

`None` indicates the subscript/superscript value is inherited from the style hierarchy.

property underline: bool | WD_UNDERLINE | None

The underline style for this *Font*.

The value is one of `None`, `True`, `False`, or a member of *WD_UNDERLINE*.

`None` indicates the font inherits its underline value from the style hierarchy. `False` indicates no underline. `True` indicates single underline. The values from *WD_UNDERLINE* are used to specify other outline styles such as double, wavy, and dotted.

property web_hidden: bool | None

Read/write tri-state value.

When `True`, specifies that the contents of this run shall be hidden when the document is displayed in web page view.

3.4.6 RenderedPageBreak objects

class docx.text.pagebreak.**RenderedPageBreak**

A page-break inserted by Word during page-layout for print or display purposes.

This usually does not correspond to a "hard" page-break inserted by the document author, rather just that Word ran out of room on one page and needed to start another. The position of these can change depending on the printer and page-size, as well as margins, etc. They also will change in response to edits, but not until Word loads and saves the document.

Note these are never inserted by *python-docx* because it has no rendering function. These are generally only useful for text-extraction of existing documents when *python-docx* is being used solely as a document "reader".

NOTE: a rendered page-break can occur within a hyperlink; consider a multi-word hyperlink like "excellent Wikipedia article on LLMs" that happens to fall close to the end of the last line on a page such that the page breaks between "Wikipedia" and "article". In such a "page-breaks-in-hyperlink" case, THESE METHODS WILL "MOVE" THE PAGE-BREAK to occur after the hyperlink, such that the entire hyperlink appears in the paragraph returned by *.preceding_paragraph_fragment*. While this places the "tail" text of the hyperlink on the

"wrong" page, it avoids having two hyperlinks each with a fragment of the actual text and pointing to the same address.

property following_paragraph_fragment: *Paragraph* **| None**

A "loose" paragraph containing the content following this page-break.

HAS POTENTIALLY SURPRISING BEHAVIORS so read carefully to be sure this is what you want. This is primarily targeted toward text-extraction use-cases for which precisely associating text with the page it occurs on is important.

Compare *.preceding_paragraph_fragment* as these two are intended to be used together.

This value is *None* when no content follows this page-break. This case is unlikely to occur in practice because Word places even-paragraph-boundary page-breaks on the paragraph *following* the page-break. Still, it is possible and must be checked for. Returning *None* for this case avoids "inserting" an extra, non-existent paragraph into the content stream. Note that content can include DrawingML items like images or charts, not just text.

The returned paragraph *is divorced from the document body*. Any changes made to it will not be reflected in the document. It is intended to provide a container (*Paragraph*) with familiar properties and methods that can be used to characterize the paragraph content following a mid-paragraph page-break.

Contains no portion of the hyperlink when this break occurs within a hyperlink.

property preceding_paragraph_fragment: *Paragraph* **| None**

A "loose" paragraph containing the content preceding this page-break.

Compare *.following_paragraph_fragment* as these two are intended to be used together.

This value is *None* when no content precedes this page-break. This case is common and occurs whenever a page breaks on an even paragraph boundary. Returning *None* for this case avoids "inserting" a non-existent paragraph into the content stream. Note that content can include DrawingML items like images or charts.

Note the returned paragraph *is divorced from the document body*. Any changes made to it will not be reflected in the document. It is intended to provide a familiar container (*Paragraph*) to interrogate for the content preceding this page-break in the paragraph in which it occured.

Contains the entire hyperlink when this break occurs within a hyperlink.

3.4.7 TabStop objects

class docx.text.tabstops.**TabStop**

An individual tab stop applying to a paragraph or style.

Accessed using list semantics on its containing *TabStops* object.

property alignment

A member of *WD_TAB_ALIGNMENT* specifying the alignment setting for this tab stop.

Read/write.

property leader

A member of *WD_TAB_LEADER* specifying a repeating character used as a "leader", filling in the space spanned by this tab.

Assigning `None` produces the same result as assigning *WD_TAB_LEADER.SPACES*. Read/write.

property position

A *Length* object representing the distance of this tab stop from the inside edge of the paragraph.

May be positive or negative. Read/write.

3.4.8 TabStops objects

class docx.text.tabstops.**TabStops**

A sequence of *TabStop* objects providing access to the tab stops of a paragraph or paragraph style.

Supports iteration, indexed access, del, and len(). It is accesed using the *tab_stops* property of ParagraphFormat; it is not intended to be constructed directly.

add_tab_stop(*position*, *alignment=WD_TAB_ALIGNMENT.LEFT*, *leader=WD_TAB_LEADER.SPACES*)

Add a new tab stop at *position*, a *Length* object specifying the location of the tab stop relative to the paragraph edge.

A negative *position* value is valid and appears in hanging indentation. Tab alignment defaults to left, but may be specified by passing a member of the *WD_TAB_ALIGNMENT* enumeration as *alignment*. An optional leader character can be specified by passing a member of the *WD_TAB_LEADER* enumeration as *leader*.

clear_all()

Remove all custom tab stops.

3.5 Table objects

Table objects are constructed using the `add_table()` method on *Document*.

3.5.1 Table objects

class docx.table.**Table**(*tbl: CT_Tbl*, *parent: t.ProvidesStoryPart*)

Proxy class for a WordprocessingML <w:tbl> element.

add_column(*width:* Length)

Return a *_Column* object of *width*, newly added rightmost to the table.

add_row()

Return a *_Row* instance, newly added bottom-most to the table.

property alignment: WD_TABLE_ALIGNMENT | None

Read/write.

A member of *WD_TABLE_ALIGNMENT* or None, specifying the positioning of this table between the page margins. `None` if no setting is specified, causing the effective value to be inherited from the style hierarchy.

property autofit: bool

`True` if column widths can be automatically adjusted to improve the fit of cell contents.

`False` if table layout is fixed. Column widths are adjusted in either case if total column width exceeds page width. Read/write boolean.

cell(*row_idx: int*, *col_idx: int*) → *_Cell*

_Cell at *row_idx*, *col_idx* intersection.

(0, 0) is the top, left-most cell.

column_cells(*column_idx: int*) → list[*_Cell*]

Sequence of cells in the column at *column_idx* in this table.

columns

_Columns instance representing the sequence of columns in this table.

row_cells(*row_idx: int*) → list[*_Cell*]

DEPRECATED: Use *table.rows[row_idx].cells* instead.

Sequence of cells in the row at *row_idx* in this table.

rows

_Rows instance containing the sequence of rows in this table.

property style: _TableStyle | None

_TableStyle object representing the style applied to this table.

Read/write. The default table style for the document (often *Normal Table*) is returned if the table has no directly-applied style. Assigning `None` to this property removes any directly-applied table style causing it to inherit the default table style of the document.

Note that the style name of a table style differs slightly from that displayed in the user interface; a hyphen, if it appears, must be removed. For example, *Light Shading - Accent 1* becomes *Light Shading Accent 1*.

property table_direction: WD_TABLE_DIRECTION | None

Member of *WD_TABLE_DIRECTION* indicating cell-ordering direction.

For example: *WD_TABLE_DIRECTION.LTR*. `None` indicates the value is inherited from the style hierarchy.

3.5.2 _Cell objects

class docx.table.**_Cell**(*tc: CT_Tc*, *parent: TableParent*)

Table cell.

add_paragraph(*text: str = ''*, *style: str | ParagraphStyle | None = None*)

Return a paragraph newly added to the end of the content in this cell.

If present, *text* is added to the paragraph in a single run. If specified, the paragraph style *style* is applied. If *style* is not specified or is `None`, the result is as though the 'Normal' style was applied. Note that the formatting of text in a cell can be influenced by the table style. *text* can contain tab (`\t`) characters, which are converted to the appropriate XML form for a tab. *text* can also include newline (`\n`) or carriage return (`\r`) characters, each of which is converted to a line break.

add_table(*rows: int*, *cols: int*) → *Table*

Return a table newly added to this cell after any existing cell content.

The new table will have *rows* rows and *cols* columns.

An empty paragraph is added after the table because Word requires a paragraph element as the last element in every cell.

property grid_span: int

Number of layout-grid cells this cell spans horizontally.

A "normal" cell has a grid-span of 1. A horizontally merged cell has a grid-span of 2 or more.

iter_inner_content() → Iterator[*Paragraph* | *Table*]

Generate each *Paragraph* or *Table* in this container in document order.

merge(*other_cell: _Cell*)

Return a merged cell created by spanning the rectangular region having this cell and *other_cell* as diagonal corners.

Raises `InvalidSpanError` if the cells do not define a rectangular region.

property paragraphs

List of paragraphs in the cell.

A table cell is required to contain at least one block-level element and end with a paragraph. By default, a new cell contains a single paragraph. Read-only

property tables

List of tables in the cell, in the order they appear.

Read-only.

property text: str

The entire contents of this cell as a string of text.

Assigning a string to this property replaces all existing content with a single paragraph containing the assigned text in a single run.

property vertical_alignment

Member of *WD_CELL_VERTICAL_ALIGNMENT* or None.

A value of `None` indicates vertical alignment for this cell is inherited. Assigning `None` causes any explicitly defined vertical alignment to be removed, restoring inheritance.

property width

The width of this cell in EMU, or `None` if no explicit width is set.

3.5.3 _Row objects

class docx.table.**_Row**(*tr: CT_Row*, *parent: TableParent*)

Table row.

property cells: tuple[_Cell, ...]

Sequence of *_Cell* instances corresponding to cells in this row.

Note that Word allows table rows to start later than the first column and end before the last column.

- Only cells actually present are included in the return value.
- This implies the length of this cell sequence may differ between rows of the same table.
- If you are reading the cells from each row to form a rectangular "matrix" data structure of the table cell values, you will need to account for empty leading and/or trailing layout-grid positions using *.grid_cols_before* and *.grid_cols_after*.

property grid_cols_after: int

Count of unpopulated grid-columns after the last cell in this row.

Word allows a row to "end early", meaning that one or more cells are not present at the end of that row.

Note these are not simply "empty" cells. The renderer reads this value and "skips" this many columns after drawing the last cell.

Note this also implies that not all rows are guaranteed to have the same number of cells, e.g. *_Row.cells* could have length *n* for one row and *n - m* for the next row in the same table. Visually this appears as a column (at the beginning or end, not in the middle) with one or more cells missing.

property grid_cols_before: int

Count of unpopulated grid-columns before the first cell in this row.

Word allows a row to "start late", meaning that one or more cells are not present at the beginning of that row.

Note these are not simply "empty" cells. The renderer reads this value and skips forward to the table layout-grid position of the first cell in this row; the renderer "skips" this many columns before drawing the first cell.

Note this also implies that not all rows are guaranteed to have the same number of cells, e.g. *_Row.cells* could have length *n* for one row and *n - m* for the next row in the same table.

property height: *Length* | **None**

Return a *Length* object representing the height of this cell, or `None` if no explicit height is set.

property height_rule: **WD_ROW_HEIGHT_RULE | None**

Return the height rule of this cell as a member of the *WD_ROW_HEIGHT_RULE*.

This value is `None` if no explicit height_rule is set.

property table: *Table*

Reference to the *Table* object this row belongs to.

3.5.4 _Column objects

class docx.table.**_Column**(*gridCol: CT_TblGridCol*, *parent:* Table | _Columns | _Rows)

Table column.

property cells: **tuple[*_Cell*, ...]**

Sequence of *_Cell* instances corresponding to cells in this column.

property table: *Table*

Reference to the *Table* object this column belongs to.

property width: *Length* | **None**

The width of this column in EMU, or `None` if no explicit width is set.

3.5.5 _Rows objects

class docx.table.**_Rows**(*tbl: CT_Tbl*, *parent: TableParent*)

Sequence of *_Row* objects corresponding to the rows in a table.

Supports `len()`, iteration, indexed access, and slicing.

property table: *Table*

Reference to the *Table* object this row collection belongs to.

3.5.6 _Columns objects

class docx.table.**_Columns**(*tbl: CT_Tbl*, *parent: TableParent*)

Sequence of *_Column* instances corresponding to the columns in a table.

Supports `len()`, iteration and indexed access.

property table: *Table*

Reference to the *Table* object this column collection belongs to.

3.6 Section objects

Provides access to section properties such as margins and page orientation.

3.6.1 Sections objects

class docx.section.**Sections**(*document_elm: CT_Document*, *document_part: DocumentPart*)

Sequence of *Section* objects corresponding to the sections in the document.

Supports `len()`, iteration, and indexed access.

3.6.2 Section objects

class docx.section.**Section**(*sectPr: CT_SectPr*, *document_part: DocumentPart*)

Document section, providing access to section and page setup settings.

Also provides access to headers and footers.

property bottom_margin: *Length* | None

Read/write. Bottom margin for pages in this section, in EMU.

None when no bottom margin has been specified. Assigning `None` removes any bottom-margin setting.

property different_first_page_header_footer: bool

True if this section displays a distinct first-page header and footer.

Read/write. The definition of the first-page header and footer are accessed using *first_page_header* and *first_page_footer* respectively.

property even_page_footer: *_Footer*

_Footer object defining footer content for even pages.

The content of this footer definition is ignored unless the document setting *odd_and_even_pages_header_footer* is set True.

property even_page_header: *_Header*

_Header object defining header content for even pages.

The content of this header definition is ignored unless the document setting *odd_and_even_pages_header_footer* is set True.

property first_page_footer: *_Footer*

_Footer object defining footer content for the first page of this section.

The content of this footer definition is ignored unless the property *different_first_page_header_footer* is set True.

property first_page_header: *_Header*

_Header object defining header content for the first page of this section.

The content of this header definition is ignored unless the property *different_first_page_header_footer* is set True.

footer

_Footer object representing default page footer for this section.

The default footer is used for odd-numbered pages when separate odd/even footers are enabled. It is used for both odd and even-numbered pages otherwise.

property footer_distance: *Length* | None

Distance from bottom edge of page to bottom edge of the footer.

Read/write. `None` if no setting is present in the XML.

property gutter: *Length* | None

Length object representing page gutter size in English Metric Units.

Read/write. The page gutter is extra spacing added to the *inner* margin to ensure even margins after page binding. Generally only used in book-bound documents with double-sided and facing pages.

This setting applies to all pages in this section.

header

_Header object representing default page header for this section.

The default header is used for odd-numbered pages when separate odd/even headers are enabled. It is used for both odd and even-numbered pages otherwise.

property header_distance: *Length* | None

Distance from top edge of page to top edge of header.

Read/write. `None` if no setting is present in the XML. Assigning `None` causes default value to be used.

iter_inner_content() → Iterator[*Paragraph* | *Table*]

Generate each Paragraph or Table object in this *section*.

Items appear in document order.

property left_margin: *Length* | None

Length object representing the left margin for all pages in this section in English Metric Units.

property orientation: WD_ORIENTATION

WD_ORIENTATION member specifying page orientation for this section.

One of `WD_ORIENT.PORTRAIT` or `WD_ORIENT.LANDSCAPE`.

property page_height: *Length* | None

Total page height used for this section.

This value is inclusive of all edge spacing values such as margins.

Page orientation is taken into account, so for example, its expected value would be `Inches(8.5)` for letter-sized paper when orientation is landscape.

property page_width: *Length* | None

Total page width used for this section.

This value is like "paper size" and includes all edge spacing values such as margins.

Page orientation is taken into account, so for example, its expected value would be `Inches(11)` for letter-sized paper when orientation is landscape.

property right_margin: *Length* | None

Length object representing the right margin for all pages in this section in English Metric Units.

property start_type: WD_SECTION_START

Type of page-break (if any) inserted at the start of this section.

For exmple, `WD_SECTION_START.ODD_PAGE` if the section should begin on the next odd page, possibly inserting two page-breaks instead of one.

property top_margin: *Length* | None

Length object representing the top margin for all pages in this section in English Metric Units.

3.6.3 _Header and _Footer objects

class docx.section.**_Header**

Page header, used for all three types (default, even-page, and first-page).

Note that, like a document or table cell, a header must contain a minimum of one paragraph and a new or otherwise "empty" header contains a single empty paragraph. This first paragraph can be accessed as *header.paragraphs[0]* for purposes of adding content to it. Using *add_paragraph()* by itself to add content will leave an empty paragraph above the newly added one.

add_paragraph(*text: str = ''*, *style: str | ParagraphStyle | None = None*) → *Paragraph*

Return paragraph newly added to the end of the content in this container.

The paragraph has *text* in a single run if present, and is given paragraph style *style*.

If *style* is None, no paragraph style is applied, which has the same effect as applying the 'Normal' style.

add_table(*rows: int*, *cols: int*, *width: Length*) → *Table*

Return table of *width* having *rows* rows and *cols* columns.

The table is appended appended at the end of the content in this container.

width is evenly distributed between the table columns.

property is_linked_to_previous: bool

True if this header/footer uses the definition from the prior section.

False if this header/footer has an explicit definition.

Assigning True to this property removes the header/footer definition for this section, causing it to "inherit" the corresponding definition of the prior section. Assigning False causes a new, empty definition to be added for this section, but only if no definition is already present.

iter_inner_content() → Iterator[*Paragraph* | *Table*]

Generate each *Paragraph* or *Table* in this container in document order.

property paragraphs

A list containing the paragraphs in this container, in document order.

Read-only.

property tables

A list containing the tables in this container, in document order.

Read-only.

class docx.section.**_Footer**

Page footer, used for all three types (default, even-page, and first-page).

Note that, like a document or table cell, a footer must contain a minimum of one paragraph and a new or otherwise "empty" footer contains a single empty paragraph. This first paragraph can be accessed as *footer.paragraphs[0]* for purposes of adding content to it. Using *add_paragraph()* by itself to add content will leave an empty paragraph above the newly added one.

add_paragraph(*text: str = ''*, *style: str | ParagraphStyle | None = None*) → *Paragraph*

Return paragraph newly added to the end of the content in this container.

The paragraph has *text* in a single run if present, and is given paragraph style *style*.

If *style* is None, no paragraph style is applied, which has the same effect as applying the 'Normal' style.

add_table(*rows: int*, *cols: int*, *width:* Length) → *Table*

Return table of *width* having *rows* rows and *cols* columns.

The table is appended appended at the end of the content in this container.

width is evenly distributed between the table columns.

property is_linked_to_previous: bool

`True` if this header/footer uses the definition from the prior section.

`False` if this header/footer has an explicit definition.

Assigning `True` to this property removes the header/footer definition for this section, causing it to "inherit" the corresponding definition of the prior section. Assigning `False` causes a new, empty definition to be added for this section, but only if no definition is already present.

iter_inner_content() → Iterator[*Paragraph* | *Table*]

Generate each *Paragraph* or *Table* in this container in document order.

property paragraphs

A list containing the paragraphs in this container, in document order.

Read-only.

property tables

A list containing the tables in this container, in document order.

Read-only.

3.7 Shape-related objects

3.7.1 InlineShapes objects

class docx.shape.**InlineShapes**(*body_elm: CT_Body*, *parent: StoryPart*)

Sequence of *InlineShape* instances, supporting len(), iteration, and indexed access.

3.7.2 InlineShape objects

The `width` and `height` property of *InlineShape* provide a length object that is an instance of *Length*. These instances behave like an int, but also have built-in units conversion properties, e.g.:

```
>>> inline_shape.height
914400
>>> inline_shape.height.inches
1.0
```

class docx.shape.**InlineShape**(*inline: CT_Inline*)

Proxy for an <wp:inline> element, representing the container for an inline graphical object.

property height: *Length*

Read/write.

The display height of this inline shape as an *Emu* instance.

property type

The type of this inline shape as a member of `docx.enum.shape.WD_INLINE_SHAPE`, e.g. `LINKED_PICTURE`.

Read-only.

property width

Read/write.

The display width of this inline shape as an *Emu* instance.

3.8 DrawingML objects

Low-level drawing elements like color that appear in various document contexts.

3.8.1 ColorFormat objects

class docx.dml.color.**ColorFormat**

Provides access to color settings such as RGB color, theme color, and luminance adjustments.

property rgb

An *RGBColor* value or None if no RGB color is specified.

When *type* is *MSO_COLOR_TYPE.RGB*, the value of this property will always be an *RGBColor* value. It may also be an *RGBColor* value if *type* is *MSO_COLOR_TYPE.THEME*, as Word writes the current value of a theme color when one is assigned. In that case, the RGB value should be interpreted as no more than a good guess however, as the theme color takes precedence at rendering time. Its value is None whenever *type* is either None or *MSO_COLOR_TYPE.AUTO*.

Assigning an *RGBColor* value causes *type* to become *MSO_COLOR_TYPE.RGB* and any theme color is removed. Assigning None causes any color to be removed such that the effective color is inherited from the style hierarchy.

property theme_color

Member of *MSO_THEME_COLOR_INDEX* or None if no theme color is specified.

When *type* is *MSO_COLOR_TYPE.THEME*, the value of this property will always be a member of *MSO_THEME_COLOR_INDEX*. When *type* has any other value, the value of this property is None.

Assigning a member of *MSO_THEME_COLOR_INDEX* causes *type* to become *MSO_COLOR_TYPE.THEME*. Any existing RGB value is retained but ignored by Word. Assigning None causes any color specification to be removed such that the effective color is inherited from the style hierarchy.

property type: MSO_COLOR_TYPE

Read-only.

A member of *MSO_COLOR_TYPE*, one of RGB, THEME, or AUTO, corresponding to the way this color is defined. Its value is None if no color is applied at this level, which causes the effective color to be inherited from the style hierarchy.

3.9 Shared classes

3.9.1 Length objects

Length values in python-docx are expressed as a standardized *Length* value object. *Length* is a subclass of int, having all the behavior of an int. In addition, it has built-in units conversion properties, e.g.:

```
>>> inline_shape.height
914400
>>> inline_shape.height.inches
1.0
```

Length objects are constructed using a selection of convenience constructors, allowing values to be expressed in the units most appropriate to the context.

class docx.shared.**Length**(*emu: int*)

Base class for length constructor classes Inches, Cm, Mm, Px, and Emu.

Behaves as an int count of English Metric Units, 914,400 to the inch, 36,000 to the mm. Provides convenience unit conversion methods in the form of read-only properties. Immutable.

property cm

The equivalent length expressed in centimeters (float).

property emu

The equivalent length expressed in English Metric Units (int).

property inches

The equivalent length expressed in inches (float).

property mm

The equivalent length expressed in millimeters (float).

property pt

Floating point length in points.

property twips

The equivalent length expressed in twips (int).

class docx.shared.**Inches**(*inches: float*)

Convenience constructor for length in inches, e.g. `width = Inches(0.5)`.

class docx.shared.**Cm**(*cm: float*)

Convenience constructor for length in centimeters, e.g. `height = Cm(12)`.

class docx.shared.**Mm**(*mm: float*)

Convenience constructor for length in millimeters, e.g. `width = Mm(240.5)`.

class docx.shared.**Pt**(*points: float*)

Convenience value class for specifying a length in points.

class docx.shared.**Twips**(*twips: float*)

Convenience constructor for length in twips, e.g. `width = Twips(42)`.

A twip is a twentieth of a point, 635 EMU.

class docx.shared.**Emu**(*emu: int*)

Convenience constructor for length in English Metric Units, e.g. `width = Emu(457200)`.

3.9.2 RGBColor objects

class docx.shared.**RGBColor**(*r*, *g*, *b*)

Immutable value object defining a particular RGB color.

r, *g*, and *b* are each an integer in the range 0-255 inclusive. Using the hexidecimal integer notation, e.g. *0x42* may enhance readability where hex RGB values are in use:

```
>>> lavender = RGBColor(0xff, 0x99, 0xcc)
```

classmethod from_string(*rgb_hex_str: str*) → *RGBColor*

Return a new instance from an RGB color hex string like `'3C2F80'`.

3.10 Enumerations

Documentation for the various enumerations used for `python-docx` property settings can be found here:

3.10.1 MSO_COLOR_TYPE

Specifies the color specification scheme

Example:

```
from docx.enum.dml import MSO_COLOR_TYPE

assert font.color.type == MSO_COLOR_TYPE.THEME
```

RGB
Color is specified by an *RGBColor* value.

THEME
Color is one of the preset theme colors.

AUTO
Color is determined automatically be the application.

3.10.2 MSO_THEME_COLOR_INDEX

Indicates the Office theme color, one of those shown in the color gallery on the formatting ribbon.

Alias: `MSO_THEME_COLOR`

Example:

```
from docx.enum.dml import MSO_THEME_COLOR

font.color.theme_color = MSO_THEME_COLOR.ACCENT_1
```

NOT_THEME_COLOR
Indicates the color is not a theme color.

ACCENT_1
Specifies the Accent 1 theme color.

ACCENT_2
Specifies the Accent 2 theme color.

ACCENT_3
Specifies the Accent 3 theme color.

ACCENT_4
Specifies the Accent 4 theme color.

ACCENT_5
Specifies the Accent 5 theme color.

ACCENT_6
Specifies the Accent 6 theme color.

BACKGROUND_1
Specifies the Background 1 theme color.

BACKGROUND_2
Specifies the Background 2 theme color.

DARK_1
Specifies the Dark 1 theme color.

DARK_2
Specifies the Dark 2 theme color.

FOLLOWED_HYPERLINK
Specifies the theme color for a clicked hyperlink.

HYPERLINK
Specifies the theme color for a hyperlink.

LIGHT_1
Specifies the Light 1 theme color.

LIGHT_2
Specifies the Light 2 theme color.

TEXT_1
Specifies the Text 1 theme color.

TEXT_2
Specifies the Text 2 theme color.

MIXED
Indicates multiple theme colors are used.

3.10.3 WD_PARAGRAPH_ALIGNMENT

alias: **WD_ALIGN_PARAGRAPH**

Specifies paragraph justification type.

Example:

```
from docx.enum.text import WD_ALIGN_PARAGRAPH

paragraph = document.add_paragraph()
paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

LEFT
Left-aligned

CENTER
Center-aligned.

RIGHT
Right-aligned.

JUSTIFY
Fully justified.

DISTRIBUTE
Paragraph characters are distributed to fill the entire width of the paragraph.

JUSTIFY_MED
Justified with a medium character compression ratio.

JUSTIFY_HI
Justified with a high character compression ratio.

JUSTIFY_LOW
Justified with a low character compression ratio.

THAI_JUSTIFY
Justified according to Thai formatting layout.

3.10.4 WD_BUILTIN_STYLE

alias: **WD_STYLE**

Specifies a built-in Microsoft Word style.

Example:

```
from docx import Document
from docx.enum.style import WD_STYLE

document = Document()
styles = document.styles
style = styles[WD_STYLE.BODY_TEXT]
```

BLOCK_QUOTATION
Block Text.

BODY_TEXT
Body Text.

BODY_TEXT_2
Body Text 2.

BODY_TEXT_3
Body Text 3.

BODY_TEXT_FIRST_INDENT
Body Text First Indent.

BODY_TEXT_FIRST_INDENT_2
Body Text First Indent 2.

BODY_TEXT_INDENT
Body Text Indent.

BODY_TEXT_INDENT_2
Body Text Indent 2.

BODY_TEXT_INDENT_3
Body Text Indent 3.

BOOK_TITLE
Book Title.

CAPTION
Caption.

CLOSING
Closing.

COMMENT_REFERENCE
Comment Reference.

COMMENT_TEXT
Comment Text.

DATE
Date.

DEFAULT_PARAGRAPH_FONT
Default Paragraph Font.

EMPHASIS
Emphasis.

ENDNOTE_REFERENCE
Endnote Reference.

ENDNOTE_TEXT
Endnote Text.

ENVELOPE_ADDRESS
Envelope Address.

ENVELOPE_RETURN
Envelope Return.

FOOTER
Footer.

FOOTNOTE_REFERENCE
Footnote Reference.

FOOTNOTE_TEXT
Footnote Text.

HEADER
Header.

HEADING_1
Heading 1.

HEADING_2
Heading 2.

HEADING_3
Heading 3.

HEADING_4
Heading 4.

HEADING_5
Heading 5.

HEADING_6
Heading 6.

HEADING_7
Heading 7.

HEADING_8
Heading 8.

HEADING_9
Heading 9.

HTML_ACRONYM
HTML Acronym.

HTML_ADDRESS
HTML Address.

HTML_CITE
HTML Cite.

HTML_CODE
HTML Code.

HTML_DFN
HTML Definition.

HTML_KBD
HTML Keyboard.

HTML_NORMAL
Normal (Web).

HTML_PRE
HTML Preformatted.

HTML_SAMP
HTML Sample.

HTML_TT
HTML Typewriter.

HTML_VAR
HTML Variable.

HYPERLINK
Hyperlink.

HYPERLINK_FOLLOWED
Followed Hyperlink.

INDEX_1
Index 1.

INDEX_2
Index 2.

INDEX_3
Index 3.

INDEX_4
Index 4.

INDEX_5
Index 5.

INDEX_6
Index 6.

INDEX_7
Index 7.

INDEX_8
Index 8.

INDEX_9
Index 9.

INDEX_HEADING
Index Heading

INTENSE_EMPHASIS
Intense Emphasis.

INTENSE_QUOTE
Intense Quote.

INTENSE_REFERENCE
Intense Reference.

LINE_NUMBER
Line Number.

LIST
List.

LIST_2
List 2.

LIST_3
List 3.

LIST_4
List 4.

LIST_5
List 5.

LIST_BULLET
List Bullet.

LIST_BULLET_2
List Bullet 2.

LIST_BULLET_3
List Bullet 3.

LIST_BULLET_4
List Bullet 4.

LIST_BULLET_5
List Bullet 5.

LIST_CONTINUE
List Continue.

LIST_CONTINUE_2
List Continue 2.

LIST_CONTINUE_3
List Continue 3.

LIST_CONTINUE_4
List Continue 4.

LIST_CONTINUE_5
List Continue 5.

LIST_NUMBER
List Number.

LIST_NUMBER_2
List Number 2.

LIST_NUMBER_3
List Number 3.

LIST_NUMBER_4
List Number 4.

LIST_NUMBER_5
List Number 5.

LIST_PARAGRAPH
List Paragraph.

MACRO_TEXT
Macro Text.

MESSAGE_HEADER
Message Header.

NAV_PANE
Document Map.

NORMAL
Normal.

NORMAL_INDENT
Normal Indent.

NORMAL_OBJECT
Normal (applied to an object).

NORMAL_TABLE
Normal (applied within a table).

NOTE_HEADING
Note Heading.

PAGE_NUMBER
Page Number.

PLAIN_TEXT
Plain Text.

QUOTE
Quote.

SALUTATION
Salutation.

SIGNATURE
Signature.

STRONG
Strong.

SUBTITLE
Subtitle.

SUBTLE_EMPHASIS
Subtle Emphasis.

SUBTLE_REFERENCE
Subtle Reference.

TABLE_COLORFUL_GRID
Colorful Grid.

TABLE_COLORFUL_LIST
Colorful List.

TABLE_COLORFUL_SHADING
Colorful Shading.

TABLE_DARK_LIST
Dark List.

TABLE_LIGHT_GRID
Light Grid.

TABLE_LIGHT_GRID_ACCENT_1
Light Grid Accent 1.

TABLE_LIGHT_LIST
Light List.

TABLE_LIGHT_LIST_ACCENT_1
Light List Accent 1.

TABLE_LIGHT_SHADING
Light Shading.

TABLE_LIGHT_SHADING_ACCENT_1
Light Shading Accent 1.

TABLE_MEDIUM_GRID_1
Medium Grid 1.

TABLE_MEDIUM_GRID_2
Medium Grid 2.

TABLE_MEDIUM_GRID_3
Medium Grid 3.

TABLE_MEDIUM_LIST_1
Medium List 1.

TABLE_MEDIUM_LIST_1_ACCENT_1
Medium List 1 Accent 1.

TABLE_MEDIUM_LIST_2
Medium List 2.

TABLE_MEDIUM_SHADING_1
Medium Shading 1.

TABLE_MEDIUM_SHADING_1_ACCENT_1
Medium Shading 1 Accent 1.

TABLE_MEDIUM_SHADING_2
Medium Shading 2.

TABLE_MEDIUM_SHADING_2_ACCENT_1
Medium Shading 2 Accent 1.

TABLE_OF_AUTHORITIES
Table of Authorities.

TABLE_OF_FIGURES
Table of Figures.

TITLE
Title.

TOAHEADING
TOA Heading.

TOC_1
TOC 1.

TOC_2
TOC 2.

TOC_3
TOC 3.

TOC_4
TOC 4.

TOC_5
TOC 5.

TOC_6
TOC 6.

TOC_7
TOC 7.

TOC_8
TOC 8.

TOC_9
TOC 9.

3.10.5 WD_CELL_VERTICAL_ALIGNMENT

alias: **WD_ALIGN_VERTICAL**

Specifies the vertical alignment of text in one or more cells of a table.

Example:

```
from docx.enum.table import WD_ALIGN_VERTICAL

table = document.add_table(3, 3)
table.cell(0, 0).vertical_alignment = WD_ALIGN_VERTICAL.BOTTOM
```

TOP
Text is aligned to the top border of the cell.

CENTER
Text is aligned to the center of the cell.

BOTTOM
Text is aligned to the bottom border of the cell.

BOTH
This is an option in the OpenXml spec, but not in Word itself. It's not clear what Word behavior this setting produces. If you find out please let us know and we'll update this documentation. Otherwise, probably best to avoid this option.

3.10.6 WD_COLOR_INDEX

alias: **WD_COLOR**

Specifies a standard preset color to apply. Used for font highlighting and perhaps other applications.

AUTO
Automatic color. Default; usually black.

BLACK
Black color.

BLUE
Blue color

BRIGHT_GREEN
Bright green color.

DARK_BLUE
Dark blue color.

DARK_RED
Dark red color.

DARK_YELLOW
Dark yellow color.

GRAY_25
25% shade of gray color.

GRAY_50
50% shade of gray color.

GREEN
Green color.

PINK
Pink color.

RED
Red color.

TEAL
Teal color.

TURQUOISE
Turquoise color.

VIOLET
Violet color.

WHITE
White color.

YELLOW
Yellow color.

3.10.7 WD_LINE_SPACING

Specifies a line spacing format to be applied to a paragraph.

Example:

```
from docx.enum.text import WD_LINE_SPACING

paragraph = document.add_paragraph()
paragraph.paragraph_format.line_spacing_rule = WD_LINE_SPACING.EXACTLY
```

ONE_POINT_FIVE
Space-and-a-half line spacing.

AT_LEAST
Line spacing is always at least the specified amount. The amount is specified separately.

DOUBLE
Double spaced.

EXACTLY
Line spacing is exactly the specified amount. The amount is specified separately.

MULTIPLE
Line spacing is specified as a multiple of line heights. Changing the font size will change the line spacing proportionately.

SINGLE
Single spaced (default).

3.10.8 WD_ORIENTATION

alias: **WD_ORIENT**

Specifies the page layout orientation.

Example:

```
from docx.enum.section import WD_ORIENT

section = document.sections[-1]
section.orientation = WD_ORIENT.LANDSCAPE
```

PORTRAIT
Portrait orientation.

LANDSCAPE
Landscape orientation.

3.10.9 WD_TABLE_ALIGNMENT

Specifies table justification type.

Example:

```
from docx.enum.table import WD_TABLE_ALIGNMENT

table = document.add_table(3, 3)
table.alignment = WD_TABLE_ALIGNMENT.CENTER
```

LEFT
Left-aligned

CENTER
Center-aligned.

RIGHT
Right-aligned.

3.10.10 WD_ROW_HEIGHT_RULE

alias: **WD_ROW_HEIGHT**

Specifies the rule for determining the height of a table row

Example:

```
from docx.enum.table import WD_ROW_HEIGHT_RULE

table = document.add_table(3, 3)
table.rows[0].height_rule = WD_ROW_HEIGHT_RULE.EXACTLY
```

AUTO
The row height is adjusted to accommodate the tallest value in the row.

AT_LEAST
The row height is at least a minimum specified value.

EXACTLY
The row height is an exact value.

3.10.11 WD_SECTION_START

alias: **WD_SECTION**

Specifies the start type of a section break.

Example:

```
from docx.enum.section import WD_SECTION

section = document.sections[0]
section.start_type = WD_SECTION.NEW_PAGE
```

CONTINUOUS
Continuous section break.

NEW_COLUMN
New column section break.

NEW_PAGE
New page section break.

EVEN_PAGE
Even pages section break.

ODD_PAGE
Section begins on next odd page.

3.10.12 WD_STYLE_TYPE

Specifies one of the four style types: paragraph, character, list, or table.

Example:

```
from docx import Document
from docx.enum.style import WD_STYLE_TYPE

styles = Document().styles
assert styles[0].type == WD_STYLE_TYPE.PARAGRAPH
```

CHARACTER
Character style.

LIST
List style.

PARAGRAPH
Paragraph style.

TABLE
Table style.

3.10.13 WD_TAB_ALIGNMENT

Specifies the tab stop alignment to apply.

LEFT
Left-aligned.

CENTER
Center-aligned.

RIGHT
Right-aligned.

DECIMAL
Decimal-aligned.

BAR
Bar-aligned.

LIST
List-aligned. (deprecated)

CLEAR
Clear an inherited tab stop.

END
Right-aligned. (deprecated)

NUM
Left-aligned. (deprecated)

START
Left-aligned. (deprecated)

3.10.14 WD_TAB_LEADER

Specifies the character to use as the leader with formatted tabs.

SPACES
Spaces. Default.

DOTS
Dots.

DASHES
Dashes.

LINES
Double lines.

HEAVY
A heavy line.

MIDDLE_DOT
A vertically-centered dot.

3.10.15 WD_TABLE_DIRECTION

Specifies the direction in which an application orders cells in the specified table or row.

Example:

```
from docx.enum.table import WD_TABLE_DIRECTION

table = document.add_table(3, 3)
table.direction = WD_TABLE_DIRECTION.RTL
```

LTR
The table or row is arranged with the first column in the leftmost position.

RTL
The table or row is arranged with the first column in the rightmost position.

3.10.16 WD_UNDERLINE

Specifies the style of underline applied to a run of characters.

NONE
No underline. This setting overrides any inherited underline value, so can be used to remove underline from a run that inherits underlining from its containing paragraph. Note this is not the same as assigning `None` to Run.underline. `None` is a valid assignment value, but causes the run to inherit its underline value. Assigning `WD_UNDERLINE.NONE` causes underlining to be unconditionally turned off.

SINGLE
A single line. Note that this setting is write-only in the sense that `True` (rather than `WD_UNDERLINE.SINGLE`) is returned for a run having this setting.

WORDS
Underline individual words only.

DOUBLE
A double line.

DOTTED
Dots.

THICK
A single thick line.

DASH
Dashes.

DOT_DASH
Alternating dots and dashes.

DOT_DOT_DASH
An alternating dot-dot-dash pattern.

WAVY
A single wavy line.

DOTTED_HEAVY
Heavy dots.

DASH_HEAVY
Heavy dashes.

DOT_DASH_HEAVY
Alternating heavy dots and heavy dashes.

DOT_DOT_DASH_HEAVY
An alternating heavy dot-dot-dash pattern.

WAVY_HEAVY
A heavy wavy line.

DASH_LONG
Long dashes.

WAVY_DOUBLE
A double wavy line.

DASH_LONG_HEAVY
Long heavy dashes.

CHAPTER

FOUR

CONTRIBUTOR GUIDE

4.1 Analysis

Documentation of studies undertaken in support of API and code design.

4.1.1 Feature Analysis

Header and Footer

In a WordprocessingML document, a page header is text that is separated from the main body of text and appears at the top of a printed page. The page headers in a document are often the same from page to page, with only small differences in content, such as a section title or page number. Such a header is also known as a running head.

A page footer is analogous in every way to a page header except that it appears at the bottom of a page. It should not be confused with a footnote, which is not uniform between pages. For brevity's sake, the term *header* is often used here to refer to what may be either a header or footer object, trusting the reader to understand its applicability to both object types.

In book-printed documents, where pages are printed on both sides, when opened, the front or *recto* side of each page appears to the right of the bound edge and the back or *verso* side of each page appears on the left. The first printed page receives the page-number "1", and is always a recto page. Because pages are numbered consecutively, each recto page receives an *odd* page number and each verso page receives an *even* page number.

The header appearing on a recto page often differs from that on a verso page. Supporting this difference gives rise to the option to have an even-page header that differs from the default odd-page header in a document. This "both odd-and-even headers" option is applied at the document level and affects all sections of the document.

The header appearing on the first page of a section (e.g. a chapter) may differ from that appearing on subsequent pages. Supporting this difference gives rise to the option to set a distinct first-page header. This "different first-page-header" option is applied at the section level and may differ from section-to-section in the document.

In WordprocessingML, a header or footer appears within the margin area of a page. With a few exceptions, a header or footer may contain all the types of content that can appear in the main body, including text and images. Each header and footer has access to the styles defined in `/word/styles.xml`.

Each section has its own set of headers and footers, although a section can be configured to "inherit" headers and footers from the prior section. Each section can have three header definitions, the default header, even header, and first page header. When different even/odd headers are not enabled, the default header appears on both even and odd numbered pages. If even/odd headers are enabled, the default header is used for odd pages. A corresponding set of three footer definitions are also possible. All header/footer definitions are optional.

Open Questions

- What about a continuous section break? What is the header/footer behavior there?

Candidate Protocol

Every section has a header; it is never None:

```
>>> header = section.header
>>> header
<docx.hdrftr.Header object at 0x02468ACE>
```

There are three header properties on `Section`: *.header*, *.even_page_header*, and *.first_page_header*. All header objects share the same properties and methods. There are three corresponding properties for the footers.

Header is a subclass of `BlockItemContainer`, from which it inherits the same content editing capabilities as `Document`, such as *.add_paragraph()*.

If the *w:headerReference* element for a header is not present, the definition for that header is "inherited" from the prior section. This action is recursive, such that, for example, the header definition from the first section could be applied to the third section. A header that inherits its definition is said to be "linked to previous". Perhaps counterintuitively, a header for the first section can be "linked to previous", even though no previous section exists. The *.is_linked_to_previous* property is simply a test for the existence of a header definition in the current section:

```
>>> header.is_linked_to_previous
True
```

Editing operations transparently operate on the source header, the one in the first prior section having a header of that type (when one is not present in the current section). If no prior sections have a header, one is created in the first section of the document on the first constructive edit call:

```
>>> header = document.sections[0].header
>>> header.is_linked_to_previous
True
>>> header.text = 'foobar'
>>> header.is_linked_to_previous
False
```

Assigning False to *.is_linked_to_previous* creates a blank header for that section when one does not already exist:

```
>>> header.is_linked_to_previous
True
>>> header.is_linked_to_previous = False
>>> header.is_linked_to_previous
False
```

Conversely, an existing header is deleted from a section by assigning True to *.is_linked_to_previous*:

```
>>> header.is_linked_to_previous
False
>>> header.is_linked_to_previous = True
>>> header.is_linked_to_previous
True
```

The document settings object has a read/write *.odd_and_even_pages_header_footer* property that indicates verso and recto pages will have a different header. Any existing even page header definitions are preserved when

.odd_and_even_pages_header_footer is False; they are simply not rendered by Word. Assigning *True* to *.odd_and_even_pages_header_footer* does not automatically create new even header definitions:

```
>>> document.settings.odd_and_even_pages_header_footer
False
>>> document.settings.odd_and_even_pages_header_footer = True
>>> section.even_page_header.is_linked_to_previous
True
```

Section has a read/write *.different_first_page_header_footer* property that indicates whether the first page of the section should have a distinct header. Assigning *True* to *.different_first_page_header_footer* does not automatically create a new first page header definition:

```
>>> section.different_first_page_header_footer
False
>>> section.different_first_page_header_footer = True
>>> section.different_first_page_header_footer
True
>>> section.first_page_header.is_linked_to_previous
True
```

Specimen XML

There are seven different permutations of headers:

The same header on all pages of the document:

```
<w:sectPr>
    <w:headerReference w:type="default" r:id="rId3"/>
    ...
</w:sectPr>
```

Only an odd header. The section is exactly the same as above but *settings.xml* has the the *<w:evenAndOddHeaders>* property:

```
<w:settings xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
   ...
   <w:evenAndOddHeaders w:val="1"/>
   ...
</w:settings>
```

Different even and odd headers:

```
<w:sectPr>
    <w:headerReference w:type="default" r:id="rId3"/>
    <w:headerReference w:type="even" r:id="rId4"/>
    ...
</w:sectPr>
```

Distinct first page header, subsequent pages all have the same header:

```
<w:sectPr>
    <w:headerReference w:type="default" r:id="rId3"/>
    <w:headerReference w:type="first" r:id="rId4"/>
```

(continues on next page)

(continued from previous page)

```
    <w:titlePg/>
    ...
</w:sectPr>
```

Distinct first, even, and odd page headers:

```
<w:sectPr>
    <w:headerReference w:type="default" r:id="rId3"/>
    <w:headerReference w:type="first" r:id="rId4"/>
    <w:headerReference w:type="even" r:id="rId5"/>
    <w:titlePg/>
    ...
</w:sectPr>
```

A header part:

```
<w:hdr>
  <w:p>
    <w:pPr>
      <w:pStyle w:val="Header"/>
    </w:pPr>
    <w:r>
      <w:t>Header for section-1</w:t>
    </w:r>
  </w:p>
</w:hdr>
```

Word Behavior

- When you turn off even/odd headers, Word sets the value of *w:evenAndOddHeaders* to 0, but does not actually remove the even header.
- When you turn off first page header, Word sets the value of *w:titlePg* to 0, but does not actually remove the even header.
- Word will load a file with an even page header but no odd page header.

MS API

WdHeaderFooterIndex Enumeration:

```
EVEN_PAGES = 3
FIRST_PAGE = 2
PRIMARY    = 1
```

Create footer in MS API:

```
section = Document.Sections(1)
footers = section.Footers  # a HeadersFooters collection object
default_footer = footers(wdHeaderFooterPrimary)
default_footer.Range.Text = "Footer text"
```

PageSetup object:

```
DifferentFirstPageHeaderFooter: Read/write {True, False, WD_UNDEFINED}
OddAndEvenPagesHeaderFooter: Read/write {True, False, WD_UNDEFINED}
```

Schema Excerpt

```
<xsd:complexType name="CT_SectPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="6"/>
      <xsd:element name="headerReference" type="CT_HdrFtrRef"/>
      <xsd:element name="footerReference" type="CT_HdrFtrRef"/>
    </xsd:choice>
    <xsd:element name="footnotePr"      type="CT_FtnProps"      minOccurs="0"/>
    <xsd:element name="endnotePr"       type="CT_EdnProps"      minOccurs="0"/>
    <xsd:element name="type"            type="CT_SectType"      minOccurs="0"/>
    <xsd:element name="pgSz"            type="CT_PageSz"        minOccurs="0"/>
    <xsd:element name="pgMar"           type="CT_PageMar"       minOccurs="0"/>
    <xsd:element name="paperSrc"        type="CT_PaperSource"   minOccurs="0"/>
    <xsd:element name="pgBorders"       type="CT_PageBorders"   minOccurs="0"/>
    <xsd:element name="lnNumType"       type="CT_LineNumber"    minOccurs="0"/>
    <xsd:element name="pgNumType"       type="CT_PageNumber"    minOccurs="0"/>
    <xsd:element name="cols"            type="CT_Columns"       minOccurs="0"/>
    <xsd:element name="formProt"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="vAlign"          type="CT_VerticalJc"    minOccurs="0"/>
    <xsd:element name="noEndnote"       type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="titlePg"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="textDirection"   type="CT_TextDirection" minOccurs="0"/>
    <xsd:element name="bidi"            type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rtlGutter"       type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="docGrid"         type="CT_DocGrid"       minOccurs="0"/>
    <xsd:element name="printerSettings" type="CT_Rel"           minOccurs="0"/>
    <xsd:element name="sectPrChange"    type="CT_SectPrChange"  minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr"  type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel"  type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"    type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidSect" type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_HdrFtrRef">
  <xsd:attribute ref="r:id" use="required"/>
  <xsd:attribute name="type" type="ST_HdrFtr" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_HdrFtr">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="even"/>
    <xsd:enumeration value="default"/>
    <xsd:enumeration value="first"/>
  </xsd:restriction>
</xsd:simpleType>
```

Settings part

In WordprocessingML, document-level settings are defined in the *settings.xml* part. There are 98 distinct settings, all of which are optional (according to the spec at least).

The API does not provide for direct access to the settings part. A *Settings* proxy object is available on the *Document.settings* property and provides access to the document-level settings. The *Document* object obtains access via its document part. `DocumentPart` brokers all access to the settings part.

Candidate Protocol

```
>>> document = Document()
>>> document.settings
<docx.settings.Settings object at 0xdeadbeef4>
```

Specimen XML

Default *settings.xml* part for a new document in Word 2016:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<w:settings
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:sl="http://schemas.openxmlformats.org/schemaLibrary/2006/main"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
    xmlns:w16cid="http://schemas.microsoft.com/office/word/2016/wordml/cid"
    xmlns:w16se="http://schemas.microsoft.com/office/word/2015/wordml/symex"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    mc:Ignorable="w14 w15 w16se w16cid"
    >
  <w:zoom w:percent="150"/>
  <w:defaultTabStop w:val="720"/>
  <w:characterSpacingControl w:val="doNotCompress"/>
  <w:compat>
    <w:compatSetting w:name="compatibilityMode" w:uri="http://schemas.microsoft.com/
↪office/word" w:val="15"/>
    <w:compatSetting w:name="overrideTableStyleFontSizeAndJustification" w:uri="http://
↪schemas.microsoft.com/office/word" w:val="1"/>
    <w:compatSetting w:name="enableOpenTypeFeatures" w:uri="http://schemas.microsoft.com/
↪office/word" w:val="1"/>
    <w:compatSetting w:name="doNotFlipMirrorIndents" w:uri="http://schemas.microsoft.com/
↪office/word" w:val="1"/>
    <w:compatSetting w:name="differentiateMultirowTableHeaders" w:uri="http://schemas.
↪microsoft.com/office/word" w:val="1"/>
    <w:compatSetting w:name="useWord2013TrackBottomHyphenation" w:uri="http://schemas.
↪microsoft.com/office/word" w:val="0"/>
  </w:compat>
  <w:rsids>
    <w:rsidRoot w:val="005968A6"/>
```

(continues on next page)

(continued from previous page)

```
    <w:rsid w:val="00480A2E"/>
    <w:rsid w:val="005968A6"/>
  </w:rsids>
  <m:mathPr>
    <m:mathFont m:val="Cambria Math"/>
    <m:brkBin m:val="before"/>
    <m:brkBinSub m:val="--"/>
    <m:smallFrac m:val="0"/>
    <m:dispDef/>
    <m:lMargin m:val="0"/>
    <m:rMargin m:val="0"/>
    <m:defJc m:val="centerGroup"/>
    <m:wrapIndent m:val="1440"/>
    <m:intLim m:val="subSup"/>
    <m:naryLim m:val="undOvr"/>
  </m:mathPr>
  <w:themeFontLang w:val="en-US"/>
  <w:clrSchemeMapping w:bg1="light1" w:t1="dark1" w:bg2="light2" w:t2="dark2" w:accent1=
↪"accent1" w:accent2="accent2" w:accent3="accent3" w:accent4="accent4" w:accent5=
↪"accent5" w:accent6="accent6" w:hyperlink="hyperlink" w:followedHyperlink=
↪"followedHyperlink"/>
  <w:decimalSymbol w:val="."/>
  <w:listSeparator w:val=","/>
  <w15:chartTrackingRefBased/>
  <w15:docId w15:val="{3E989880-FF70-7C4D-8D4E-02DC7E104B81}"/>
</w:settings>
```

Schema Excerpts

```
<xsd:complexType name="CT_Settings">
  <xsd:sequence>
    <xsd:element name="writeProtection"           type="CT_WriteProtection" minOccurs="0
↪"/>
    <xsd:element name="view"                      type="CT_View"            minOccurs="0
↪"/>
    <xsd:element name="zoom"                      type="CT_Zoom"            minOccurs="0
↪"/>
    <xsd:element name="removePersonalInformation" type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="removeDateAndTime"         type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="doNotDisplayPageBoundaries" type="CT_OnOff"          minOccurs="0
↪"/>
    <xsd:element name="displayBackgroundShape"    type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="printPostScriptOverText"   type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="printFractionalCharacterWidth" type="CT_OnOff"       minOccurs="0
↪"/>
    <xsd:element name="printFormsData"            type="CT_OnOff"           minOccurs="0
↪"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="embedTrueTypeFonts"          type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="embedSystemFonts"            type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="saveSubsetFonts"             type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="saveFormsData"               type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="mirrorMargins"               type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="alignBordersAndEdges"        type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="bordersDoNotSurroundHeader" type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="bordersDoNotSurroundFooter" type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="gutterAtTop"                 type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="hideSpellingErrors"          type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="hideGrammaticalErrors"       type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="activeWritingStyle"          type="CT_WritingStyle"    minOccurs="0
↪" maxOccurs="unbounded"/>
    <xsd:element name="proofState"                  type="CT_Proof"           minOccurs="0
↪"/>
    <xsd:element name="formsDesign"                 type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="attachedTemplate"            type="CT_Rel"             minOccurs="0
↪"/>
    <xsd:element name="linkStyles"                  type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="stylePaneFormatFilter"       type="CT_StylePaneFilter" minOccurs="0
↪"/>
    <xsd:element name="stylePaneSortMethod"         type="CT_StyleSort"       minOccurs="0
↪"/>
    <xsd:element name="documentType"                type="CT_DocType"         minOccurs="0
↪"/>
    <xsd:element name="mailMerge"                   type="CT_MailMerge"       minOccurs="0
↪"/>
    <xsd:element name="revisionView"                type="CT_TrackChangesView" minOccurs=
↪"0"/>
    <xsd:element name="trackRevisions"              type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="doNotTrackMoves"             type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="doNotTrackFormatting"        type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="documentProtection"          type="CT_DocProtect"      minOccurs="0
↪"/>
    <xsd:element name="autoFormatOverride"          type="CT_OnOff"           minOccurs="0
↪"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="styleLockTheme"              type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="styleLockQFSet"              type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="defaultTabStop"              type="CT_TwipsMeasure"    minOccurs="0
↪"/>
    <xsd:element name="autoHyphenation"             type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="consecutiveHyphenLimit"      type="CT_DecimalNumber"   minOccurs="0
↪"/>
    <xsd:element name="hyphenationZone"             type="CT_TwipsMeasure"    minOccurs="0
↪"/>
    <xsd:element name="doNotHyphenateCaps"          type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="showEnvelope"                type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="summaryLength"               type="CT_DecimalNumberOrPrecent"␣
↪minOccurs="0"/>
    <xsd:element name="clickAndTypeStyle"           type="CT_String"          minOccurs="0
↪"/>
    <xsd:element name="defaultTableStyle"           type="CT_String"          minOccurs="0
↪"/>
    <xsd:element name="evenAndOddHeaders"           type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="bookFoldRevPrinting"         type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="bookFoldPrinting"            type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="bookFoldPrintingSheets"      type="CT_DecimalNumber"   minOccurs=
↪"0"/>
    <xsd:element name="drawingGridHorizontalSpacing"        type="CT_TwipsMeasure" ␣
↪minOccurs="0"/>
    <xsd:element name="drawingGridVerticalSpacing"          type="CT_TwipsMeasure" ␣
↪minOccurs="0"/>
    <xsd:element name="displayHorizontalDrawingGridEvery"   type="CT_DecimalNumber"␣
↪minOccurs="0"/>
    <xsd:element name="displayVerticalDrawingGridEvery"     type="CT_DecimalNumber"␣
↪minOccurs="0"/>
    <xsd:element name="doNotUseMarginsForDrawingGridOrigin" type="CT_OnOff"        ␣
↪minOccurs="0"/>
    <xsd:element name="drawingGridHorizontalOrigin"         type="CT_TwipsMeasure" ␣
↪minOccurs="0"/>
    <xsd:element name="drawingGridVerticalOrigin"  type="CT_TwipsMeasure"    minOccurs="0
↪"/>
    <xsd:element name="doNotShadeFormData"          type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="noPunctuationKerning"        type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="characterSpacingControl"     type="CT_CharacterSpacing" minOccurs=
↪"0"/>
    <xsd:element name="printTwoOnOne"               type="CT_OnOff"           minOccurs="0
↪"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="strictFirstAndLastChars"    type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="noLineBreaksAfter"          type="CT_Kinsoku"         minOccurs="0
↪"/>
    <xsd:element name="noLineBreaksBefore"         type="CT_Kinsoku"         minOccurs="0
↪"/>
    <xsd:element name="savePreviewPicture"         type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="doNotValidateAgainstSchema" type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="saveInvalidXml"             type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="ignoreMixedContent"         type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="alwaysShowPlaceholderText"  type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="doNotDemarcateInvalidXml"   type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="saveXmlDataOnly"            type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="useXSLTWhenSaving"          type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="saveThroughXslt"            type="CT_SaveThroughXslt" minOccurs="0
↪"/>
    <xsd:element name="showXMLTags"                type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="alwaysMergeEmptyNamespace"  type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="updateFields"               type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="hdrShapeDefaults"           type="CT_ShapeDefaults"   minOccurs="0
↪"/>
    <xsd:element name="footnotePr"                 type="CT_FtnDocProps"     minOccurs="0
↪"/>
    <xsd:element name="endnotePr"                  type="CT_EdnDocProps"     minOccurs="0
↪"/>
    <xsd:element name="compat"                     type="CT_Compat"          minOccurs="0
↪"/>
    <xsd:element name="docVars"                    type="CT_DocVars"         minOccurs="0
↪"/>
    <xsd:element name="rsids"                      type="CT_DocRsids"        minOccurs="0
↪"/>
    <xsd:element  ref="m:mathPr"                                             minOccurs="0
↪"/>
    <xsd:element name="attachedSchema"             type="CT_String"          minOccurs="0
↪" maxOccurs="unbounded"/>
    <xsd:element name="themeFontLang"              type="CT_Language"        minOccurs="0
↪"/>
    <xsd:element name="clrSchemeMapping"           type="CT_ColorSchemeMapping"␣
↪minOccurs="0"/>
    <xsd:element name="doNotIncludeSubdocsInStats" type="CT_OnOff"           minOccurs="0
↪"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="doNotAutoCompressPictures"  type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="forceUpgrade"               type="CT_Empty"           minOccurs="0
↪"/>
    <xsd:element name="captions"                   type="CT_Captions"        minOccurs="0
↪"/>
    <xsd:element name="readModeInkLockDown"        type="CT_ReadingModeInkLockDown"␣
↪minOccurs="0"/>
    <xsd:element name="smartTagType"               type="CT_SmartTagType"    minOccurs="0
↪" maxOccurs="unbounded"/>
    <xsd:element  ref="sl:schemaLibrary"                                     minOccurs="0
↪"/>
    <xsd:element name="shapeDefaults"              type="CT_ShapeDefaults"   minOccurs="0
↪"/>
    <xsd:element name="doNotEmbedSmartTags"        type="CT_OnOff"           minOccurs="0
↪"/>
    <xsd:element name="decimalSymbol"              type="CT_String"          minOccurs="0
↪"/>
    <xsd:element name="listSeparator"              type="CT_String"          minOccurs="0
↪"/>
  </xsd:sequence>
</xsd:complexType>
```

Text

Hyperlink

Word allows a hyperlink to be placed in a document wherever a paragraph can appear. The actual hyperlink element is a peer of *Run*.

The link may be to an external resource such as a web site, or internal, to another location in the document. The link may also be a *mailto:* URI or a reference to a file on an accessible local or network filesystem.

The visible text of a hyperlink is held in one or more runs. Technically a hyperlink can have zero runs, but this occurs only in contrived cases (otherwise there would be nothing to click on). As usual, each run can have its own distinct text formatting (font), so for example one word in the hyperlink can be bold, etc. By default, Word applies the built-in *Hyperlink* character style to a newly inserted hyperlink. Like other text, the hyperlink text may often be broken into multiple runs as a result of edits in different "revision-save" editing sessions (between "Save" commands).

Note that rendered page-breaks can occur in the middle of a hyperlink.

A *Hyperlink* is a child of *Paragraph*, a peer of *Run*.

TODO: What about URL-encoding/decoding (like %20) behaviors, if any?

Candidate protocol

An external hyperlink has an address and an optional anchor. An internal hyperlink has only an anchor. An anchor is more precisely known as a *URI fragment* in a web URL and follows a hash mark ("#"). The fragment-separator hash character is not stored in the XML.

Note that the anchor and address are stored in two distinct attributes, so you need to concatenate *.address* and *.anchor* like *f"{address}#{anchor}"* if you want the whole thing.

Also note that Word does not rigorously separate a fragment in a web URI so it may appear as part of the address or separately in the anchor attribute, depending on how the hyperlink was authored. Hyperlinks inserted using the

dialog-box seem to separate it and addresses typed into the document directly don't, based on my limited experience.

Access hyperlinks in a paragraph:

```
>>> hyperlinks = paragraph.hyperlinks
[<docx.text.hyperlink.Hyperlink at 0x7f...>]
```

Access hyperlinks in a paragraph in document order with runs:

```
>>> list(paragraph.iter_inner_content())
[
  <docx.text.run.Run at 0x7f...>
  <docx.text.hyperlink.Hyperlink at 0x7f...>
  <docx.text.run.Run at 0x7f...>
]
```

Access hyperlink address:

```
>>> hyperlink.address
'https://google.com/'
```

Access hyperlink fragment:

```
>>> hyperlink.fragment
'introduction'
```

Access hyperlink history (visited or not, True means not visited yet):

```
>>> hyperlink.history
True
```

Access hyperlinks runs:

```
>>> hyperlink.runs
[
  <docx.text.run.Run at 0x7f...>
  <docx.text.run.Run at 0x7f...>
  <docx.text.run.Run at 0x7f...>
]
```

Access hyperlink URL:

```
>>> hyperlink.url
'https://us.com#introduction'
```

Determine whether a hyperlink contains a rendered page-break:

```
>>> hyperlink.contains_page_break
False
```

Access visible text of a hyperlink:

```
>>> hyperlink.text
'an excellent Wikipedia article on ferrets'
```

Add an external hyperlink (not yet implemented):

```
>>> hyperlink = paragraph.add_hyperlink(
...   'About', address='http://us.com', fragment='about'
... )
>>> hyperlink
<docx.text.hyperlink.Hyperlink at 0x7f...>
>>> hyperlink.text
'About'
>>> hyperlink.address
'http://us.com'
>>> hyperlink.fragment
'about'
>>> hyperlink.url
'http://us.com#about'
```

Add an internal hyperlink (to a bookmark):

```
>>> hyperlink = paragraph.add_hyperlink('Section 1', fragment='Section_1')
>>> hyperlink.text
'Section 1'
>>> hyperlink.fragment
'Section_1'
>>> hyperlink.address
''
```

Modify hyperlink properties:

```
>>> hyperlink.text = 'Froogle'
>>> hyperlink.text
'Froogle'
>>> hyperlink.address = 'mailto:info@froogle.com?subject=sup dawg?'
>>> hyperlink.address
'mailto:info@froogle.com?subject=sup%20dawg%3F'
>>> hyperlink.anchor = None
>>> hyperlink.anchor
None
```

Add additional runs to a hyperlink:

```
>>> hyperlink.text = 'A '
>>> # .insert_run inserts a new run at idx, defaults to idx=-1
>>> hyperlink.insert_run(' link').bold = True
>>> hyperlink.insert_run('formatted', idx=1).bold = True
>>> hyperlink.text
'A formatted link'
>>> [r for r in hyperlink.iter_runs()]
[<docx.text.run.Run at 0x7fa...>,
 <docx.text.run.Run at 0x7fb...>,
 <docx.text.run.Run at 0x7fc...>]
```

Iterate over the run-level items a paragraph contains:

```
>>> paragraph = document.add_paragraph('A paragraph having a link to: ')
>>> paragraph.add_hyperlink(text='github', address='http://github.com')
```

(continues on next page)

(continued from previous page)

```
>>> [item for item in paragraph.iter_run_level_items()]:
[<docx.text.paragraph.Run at 0x7fd...>, <docx.text.paragraph.Hyperlink at 0x7fe...>]
```

Paragraph.text now includes text contained in a hyperlink:

```
>>> paragraph.text
'A paragraph having a link to: github'
```

Word Behaviors

- What are the semantics of the w:history attribute on w:hyperlink? I'm suspecting this indicates whether the link should show up blue (unvisited) or purple (visited). I'm inclined to think we need that as a read/write property on hyperlink. We should see what the MS API does on this count.
- We probably need to enforce some character-set restrictions on w:anchor. Word doesn't seem to like spaces or hyphens, for example. The simple type ST_String doesn't look like it takes care of this.
- We'll need to test URL escaping of special characters like spaces and question marks in Hyperlink.address.
- What does Word do when loading a document containing an internal hyperlink having an anchor value that doesn't match an existing bookmark? We'll want to know because we're sure to get support inquiries from folks who don't match those up and wonder why they get a repair error or whatever.

Specimen XML

External links

The address (URL) of an external hyperlink is stored in the document.xml.rels file, keyed by the w:hyperlink@r:id attribute:

```
<w:p>
  <w:r>
    <w:t xml:space="preserve">This is an external link to </w:t>
  </w:r>
  <w:hyperlink r:id="rId4">
    <w:r>
      <w:rPr>
        <w:rStyle w:val="Hyperlink"/>
      </w:rPr>
      <w:t>Google</w:t>
    </w:r>
  </w:hyperlink>
</w:p>
```

… mapping to relationship in document.xml.rels:

```
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId4" Mode="External" Type="http://..." Target="http://google.com/"/>
</Relationships>
```

A hyperlink can contain multiple runs of text (and a whole lot of other stuff, at least as far as the schema indicates):

```
<w:p>
  <w:hyperlink r:id="rId2">
```

(continues on next page)

(continued from previous page)

```
    <w:r>
      <w:rPr>
        <w:rStyle w:val="Hyperlink"/>
      </w:rPr>
      <w:t xml:space="preserve">A hyperlink containing an </w:t>
    </w:r>
    <w:r>
      <w:rPr>
        <w:rStyle w:val="Hyperlink"/>
        <w:i/>
      </w:rPr>
      <w:t>italicized</w:t>
    </w:r>
    <w:r>
      <w:rPr>
        <w:rStyle w:val="Hyperlink"/>
      </w:rPr>
      <w:t xml:space="preserve"> word</w:t>
    </w:r>
  </w:hyperlink>
</w:p>
```

Internal links

An internal link provides "jump to another document location" behavior in the Word UI. An internal link is distinguished by the absence of an r:id attribute. In this case, the w:anchor attribute is required. The value of the anchor attribute is the name of a bookmark in the document.

Example:

```
<w:p>
  <w:r>
    <w:t xml:space="preserve">See </w:t>
  </w:r>
  <w:hyperlink w:anchor="Section_4">
    <w:r>
      <w:rPr>
        <w:rStyle w:val="Hyperlink"/>
      </w:rPr>
      <w:t>Section 4</w:t>
    </w:r>
  </w:hyperlink>
  <w:r>
    <w:t xml:space="preserve"> for more details.</w:t>
  </w:r>
</w:p>
```

. . . referring to this bookmark elsewhere in the document:

```
<w:p>
  <w:bookmarkStart w:id="0" w:name="Section_4"/>
    <w:r>
```

(continues on next page)

(continued from previous page)

```
      <w:t>Section 4</w:t>
    </w:r>
  <w:bookmarkEnd w:id="0"/>
</w:p>
```

Schema excerpt

```
<xsd:complexType name="CT_P">
  <xsd:sequence>
    <xsd:element name="pPr" type="CT_PPr" minOccurs="0"/>
    <xsd:group   ref="EG_PContent"        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr"      type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"        type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel"      type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidP"        type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidRDefault" type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:group name="EG_PContent">  <!-- denormalized -->
  <xsd:choice>
    <xsd:element name="r"         type="CT_R"/>
    <xsd:element name="hyperlink" type="CT_Hyperlink"/>
    <xsd:element name="fldSimple" type="CT_SimpleField"/>
    <xsd:element name="sdt"       type="CT_SdtRun"/>
    <xsd:element name="customXml" type="CT_CustomXmlRun"/>
    <xsd:element name="smartTag"  type="CT_SmartTagRun"/>
    <xsd:element name="dir"       type="CT_DirContentRun"/>
    <xsd:element name="bdo"       type="CT_BdoContentRun"/>
    <xsd:element name="subDoc"    type="CT_Rel"/>
    <xsd:group ref="EG_RunLevelElts"/>
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_Hyperlink">
  <xsd:group ref="EG_PContent" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:attribute name="tgtFrame"    type="s:ST_String"/>
  <xsd:attribute name="tooltip"     type="s:ST_String"/>
  <xsd:attribute name="docLocation" type="s:ST_String"/>
  <xsd:attribute name="history"     type="s:ST_OnOff"/>
  <xsd:attribute name="anchor"      type="s:ST_String"/>
  <xsd:attribute ref="r:id"/>
</xsd:complexType>

<xsd:group name="EG_RunLevelElts">
  <xsd:choice>
    <xsd:element name="proofErr"                    type="CT_ProofErr"/>
    <xsd:element name="permStart"                   type="CT_PermStart"/>
    <xsd:element name="permEnd"                     type="CT_Perm"/>
    <xsd:element name="bookmarkStart"               type="CT_Bookmark"/>
    <xsd:element name="bookmarkEnd"                 type="CT_MarkupRange"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="moveFromRangeStart"          type="CT_MoveBookmark"/>
    <xsd:element name="moveFromRangeEnd"            type="CT_MarkupRange"/>
    <xsd:element name="moveToRangeStart"            type="CT_MoveBookmark"/>
    <xsd:element name="moveToRangeEnd"              type="CT_MarkupRange"/>
    <xsd:element name="commentRangeStart"           type="CT_MarkupRange"/>
    <xsd:element name="commentRangeEnd"             type="CT_MarkupRange"/>
    <xsd:element name="customXmlInsRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlInsRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlDelRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlDelRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
    <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
    <xsd:element name="ins"                         type="CT_RunTrackChange"/>
    <xsd:element name="del"                         type="CT_RunTrackChange"/>
    <xsd:element name="moveFrom"                    type="CT_RunTrackChange"/>
    <xsd:element name="moveTo"                      type="CT_RunTrackChange"/>
    <xsd:group ref="EG_MathContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_R">
  <xsd:sequence>
    <xsd:group ref="EG_RPr"             minOccurs="0"/>
    <xsd:group ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_RelationshipId">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="ST_String">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

Tab Stops

WordprocessingML allows for custom specification of tab stops at the paragraph level. Tab stop spacing is a subset of paragraph formatting in this system, so will be implemented within the docx.text.parfmt.ParagraphFormatting object. Tab stops will be handled as a List-like TabStops object made up of TabStop objects.

A TabStop object has three properties, alignment, leader, and position. Alignment is a WD_TAB_ALIGNMENT member and position is a Length() object.

Tab stops are always sorted in position order. Alignment defaults to WD_TAB_ALIGNMENT.LEFT, and leader defaults to WD_TAB_LEADER.SPACES.

Tab stops specify how tab characters in a paragraph are rendered. Insertion of tab characters is accomplished using the Run object.

Protocol

Getting and setting tab stops:

```
>>> tab_stops = paragraph.paragraph_format.tab_stops
>>> tab_stops
<docx.text.parfmt.TabStops object at 0x104ea8c30>

>>> tab_stop = tab_stops.add_tab_stop(Inches(2), WD_TAB_ALIGNMENT.LEFT, WD_TAB_LEADER.
↪DOTS)

# add_tab_stop defaults to WD_TAB_ALIGNMENT.LEFT, WD_TAB_LEADER.SPACES

>>> tab_stop = tab_stops.add_tab_stop(Inches(0.5))
>>> tab_stop.alignment
WD_TAB_ALIGNMENT.LEFT
>>> tab_stop.leader
WD_TAB_LEADER.SPACES

# TabStop properties are read/write

>>> tab_stop.position = Inches(2.5)
>>> tab_stop.alignment = WD_TAB_ALIGNMENT.CENTER
>>> tab_stop.leader = WD_TAB_LEADER.DASHES

# Tab stops are sorted into position order as created or modified

>>> [(t.position, t.alignment) for t in tab_stops]
[(914400, WD_TAB_ALIGNMENT.LEFT), (2286000, WD_TAB_ALIGNMENT.CENTER)]

# A tab stop is deleted using del statement

>>> len(tab_stops)
2
>>> del tab_stops[1]
>>> len(tab_stops)
1

# Restore default tabs
```

(continues on next page)

(continued from previous page)

```
>>> tab_stops.clear()
```

Word Behavior

When the w:tabs element is empty or not present, Word uses default tab stops (typically every half inch).

Word resumes using default tab stops following the last specified tab stop.

TabStops must be in position order within the XML. If they are not, the out- of-order tab stop will appear in the ruler and in the properties dialog, but will not actually be used by Word.

XML Semantics

- Both "num" and "list" alignment are a legacy from early versions of Word before hanging indents were available. Both are deprecated.
- "start" alignment is equivalent to "left", and "end" alignment are equivalent to "right". (Confirmed with manually edited XML.)
- A "clear" tab stop is not shown in Word's tab bar and default tab behavior is followed in the document. That is, Word ignores that tab stop specification completely, acting as if it were not there at all. This allows a tab stop inherited from a style, for example, to be ignored.
- The w:pos attribute uses twips rather than EMU.
- The w:tabs element must be removed when empty. If present, it must contain at least one w:tab element.

Specimen XML

```
<w:pPr>
  <w:tabs>
    <w:tab w:val="left" w:leader="dot" w:pos="2880"/>
    <w:tab w:val="decimal" w:pos="6480"/>
  </w:tabs>
</w:pPr>
```

Enumerations

- WdTabAlignment Enumeration on MSDN

Name	XML	Value
wdAlignTabBar	bar	4
wdAlignTabCenter	center	1
wdAlignTabDecimal	decimal	3
wdAlignTabLeft	left	0
wdAlignTabList	list	6
wdAlignTabRight	right	2

Additional Enumeration values not appearing in WdTabAlignment

Name	XML	Value
wdAlignTabClear	clear	101
wdAlignTabEnd	end	102
wdAlignTabNum	num	103
wdAlignTabStart	start	104

- WdTabLeader Enumeration on MSDN

Name	XML	Value
wdTabLeaderDashes	hyphen	2
wdTabLeaderDots	dot	1
wdTabLeaderHeavy	heavy	4
wdTabLeaderLines	underscore	3
wdTabLeaderMiddleDot	middleDot	5
wdTabLeaderSpaces	none	0

MS API Protocol

The MS API defines a TabStops object which is a collection of TabStop objects.

Schema excerpt

```
<xsd:complexType name="CT_PPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="pStyle"              type="CT_String"          minOccurs="0"/>
    <xsd:element name="keepNext"            type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="keepLines"           type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="pageBreakBefore"     type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="framePr"             type="CT_FramePr"         minOccurs="0"/>
    <xsd:element name="widowControl"        type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="numPr"               type="CT_NumPr"           minOccurs="0"/>
    <xsd:element name="suppressLineNumbers" type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="pBdr"                type="CT_PBdr"            minOccurs="0"/>
    <xsd:element name="shd"                 type="CT_Shd"             minOccurs="0"/>
    <xsd:element name="tabs"                type="CT_Tabs"            minOccurs="0"/>
    <xsd:element name="suppressAutoHyphens" type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="kinsoku"             type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="wordWrap"            type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="overflowPunct"       type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="topLinePunct"        type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="autoSpaceDE"         type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="autoSpaceDN"         type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="bidi"                type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="adjustRightInd"      type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="snapToGrid"          type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="spacing"             type="CT_Spacing"         minOccurs="0"/>
    <xsd:element name="ind"                 type="CT_Ind"             minOccurs="0"/>
    <xsd:element name="contextualSpacing"   type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="mirrorIndents"       type="CT_OnOff"           minOccurs="0"/>
    <xsd:element name="suppressOverlap"     type="CT_OnOff"           minOccurs="0"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="jc"                 type="CT_Jc"               minOccurs="0"/>
    <xsd:element name="textDirection"      type="CT_TextDirection"    minOccurs="0"/>
    <xsd:element name="textAlignment"      type="CT_TextAlignment"    minOccurs="0"/>
    <xsd:element name="textboxTightWrap"   type="CT_TextboxTightWrap" minOccurs="0"/>
    <xsd:element name="outlineLvl"         type="CT_DecimalNumber"    minOccurs="0"/>
    <xsd:element name="divId"              type="CT_DecimalNumber"    minOccurs="0"/>
    <xsd:element name="cnfStyle"           type="CT_Cnf"              minOccurs="0"/>
    <xsd:element name="rPr"                type="CT_ParaRPr"          minOccurs="0"/>
    <xsd:element name="sectPr"             type="CT_SectPr"           minOccurs="0"/>
    <xsd:element name="pPrChange"          type="CT_PPrChange"        minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Tabs">
  <xsd:sequence>
    <xsd:element name="tab" type="CT_TabStop" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_TabStop">
  <xsd:attribute name="val"    type="ST_TabJc"              use="required"/>
  <xsd:attribute name="leader" type="ST_TabTlc"             use="optional"/>
  <xsd:attribute name="pos"    type="ST_SignedTwipsMeasure" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_TabJc">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="clear"/>
    <xsd:enumeration value="start"/>
    <xsd:enumeration value="center"/>
    <xsd:enumeration value="end"/>
    <xsd:enumeration value="decimal"/>
    <xsd:enumeration value="bar"/>
    <xsd:enumeration value="num"/>
    <xsd:enumeration value="left"/>
    <xsd:enumeration value="right"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_TabTlc">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="none"/>
    <xsd:enumeration value="dot"/>
    <xsd:enumeration value="hyphen"/>
    <xsd:enumeration value="underscore"/>
    <xsd:enumeration value="heavy"/>
    <xsd:enumeration value="middleDot"/>
  </xsd:restriction>
</xsd:simpleType>
```

Font highlight color

Text in a Word document can be "highlighted" with a number of colors, providing text background color. The visual effect is similar to that produced using a highlighter (often fluorescent yellow) on a printed page.

Protocol

Text is highlighted by assigning a member of *WD_COLOR_INDEX* to *Font.highlight_color*.

```
>>> font = paragraph.add_run().font
>>> font.highlight_color
None
>>> font.highlight_color = WD_COLOR_INDEX.YELLOW
>>> font.highlight_color
YELLOW (7)
>>> font.highlight_color = WD_COLOR_INDEX.TURQUOISE
>>> font.highlight_color
TURQUOISE (3)
>>> font.highlight_color = None
>>> font.highlight_color
None
```

Enumerations

- WdColorIndex Enumeration on MSDN

XML Semantics

Mapping of *WD_COLOR_INDEX* members to *ST_Highlight* values:

```
AUTO = 'default'
BLACK = 'black'
BLUE = 'blue'
BRIGHTGREEN = 'green'
DARKBLUE = 'darkBlue'
DARKRED = 'darkRed'
DARKYELLOW = 'darkYellow'
GRAY25 = 'lightGray'
GRAY50 = 'darkGray'
GREEN = 'darkGreen'
PINK = 'magenta'
RED = 'red'
TEAL = 'darkCyan'
TURQUOISE = 'cyan'
VOILET = 'darkMagenta'
WHITE = 'white'
YELLOW = 'yellow'
```

Specimen XML

Baseline run:

```
<w:r>
  <w:t>Black text on white background</w:t>
</w:r>
```

Blue text, Bright Green Highlight:

```
<w:r>
  <w:rPr>
    <w:highlight w:val="green"/>
  </w:rPr>
  <w:t>Blue text on bright green background</w:t>
</w:r>
```

Red text, Green Highlight:

```
<w:r>
  <w:rPr>
    <w:highlight w:val="darkGreen"/>
  </w:rPr>
  <w:t>Red text on green background</w:t>
</w:r>
```

Schema excerpt

According to the schema, run properties may appear in any order and may appear multiple times each. Not sure what the semantics of that would be or why one would want to do it, but something to note. Word seems to place them in the order below when it writes the file.:

```
<xsd:complexType name="CT_RPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="rStyle"          type="CT_String"/>
      <xsd:element name="rFonts"          type="CT_Fonts"/>
      <xsd:element name="b"               type="CT_OnOff"/>
      <xsd:element name="bCs"             type="CT_OnOff"/>
      <xsd:element name="i"               type="CT_OnOff"/>
      <xsd:element name="iCs"             type="CT_OnOff"/>
      <xsd:element name="caps"            type="CT_OnOff"/>
      <xsd:element name="smallCaps"       type="CT_OnOff"/>
      <xsd:element name="strike"          type="CT_OnOff"/>
      <xsd:element name="dstrike"         type="CT_OnOff"/>
      <xsd:element name="outline"         type="CT_OnOff"/>
      <xsd:element name="shadow"          type="CT_OnOff"/>
      <xsd:element name="emboss"          type="CT_OnOff"/>
      <xsd:element name="imprint"         type="CT_OnOff"/>
      <xsd:element name="noProof"         type="CT_OnOff"/>
      <xsd:element name="snapToGrid"      type="CT_OnOff"/>
      <xsd:element name="vanish"          type="CT_OnOff"/>
      <xsd:element name="webHidden"       type="CT_OnOff"/>
      <xsd:element name="color"           type="CT_Color"/>
      <xsd:element name="spacing"         type="CT_SignedTwipsMeasure"/>
      <xsd:element name="w"               type="CT_TextScale"/>
      <xsd:element name="kern"            type="CT_HpsMeasure"/>
```

(continues on next page)

(continued from previous page)

```
      <xsd:element name="position"        type="CT_SignedHpsMeasure"/>
      <xsd:element name="sz"              type="CT_HpsMeasure"/>
      <xsd:element name="szCs"            type="CT_HpsMeasure"/>
      <xsd:element name="highlight"       type="CT_Highlight"/>
      <xsd:element name="u"               type="CT_Underline"/>
      <xsd:element name="effect"          type="CT_TextEffect"/>
      <xsd:element name="bdr"             type="CT_Border"/>
      <xsd:element name="shd"             type="CT_Shd"/>
      <xsd:element name="fitText"         type="CT_FitText"/>
      <xsd:element name="vertAlign"       type="CT_VerticalAlignRun"/>
      <xsd:element name="rtl"             type="CT_OnOff"/>
      <xsd:element name="cs"              type="CT_OnOff"/>
      <xsd:element name="em"              type="CT_Em"/>
      <xsd:element name="lang"            type="CT_Language"/>
      <xsd:element name="eastAsianLayout" type="CT_EastAsianLayout"/>
      <xsd:element name="specVanish"      type="CT_OnOff"/>
      <xsd:element name="oMath"           type="CT_OnOff"/>
    </xsd:choice>
    <xsd:element name="rPrChange" type="CT_RPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

<!-- complex types -->

<xsd:complexType name="CT_Highlight">
  <xsd:attribute name="val" type="ST_Highlight" use="required"/>
</xsd:complexType>

<!-- simple types -->

<xsd:simpleType name="ST_Highlight">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="default"/>
    <xsd:enumeration value="black"/>
    <xsd:enumeration value="blue"/>
    <xsd:enumeration value="green"/>
    <xsd:enumeration value="darkBlue"/>
    <xsd:enumeration value="darkRed"/>
    <xsd:enumeration value="darkYellow"/>
    <xsd:enumeration value="lightGray"/>
    <xsd:enumeration value="darkGray"/>
    <xsd:enumeration value="darkGreen"/>
    <xsd:enumeration value="magenta"/>
    <xsd:enumeration value="red"/>
    <xsd:enumeration value="darkCyan"/>
    <xsd:enumeration value="cyan"/>
    <xsd:enumeration value="darkMagenta"/>
    <xsd:enumeration value="white"/>
    <xsd:enumeration value="yellow"/>
  </xsd:restriction>
</xsd:simpleType>
```

Paragraph formatting

WordprocessingML supports a variety of paragraph formatting attributes to control layout characteristics such as justification, indentation, line spacing, space before and after, and widow/orphan control.

Alignment (justification)

In Word, each paragraph has an *alignment* attribute that specifies how to justify the lines of the paragraph when the paragraph is laid out on the page. Common values are left, right, centered, and justified.

Protocol

Getting and setting paragraph alignment:

```
>>> paragraph = body.add_paragraph()
>>> paragraph.alignment
None
>>> paragraph.alignment = WD_ALIGN_PARAGRAPH.RIGHT
>>> paragraph.alignment
RIGHT (2)
>>> paragraph.alignment = None
>>> paragraph.alignment
None
```

XML Semantics

If the <w:jc> element is not present on a paragraph, the alignment value for that paragraph is inherited from its style hierarchy. If the element is present, its value overrides any inherited value. From the API, a value of `None` on the `Paragraph.alignment` property corresponds to no <w:jc> element being present. If `None` is assigned to `Paragraph.alignment`, the <w:jc> element is removed.

Paragraph spacing

Spacing between subsequent paragraphs is controlled by the paragraph spacing attributes. Spacing can be applied either before the paragraph, after it, or both. The concept is similar to that of *padding* or *margin* in CSS. WordprocessingML supports paragraph spacing specified as either a length value or as a multiple of the line height; however only a length value is supported via the Word UI. Inter-paragraph spacing "overlaps", such that the rendered spacing between two paragraphs is the maximum of the space after the first paragraph and the space before the second.

Protocol

Getting and setting paragraph spacing:

```
>>> paragraph_format = document.styles['Normal'].paragraph_format
>>> paragraph_format.space_before
None
>>> paragraph_format.space_before = Pt(12)
>>> paragraph_format.space_before.pt
12.0
```

XML Semantics

- Paragraph spacing is specified using the *w:pPr/w:spacing* element, which also controls line spacing. Spacing is specified in twips.
- If the *w:spacing* element is not present, paragraph spacing is inherited from the style hierarchy.
- If not present in the style hierarchy, the paragraph will have no spacing.
- If the *w:spacing* element is present but the specific attribute (e.g. *w:before*) is not, its value is inherited.

Specimen XML

12 pt space before, 0 after:

```
<w:pPr>
  <w:spacing w:before="240" w:after="0"/>
</w:pPr>
```

Line spacing

Line spacing can be specified either as a specific length or as a multiple of the line height (font size). Line spacing is specified by the combination of values in *w:spacing/@w:line* and *w:spacing/@w:lineRule*. The *ParagraphFormat.line_spacing* property determines which method to use based on whether the assigned value is an instance of *Length*.

Protocol

Getting and setting line spacing:

```
>>> paragraph_format.line_spacing, paragraph_format.line_spacing_rule
(None, None)

>>> paragraph_format.line_spacing = Pt(18)
>>> paragraph_format.line_spacing, paragraph_format.line_spacing_rule
(228600, WD_LINE_SPACING.EXACTLY (4))

>>> paragraph_format.line_spacing = 1
>>> paragraph_format.line_spacing, paragraph_format.line_spacing_rule
(152400, WD_LINE_SPACING.SINGLE (0))

>>> paragraph_format.line_spacing = 0.9
>>> paragraph_format.line_spacing, paragraph_format.line_spacing_rule
(137160, WD_LINE_SPACING.MULTIPLE (5))
```

XML Semantics

- Line spacing is specified by the combination of the values in *w:spacing/@w:line* and *w:spacing/@w:lineRule*.
- *w:spacing/@w:line* is specified in twips. If *@w:lineRule* is 'auto' (or missing), *@w:line* is interpreted as 240ths of a line. For all other values of *@w:lineRule*, the value of *@w:line* is interpreted as a specific length in twips.
- If the *w:spacing* element is not present, line spacing is inherited.
- If *@w:line* is not present, line spacing is inherited.
- If not present, *@w:lineRule* defaults to 'auto'.

- If not present in the style hierarchy, line spacing defaults to single spaced.
- The 'atLeast' value for *@w:lineRule* indicates the line spacing will be *@w:line* twips or single spaced, whichever is greater.

Specimen XML

14 points:

```
<w:pPr>
  <w:spacing w:line="280"/>
</w:pPr>
```

double-spaced:

```
<w:pPr>
  <w:spacing w:line="480" w:lineRule="exact"/>
</w:pPr>
```

Indentation

Paragraph indentation is specified using the *w:pPr/w:ind* element. Left, right, first line, and hanging indent can be specified. Indentation can be specified as a length or in hundredths of a character width. Only length is supported by `python-docx`. Both first line indent and hanging indent are specified using the *ParagraphFormat.first_line_indent* property. Assigning a positive value produces an indented first line. A negative value produces a hanging indent.

Protocol

Getting and setting indentation:

```
>>> paragraph_format.left_indent
None
>>> paragraph_format.right_indent
None
>>> paragraph_format.first_line_indent
None

>>> paragraph_format.left_indent = Pt(36)
>>> paragraph_format.left_indent.pt
36.0

>>> paragraph_format.right_indent = Inches(0.25)
>>> paragraph_format.right_indent.pt
18.0

>>> paragraph_format.first_line_indent = Pt(-18)
>>> paragraph_format.first_line_indent.pt
-18.0
```

XML Semantics

- Indentation is specified by *w:ind/@w:start*, *w:ind/@w:end*, *w:ind/@w:firstLine*, and *w:ind/@w:hanging*.
- *w:firstLine* and *w:hanging* are mutually exclusive, if both are specified, *w:firstLine* is ignored.
- All four attributes are specified in twips.
- *w:start* controls left indent for a left-to-right paragraph or right indent for a right-to-left paragraph. *w:end* controls the other side. If mirrorIndents is specified, *w:start* controls the inside margin and *w:end* the outside. Negative values are permitted and cause the text to move past the text margin.
- If *w:ind* is not present, indentation is inherited.
- Any omitted attributes are inherited.
- If not present in the style hierarchy, indentation values default to zero.

Specimen XML

1 inch left, 0.5 inch (additional) first line, 0.5 inch right:

```
<w:pPr>
  <w:ind w:start="1440" w:end="720" w:firstLine="720"/>
</w:pPr>
```

0.5 inch left, 0.5 inch hanging indent:

```
<w:pPr>
  <w:ind w:start="720" w:hanging="720"/>
</w:pPr>
```

Page placement

There are a handful of page placement properties that control such things as keeping the lines of a paragraph together on the same page, keeing a paragraph (such as a heading) on the same page as the subsequent paragraph, and placing the paragraph at the top of a new page. Each of these are tri-state boolean properties where `None` indicates "inherit".

Protocol

Getting and setting indentation:

```
>>> paragraph_format.keep_with_next
None
>>> paragraph_format.keep_together
None
>>> paragraph_format.page_break_before
None
>>> paragraph_format.widow_control
None

>>> paragraph_format.keep_with_next = True
>>> paragraph_format.keep_with_next
True

>>> paragraph_format.keep_together = False
```

(continues on next page)

(continued from previous page)

```
>>> paragraph_format.keep_together
False

>>> paragraph_format.page_break_before = True
>>> paragraph_format.widow_control = None
```

XML Semantics

- All four elements have "On/Off" semantics.
- If not present, their value is inherited.
- If not present in the style hierarchy, values default to False.

Specimen XML

keep with next, keep together, no page break before, and widow/orphan control:

```
<w:pPr>
  <w:keepNext/>
  <w:keepLines/>
  <w:pageBreakBefore w:val="0"/>
  <w:widowControl/>
</w:pPr>
```

Enumerations

- *WD_LINE_SPACING*
- *WD_PARAGRAPH_ALIGNMENT*

Specimen XML

A paragraph with inherited alignment:

```
<w:p>
  <w:r>
    <w:t>Inherited paragraph alignment.</w:t>
  </w:r>
</w:p>
```

A right-aligned paragraph:

```
<w:p>
  <w:pPr>
    <w:jc w:val="right"/>
  </w:pPr>
  <w:r>
    <w:t>Right-aligned paragraph.</w:t>
  </w:r>
</w:p>
```

Schema excerpt

```
<xsd:complexType name="CT_PPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="pStyle"              type="CT_String"           minOccurs="0"/>
    <xsd:element name="keepNext"            type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="keepLines"           type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="pageBreakBefore"     type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="framePr"             type="CT_FramePr"          minOccurs="0"/>
    <xsd:element name="widowControl"        type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="numPr"               type="CT_NumPr"            minOccurs="0"/>
    <xsd:element name="suppressLineNumbers" type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="pBdr"                type="CT_PBdr"             minOccurs="0"/>
    <xsd:element name="shd"                 type="CT_Shd"              minOccurs="0"/>
    <xsd:element name="tabs"                type="CT_Tabs"             minOccurs="0"/>
    <xsd:element name="suppressAutoHyphens" type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="kinsoku"             type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="wordWrap"            type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="overflowPunct"       type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="topLinePunct"        type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="autoSpaceDE"         type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="autoSpaceDN"         type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="bidi"                type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="adjustRightInd"      type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="snapToGrid"          type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="spacing"             type="CT_Spacing"          minOccurs="0"/>
    <xsd:element name="ind"                 type="CT_Ind"              minOccurs="0"/>
    <xsd:element name="contextualSpacing"   type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="mirrorIndents"       type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="suppressOverlap"     type="CT_OnOff"            minOccurs="0"/>
    <xsd:element name="jc"                  type="CT_Jc"               minOccurs="0"/>
    <xsd:element name="textDirection"       type="CT_TextDirection"    minOccurs="0"/>
    <xsd:element name="textAlignment"       type="CT_TextAlignment"    minOccurs="0"/>
    <xsd:element name="textboxTightWrap"    type="CT_TextboxTightWrap" minOccurs="0"/>
    <xsd:element name="outlineLvl"          type="CT_DecimalNumber"    minOccurs="0"/>
    <xsd:element name="divId"               type="CT_DecimalNumber"    minOccurs="0"/>
    <xsd:element name="cnfStyle"            type="CT_Cnf"              minOccurs="0"/>
    <xsd:element name="rPr"                 type="CT_ParaRPr"          minOccurs="0"/>
    <xsd:element name="sectPr"              type="CT_SectPr"           minOccurs="0"/>
    <xsd:element name="pPrChange"           type="CT_PPrChange"        minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_FramePr">
  <xsd:attribute name="dropCap"    type="ST_DropCap"/>
  <xsd:attribute name="lines"      type="ST_DecimalNumber"/>
  <xsd:attribute name="w"          type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="h"          type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="vSpace"     type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="hSpace"     type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="wrap"       type="ST_Wrap"/>
  <xsd:attribute name="hAnchor"    type="ST_HAnchor"/>
  <xsd:attribute name="vAnchor"    type="ST_VAnchor"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="x"          type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="xAlign"     type="s:ST_XAlign"/>
  <xsd:attribute name="y"          type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="yAlign"     type="s:ST_YAlign"/>
  <xsd:attribute name="hRule"      type="ST_HeightRule"/>
  <xsd:attribute name="anchorLock" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_Ind">
  <xsd:attribute name="start"          type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="startChars"     type="ST_DecimalNumber"/>
  <xsd:attribute name="end"            type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="endChars"       type="ST_DecimalNumber"/>
  <xsd:attribute name="left"           type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="leftChars"      type="ST_DecimalNumber"/>
  <xsd:attribute name="right"          type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="rightChars"     type="ST_DecimalNumber"/>
  <xsd:attribute name="hanging"        type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="hangingChars"   type="ST_DecimalNumber"/>
  <xsd:attribute name="firstLine"      type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="firstLineChars" type="ST_DecimalNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_Jc">
  <xsd:attribute name="val" type="ST_Jc" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_Spacing">
  <xsd:attribute name="before"            type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="beforeLines"       type="ST_DecimalNumber"/>
  <xsd:attribute name="beforeAutospacing" type="s:ST_OnOff"/>
  <xsd:attribute name="after"             type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="afterLines"        type="ST_DecimalNumber"/>
  <xsd:attribute name="afterAutospacing"  type="s:ST_OnOff"/>
  <xsd:attribute name="line"              type="ST_SignedTwipsMeasure"/>
  <xsd:attribute name="lineRule"          type="ST_LineSpacingRule"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_Tabs">
  <xsd:sequence>
    <xsd:element name="tab" type="CT_TabStop" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<!-- simple types -->

<xsd:simpleType name="ST_Jc">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="start"/>
    <xsd:enumeration value="center"/>
    <xsd:enumeration value="end"/>
    <xsd:enumeration value="both"/>
    <xsd:enumeration value="mediumKashida"/>
    <xsd:enumeration value="distribute"/>
    <xsd:enumeration value="numTab"/>
    <xsd:enumeration value="highKashida"/>
    <xsd:enumeration value="lowKashida"/>
    <xsd:enumeration value="thaiDistribute"/>
    <xsd:enumeration value="left"/>
    <xsd:enumeration value="right"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_LineSpacingRule">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="auto"/>  <!-- default -->
    <xsd:enumeration value="exact"/>
    <xsd:enumeration value="atLeast"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>
```

Font

Word supports a rich variety of character formatting. Character formatting can be applied at various levels in the *style hierarchy*. At the lowest level, it can be applied directly to a run of text content. Above that, it can be applied to character, paragraph and table styles. It can also be applied to an abstract numbering definition. At the highest levels it can be applied via a theme or document defaults.

Typeface name

Word allows multiple typefaces to be specified for character content in a single run. This allows different Unicode character ranges such as ASCII and Arabic to be used in a single run, each being rendered in the typeface specified for that range.

Up to eight distinct typefaces may be specified for a font. Four are used to specify a typeface for a distinct code point range. These are:

- *w:ascii* - used for the first 128 Unicode code points
- *w:cs* - used for complex script code points
- *w:eastAsia* - used for East Asian code points
- *w:hAnsi* - standing for *high ANSI*, but effectively the catch-all for any code points not specified by one of the other three.

The other four, *w:asciiTheme*, *w:csTheme*, *w:eastAsiaTheme*, and *w:hAnsiTheme* are used to indirectly specify a theme-defined font. This allows the typeface to be set centrally in the document. These four attributes have lower precedence than the first four, so for example the value of *w:asciiTheme* is ignored if a *w:ascii* attribute is also present.

The typeface name used for a run is specified in the *w:rPr/w:rFonts* element. There are 8 attributes that in combination specify the typeface to be used.

Protocol

Initially, only the base typeface name is supported by the API, using the *name* property. Its value is the that of the *w:rFonts/@w:ascii* attribute or `None` if not present. Assignment to this property sets both the *w:ascii* and the *w:hAnsi* attribute to the assigned string or removes them both if `None` is assigned:

```
>>> font = document.styles['Normal'].font
>>> font.name
None
>>> font.name = 'Arial'
>>> font.name
'Arial'
```

Boolean run properties

Character formatting that is either on or off, such as bold, italic, and small caps. Certain of these properties are *toggle properties* that may cancel each other out if they appear more than once in the style hierarchy. See §17.7.3 for more details on toggle properties. They don't affect the API specified here.

The following run properties are boolean (tri-state) properties:

element	spec	name
<b/>	§17.3.2.1	Bold
<bCs/>	§17.3.2.2	Complex Script Bold
<caps/>	§17.3.2.5	Display All Characters as Capital Letters
<cs/>	§17.3.2.7	Use Complex Script Formatting on Run
<dstrike/>	§17.3.2.9	Double Strikethrough
<emboss/>	§17.3.2.13	Embossing
<i/>	§17.3.2.16	Italics
<iCs/>	§17.3.2.17	Complex Script Italics
<imprint/>	§17.3.2.18	Imprinting
<noProof/>	§17.3.2.21	Do Not Check Spelling or Grammar
<oMath/>	§17.3.2.22	Office Open XML Math
<outline/>	§17.3.2.23	Display Character Outline
<rtl/>	§17.3.2.30	Right To Left Text
<shadow/>	§17.3.2.31	Shadow
<smallCaps/>	§17.3.2.33	Small Caps
<snapToGrid/>	§17.3.2.34	Use Document Grid Settings For Inter- Character Spacing
<specVanish/>	§17.3.2.36	Paragraph Mark is Always Hidden
<strike/>	§17.3.2.37	Single Strikethrough
<vanish/>	§17.3.2.41	Hidden Text
<webHidden/>	§17.3.2.44	Web Hidden Text

Protocol

At the API level, each of the boolean run properties is a read/write 'tri-state' property, having the possible values `True`, `False`, and `None`.

The following interactive session demonstrates the protocol for querying and applying run-level properties:

```
>>> run = p.add_run()
>>> run.bold
None
>>> run.bold = True
>>> run.bold
True
>>> run.bold = False
>>> run.bold
False
>>> run.bold = None
>>> run.bold
None
```

The semantics of the three values are as follows:

value	meaning
True	The effective value of the property is unconditionally *on*. Contrary settings in the style hierarchy have no effect.
False	The effective value of the property is unconditionally *off*. Contrary settings in the style hierarchy have no effect.
None	The element is not present. The effective value is inherited from the style hierarchy. If no value for this property is present in the style hierarchy, the effective value is *off*.

Toggle properties

Certain of the boolean run properties are *toggle properties*. A toggle property is one that behaves like a *toggle* at certain places in the style hierarchy. Toggle here means that setting the property on has the effect of reversing the prior setting rather than unconditionally setting the property on.

This behavior allows these properties to be overridden (turned off) in inheriting styles. For example, consider a character style *emphasized* that sets bold on. Another style, *strong* inherits from *emphasized*, but should display in italic rather than bold. Setting bold off has no effect because it is overridden by the bold in *strong* (I think). Because bold is a toggle property, setting bold on in *emphasized* causes its value to be toggled, to False, achieving the desired effect. See §17.7.3 for more details on toggle properties.

The following run properties are toggle properties:

element	spec	name
<b/>	§17.3.2.1	Bold
<bCs/>	§17.3.2.2	Complex Script Bold
<caps/>	§17.3.2.5	Display All Characters as Capital Letters
<emboss/>	§17.3.2.13	Embossing
<i/>	§17.3.2.16	Italics
<iCs/>	§17.3.2.17	Complex Script Italics
<imprint/>	§17.3.2.18	Imprinting
<outline/>	§17.3.2.23	Display Character Outline
<shadow/>	§17.3.2.31	Shadow
<smallCaps/>	§17.3.2.33	Small Caps
<strike/>	§17.3.2.37	Single Strikethrough
<vanish/>	§17.3.2.41	Hidden Text

Specimen XML

```
<w:r>
  <w:rPr>
    <w:b/>
    <w:i/>
    <w:smallCaps/>
    <w:strike/>
    <w:sz w:val="28"/>
    <w:szCs w:val="28"/>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>bold, italic, small caps, strike, 14 pt, and underline</w:t>
</w:r>
```

Schema excerpt

It appears the run properties may appear in any order and may appear multiple times each. Not sure what the semantics of that would be or why one would want to do it, but something to note. Word seems to place them in the order below when it writes the file.:

```
<xsd:complexType name="CT_RPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
```

(continues on next page)

(continued from previous page)

```
      <xsd:element name="rStyle"          type="CT_String"/>
      <xsd:element name="rFonts"          type="CT_Fonts"/>
      <xsd:element name="b"               type="CT_OnOff"/>
      <xsd:element name="bCs"             type="CT_OnOff"/>
      <xsd:element name="i"               type="CT_OnOff"/>
      <xsd:element name="iCs"             type="CT_OnOff"/>
      <xsd:element name="caps"            type="CT_OnOff"/>
      <xsd:element name="smallCaps"       type="CT_OnOff"/>
      <xsd:element name="strike"          type="CT_OnOff"/>
      <xsd:element name="dstrike"         type="CT_OnOff"/>
      <xsd:element name="outline"         type="CT_OnOff"/>
      <xsd:element name="shadow"          type="CT_OnOff"/>
      <xsd:element name="emboss"          type="CT_OnOff"/>
      <xsd:element name="imprint"         type="CT_OnOff"/>
      <xsd:element name="noProof"         type="CT_OnOff"/>
      <xsd:element name="snapToGrid"      type="CT_OnOff"/>
      <xsd:element name="vanish"          type="CT_OnOff"/>
      <xsd:element name="webHidden"       type="CT_OnOff"/>
      <xsd:element name="color"           type="CT_Color"/>
      <xsd:element name="spacing"         type="CT_SignedTwipsMeasure"/>
      <xsd:element name="w"               type="CT_TextScale"/>
      <xsd:element name="kern"            type="CT_HpsMeasure"/>
      <xsd:element name="position"        type="CT_SignedHpsMeasure"/>
      <xsd:element name="sz"              type="CT_HpsMeasure"/>
      <xsd:element name="szCs"            type="CT_HpsMeasure"/>
      <xsd:element name="highlight"       type="CT_Highlight"/>
      <xsd:element name="u"               type="CT_Underline"/>
      <xsd:element name="effect"          type="CT_TextEffect"/>
      <xsd:element name="bdr"             type="CT_Border"/>
      <xsd:element name="shd"             type="CT_Shd"/>
      <xsd:element name="fitText"         type="CT_FitText"/>
      <xsd:element name="vertAlign"       type="CT_VerticalAlignRun"/>
      <xsd:element name="rtl"             type="CT_OnOff"/>
      <xsd:element name="cs"              type="CT_OnOff"/>
      <xsd:element name="em"              type="CT_Em"/>
      <xsd:element name="lang"            type="CT_Language"/>
      <xsd:element name="eastAsianLayout" type="CT_EastAsianLayout"/>
      <xsd:element name="specVanish"      type="CT_OnOff"/>
      <xsd:element name="oMath"           type="CT_OnOff"/>
    </xsd:choice>
    <xsd:element name="rPrChange" type="CT_RPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="CT_Fonts">
  <xsd:attribute name="hint"          type="ST_Hint"/>
  <xsd:attribute name="ascii"         type="s:ST_String"/>
  <xsd:attribute name="hAnsi"         type="s:ST_String"/>
  <xsd:attribute name="eastAsia"      type="s:ST_String"/>
  <xsd:attribute name="cs"            type="s:ST_String"/>
  <xsd:attribute name="asciiTheme"    type="ST_Theme"/>
  <xsd:attribute name="hAnsiTheme"    type="ST_Theme"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="eastAsiaTheme" type="ST_Theme"/>
  <xsd:attribute name="cstheme"       type="ST_Theme"/>
</xsd:complexType>

<xsd:complexType name="CT_HpsMeasure">
  <xsd:attribute name="val" type="ST_HpsMeasure" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_SignedHpsMeasure">
  <xsd:attribute name="val" type="ST_SignedHpsMeasure" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_Underline">
  <xsd:attribute name="val"        type="ST_Underline"/>
  <xsd:attribute name="color"      type="ST_HexColor"/>
  <xsd:attribute name="themeColor" type="ST_ThemeColor"/>
  <xsd:attribute name="themeTint"  type="ST_UcharHexNumber"/>
  <xsd:attribute name="themeShade" type="ST_UcharHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_VerticalAlignRun">
  <xsd:attribute name="val" type="s:ST_VerticalAlignRun" use="required"/>
</xsd:complexType>

<!-- simple types -->

<xsd:simpleType name="ST_Hint">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="default"/>
    <xsd:enumeration value="eastAsia"/>
    <xsd:enumeration value="cs"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_HpsMeasure">
  <xsd:union memberTypes="s:ST_UnsignedDecimalNumber
                          s:ST_PositiveUniversalMeasure"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
```

(continues on next page)

(continued from previous page)

```
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_PositiveUniversalMeasure">
  <xsd:restriction base="ST_UniversalMeasure">
    <xsd:pattern value="[0-9]+(\.[0-9]+)?(mm|cm|in|pt|pc|pi)"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_SignedHpsMeasure">
  <xsd:union memberTypes="xsd:integer s:ST_UniversalMeasure"/>
</xsd:simpleType>

<xsd:simpleType name="ST_Theme">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="majorEastAsia"/>
    <xsd:enumeration value="majorBidi"/>
    <xsd:enumeration value="majorAscii"/>
    <xsd:enumeration value="majorHAnsi"/>
    <xsd:enumeration value="minorEastAsia"/>
    <xsd:enumeration value="minorBidi"/>
    <xsd:enumeration value="minorAscii"/>
    <xsd:enumeration value="minorHAnsi"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_Underline">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="single"/>
    <xsd:enumeration value="words"/>
    <xsd:enumeration value="double"/>
    <xsd:enumeration value="thick"/>
    <xsd:enumeration value="dotted"/>
    <xsd:enumeration value="dottedHeavy"/>
    <xsd:enumeration value="dash"/>
    <xsd:enumeration value="dashedHeavy"/>
    <xsd:enumeration value="dashLong"/>
    <xsd:enumeration value="dashLongHeavy"/>
    <xsd:enumeration value="dotDash"/>
    <xsd:enumeration value="dashDotHeavy"/>
    <xsd:enumeration value="dotDotDash"/>
    <xsd:enumeration value="dashDotDotHeavy"/>
    <xsd:enumeration value="wave"/>
    <xsd:enumeration value="wavyHeavy"/>
    <xsd:enumeration value="wavyDouble"/>
    <xsd:enumeration value="none"/>
  </xsd:restriction>
</xsd:simpleType>
```

(continues on next page)

(continued from previous page)

```
<xsd:simpleType name="ST_UnsignedDecimalNumber">
  <xsd:restriction base="xsd:unsignedLong"/>
</xsd:simpleType>

<xsd:simpleType name="ST_VerticalAlignRun">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="baseline"/>
    <xsd:enumeration value="superscript"/>
    <xsd:enumeration value="subscript"/>
  </xsd:restriction>
</xsd:simpleType>
```

Font Color

Color, as a topic, extends beyond the *Font* object; font color is just the first place it's come up. Accordingly, it bears a little deeper thought than usual since we'll want to reuse the same objects and protocol to specify color in the other contexts; it makes sense to craft a general solution that will bear the expected reuse.

There are three historical sources to draw from for this API.

1. The *w:rPr/w:color* element. This is used by default when applying color directly to text or when setting the text color of a style. This corresponds to the *Font.Color* property (undocumented, unfortunately). This element supports RGB colors, theme colors, and a tint or shade of a theme color.
2. The *w:rPr/w14:textFill* element. This is used by Word for fancy text like gradient and shadow effects. This corresponds to the *Font.Fill* property.
3. The PowerPoint font color UI. This seems like a reasonable compromise between the prior two, allowing direct-ish access to common color options while holding the door open for the *Font.fill* operations to be added later if required.

Candidate Protocol

docx.text.run.Run has a font property:

```
>>> from docx import Document
>>> from docx.text.run import Font, Run
>>> run = Document().add_paragraph().add_run()
>>> isinstance(run, Run)
True
>>> font = run.font
>>> isinstance(font, Font)
True
```

docx.text.run.Font has a read-only color property, returning a *docx.dml.color.ColorFormat* object:

```
>>> from docx.dml.color import ColorFormat
>>> color = font.color
>>> isinstance(font.color, ColorFormat)
True
>>> font.color = 'anything'
AttributeError: can't set attribute
```

docx.dml.color.ColorFormat has a read-only `type` property and read/write `rgb`, `theme_color`, and `brightness` properties.

`ColorFormat.type` returns one of *MSO_COLOR_TYPE.RGB*, *MSO_COLOR_TYPE.THEME*, *MSO_COLOR_TYPE.AUTO*, or `None`, the latter indicating font has no directly-applied color:

```
>>> font.color.type
None
```

`ColorFormat.rgb` returns an *RGBColor* object when *type* is *MSO_COLOR_TYPE.RGB*. It may also report an RGBColor value when *type* is *MSO_COLOR_TYPE.THEME*, since an RGB color may also be present in that case. According to the spec, the RGB color value is ignored when a theme color is specified, but Word writes the current RGB value of the theme color along with the theme color name (e.g. 'accent1') when assigning a theme color; perhaps as a convenient value for a file browser to use. The value of *.type* must be consulted to determine whether the RGB value is operative or a "best-guess":

```
>>> font.color.type
RGB (1)
>>> font.color.rgb
RGBColor(0x3f, 0x2c, 0x36)
```

Assigning an *RGBColor* value to `ColorFormat.rgb` causes `ColorFormat.type` to become *MSO_COLOR_TYPE.RGB*:

```
>>> font.color.type
None
>>> font.color.rgb = RGBColor(0x3f, 0x2c, 0x36)
>>> font.color.type
RGB (1)
>>> font.color.rgb
RGBColor(0x3f, 0x2c, 0x36)
```

`ColorFormat.theme_color` returns a member of *MSO_THEME_COLOR_INDEX* when *type* is *MSO_COLOR_TYPE.THEME*:

```
>>> font.color.type
THEME (2)
>>> font.color.theme_color
ACCENT_1 (5)
```

Assigning a member of *MSO_THEME_COLOR_INDEX* to `ColorFormat.theme_color` causes `ColorFormat.type` to become *MSO_COLOR_TYPE.THEME*:

```
>>> font.color.type
RGB (1)
>>> font.color.theme_color = MSO_THEME_COLOR.ACCENT_2
>>> font.color.type
THEME (2)
>>> font.color.theme_color
ACCENT_2 (6)
```

The `ColorFormat.brightness` attribute can be used to select a tint or shade of a theme color. Assigning the value 0.1 produces a color 10% brighter (a tint); assigning -0.1 produces a color 10% darker (a shade):

```
>>> font.color.type
None
>>> font.color.brightness
```

(continues on next page)

(continued from previous page)

```
0.0
>>> font.color.brightness = 0.4
ValueError: not a theme color

>>> font.color.theme_color = MSO_THEME_COLOR.TEXT_1
>>> font.color.brightness = 0.4
>>> font.color.brightness
0.4
```

Specimen XML

Baseline paragraph with no font color:

```
<w:p>
  <w:r>
    <w:t>Text with no color.</w:t>
  </w:r>
</w:p>
```

Paragraph with directly-applied RGB color:

```
<w:p>
  <w:pPr>
    <w:rPr>
      <w:color w:val="0000FF"/>
    </w:rPr>
  </w:pPr>
  <w:r>
    <w:rPr>
      <w:color w:val="0000FF"/>
    </w:rPr>
    <w:t>Directly-applied color Blue.</w:t>
  </w:r>
</w:p>
```

Run with directly-applied theme color:

```
<w:r>
  <w:rPr>
    <w:color w:val="4F81BD" w:themeColor="accent1"/>
  </w:rPr>
  <w:t>Theme color Accent 1.</w:t>
</w:r>
```

Run with 40% tint of Text 2 theme color:

```
<w:r>
  <w:rPr>
    <w:color w:val="548DD4" w:themeColor="text2" w:themeTint="99"/>
  </w:rPr>
  <w:t>Theme color with 40% tint.</w:t>
</w:r>
```

Run with 25% shade of Accent 2 theme color:

```
<w:r>
  <w:rPr>
    <w:color w:val="943634" w:themeColor="accent2" w:themeShade="BF"/>
  </w:rPr>
  <w:t>Theme color with 25% shade.</w:t>
</w:r>
```

Schema excerpt

```
<xsd:complexType name="CT_RPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="rStyle"          type="CT_String"/>
      <xsd:element name="rFonts"          type="CT_Fonts"/>
      <xsd:element name="b"               type="CT_OnOff"/>
      <xsd:element name="bCs"             type="CT_OnOff"/>
      <xsd:element name="i"               type="CT_OnOff"/>
      <xsd:element name="iCs"             type="CT_OnOff"/>
      <xsd:element name="caps"            type="CT_OnOff"/>
      <xsd:element name="smallCaps"       type="CT_OnOff"/>
      <xsd:element name="strike"          type="CT_OnOff"/>
      <xsd:element name="dstrike"         type="CT_OnOff"/>
      <xsd:element name="outline"         type="CT_OnOff"/>
      <xsd:element name="shadow"          type="CT_OnOff"/>
      <xsd:element name="emboss"          type="CT_OnOff"/>
      <xsd:element name="imprint"         type="CT_OnOff"/>
      <xsd:element name="noProof"         type="CT_OnOff"/>
      <xsd:element name="snapToGrid"      type="CT_OnOff"/>
      <xsd:element name="vanish"          type="CT_OnOff"/>
      <xsd:element name="webHidden"       type="CT_OnOff"/>
      <xsd:element name="color"           type="CT_Color"/>
      <xsd:element name="spacing"         type="CT_SignedTwipsMeasure"/>
      <xsd:element name="w"               type="CT_TextScale"/>
      <xsd:element name="kern"            type="CT_HpsMeasure"/>
      <xsd:element name="position"        type="CT_SignedHpsMeasure"/>
      <xsd:element name="sz"              type="CT_HpsMeasure"/>
      <xsd:element name="szCs"            type="CT_HpsMeasure"/>
      <xsd:element name="highlight"       type="CT_Highlight"/>
      <xsd:element name="u"               type="CT_Underline"/>
      <xsd:element name="effect"          type="CT_TextEffect"/>
      <xsd:element name="bdr"             type="CT_Border"/>
      <xsd:element name="shd"             type="CT_Shd"/>
      <xsd:element name="fitText"         type="CT_FitText"/>
      <xsd:element name="vertAlign"       type="CT_VerticalAlignRun"/>
      <xsd:element name="rtl"             type="CT_OnOff"/>
      <xsd:element name="cs"              type="CT_OnOff"/>
      <xsd:element name="em"              type="CT_Em"/>
      <xsd:element name="lang"            type="CT_Language"/>
      <xsd:element name="eastAsianLayout" type="CT_EastAsianLayout"/>
      <xsd:element name="specVanish"      type="CT_OnOff"/>
      <xsd:element name="oMath"           type="CT_OnOff"/>
```

(continues on next page)

(continued from previous page)

```
    </xsd:choice>
    <xsd:element name="rPrChange" type="CT_RPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="CT_Color">
  <xsd:attribute name="val"        type="ST_HexColor" use="required"/>
  <xsd:attribute name="themeColor" type="ST_ThemeColor"/>
  <xsd:attribute name="themeTint"  type="ST_UcharHexNumber"/>
  <xsd:attribute name="themeShade" type="ST_UcharHexNumber"/>
</xsd:complexType>

<!-- simple types -->

<xsd:simpleType name="ST_HexColor">
  <xsd:union memberTypes="ST_HexColorAuto s:ST_HexColorRGB"/>
</xsd:simpleType>

<xsd:simpleType name="ST_HexColorAuto">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="auto"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_HexColorRGB">
  <xsd:restriction base="xsd:hexBinary">
    <xsd:length value="3" fixed="true"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_ThemeColor">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="dark1"/>
    <xsd:enumeration value="light1"/>
    <xsd:enumeration value="dark2"/>
    <xsd:enumeration value="light2"/>
    <xsd:enumeration value="accent1"/>
    <xsd:enumeration value="accent2"/>
    <xsd:enumeration value="accent3"/>
    <xsd:enumeration value="accent4"/>
    <xsd:enumeration value="accent5"/>
    <xsd:enumeration value="accent6"/>
    <xsd:enumeration value="hyperlink"/>
    <xsd:enumeration value="followedHyperlink"/>
    <xsd:enumeration value="none"/>
    <xsd:enumeration value="background1"/>
    <xsd:enumeration value="text1"/>
    <xsd:enumeration value="background2"/>
    <xsd:enumeration value="text2"/>
  </xsd:restriction>
</xsd:simpleType>
```

(continues on next page)

(continued from previous page)

```
<xsd:simpleType name="ST_UcharHexNumber">
  <xsd:restriction base="xsd:hexBinary">
    <xsd:length value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```

Underline

Text in a Word document can be underlined in a variety of styles.

Protocol

The call protocol for underline is overloaded such that it works like `.bold` and `.italic` for single underline, but also allows an enumerated value to be assigned to specify more sophisticated underlining such as dashed, wavy, and double-underline:

```
>>> run = paragraph.add_run()
>>> run.underline
None
>>> run.underline = True
>>> run.underline
True
>>> run.underline = WD_UNDERLINE.SINGLE
>>> run.underline
True
>>> run.underline = WD_UNDERLINE.DOUBLE
>>> str(run.underline)
DOUBLE (3)
>>> run.underline = False
>>> run.underline
False
>>> run.underline = WD_UNDERLINE.NONE
>>> run.underline
False
>>> run.underline = None
>>> run.underline
None
```

Enumerations

- WdUnderline Enumeration on MSDN

Specimen XML

Baseline run:

```
<w:r>
  <w:t>underlining determined by inheritance</w:t>
</w:r>
```

Single underline:

```
<w:r>
  <w:rPr>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>single underlined</w:t>
</w:r>
```

Double underline:

```
<w:r>
  <w:rPr>
    <w:u w:val="double"/>
  </w:rPr>
  <w:t>single underlined</w:t>
</w:r>
```

Directly-applied no-underline, overrides inherited value:

```
<w:r>
  <w:rPr>
    <w:u w:val="none"/>
  </w:rPr>
  <w:t>not underlined</w:t>
</w:r>
```

Schema excerpt

Note that the `w:val` attribute on `CT_Underline` is optional. When it is not present no underline appears on the run.

```
<xsd:complexType name="CT_R">  <!-- flattened for readibility -->
  <xsd:sequence>
    <xsd:element name="rPr" type="CT_RPr" minOccurs="0"/>
    <xsd:group   ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_RPr">  <!-- flattened for readibility -->
  <xsd:sequence>
    <xsd:group   ref="EG_RPrBase" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="rPrChange" type="CT_RPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:group name="EG_RPrBase">
  <xsd:choice>
    <xsd:element name="rStyle"          type="CT_String"/>
    <xsd:element name="b"               type="CT_OnOff"/>
    <xsd:element name="i"               type="CT_OnOff"/>
    <xsd:element name="color"           type="CT_Color"/>
    <xsd:element name="sz"              type="CT_HpsMeasure"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="u"               type="CT_Underline"/>
    <!-- 33 others -->
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_Underline">
  <xsd:attribute name="val"         type="ST_Underline"/>
  <xsd:attribute name="color"       type="ST_HexColor"/>
  <xsd:attribute name="themeColor"  type="ST_ThemeColor"/>
  <xsd:attribute name="themeTint"   type="ST_UcharHexNumber"/>
  <xsd:attribute name="themeShade"  type="ST_UcharHexNumber"/>
</xsd:complexType>

<xsd:simpleType name="ST_Underline">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="single"/>
    <xsd:enumeration value="words"/>
    <xsd:enumeration value="double"/>
    <xsd:enumeration value="thick"/>
    <xsd:enumeration value="dotted"/>
    <xsd:enumeration value="dottedHeavy"/>
    <xsd:enumeration value="dash"/>
    <xsd:enumeration value="dashedHeavy"/>
    <xsd:enumeration value="dashLong"/>
    <xsd:enumeration value="dashLongHeavy"/>
    <xsd:enumeration value="dotDash"/>
    <xsd:enumeration value="dashDotHeavy"/>
    <xsd:enumeration value="dotDotDash"/>
    <xsd:enumeration value="dashDotDotHeavy"/>
    <xsd:enumeration value="wave"/>
    <xsd:enumeration value="wavyHeavy"/>
    <xsd:enumeration value="wavyDouble"/>
    <xsd:enumeration value="none"/>
  </xsd:restriction>
</xsd:simpleType>
```

Run-level content

A run is the object most closely associated with inline content; text, pictures, and other items that are flowed between the block-item boundaries within a paragraph.

main content child elements:

- <w:t>
- <w:br>
- <w:drawing>
- <w:tab>
- <w:cr>

Schema excerpt

```
<xsd:complexType name="CT_R">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="rPr" type="CT_RPr" minOccurs="0"/>
    <xsd:group   ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:group name="EG_RunInnerContent">
  <xsd:choice>
    <xsd:element name="t"                    type="CT_Text"/>
    <xsd:element name="br"                   type="CT_Br"/>
    <xsd:element name="cr"                   type="CT_Empty"/>
    <xsd:element name="tab"                  type="CT_Empty"/>
    <xsd:element name="ptab"                 type="CT_PTab"/>
    <xsd:element name="sym"                  type="CT_Sym"/>
    <xsd:element name="noBreakHyphen"        type="CT_Empty"/>
    <xsd:element name="softHyphen"           type="CT_Empty"/>
    <xsd:element name="fldChar"              type="CT_FldChar"/>
    <xsd:element name="drawing"              type="CT_Drawing"/>
    <xsd:element name="object"               type="CT_Object"/>

    <xsd:element name="footnoteReference"    type="CT_FtnEdnRef"/>
    <xsd:element name="footnoteRef"          type="CT_Empty"/>
    <xsd:element name="endnoteReference"     type="CT_FtnEdnRef"/>
    <xsd:element name="endnoteRef"           type="CT_Empty"/>
    <xsd:element name="separator"            type="CT_Empty"/>
    <xsd:element name="continuationSeparator" type="CT_Empty"/>
    <xsd:element name="commentReference"     type="CT_Markup"/>
    <xsd:element name="annotationRef"        type="CT_Empty"/>

    <xsd:element name="contentPart"          type="CT_Rel"/>
    <xsd:element name="delText"              type="CT_Text"/>
    <xsd:element name="instrText"            type="CT_Text"/>
    <xsd:element name="delInstrText"         type="CT_Text"/>

    <xsd:element name="dayShort"             type="CT_Empty"/>
    <xsd:element name="monthShort"           type="CT_Empty"/>
    <xsd:element name="yearShort"            type="CT_Empty"/>
    <xsd:element name="dayLong"              type="CT_Empty"/>
    <xsd:element name="monthLong"            type="CT_Empty"/>
    <xsd:element name="yearLong"             type="CT_Empty"/>

    <xsd:element name="pgNum"                type="CT_Empty"/>
    <xsd:element name="pict"                 type="CT_Picture"/>
    <xsd:element name="ruby"                 type="CT_Ruby"/>
    <xsd:element name="lastRenderedPageBreak" type="CT_Empty"/>
  </xsd:choice>
</xsd:group>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_Empty"/>
```

Breaks

Word supports a variety of breaks that interrupt the flow of text in the document:

- line break
- page break
- column break
- section break (new page, even page, odd page)

In addition, a page break can be forced by formatting a paragraph with the "page break before" setting.

This analysis is limited to line, page, and column breaks. A section break is implemented using a completely different set of elements and is covered separately.

Candidate protocol – run.add_break()

The following interactive session demonstrates the protocol for adding a page break:

```
>>> run = p.add_run()
>>> run.breaks
[]

>>> run.add_break()  # by default adds WD_BREAK.LINE
>>> run.breaks
[<docx.text.Break object at 0x10a7c4f50>]
>>> run.breaks[0].type.__name__
WD_BREAK.LINE

>>> run.add_break(WD_BREAK.LINE)
>>> run.breaks
[<docx.text.Break object at 0x10a7c4f50>, <docx.text.Break object at 0x10a7c4f58>]

>>> run.add_break(WD_BREAK.PAGE)
>>> run.add_break(WD_BREAK.COLUMN)
>>> run.add_break(WD_BREAK.LINE_CLEAR_LEFT)
>>> run.add_break(WD_BREAK.LINE_CLEAR_RIGHT)
>>> run.add_break(WD_BREAK.TEXT_WRAPPING)
```

Enumeration – WD_BREAK_TYPE

- WD_BREAK.LINE
- WD_BREAK.LINE_CLEAR_LEFT
- WD_BREAK.LINE_CLEAR_RIGHT
- WD_BREAK.TEXT_WRAPPING (e.g. LINE_CLEAR_ALL)
- WD_BREAK.PAGE
- WD_BREAK.COLUMN

- WD_BREAK.SECTION_NEXT_PAGE
- WD_BREAK.SECTION_CONTINUOUS
- WD_BREAK.SECTION_EVEN_PAGE
- WD_BREAK.SECTION_ODD_PAGE

Specimen XML

Line break

This XML is produced by Word after inserting a line feed with Shift-Enter:

```
<w:p>
  <w:r>
    <w:t>Text before</w:t>
  </w:r>
  <w:r>
    <w:br/>
    <w:t>and after line break</w:t>
  </w:r>
</w:p>
```

Word loads this more straightforward generation just fine, although it changes it back on next save. I'm not sure of the advantage in creating a fresh run such that the <w:br/> element is the first child:

```
<w:p>
  <w:r>
    <w:t>Text before</w:t>
    <w:br/>
    <w:t>and after line break</w:t>
  </w:r>
</w:p>
```

Page break

Starting with this XML …

```
<w:p>
  <w:r>
    <w:t>Before inserting a page break, the cursor was here }</w:t>
  </w:r>
</w:p>
<w:p>
  <w:r>
    <w:t>This was the following paragraph, the last in the document</w:t>
  </w:r>
</w:p>
```

… this XML is produced by Word on inserting a hard page:

```
<w:p>
  <w:r>
    <w:t>Before inserting a page break, the cursor was here }</w:t>
```

(continues on next page)

(continued from previous page)

```
  </w:r>
</w:p>
<w:p>
  <w:r>
    <w:br w:type="page"/>
  </w:r>
</w:p>
<w:p>
  <w:bookmarkStart w:id="0" w:name="_GoBack"/>
  <w:bookmarkEnd w:id="0"/>
</w:p>
<w:p>
  <w:r>
    <w:t>This was the following paragraph, the last in the document</w:t>
  </w:r>
</w:p>
```

Word loads the following simplified form fine …

```
<w:p>
  <w:r>
    <w:t>Text before an intra-run page break</w:t>
    <w:br w:type="page"/>
    <w:t>Text after an intra-run page break</w:t>
  </w:r>
</w:p>
<w:p>
  <w:r>
    <w:t>following paragraph</w:t>
  </w:r>
</w:p>
```

… although on saving it converts it to this:

```
<w:p>
  <w:r>
    <w:t>Text before an intra-run page break</w:t>
  </w:r>
  <w:r>
    <w:br w:type="page"/>
  </w:r>
  <w:r>
    <w:lastRenderedPageBreak/>
    <w:t>Text after an intra-run page break</w:t>
  </w:r>
</w:p>
<w:p>
  <w:r>
    <w:t>following paragraph</w:t>
  </w:r>
</w:p>
```

Schema excerpt

```
<xsd:complexType name="CT_R">
  <xsd:sequence>
    <xsd:group ref="EG_RPr"             minOccurs="0"/>
    <xsd:group ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:group name="EG_RunInnerContent">
  <xsd:choice>
    <xsd:element name="br"                    type="CT_Br"/>
    <xsd:element name="t"                     type="CT_Text"/>
    <xsd:element name="contentPart"           type="CT_Rel"/>
    <xsd:element name="delText"               type="CT_Text"/>
    <xsd:element name="instrText"             type="CT_Text"/>
    <xsd:element name="delInstrText"          type="CT_Text"/>
    <xsd:element name="noBreakHyphen"         type="CT_Empty"/>
    <xsd:element name="softHyphen"            type="CT_Empty"/>
    <xsd:element name="dayShort"              type="CT_Empty"/>
    <xsd:element name="monthShort"            type="CT_Empty"/>
    <xsd:element name="yearShort"             type="CT_Empty"/>
    <xsd:element name="dayLong"               type="CT_Empty"/>
    <xsd:element name="monthLong"             type="CT_Empty"/>
    <xsd:element name="yearLong"              type="CT_Empty"/>
    <xsd:element name="annotationRef"         type="CT_Empty"/>
    <xsd:element name="footnoteRef"           type="CT_Empty"/>
    <xsd:element name="endnoteRef"            type="CT_Empty"/>
    <xsd:element name="separator"             type="CT_Empty"/>
    <xsd:element name="continuationSeparator" type="CT_Empty"/>
    <xsd:element name="sym"                   type="CT_Sym"/>
    <xsd:element name="pgNum"                 type="CT_Empty"/>
    <xsd:element name="cr"                    type="CT_Empty"/>
    <xsd:element name="tab"                   type="CT_Empty"/>
    <xsd:element name="object"                type="CT_Object"/>
    <xsd:element name="pict"                  type="CT_Picture"/>
    <xsd:element name="fldChar"               type="CT_FldChar"/>
    <xsd:element name="ruby"                  type="CT_Ruby"/>
    <xsd:element name="footnoteReference"     type="CT_FtnEdnRef"/>
    <xsd:element name="endnoteReference"      type="CT_FtnEdnRef"/>
    <xsd:element name="commentReference"      type="CT_Markup"/>
    <xsd:element name="drawing"               type="CT_Drawing"/>
    <xsd:element name="ptab"                  type="CT_PTab"/>
    <xsd:element name="lastRenderedPageBreak" type="CT_Empty"/>
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_Br">
  <xsd:attribute name="type"  type="ST_BrType"/>
  <xsd:attribute name="clear" type="ST_BrClear"/>
```

(continues on next page)

(continued from previous page)

```
</xsd:complexType>

<xsd:simpleType name="ST_BrType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="page"/>
    <xsd:enumeration value="column"/>
    <xsd:enumeration value="textWrapping"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_BrClear">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="none"/>
    <xsd:enumeration value="left"/>
    <xsd:enumeration value="right"/>
    <xsd:enumeration value="all"/>
  </xsd:restriction>
</xsd:simpleType>
```

Resources

- WdBreakType Enumeration on MSDN
- Range.InsertBreak Method (Word) on MSDN

Relevant sections in the ISO Spec

- 17.18.3 ST_BrClear (Line Break Text Wrapping Restart Location)

Table

A table is composed of rows of cells. An implicit sequence of *grid columns* align cells across rows. If there are no merged cells, the grid columns correspond directly to the visual columns.

All table content is contained in its cells.

In addition to this overview, there are the following more specialized feature analyses:

Table Properties

Alignment

Word allows a table to be aligned between the page margins either left, right, or center.

The read/write `Table.alignment` property specifies the alignment for a table:

```
>>> table = document.add_table(rows=2, cols=2)
>>> table.alignment
None
>>> table.alignment = WD_TABLE_ALIGNMENT.RIGHT
>>> table.alignment
RIGHT (2)
```

Autofit

Word has two algorithms for laying out a table, *fixed-width* or *autofit*. The default is autofit. Word will adjust column widths in an autofit table based on cell contents. A fixed-width table retains its column widths regardless of the contents. Either algorithm will adjust column widths proportionately when total table width exceeds page width.

The read/write `Table.allow_autofit` property specifies which algorithm is used:

```
>>> table = document.add_table(rows=2, cols=2)
>>> table.allow_autofit
True
>>> table.allow_autofit = False
>>> table.allow_autofit
False
```

Specimen XML

The following XML represents a 2x2 table:

```
<w:tbl>
  <w:tblPr>
    <w:tblStyle w:val="TableGrid"/>
    <w:tblW w:type="auto" w:w="0"/>
    <w:jc w:val="right"/>
    <w:tblLook w:firstColumn="1" w:firstRow="1" w:lastColumn="0"
               w:lastRow="0" w:noHBand="0" w:noVBand="1" w:val="04A0"/>
  </w:tblPr>
  <w:tblGrid>
    <w:gridCol w:w="4788"/>
    <w:gridCol w:w="4788"/>
  </w:tblGrid>
  <w:tr>
    <w:tc/>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
  </w:tr>
  <w:tr>
    <w:tc>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
    <w:tc>
      <w:tcPr>
```

(continues on next page)

(continued from previous page)

```
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
  </w:tr>
</w:tbl>
```

Layout behavior

Auto-layout causes actual column widths to be both unpredictable and unstable. Changes to the content can make the table layout shift.

Semantics of CT_TblWidth element

e.g. `tcW`:

```
<w:tcW w:w="42.4mm"/>

<w:tcW w:w="1800" w:type="dxa"/>

<w:tcW w:w="20%" w:type="pct"/>

<w:tcW w:w="0" w:type="auto"/>

<w:tcW w:type="nil"/>


ST_MeasurementOrPercent
|
+-- ST_DecimalNumberOrPercent
|   |
|   +-- ST_UnqualifiedPercentage
|   |   |
|   |   +-- XsdInteger e.g. '1440'
|   |
|   +-- ST_Percentage e.g. '-07.43%'
|
+-- ST_UniversalMeasure  e.g. '-04.34mm'
```

Schema Definitions

```
<xsd:complexType name="CT_Tbl">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:group    ref="EG_RangeMarkupElements"        minOccurs="0" maxOccurs="unbounded
↪"/>
    <xsd:element name="tblPr"       type="CT_TblPr"/>
    <xsd:element name="tblGrid"     type="CT_TblGrid"/>
    <xsd:choice                                       minOccurs="0" maxOccurs="unbounded
↪">
      <xsd:element name="tr"        type="CT_Row"/>
      <xsd:element name="customXml" type="CT_CustomXmlRow"/>
```

(continues on next page)

(continued from previous page)

```
      <xsd:element name="sdt"       type="CT_SdtRow"/>
      <xsd:group    ref="EG_RunLevelElts"             minOccurs="0" maxOccurs="unbounded
↪"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_TblPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="tblStyle"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblpPr"              type="CT_TblPPr"        minOccurs="0"/>
    <xsd:element name="tblOverlap"          type="CT_TblOverlap"    minOccurs="0"/>
    <xsd:element name="bidiVisual"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="tblStyleRowBandSize" type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="tblStyleColBandSize" type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="tblW"                type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="jc"                  type="CT_JcTable"       minOccurs="0"/>
    <xsd:element name="tblCellSpacing"      type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="tblInd"              type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="tblBorders"          type="CT_TblBorders"    minOccurs="0"/>
    <xsd:element name="shd"                 type="CT_Shd"           minOccurs="0"/>
    <xsd:element name="tblLayout"           type="CT_TblLayoutType" minOccurs="0"/>
    <xsd:element name="tblCellMar"          type="CT_TblCellMar"    minOccurs="0"/>
    <xsd:element name="tblLook"             type="CT_TblLook"       minOccurs="0"/>
    <xsd:element name="tblCaption"          type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblDescription"      type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblPrChange"         type="CT_TblPrChange"   minOccurs="0"/>
  </xsd:sequence>

<!-- table alignment ---------------------------------- -->

<xsd:complexType name="CT_JcTable">
  <xsd:attribute name="val" type="ST_JcTable" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_JcTable">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="center"/>
    <xsd:enumeration value="end"/>
    <xsd:enumeration value="left"/>
    <xsd:enumeration value="right"/>
    <xsd:enumeration value="start"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- table width ------------------------------------ -->

<xsd:complexType name="CT_TblWidth">
  <xsd:attribute name="w"    type="ST_MeasurementOrPercent"/>
  <xsd:attribute name="type" type="ST_TblWidth"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:simpleType name="ST_MeasurementOrPercent">
  <xsd:union memberTypes="ST_DecimalNumberOrPercent s:ST_UniversalMeasure"/>
</xsd:simpleType>

<xsd:simpleType name="ST_DecimalNumberOrPercent">
  <xsd:union memberTypes="ST_UnqualifiedPercentage s:ST_Percentage"/>
</xsd:simpleType>

<xsd:simpleType name="ST_UniversalMeasure">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="-?[0-9]+(\.[0-9]+)?(mm|cm|in|pt|pc|pi)"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_UnqualifiedPercentage">
  <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="ST_Percentage">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="-?[0-9]+(\.[0-9]+)?%"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_TblWidth">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="nil"/>
    <xsd:enumeration value="pct"/>
    <xsd:enumeration value="dxa"/>
    <xsd:enumeration value="auto"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- table layout ------------------------------------ -->

<xsd:complexType name="CT_TblLayoutType">
  <xsd:attribute name="type" type="ST_TblLayoutType"/>
</xsd:complexType>

<xsd:simpleType name="ST_TblLayoutType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="fixed"/>
    <xsd:enumeration value="autofit"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- table look ------------------------------------- -->

<xsd:complexType name="CT_TblLook">
  <xsd:attribute name="firstRow"    type="s:ST_OnOff"/>
  <xsd:attribute name="lastRow"     type="s:ST_OnOff"/>
  <xsd:attribute name="firstColumn" type="s:ST_OnOff"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="lastColumn"  type="s:ST_OnOff"/>
  <xsd:attribute name="noHBand"     type="s:ST_OnOff"/>
  <xsd:attribute name="noVBand"     type="s:ST_OnOff"/>
  <xsd:attribute name="val"         type="ST_ShortHexNumber"/>
</xsd:complexType>
```

Table Row

A table row has certain properties such as height.

Row.height

Candidate protocol:

```
>>> from docx.enum.table import WD_ROW_HEIGHT
>>> row = table.add_row()
>>> row
<docx.table._Row object at 0x...>
>>> row.height_rule
None
>>> row.height_rule = WD_ROW_HEIGHT.EXACTLY
>>> row.height
None
>>> row.height = Pt(24)
```

MS API

https://msdn.microsoft.com/en-us/library/office/ff193915.aspx

Methods

- Delete()
- SetHeight()
- SetLeftIndent()

Properties

- Alignment
- AllowBreakAcrossPages
- Borders
- Cells
- HeadingFormat
- Height
- HeightRule
- Index
- IsFirst
- IsLast

- LeftIndent
- NestingLevel
- Next
- Previous
- Shading
- SpaceBetweenColumns

WD_ROW_HEIGHT_RULE Enumeration

Alias: WD_ROW_HEIGHT

- wdRowHeightAtLeast (1) The row height is at least a minimum specified value.
- wdRowHeightAuto (0) The row height is adjusted to accommodate the tallest value in the row.
- wdRowHeightExactly (2) The row height is an exact value.

Schema Definitions

```
<xsd:complexType name="CT_Tbl">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:group    ref="EG_RangeMarkupElements"        minOccurs="0" maxOccurs="unbounded
↪"/>
    <xsd:element name="tblPr"       type="CT_TblPr"/>
    <xsd:element name="tblGrid"     type="CT_TblGrid"/>
    <xsd:choice                                       minOccurs="0" maxOccurs="unbounded
↪">
      <xsd:element name="tr"        type="CT_Row"/>
      <xsd:element name="customXml" type="CT_CustomXmlRow"/>
      <xsd:element name="sdt"       type="CT_SdtRow"/>
      <xsd:group    ref="EG_RunLevelElts"             minOccurs="0" maxOccurs="unbounded
↪"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Row">
  <xsd:sequence>
    <xsd:element name="tblPrEx" type="CT_TblPrEx" minOccurs="0"/>
    <xsd:element name="trPr"    type="CT_TrPr"    minOccurs="0"/>
    <xsd:group    ref="EG_ContentCellContent"     minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidTr"  type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_TrPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="cnfStyle"       type="CT_Cnf"           minOccurs="0"/>
    <xsd:element name="divId"          type="CT_DecimalNumber" minOccurs="0"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="gridBefore"     type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="gridAfter"      type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="wBefore"        type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="wAfter"         type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="cantSplit"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="trHeight"       type="CT_Height"        minOccurs="0"/>
    <xsd:element name="tblHeader"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="tblCellSpacing" type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="jc"             type="CT_JcTable"       minOccurs="0"/>
    <xsd:element name="hidden"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="ins"            type="CT_TrackChange"   minOccurs="0"/>
    <xsd:element name="del"            type="CT_TrackChange"   minOccurs="0"/>
    <xsd:element name="trPrChange"     type="CT_TrPrChange"    minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Height">
  <xsd:attribute name="val"   type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="hRule" type="ST_HeightRule"/>
</xsd:complexType>

<xsd:simpleType name="ST_HeightRule">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="auto"/>
    <xsd:enumeration value="exact"/>
    <xsd:enumeration value="atLeast"/>
  </xsd:restriction>
</xsd:simpleType>
```

Table Cell

All content in a table is contained in a cell. A cell also has several properties affecting its size, appearance, and how the content it contains is formatted.

Candidate protocol

Cell.vertical_alignment:

```
>>> from docx.enum.table import WD_CELL_ALIGN_VERTICAL
>>> cell = table.add_row().cells[0]
>>> cell
<docx.table._Cell object at 0x...>
>>> cell.vertical_alignment
None
>>> cell.vertical_alignment = WD_CELL_ALIGN_VERTICAL.CENTER
>>> print(cell.vertical_alignment)
CENTER (1)
```

MS API - Partial Summary

- Merge()
- Split()
- Borders
- BottomPadding (and Left, Right, Top)
- Column
- ColumnIndex
- FitText
- Height
- HeightRule (one of WdRowHeightRule enumeration)
- Preferred Width
- Row
- RowIndex
- Shading
- Tables
- VerticalAlignment
- Width
- WordWrap

WD_ALIGN_VERTICAL Enumeration

wdAlignVerticalBoth (101)
This is an option in the OpenXml spec, but not in Word itself. It's not clear what Word behavior this setting produces. If you find out please let us know and we'll update the documentation. Otherwise, probably best to avoid this option.

wdAlignVerticalBottom (3)
Text is aligned to the bottom border of the cell.

wdAlignVerticalCenter (1)
Text is aligned to the center of the cell.

wdAlignVerticalTop (0)
Text is aligned to the top border of the cell.

Specimen XML

```
<w:tc>
  <w:tcPr>
    <w:tcW w:w="7038" w:type="dxa"/>
    <w:vAlign w:val="bottom"/>
  </w:tcPr>
  <w:p>
    <w:pPr>
      <w:pStyle w:val="ListBullet"/>
    </w:pPr>
```

(continues on next page)

(continued from previous page)

```
    <w:r>
      <w:t>Amy earned her BA in American Studies</w:t>
    </w:r>
  </w:p>
</w:tc>
```

Schema Definitions

```
<xsd:complexType name="CT_Tc">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="tcPr" type="CT_TcPr" minOccurs="0"/>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="p"                          type="CT_P"/>
      <xsd:element name="tbl"                        type="CT_Tbl"/>
      <xsd:element name="customXml"                  type="CT_CustomXmlBlock"/>
      <xsd:element name="sdt"                        type="CT_SdtBlock"/>
      <xsd:element name="proofErr"                   type="CT_ProofErr"/>
      <xsd:element name="permStart"                  type="CT_PermStart"/>
      <xsd:element name="permEnd"                    type="CT_Perm"/>
      <xsd:element name="ins"                        type="CT_RunTrackChange"/>
      <xsd:element name="del"                        type="CT_RunTrackChange"/>
      <xsd:element name="moveFrom"                   type="CT_RunTrackChange"/>
      <xsd:element name="moveTo"                     type="CT_RunTrackChange"/>
      <xsd:element  ref="m:oMathPara"                type="CT_OMathPara"/>
      <xsd:element  ref="m:oMath"                    type="CT_OMath"/>
      <xsd:element name="bookmarkStart"              type="CT_Bookmark"/>
      <xsd:element name="bookmarkEnd"                type="CT_MarkupRange"/>
      <xsd:element name="moveFromRangeStart"         type="CT_MoveBookmark"/>
      <xsd:element name="moveFromRangeEnd"           type="CT_MarkupRange"/>
      <xsd:element name="moveToRangeStart"           type="CT_MoveBookmark"/>
      <xsd:element name="moveToRangeEnd"             type="CT_MarkupRange"/>
      <xsd:element name="commentRangeStart"          type="CT_MarkupRange"/>
      <xsd:element name="commentRangeEnd"            type="CT_MarkupRange"/>
      <xsd:element name="customXmlInsRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlInsRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlDelRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlDelRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveFromRangeEnd"  type="CT_Markup"/>
      <xsd:element name="customXmlMoveToRangeStart"  type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveToRangeEnd"    type="CT_Markup"/>
      <xsd:element name="altChunk"                   type="CT_AltChunk"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="id" type="s:ST_String" use="optional"/>
</xsd:complexType>

<xsd:complexType name="CT_TcPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="cnfStyle"            type="CT_Cnf"          minOccurs="0"/>
    <xsd:element name="tcW"                 type="CT_TblWidth"     minOccurs="0"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="gridSpan"            type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="hMerge"              type="CT_HMerge"        minOccurs="0"/>
    <xsd:element name="vMerge"              type="CT_VMerge"        minOccurs="0"/>
    <xsd:element name="tcBorders"           type="CT_TcBorders"     minOccurs="0"/>
    <xsd:element name="shd"                 type="CT_Shd"           minOccurs="0"/>
    <xsd:element name="noWrap"              type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="tcMar"               type="CT_TcMar"         minOccurs="0"/>
    <xsd:element name="textDirection"       type="CT_TextDirection" minOccurs="0"/>
    <xsd:element name="tcFitText"           type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="vAlign"              type="CT_VerticalJc"    minOccurs="0"/>
    <xsd:element name="hideMark"            type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="headers"             type="CT_Headers"       minOccurs="0"/>
    <xsd:choice                                                     minOccurs="0"/>
      <xsd:element name="cellIns"           type="CT_TrackChange"/>
      <xsd:element name="cellDel"           type="CT_TrackChange"/>
      <xsd:element name="cellMerge"         type="CT_CellMergeTrackChange"/>
    </xsd:choice>
    <xsd:element name="tcPrChange"          type="CT_TcPrChange"    minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_TblWidth">
  <xsd:attribute name="w"    type="ST_MeasurementOrPercent"/>
  <xsd:attribute name="type" type="ST_TblWidth"/>
</xsd:complexType>

<xsd:complexType name="CT_VerticalJc">
  <xsd:attribute name="val" type="ST_VerticalJc" use="required"/>
</xsd:complexType>

<!-- simple types -->

<xsd:simpleType name="ST_DecimalNumberOrPercent">
  <xsd:union memberTypes="ST_UnqualifiedPercentage s:ST_Percentage"/>
</xsd:simpleType>

<xsd:simpleType name="ST_MeasurementOrPercent">
  <xsd:union memberTypes="ST_DecimalNumberOrPercent s:ST_UniversalMeasure"/>
</xsd:simpleType>

<xsd:simpleType name="ST_Percentage">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="-?[0-9]+(\.[0-9]+)?%"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_TblWidth">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="nil"/>
    <xsd:enumeration value="pct"/>
    <xsd:enumeration value="dxa"/>
    <xsd:enumeration value="auto"/>
```

(continues on next page)

(continued from previous page)

```
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_UniversalMeasure">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="-?[0-9]+(\.[0-9]+)?(mm|cm|in|pt|pc|pi)"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_UnqualifiedPercentage">
  <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="ST_VerticalJc">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="top"/>
    <xsd:enumeration value="center"/>
    <xsd:enumeration value="both"/>
    <xsd:enumeration value="bottom"/>
  </xsd:restriction>
</xsd:simpleType>
```

Table - Merge Cells

Word allows contiguous table cells to be merged, such that two or more cells appear to be a single cell. Cells can be merged horizontally (spanning multple columns) or vertically (spanning multiple rows). Cells can also be merged both horizontally and vertically at the same time, producing a cell that spans both rows and columns. Only rectangular ranges of cells can be merged.

Table diagrams

Diagrams like the one below are used to depict tables in this analysis. Horizontal spans are depicted as a continuous horizontal cell without vertical dividers within the span. Vertical spans are depicted as a vertical sequence of cells of the same width where continuation cells are separated by a dashed top border and contain a caret ('^') to symbolize the continuation of the cell above. Cell 'addresses' are depicted at the column and row grid lines. This is conceptually convenient as it reuses the notion of list indices (and slices) and makes certain operations more intuitive to specify. The merged cell *A* below has top, left, bottom, and right values of 0, 0, 2, and 2 respectively:

```
\ 0   1   2   3
0 +---+---+---+
  | A     |   |
1 + - - - +---+
  | ^     |   |
2 +---+---+---+
  |   |   |   |
3 +---+---+---+
```

Basic cell access protocol

There are three ways to access a table cell:

- `Table.cell(row_idx, col_idx)`
- `Row.cells[col_idx]`
- `Column.cells[col_idx]`

Accessing the middle cell of a 3 x 3 table:

```
>>> table = document.add_table(3, 3)
>>> middle_cell = table.cell(1, 1)
>>> table.rows[1].cells[1] == middle_cell
True
>>> table.columns[1].cells[1] == middle_cell
True
```

Basic merge protocol

A merge is specified using two diagonal cells:

```
>>> table = document.add_table(3, 3)
>>> a = table.cell(0, 0)
>>> b = table.cell(1, 1)
>>> A = a.merge(b)
```

```
\ 0   1   2   3
0 +---+---+---+        +---+---+---+
  | a |   |   |        | A     |   |
1 +---+---+---+        + - - - +---+
  |   | b |   |  -->   | ^     |   |
2 +---+---+---+        +---+---+---+
  |   |   |   |        |   |   |   |
3 +---+---+---+        +---+---+---+
```

Accessing a merged cell

A cell is accessed by its "layout grid" position regardless of any spans that may be present. A grid address that falls in a span returns the top-leftmost cell in that span. This means a span has as many addresses as layout grid cells it spans. For example, the merged cell *A* above can be addressed as (0, 0), (0, 1), (1, 0), or (1, 1). This addressing scheme leads to desirable access behaviors when spans are present in the table.

The length of Row.cells is always equal to the number of grid columns, regardless of any spans that are present. Likewise, the length of Column.cells is always equal to the number of table rows, regardless of any spans.

```
>>> table = document.add_table(2, 3)
>>> row = table.rows[0]
>>> len(row.cells)
3
>>> row.cells[0] == row.cells[1]
False

>>> a, b = row.cells[:2]
```

(continues on next page)

(continued from previous page)

```
>>> a.merge(b)

>>> len(row.cells)
3
>>> row.cells[0] == row.cells[1]
True
```

```
\ 0   1   2   3
0 +---+---+---+         +---+---+---+
  | a | b |   |         | A     |   |
1 +---+---+---+  -->    +---+---+---+
  |   |   |   |         |   |   |   |
2 +---+---+---+         +---+---+---+
```

Cell content behavior on merge

When two or more cells are merged, any existing content is concatenated and placed in the resulting merged cell. Content from each original cell is separated from that in the prior original cell by a paragraph mark. An original cell having no content is skipped in the contatenation process. In Python, the procedure would look roughly like this:

```
merged_cell_text = '\n'.join(
    cell.text for cell in original_cells if cell.text
)
```

Merging four cells with content `'a'`, `'b'`, `''`, and `'d'` respectively results in a merged cell having text `'a\nb\nd'`.

Cell size behavior on merge

Cell width and height, if present, are added when cells are merged:

```
>>> a, b = row.cells[:2]
>>> a.width.inches, b.width.inches
(1.0, 1.0)
>>> A = a.merge(b)
>>> A.width.inches
2.0
```

Removing a redundant row or column

Collapsing a column. When all cells in a grid column share the same `w:gridSpan` specification, the spanned columns can be collapsed into a single column by removing the `w:gridSpan` attributes.

Word behavior

- Row and Column access in the MS API just plain breaks when the table is not uniform. *Table.Rows(n)* and *Cell.Row* raise *EnvironmentError* when a table contains a vertical span, and *Table.Columns(n)* and *Cell.Column* unconditionally raise *EnvironmentError* when the table contains a horizontal span. We can do better.
- *Table.Cell(n, m)* works on any non-uniform table, although it uses a *visual grid* that greatly complicates access. It raises an error for *n* or *m* out of visual range, and provides no way other than try/except to determine what that visual range is, since *Row.Count* and *Column.Count* are unavailable.

- In a merge operation, the text of the continuation cells is appended to that of the origin cell as separate paragraph(s).
- If a merge range contains previously merged cells, the range must completely enclose the merged cells.
- Word resizes a table (adds rows) when a cell is referenced by an out-of-bounds row index. If the column identifier is out of bounds, an exception is raised. This behavior will not be implemented in `python-docx`.

Glossary

layout grid
The regular two-dimensional matrix of rows and columns that determines the layout of cells in the table. The grid is primarily defined by the *w:gridCol* elements that define the layout columns for the table. Each row essentially duplicates that layout for an additional row, although its height can differ from other rows. Every actual cell in the table must begin and end on a layout grid "line", whether the cell is merged or not.

span
The single "combined" cell occupying the area of a set of merged cells.

skipped cell
The WordprocessingML (WML) spec allows for 'skipped' cells, where a layout cell location contains no actual cell. I can't find a way to make a table like this using the Word UI and haven't experimented yet to see whether Word will load one constructed by hand in the XML.

uniform table
A table in which each cell corresponds exactly to a layout cell. A uniform table contains no spans or skipped cells.

non-uniform table
A table that contains one or more spans, such that not every cell corresponds to a single layout cell. I suppose it would apply when there was one or more skipped cells too, but in this analysis the term is only used to indicate a table with one or more spans.

uniform cell
A cell not part of a span, occupying a single cell in the layout grid.

origin cell
The top-leftmost cell in a span. Contrast with *continuation cell*.

continuation cell
A layout cell that has been subsumed into a span. A continuation cell is mostly an abstract concept, although a actual *w:tc* element will always exist in the XML for each continuation cell in a vertical span.

Understanding merge XML intuitively

A key insight is that merged cells always look like the diagram below. Horizontal spans are accomplished with a single *w:tc* element in each row, using the *gridSpan* attribute to span additional grid columns. Vertical spans are accomplished with an identical cell in each continuation row, having the same *gridSpan* value, and having vMerge set to *continue* (the default). These vertical continuation cells are depicted in the diagrams below with a dashed top border and a caret ('^') in the left-most grid column to symbolize the continuation of the cell above.:

```
\ 0   1   2   3
0 +---+---+---+
  | A     |   |
1 + - - - +---+
  | ^     |   |
2 +---+---+---+
```

(continues on next page)

(continued from previous page)

```
  |   |   |   |
3 +---+---+---+
```

The table depicted above corresponds to this XML (minimized for clarity):

```
<w:tbl>
  <w:tblGrid>
     <w:gridCol/>
     <w:gridCol/>
     <w:gridCol/>
  </w:tblGrid>
  <w:tr>
     <w:tc>
        <w:tcPr>
           <w:gridSpan w:val="2"/>
           <w:vMerge w:val="restart"/>
        </w:tcPr>
     </w:tc>
     <w:tc/>
  </w:tr>
  <w:tr>
     <w:tc>
        <w:tcPr>
           <w:gridSpan w:val="2"/>
           <w:vMerge/>
        </w:tcPr>
     </w:tc>
     <w:tc/>
  </w:tr>
  <w:tr>
     <w:tc/>
     <w:tc/>
     <w:tc/>
  </w:tr>
</w:tbl>
```

XML Semantics

In a horizontal merge, the `<w:tc w:gridSpan="?">` attribute indicates the number of columns the cell should span. Only the leftmost cell is preserved; the remaining cells in the merge are deleted.

For merging vertically, the `w:vMerge` table cell property of the uppermost cell of the column is set to the value "restart" of type `w:ST_Merge`. The following, lower cells included in the vertical merge must have the `w:vMerge` element present in their cell property (`w:TcPr`) element. Its value should be set to "continue", although it is not necessary to explicitely define it, as it is the default value. A vertical merge ends as soon as a cell `w:TcPr` element lacks the `w:vMerge` element. Similarly to the `w:gridSpan` element, the `w:vMerge` elements are only required when the table's layout is not uniform across its different columns. In the case it is, only the topmost cell is kept; the other lower cells in the merged area are deleted along with their `w:vMerge` elements and the `w:trHeight` table row property is used to specify the combined height of the merged cells.

len() implementation for Row.cells and Column.cells

Each `Row` and `Column` object provides access to the collection of cells it contains. The length of these cell collections is unaffected by the presence of merged cells.

len() always bases its count on the layout grid, as though there were no merged cells.

- `len(Table.columns)` is the number of *w:gridCol* elements, representing the number of grid columns, without regard to the presence of merged cells in the table.
- `len(Table.rows)` is the number of *w:tr* elements, regardless of any merged cells that may be present in the table.
- `len(Row.cells)` is the number of grid columns, regardless of whether any cells in the row are merged.
- `len(Column.cells)` is the number of rows in the table, regardless of whether any cells in the column are merged.

Merging a cell already containing a span

One or both of the "diagonal corner" cells in a merge operation may itself be a merged cell, as long as the specified region is rectangular.

For example:

```
\   0   1   2   3
  +---+---+---+---+       +---+---+---+---+
0 | a     | b |   |       | a\nb\nC   |   |
  + - - - +---+---+       + - - - - - +---+
1 | ^     | C |   |       | ^         |   |
  +---+---+---+---+  -->  +---+---+---+---+
2 |   |   |   |   |       |   |   |   |   |
  +---+---+---+---+       +---+---+---+---+
3 |   |   |   |   |       |   |   |   |   |
  +---+---+---+---+       +---+---+---+---+

  cell(0, 0).merge(cell(1, 2))
```

or:

```
    0   1   2   3   4
  +---+---+---+---+---+       +---+---+---+---+---+
0 | a     | b | c |   |       | abcD          |   |
  + - - - +---+---+---+       + - - - - - - - +---+
1 | ^     | D     |   |       | ^             |   |
  +---+---+---+---+---+  -->  +---+---+---+---+---+
2 |   |   |   |   |   |       |   |   |   |   |   |
  +---+ - - - +---+---+       +---+---+---+---+---+
3 |   |   |   |   |   |       |   |   |   |   |   |
  +---+---+---+---+---+       +---+---+---+---+---+

  cell(0, 0).merge(cell(1, 2))
```

Conversely, either of these two merge operations would be illegal:

```
\ 0   1   2   3   4      0   1   2   3   4
0 +---+---+---+---+    0 +---+---+---+---+
```

(continues on next page)

(continued from previous page)

```
  |   |   | b |   |      |   |   |   |   |
1 +---+---+ - +---+    1 +---+---+---+---+
  |   | a | ^ |   |      |   | a |   |   |
2 +---+---+ - +---+    2 +---+---+---+---+
  |   |   | ^ |   |      | b         |   |
3 +---+---+---+---+    3 +---+---+---+---+
  |   |   |   |   |      |   |   |   |   |
4 +---+---+---+---+    4 +---+---+---+---+

  a.merge(b)
```

General algorithm

- find top-left and target width, height
- for each tr in target height, tc.grow_right(target_width)

Specimen XML

A 3 x 3 table where an area defined by the 2 x 2 topleft cells has been merged, demonstrating the combined use of the `w:gridSpan` as well as the `w:vMerge` elements, as produced by Word:

```
<w:tbl>
  <w:tblPr>
     <w:tblW w:w="0" w:type="auto" />
  </w:tblPr>
  <w:tblGrid>
     <w:gridCol w:w="3192" />
     <w:gridCol w:w="3192" />
     <w:gridCol w:w="3192" />
  </w:tblGrid>
  <w:tr>
     <w:tc>
        <w:tcPr>
           <w:tcW w:w="6384" w:type="dxa" />
           <w:gridSpan w:val="2" />
           <w:vMerge w:val="restart" />
        </w:tcPr>
     </w:tc>
     <w:tc>
        <w:tcPr>
           <w:tcW w:w="3192" w:type="dxa" />
        </w:tcPr>
     </w:tc>
  </w:tr>
  <w:tr>
     <w:tc>
        <w:tcPr>
           <w:tcW w:w="6384" w:type="dxa" />
           <w:gridSpan w:val="2" />
           <w:vMerge />
        </w:tcPr>
```

(continues on next page)

(continued from previous page)

```
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="3192" w:type="dxa" />
      </w:tcPr>
    </w:tc>
  </w:tr>
  <w:tr>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="3192" w:type="dxa" />
      </w:tcPr>
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="3192" w:type="dxa" />
      </w:tcPr>
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="3192" w:type="dxa" />
      </w:tcPr>
    </w:tc>
  </w:tr>
</w:tbl>
```

Schema excerpt

```
<xsd:complexType name="CT_Tc">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="tcPr" type="CT_TcPr" minOccurs="0"/>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="p"                          type="CT_P"/>
      <xsd:element name="tbl"                        type="CT_Tbl"/>
      <xsd:element name="customXml"                  type="CT_CustomXmlBlock"/>
      <xsd:element name="sdt"                        type="CT_SdtBlock"/>
      <xsd:element name="proofErr"                   type="CT_ProofErr"/>
      <xsd:element name="permStart"                  type="CT_PermStart"/>
      <xsd:element name="permEnd"                    type="CT_Perm"/>
      <xsd:element name="ins"                        type="CT_RunTrackChange"/>
      <xsd:element name="del"                        type="CT_RunTrackChange"/>
      <xsd:element name="moveFrom"                   type="CT_RunTrackChange"/>
      <xsd:element name="moveTo"                     type="CT_RunTrackChange"/>
      <xsd:element  ref="m:oMathPara"                type="CT_OMathPara"/>
      <xsd:element  ref="m:oMath"                    type="CT_OMath"/>
      <xsd:element name="bookmarkStart"              type="CT_Bookmark"/>
      <xsd:element name="bookmarkEnd"                type="CT_MarkupRange"/>
      <xsd:element name="moveFromRangeStart"         type="CT_MoveBookmark"/>
      <xsd:element name="moveFromRangeEnd"           type="CT_MarkupRange"/>
      <xsd:element name="moveToRangeStart"           type="CT_MoveBookmark"/>
      <xsd:element name="moveToRangeEnd"             type="CT_MarkupRange"/>
```

(continues on next page)

(continued from previous page)

```
      <xsd:element name="commentRangeStart"           type="CT_MarkupRange"/>
      <xsd:element name="commentRangeEnd"             type="CT_MarkupRange"/>
      <xsd:element name="customXmlInsRangeStart"      type="CT_TrackChange"/>
      <xsd:element name="customXmlInsRangeEnd"        type="CT_Markup"/>
      <xsd:element name="customXmlDelRangeStart"      type="CT_TrackChange"/>
      <xsd:element name="customXmlDelRangeEnd"        type="CT_Markup"/>
      <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
      <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
      <xsd:element name="altChunk"                    type="CT_AltChunk"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="id" type="s:ST_String" use="optional"/>
</xsd:complexType>

<xsd:complexType name="CT_TcPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="cnfStyle"             type="CT_Cnf"           minOccurs="0"/>
    <xsd:element name="tcW"                  type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="gridSpan"             type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="hMerge"               type="CT_HMerge"        minOccurs="0"/>
    <xsd:element name="vMerge"               type="CT_VMerge"        minOccurs="0"/>
    <xsd:element name="tcBorders"            type="CT_TcBorders"     minOccurs="0"/>
    <xsd:element name="shd"                  type="CT_Shd"           minOccurs="0"/>
    <xsd:element name="noWrap"               type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="tcMar"                type="CT_TcMar"         minOccurs="0"/>
    <xsd:element name="textDirection"        type="CT_TextDirection" minOccurs="0"/>
    <xsd:element name="tcFitText"            type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="vAlign"               type="CT_VerticalJc"    minOccurs="0"/>
    <xsd:element name="hideMark"             type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="headers"              type="CT_Headers"       minOccurs="0"/>
    <xsd:choice  minOccurs="0">
      <xsd:element name="cellIns"            type="CT_TrackChange"/>
      <xsd:element name="cellDel"            type="CT_TrackChange"/>
      <xsd:element name="cellMerge"          type="CT_CellMergeTrackChange"/>
    </xsd:choice>
    <xsd:element name="tcPrChange"           type="CT_TcPrChange"    minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_DecimalNumber">
  <xsd:attribute name="val" type="ST_DecimalNumber" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_DecimalNumber">
   <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:complexType name="CT_VMerge">
  <xsd:attribute name="val" type="ST_Merge"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_HMerge">
  <xsd:attribute name="val" type="ST_Merge"/>
</xsd:complexType>

<xsd:simpleType name="ST_Merge">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="continue"/>
    <xsd:enumeration value="restart"/>
  </xsd:restriction>
</xsd:simpleType>
```

Open Issues

- Does Word allow "skipped" cells at the beginning of a row (*w:gridBefore* element)? These are described in the spec, but I don't see a way in the Word UI to create such a table.

Ressources

- Cell.Merge Method on MSDN

Relevant sections in the ISO Spec

- 17.4.17 gridSpan (Grid Columns Spanned by Current Table Cell)
- 17.4.84 vMerge (Vertically Merged Cell)
- 17.18.57 ST_Merge (Merged Cell Type)

Specimen XML

The following XML is generated by Word when inserting a 2x2 table:

```
<w:tbl>
  <w:tblPr>
    <w:tblStyle w:val="TableGrid"/>
    <w:tblW w:type="auto" w:w="0"/>
    <w:tblLook w:firstColumn="1" w:firstRow="1" w:lastColumn="0"
               w:lastRow="0" w:noHBand="0" w:noVBand="1" w:val="04A0"/>
  </w:tblPr>
  <w:tblGrid>
    <w:gridCol w:w="4788"/>
    <w:gridCol w:w="4788"/>
  </w:tblGrid>
  <w:tr>
    <w:tc/>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
    <w:tc>
      <w:tcPr>
```

(continues on next page)

(continued from previous page)

```
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
  </w:tr>
  <w:tr>
    <w:tc>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:type="dxa" w:w="4788"/>
      </w:tcPr>
      <w:p/>
    </w:tc>
  </w:tr>
</w:tbl>
```

Schema Definitions

```
<xsd:complexType name="CT_Tbl">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:group    ref="EG_RangeMarkupElements"        minOccurs="0" maxOccurs="unbounded
↪"/>
    <xsd:element name="tblPr"       type="CT_TblPr"/>
    <xsd:element name="tblGrid"     type="CT_TblGrid"/>
    <xsd:choice                                       minOccurs="0" maxOccurs="unbounded
↪">
      <xsd:element name="tr"        type="CT_Row"/>
      <xsd:element name="customXml" type="CT_CustomXmlRow"/>
      <xsd:element name="sdt"       type="CT_SdtRow"/>
      <xsd:group    ref="EG_RunLevelElts"             minOccurs="0" maxOccurs="unbounded
↪"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_TblPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="tblStyle"           type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblpPr"             type="CT_TblPPr"        minOccurs="0"/>
    <xsd:element name="tblOverlap"         type="CT_TblOverlap"    minOccurs="0"/>
    <xsd:element name="bidiVisual"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="tblStyleRowBandSize" type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="tblStyleColBandSize" type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="tblW"               type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="jc"                 type="CT_JcTable"       minOccurs="0"/>
    <xsd:element name="tblCellSpacing"     type="CT_TblWidth"      minOccurs="0"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="tblInd"              type="CT_TblWidth"      minOccurs="0"/>
    <xsd:element name="tblBorders"          type="CT_TblBorders"    minOccurs="0"/>
    <xsd:element name="shd"                 type="CT_Shd"           minOccurs="0"/>
    <xsd:element name="tblLayout"           type="CT_TblLayoutType" minOccurs="0"/>
    <xsd:element name="tblCellMar"          type="CT_TblCellMar"    minOccurs="0"/>
    <xsd:element name="tblLook"             type="CT_TblLook"       minOccurs="0"/>
    <xsd:element name="tblCaption"          type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblDescription"      type="CT_String"        minOccurs="0"/>
    <xsd:element name="tblPrChange"         type="CT_TblPrChange"   minOccurs="0"/>
  </xsd:sequence>

<xsd:complexType name="CT_TblGrid">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:element name="gridCol"       type="CT_TblGridCol"    minOccurs="0" maxOccurs=
↪"unbounded"/>
    <xsd:element name="tblGridChange" type="CT_TblGridChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_TblGridCol">
  <xsd:attribute name="w" type="s:ST_TwipsMeasure"/>
</xsd:complexType>

<xsd:complexType name="CT_Row">
  <xsd:sequence>
    <xsd:element name="tblPrEx" type="CT_TblPrEx" minOccurs="0"/>
    <xsd:element name="trPr"    type="CT_TrPr"    minOccurs="0"/>
    <xsd:group   ref="EG_ContentCellContent"      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidTr"  type="ST_LongHexNumber"/>
</xsd:complexType>

<!-- component types ------------------------------- -->

<xsd:group name="EG_ContentCellContent">
  <xsd:choice>
    <xsd:element name="tc"        type="CT_Tc"            minOccurs="0" maxOccurs=
↪"unbounded"/>
    <xsd:element name="customXml" type="CT_CustomXmlCell"/>
    <xsd:element name="sdt"       type="CT_SdtCell"/>
    <xsd:group   ref="EG_RunLevelElts"                    minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_RunLevelElts">  <!-- denormalized -->
  <xsd:choice>
    <xsd:element name="proofErr"                  type="CT_ProofErr"/>
    <xsd:element name="permStart"                 type="CT_PermStart"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="permEnd"                      type="CT_Perm"/>
    <xsd:element name="ins"                          type="CT_RunTrackChange"/>
    <xsd:element name="del"                          type="CT_RunTrackChange"/>
    <xsd:element name="moveFrom"                     type="CT_RunTrackChange"/>
    <xsd:element name="moveTo"                       type="CT_RunTrackChange"/>
    <xsd:element  ref="m:oMathPara"                  type="CT_OMathPara"/>
    <xsd:element  ref="m:oMath"                      type="CT_OMath"/>
    <xsd:element name="bookmarkStart"                type="CT_Bookmark"/>
    <xsd:element name="bookmarkEnd"                  type="CT_MarkupRange"/>
    <xsd:element name="moveFromRangeStart"           type="CT_MoveBookmark"/>
    <xsd:element name="moveFromRangeEnd"             type="CT_MarkupRange"/>
    <xsd:element name="moveToRangeStart"             type="CT_MoveBookmark"/>
    <xsd:element name="moveToRangeEnd"               type="CT_MarkupRange"/>
    <xsd:element name="commentRangeStart"            type="CT_MarkupRange"/>
    <xsd:element name="commentRangeEnd"              type="CT_MarkupRange"/>
    <xsd:element name="customXmlInsRangeStart"       type="CT_TrackChange"/>
    <xsd:element name="customXmlInsRangeEnd"         type="CT_Markup"/>
    <xsd:element name="customXmlDelRangeStart"       type="CT_TrackChange"/>
    <xsd:element name="customXmlDelRangeEnd"         type="CT_Markup"/>
    <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
    <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_RangeMarkupElements">
  <xsd:choice>
    <xsd:element name="bookmarkStart"                type="CT_Bookmark"/>
    <xsd:element name="bookmarkEnd"                  type="CT_MarkupRange"/>
    <xsd:element name="moveFromRangeStart"           type="CT_MoveBookmark"/>
    <xsd:element name="moveFromRangeEnd"             type="CT_MarkupRange"/>
    <xsd:element name="moveToRangeStart"             type="CT_MoveBookmark"/>
    <xsd:element name="moveToRangeEnd"               type="CT_MarkupRange"/>
    <xsd:element name="commentRangeStart"            type="CT_MarkupRange"/>
    <xsd:element name="commentRangeEnd"              type="CT_MarkupRange"/>
    <xsd:element name="customXmlInsRangeStart"       type="CT_TrackChange"/>
    <xsd:element name="customXmlInsRangeEnd"         type="CT_Markup"/>
    <xsd:element name="customXmlDelRangeStart"       type="CT_TrackChange"/>
    <xsd:element name="customXmlDelRangeEnd"         type="CT_Markup"/>
    <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
    <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
  </xsd:choice>
</xsd:group>

<xsd:simpleType name="ST_TwipsMeasure">
  <xsd:union memberTypes="ST_UnsignedDecimalNumber ST_PositiveUniversalMeasure"/>
</xsd:simpleType>

<xsd:simpleType name="ST_UnsignedDecimalNumber">
```

(continues on next page)

(continued from previous page)

```
  <xsd:restriction base="xsd:unsignedLong"/>
</xsd:simpleType>

<xsd:simpleType name="ST_PositiveUniversalMeasure">
  <xsd:restriction base="ST_UniversalMeasure">
    <xsd:pattern value="[0-9]+(\.[0-9]+)?(mm|cm|in|pt|pc|pi)"/>
  </xsd:restriction>
</xsd:simpleType>
```

Styles

Styles collection

Candidate protocols

Access:

```
>>> styles = document.styles  # default styles part added if not present
>>> styles
<docx.styles.styles.Styles object at 0x1045dd550>
```

Iteration and length:

```
>>> len(styles)
10
>>> list_styles = [s for s in styles if s.type == WD_STYLE_TYPE.LIST]
>>> len(list_styles)
3
```

Access style by name (or style id):

```
>>> styles['Normal']
<docx.styles.style._ParagraphStyle object at 0x1045dd550>

>>> styles['undefined-style']
KeyError: no style with id or name 'undefined-style'
```

Styles.add_style():

```
>>> style = styles.add_style('Citation', WD_STYLE_TYPE.PARAGRAPH)
>>> style.name
'Citation'
>>> style.type
PARAGRAPH (1)
>>> style.builtin
False
```

Feature Notes

- could add a default builtin style from known specs on first access via WD_BUILTIN_STYLE enumeration:

```
>>> style = document.styles['Heading1']
KeyError: no style with id or name 'Heading1'
```

(continues on next page)

(continued from previous page)

```
>>> style = document.styles[WD_STYLE.HEADING_1]
>>> assert style == document.styles['Heading1']
```

Example XML

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<w:styles
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    mc:Ignorable="w14"
    >

  <w:docDefaults>
    <w:rPrDefault>
      <w:rPr>
        <w:rFonts w:asciiTheme="minorHAnsi" w:eastAsiaTheme="minorEastAsia"
                  w:hAnsiTheme="minorHAnsi" w:cstheme="minorBidi"/>
        <w:sz w:val="24"/>
        <w:szCs w:val="24"/>
        <w:lang w:val="en-US" w:eastAsia="en-US" w:bidi="ar-SA"/>
      </w:rPr>
    </w:rPrDefault>
    <w:pPrDefault/>
  </w:docDefaults>

  <w:latentStyles w:defLockedState="0" w:defUIPriority="99" w:defSemiHidden="1"
                  w:defUnhideWhenUsed="1" w:defQFormat="0" w:count="276">
    <w:lsdException w:name="Normal" w:semiHidden="0" w:uiPriority="0"
                    w:unhideWhenUsed="0" w:qFormat="1"/>
    <w:lsdException w:name="heading 1" w:semiHidden="0" w:uiPriority="9"
                    w:unhideWhenUsed="0" w:qFormat="1"/>
    <w:lsdException w:name="heading 2" w:uiPriority="9" w:qFormat="1"/>
    <w:lsdException w:name="Default Paragraph Font" w:uiPriority="1"/>
  </w:latentStyles>

  <w:style w:type="paragraph" w:default="1" w:styleId="Normal">
    <w:name w:val="Normal"/>
    <w:qFormat/>
  </w:style>
  <w:style w:type="character" w:default="1" w:styleId="DefaultParagraphFont">
    <w:name w:val="Default Paragraph Font"/>
    <w:uiPriority w:val="1"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>
  <w:style w:type="table" w:default="1" w:styleId="TableNormal">
    <w:name w:val="Normal Table"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
```

(continues on next page)

(continued from previous page)

```
    <w:unhideWhenUsed/>
    <w:tblPr>
      <w:tblInd w:w="0" w:type="dxa"/>
      <w:tblCellMar>
        <w:top w:w="0" w:type="dxa"/>
        <w:left w:w="108" w:type="dxa"/>
        <w:bottom w:w="0" w:type="dxa"/>
        <w:right w:w="108" w:type="dxa"/>
      </w:tblCellMar>
    </w:tblPr>
  </w:style>
  <w:style w:type="numbering" w:default="1" w:styleId="NoList">
    <w:name w:val="No List"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>
  <w:style w:type="paragraph" w:customStyle="1" w:styleId="Foobar">
    <w:name w:val="Foobar"/>
    <w:qFormat/>
    <w:rsid w:val="004B54E0"/>
  </w:style>

</w:styles>
```

Schema excerpt

```
<xsd:complexType name="CT_Styles">
  <xsd:sequence>
    <xsd:element name="docDefaults"  type="CT_DocDefaults"  minOccurs="0"/>
    <xsd:element name="latentStyles" type="CT_LatentStyles" minOccurs="0"/>
    <xsd:element name="style"        type="CT_Style"        minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_DocDefaults">
  <xsd:sequence>
    <xsd:element name="rPrDefault" type="CT_RPrDefault" minOccurs="0"/>
    <xsd:element name="pPrDefault" type="CT_PPrDefault" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_LatentStyles">
  <xsd:sequence>
    <xsd:element name="lsdException" type="CT_LsdException" minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="defLockedState"    type="s:ST_OnOff"/>
  <xsd:attribute name="defUIPriority"     type="ST_DecimalNumber"/>
  <xsd:attribute name="defSemiHidden"     type="s:ST_OnOff"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="defUnhideWhenUsed" type="s:ST_OnOff"/>
  <xsd:attribute name="defQFormat"        type="s:ST_OnOff"/>
  <xsd:attribute name="count"             type="ST_DecimalNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_Style">
  <xsd:sequence>
    <xsd:element name="name"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="aliases"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="basedOn"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="next"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="link"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="autoRedefine"    type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="hidden"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="uiPriority"      type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="semiHidden"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="unhideWhenUsed"  type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="qFormat"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="locked"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personal"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalCompose" type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalReply"   type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rsid"            type="CT_LongHexNumber" minOccurs="0"/>
    <xsd:element name="pPr"             type="CT_PPrGeneral"    minOccurs="0"/>
    <xsd:element name="rPr"             type="CT_RPr"           minOccurs="0"/>
    <xsd:element name="tblPr"           type="CT_TblPrBase"     minOccurs="0"/>
    <xsd:element name="trPr"            type="CT_TrPr"          minOccurs="0"/>
    <xsd:element name="tcPr"            type="CT_TcPr"          minOccurs="0"/>
    <xsd:element name="tblStylePr"      type="CT_TblStylePr"    minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type"        type="ST_StyleType"/>
  <xsd:attribute name="styleId"     type="s:ST_String"/>
  <xsd:attribute name="default"     type="s:ST_OnOff"/>
  <xsd:attribute name="customStyle" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_StyleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="paragraph"/>
    <xsd:enumeration value="character"/>
    <xsd:enumeration value="table"/>
    <xsd:enumeration value="numbering"/>
  </xsd:restriction>
</xsd:simpleType>
```

Style objects

A style is one of four types; character, paragraph, table, or numbering. All style objects have behavioral properties and formatting properties. The set of formatting properties varies depending on the style type. In general, formatting properties are inherited along this hierarchy: character -> paragraph -> table. A numbering style has no formatting properties and does not inherit.

Behavioral properties

There are six behavior properties:

hidden
: Style operates to assign formatting properties, but does not appear in the UI under any circumstances. Used for *internal* styles assigned by an application that should not be under the control of an end-user.

priority
: Determines the sort order of the style in sequences presented by the UI.

semi-hidden
: The style is hidden from the so-called "main" user interface. In Word this means the *recommended list* and the style gallery. The style still appears in the *all styles* list.

unhide_when_used
: Flag to the application to set semi-hidden False when the style is next used.

quick_style
: Show the style in the style gallery when it is not hidden.

locked
: Style is hidden and cannot be applied when document formatting protection is active.

hidden

The *hidden* attribute doesn't work on built-in styles and its behavior on custom styles is spotty. Skipping this attribute for now. Will reconsider if someone requests it and can provide a specific use case.

Behavior

Scope. *hidden* doesn't work at all on 'Normal' or 'Heading 1' style. It doesn't work on Salutation either. There is no *w:defHidden* attribute on *w:latentStyles*, lending credence to the hypothesis it is not enabled for built-in styles. *Hypothesis:* Doesn't work on built-in styles.

UI behavior. A custom style having *w:hidden* set `True` is hidden from the gallery and all styles pane lists. It does however appear in the "Current style of selected text" box in the styles pane when the cursor is on a paragraph of that style. The style can be modified by the user from this current style UI element. The user can assign a new style to a paragraph having a hidden style.

priority

The *priority* attribute is the integer primary sort key determining the position of a style in a UI list. The secondary sort is alphabetical by name. Negative values are valid, although not assigned by Word itself and appear to be treated as 0.

Behavior

Default. Word behavior appears to default priority to 0 for custom styles. The spec indicates the effective default value is conceptually infinity, such that the style appears at the end of the styles list, presumably alphabetically among other styles having no priority assigned.

Candidate protocol

```
>>> style = document.styles['Foobar']
>>> style.priority
None
>>> style.priority = 7
>>> style.priority
7
>>> style.priority = -42
>>> style.priority
0
```

semi-hidden

The *w:semiHidden* element specifies visibility of the style in the so-called *main* user interface. For Word, this means the style gallery and the recommended, styles-in-use, and in-current-document lists. The all-styles list and current-style dropdown in the styles pane would then be considered part of an *advanced* user interface.

Behavior

Default. If the *w:semiHidden* element is omitted, its effective value is `False`. There is no inheritance of this value.

Scope. Works on both built-in and custom styles.

Word behavior. Word does not use the *@w:val* attribute. It writes *<w:semiHidden/>* for `True` and omits the element for `False`.

Candidate protocol

```
>>> style = document.styles['Foo']
>>> style.hidden
False
>>> style.hidden = True
>>> style.hidden
True
```

Example XML

style.hidden = True:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:semiHidden/>
</w:style>
```

style.hidden = False:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
</w:style>
```

Alternate constructions should also report the proper value but not be used when writing XML:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:semiHidden w:val="0"/>  <!-- style.hidden is False -->
</w:style>

<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:semiHidden w:val="1"/>  <!-- style.hidden is True -->
</w:style>
```

unhide-when-used

The *w:unhideWhenUsed* element signals an application that this style should be made visibile the next time it is used.

Behavior

Default. If the *w:unhideWhenUsed* element is omitted, its effective value is `False`. There is no inheritance of this value.

Word behavior. The *w:unhideWhenUsed* element is not changed or removed when the style is next used. Only the *w:semiHidden* element is affected, if present. Presumably this is so a style can be re-hidden, to be unhidden on the subsequent use.

Note that this behavior in Word is only triggered by a user actually applying a style. Merely loading a document having the style applied somewhere in its contents does not cause the *w:semiHidden* element to be removed.

Candidate protocol

```
>>> style = document.styles['Foo']
>>> style.unhide_when_used
False
>>> style.unhide_when_used = True
>>> style.unhide_when_used
True
```

Example XML

style.unhide_when_used = True:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:semiHidden/>
  <w:unhideWhenUsed/>
</w:style>
```

style.unhide_when_used = False:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
</w:style>
```

Alternate constructions should also report the proper value but not be used when writing XML:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:unhideWhenUsed w:val="0"/>  <!-- style.unhide_when_used is False -->
</w:style>

<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:unhideWhenUsed w:val="1"/>  <!-- style.unhide_when_used is True -->
</w:style>
```

quick-style

The *w:qFormat* element specifies whether Word should display this style in the style gallery. In order to appear in the gallery, this attribute must be `True` and *hidden* must be `False`.

Behavior

Default. If the *w:qFormat* element is omitted, its effective value is `False`. There is no inheritance of this value.

Word behavior. If *w:qFormat* is `True` and the style is not hidden, it will appear in the gallery in the order specified by *w:uiPriority*.

Candidate protocol

```
>>> style = document.styles['Foo']
>>> style.quick_style
False
>>> style.quick_style = True
>>> style.quick_style
True
```

Example XML

style.quick_style = True:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:qFormat/>
</w:style>
```

style.quick_style = False:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
</w:style>
```

Alternate constructions should also report the proper value but not be used when writing XML:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:qFormat w:val="0"/>  <!-- style.quick_style is False -->
</w:style>

<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:qFormat w:val="1"/>  <!-- style.quick_style is True -->
</w:style>
```

locked

The *w:locked* element specifies whether Word should prevent this style from being applied to content. This behavior is only active if formatting protection is turned on.

Behavior

Default. If the *w:locked* element is omitted, its effective value is `False`. There is no inheritance of this value.

Candidate protocol

```
>>> style = document.styles['Foo']
>>> style.locked
False
>>> style.locked = True
>>> style.locked
True
```

Example XML

style.locked = True:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:locked/>
</w:style>
```

style.locked = False:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
</w:style>
```

Alternate constructions should also report the proper value but not be used when writing XML:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:locked w:val="0"/>  <!-- style.locked is False -->
</w:style>

<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:locked w:val="1"/>  <!-- style.locked is True -->
</w:style>
```

Candidate protocols

Identification:

```
>>> style = document.styles['Body Text']
>>> style.name
'Body Text'
>>> style.style_id
'BodyText'
>>> style.type
WD_STYLE_TYPE.PARAGRAPH (1)
```

delete():

```
>>> len(styles)
6
>>> style.delete()
>>> len(styles)
5
>>> styles['Citation']
KeyError: no style with id or name 'Citation'
```

Style.base_style:

```
>>> style = styles.add_style('Citation', WD_STYLE_TYPE.PARAGRAPH)
>>> style.base_style
None
>>> style.base_style = styles['Normal']
>>> style.base_style
<docx.styles.style._ParagraphStyle object at 0x10a7a9550>
>>> style.base_style.name
'Normal'
```

Example XML

```
<w:styles>

  <!-- ... -->

  <w:style w:type="paragraph" w:default="1" w:styleId="Normal">
    <w:name w:val="Normal"/>
    <w:qFormat/>
  </w:style>
  <w:style w:type="character" w:default="1" w:styleId="DefaultParagraphFont">
    <w:name w:val="Default Paragraph Font"/>
    <w:uiPriority w:val="1"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>
  <w:style w:type="table" w:default="1" w:styleId="TableNormal">
    <w:name w:val="Normal Table"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
    <w:tblPr>
      <w:tblInd w:w="0" w:type="dxa"/>
      <w:tblCellMar>
        <w:top w:w="0" w:type="dxa"/>
        <w:left w:w="108" w:type="dxa"/>
        <w:bottom w:w="0" w:type="dxa"/>
        <w:right w:w="108" w:type="dxa"/>
      </w:tblCellMar>
    </w:tblPr>
  </w:style>
  <w:style w:type="numbering" w:default="1" w:styleId="NoList">
    <w:name w:val="No List"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>

  <w:style w:type="paragraph" w:customStyle="1" w:styleId="Foobar">
    <w:name w:val="Foobar"/>
    <w:basedOn w:val="Normal"/>
    <w:qFormat/>
  </w:style>

</w:styles>
```

Schema excerpt

```
<xsd:complexType name="CT_Style">
  <xsd:sequence>
    <xsd:element name="name"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="aliases"         type="CT_String"        minOccurs="0"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="basedOn"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="next"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="link"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="autoRedefine"    type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="hidden"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="uiPriority"      type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="semiHidden"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="unhideWhenUsed"  type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="qFormat"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="locked"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personal"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalCompose" type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalReply"   type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rsid"            type="CT_LongHexNumber" minOccurs="0"/>
    <xsd:element name="pPr"             type="CT_PPrGeneral"    minOccurs="0"/>
    <xsd:element name="rPr"             type="CT_RPr"           minOccurs="0"/>
    <xsd:element name="tblPr"           type="CT_TblPrBase"     minOccurs="0"/>
    <xsd:element name="trPr"            type="CT_TrPr"          minOccurs="0"/>
    <xsd:element name="tcPr"            type="CT_TcPr"          minOccurs="0"/>
    <xsd:element name="tblStylePr"      type="CT_TblStylePr"    minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type"        type="ST_StyleType"/>
  <xsd:attribute name="styleId"     type="s:ST_String"/>
  <xsd:attribute name="default"     type="s:ST_OnOff"/>
  <xsd:attribute name="customStyle" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_StyleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="paragraph"/>
    <xsd:enumeration value="character"/>
    <xsd:enumeration value="table"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:enumeration value="numbering"/>
  </xsd:restriction>
</xsd:simpleType>
```

Paragraph Style

A paragraph style provides character formatting (font) as well as paragraph formatting properties. Character formatting is inherited from *CharacterStyle* and is predominantly embodied in the `font` property. Likewise, most paragraph-specific properties come from the *ParagraphFormat* object available on the `paragraph_format` property.

A handful of other properties are specific to a paragraph style.

next_paragraph_style

The *next_paragraph_style* property provides access to the style that will automatically be assigned by Word to a new paragraph inserted after a paragraph with this style. This property is most useful for a style that would normally appear only once in a sequence, such as a heading.

The default is to use the same style for an inserted paragraph. This addresses the most common case; for example, a body paragraph having *Body Text* style would normally be followed by a paragraph of the same style.

Expected usage

The priority use case for this property is to provide a working style that can be assigned to a paragraph. The property will always provide a valid paragraph style, defaulting to the current style whenever a more specific one cannot be determined.

While this obscures some specifics of the situation from the API, it addresses the expected most common use case. Developers needing to detect, for example, missing styles can readily use the oxml layer to inspect the XML and further features can be added if those use cases turn out to be more common than expected.

Behavior

Default. The default next paragraph style is the same paragraph style.

The default is used whenever the next paragraph style is not specified or is invalid, including these conditions:

- No *w:next* child element is present
- A style having the styleId specified in *w:next/@w:val* is not present in the document.
- The style specified in *w:next/@w:val* is not a paragraph style.

In all these cases the current style (*self*) is returned.

Example XML

paragraph_style.next_paragraph_style is styles['Bar']:

```
<w:style w:type="paragraph" w:styleId="Foo">
  <w:name w:val="Foo"/>
  <w:next w:val="Bar"/>
</w:style>
```

Semantics. The *w:next* child element is optional.

- When omitted, the next style is the same as the current style.

- If no style with a matching styleId exists, the *w:next* element is ignored and the next style is the same as the current style.
- If a style is found but is of a style type other than paragraph, the *w:next* element is ignored and the next style is the same as the current style.

Candidate protocol

```
>>> styles = document.styles

>>> paragraph_style = styles['Foo']
>>> paragraph_style.next_paragraph_style == paragraph_style
True

>>> paragraph_style.next_paragraph_style = styles['Bar']
>>> paragraph_style.next_paragraph_style == styles['Bar']
True

>>> paragraph_style.next_paragraph_style = None
>>> paragraph_style.next_paragraph_style == paragraph_style
True
```

Schema excerpt

```
<xsd:complexType name="CT_Style">
  <xsd:sequence>
    <xsd:element name="name"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="aliases"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="basedOn"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="next"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="link"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="autoRedefine"    type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="hidden"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="uiPriority"      type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="semiHidden"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="unhideWhenUsed"  type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="qFormat"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="locked"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personal"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalCompose" type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalReply"   type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rsid"            type="CT_LongHexNumber" minOccurs="0"/>
    <xsd:element name="pPr"             type="CT_PPrGeneral"    minOccurs="0"/>
    <xsd:element name="rPr"             type="CT_RPr"           minOccurs="0"/>
    <xsd:element name="tblPr"           type="CT_TblPrBase"     minOccurs="0"/>
    <xsd:element name="trPr"            type="CT_TrPr"          minOccurs="0"/>
    <xsd:element name="tcPr"            type="CT_TcPr"          minOccurs="0"/>
    <xsd:element name="tblStylePr"      type="CT_TblStylePr"    minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type"        type="ST_StyleType"/>
  <xsd:attribute name="styleId"     type="s:ST_String"/>
  <xsd:attribute name="default"     type="s:ST_OnOff"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="customStyle" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>
```

Character Style

Word allows a set of run-level properties to be given a name. The set of properties is called a *character style*. All the settings may be applied to a run in a single action by setting the style of the run.

Protocol

There are two call protocols related to character style: getting and setting the character style of a run, and specifying a style when creating a run.

Get run style:

```
>>> run = p.add_run()

>>> run.style
<docx.styles.style._CharacterStyle object at 0x1053ab5d0>
>>> run.style.name
'Default Paragraph Font'
```

Set run style using character style name:

```
>>> run.style = 'Emphasis'
>>> run.style.name
'Emphasis'
```

Set run style using character style object:

```
>>> run.style = document.styles['Strong']
>>> run.style.name
'Strong'
```

Assigning `None` to *Run.style* causes any applied character style to be removed. A run without a character style inherits the default character style of the document:

```
>>> run.style = None
>>> run.style.name
'Default Paragraph Font'
```

Specifying the style of a run on creation:

```
>>> run = p.add_run(style='Strong')
>>> run.style.name
'Strong'
```

Specimen XML

A baseline regular run:

```
<w:p>
  <w:r>
    <w:t>This is a regular paragraph.</w:t>
  </w:r>
</w:p>
```

Adding *Emphasis* character style:

```
<w:p>
  <w:r>
    <w:rPr>
      <w:rStyle w:val="Emphasis"/>
    </w:rPr>
    <w:t>This paragraph appears in Emphasis character style.</w:t>
  </w:r>
</w:p>
```

A style that appears in the Word user interface (UI) with one or more spaces in its name, such as "Subtle Emphasis", will generally have a style ID with those spaces removed. In this example, "Subtle Emphasis" becomes "SubtleEmphasis":

```
<w:p>
  <w:r>
    <w:rPr>
      <w:rStyle w:val="SubtleEmphasis"/>
    </w:rPr>
    <w:t>a few words in Subtle Emphasis style</w:t>
  </w:r>
</w:p>
```

Schema excerpt

```
<xsd:complexType name="CT_R">  <!-- flattened for readibility -->
  <xsd:sequence>
    <xsd:element name="rPr" type="CT_RPr" minOccurs="0"/>
    <xsd:group   ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_RPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="rStyle"          type="CT_String"/>
      <xsd:element name="rFonts"          type="CT_Fonts"/>
      <xsd:element name="b"               type="CT_OnOff"/>
      <xsd:element name="bCs"             type="CT_OnOff"/>
      <xsd:element name="i"               type="CT_OnOff"/>
```

(continues on next page)

(continued from previous page)

```
      <xsd:element name="iCs"             type="CT_OnOff"/>
      <xsd:element name="caps"            type="CT_OnOff"/>
      <xsd:element name="smallCaps"       type="CT_OnOff"/>
      <xsd:element name="strike"          type="CT_OnOff"/>
      <xsd:element name="dstrike"         type="CT_OnOff"/>
      <xsd:element name="outline"         type="CT_OnOff"/>
      <xsd:element name="shadow"          type="CT_OnOff"/>
      <xsd:element name="emboss"          type="CT_OnOff"/>
      <xsd:element name="imprint"         type="CT_OnOff"/>
      <xsd:element name="noProof"         type="CT_OnOff"/>
      <xsd:element name="snapToGrid"      type="CT_OnOff"/>
      <xsd:element name="vanish"          type="CT_OnOff"/>
      <xsd:element name="webHidden"       type="CT_OnOff"/>
      <xsd:element name="color"           type="CT_Color"/>
      <xsd:element name="spacing"         type="CT_SignedTwipsMeasure"/>
      <xsd:element name="w"               type="CT_TextScale"/>
      <xsd:element name="kern"            type="CT_HpsMeasure"/>
      <xsd:element name="position"        type="CT_SignedHpsMeasure"/>
      <xsd:element name="sz"              type="CT_HpsMeasure"/>
      <xsd:element name="szCs"            type="CT_HpsMeasure"/>
      <xsd:element name="highlight"       type="CT_Highlight"/>
      <xsd:element name="u"               type="CT_Underline"/>
      <xsd:element name="effect"          type="CT_TextEffect"/>
      <xsd:element name="bdr"             type="CT_Border"/>
      <xsd:element name="shd"             type="CT_Shd"/>
      <xsd:element name="fitText"         type="CT_FitText"/>
      <xsd:element name="vertAlign"       type="CT_VerticalAlignRun"/>
      <xsd:element name="rtl"             type="CT_OnOff"/>
      <xsd:element name="cs"              type="CT_OnOff"/>
      <xsd:element name="em"              type="CT_Em"/>
      <xsd:element name="lang"            type="CT_Language"/>
      <xsd:element name="eastAsianLayout" type="CT_EastAsianLayout"/>
      <xsd:element name="specVanish"      type="CT_OnOff"/>
      <xsd:element name="oMath"           type="CT_OnOff"/>
    </xsd:choice>
    <xsd:element name="rPrChange" type="CT_RPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_String">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
```

Latent Styles

Latent style definitions are a "stub" style definition specifying behavioral (UI display) attributes for built-in styles.

Latent style collection

The latent style collection for a document is accessed using the *latent_styles* property on *Styles*:

```
>>> latent_styles = document.styles.latent_styles
>>> latent_styles
<docx.styles.LatentStyles object at 0x1045dd550>
```

Iteration. *LatentStyles* should support iteration of contained *_LatentStyle* objects in document order.

Latent style access. A latent style can be accessed by name using dictionary-style notation.

len(). *LatentStyles* supports `len()`, reporting the number of *_LatentStyle* objects it contains.

`LatentStyles` properties

default_priority

XML semantics. According to ISO 29500, the default value if the *w:defUIPriority* attribute is omitted is 99. 99 is explictly set in the default Word *styles.xml*, so will generally be what one finds.

Protocol:

```
>>> # return None if attribute is omitted
>>> latent_styles.default_priority
None
>>> # but expect is will almost always be explicitly 99
>>> latent_styles.default_priority
99
>>> latent_styles.default_priority = 42
>>> latent_styles.default_priority
42
```

load_count

XML semantics. No default is stated in the spec. Don't allow assignment of `None`.

Protocol:

```
>>> latent_styles.load_count
276
>>> latent_styles.load_count = 242
>>> latent_styles.load_count
242
```

Boolean properties

There are four boolean properties that all share the same protocol:

- default_to_hidden
- default_to_locked

- default_to_quick_style
- default_to_unhide_when_used

XML semantics. Defaults to `False` if the attribute is omitted. However, the attribute should always be written explicitly on update.

Protocol:

```
>>> latent_styles.default_to_hidden
False
>>> latent_styles.default_to_hidden = True
>>> latent_styles.default_to_hidden
True
```

Specimen XML

The *w:latentStyles* element used in the default Word 2011 template:

```
<w:latentStyles w:defLockedState="0" w:defUIPriority="99" w:defSemiHidden="1"
                w:defUnhideWhenUsed="1" w:defQFormat="0" w:count="276">
```

`_LatentStyle` properties

```
>>> latent_style = latent_styles.latent_styles[0]

>>> latent_style.name
'Normal'

>>> latent_style.priority
None
>>> latent_style.priority = 10
>>> latent_style.priority
10

>>> latent_style.locked
None
>>> latent_style.locked = True
>>> latent_style.locked
True

>>> latent_style.quick_style
None
>>> latent_style.quick_style = True
>>> latent_style.quick_style
True
```

Latent style behavior

- A style has two categories of attribute, *behavioral* and *formatting*. Behavioral attributes specify where and when the style should appear in the user interface. Behavioral attributes can be specified for latent styles using the <`w:latentStyles`> element and its <`w:lsdException`> child elements. The 5 behavioral attributes are:
  - locked

  - uiPriority
  - semiHidden
  - unhideWhenUsed
  - qFormat

- **locked**. The *locked* attribute specifies that the style should not appear in any list or the gallery and may not be applied to content. This behavior is only active when restricted formatting is turned on.

  Locking is turned on via the menu: Developer Tab > Protect Document > Formatting Restrictions (Windows only).

- **uiPriority**. The *uiPriority* attribute acts as a sort key for sequencing style names in the user interface. Both the lists in the styles panel and the Style Gallery are sensitive to this setting. Its effective value is 0 if not specified.

- **semiHidden**. The *semiHidden* attribute causes the style to be excluded from the recommended list. The notion of *semi* in this context is that while the style is hidden from the recommended list, it still appears in the "All Styles" list. This attribute is removed on first application of the style if an *unhideWhenUsed* attribute set `True` is also present.

- **unhideWhenUsed**. The *unhideWhenUsed* attribute causes any *semiHidden* attribute to be removed when the style is first applied to content. Word does *not* remove the *semiHidden* attribute just because there exists an object in the document having that style. The *unhideWhenUsed* attribute is not removed along with the *semiHidden* attribute when the style is applied.

  The *semiHidden* and *unhideWhenUsed* attributes operate in combination to produce *hide-until-used* behavior.

  Hypothesis. The persistance of the *unhideWhenUsed* attribute after removing the *semiHidden* attribute on first application of the style is necessary to produce appropriate behavior in style inheritance situations. In that case, the *semiHidden* attribute may be explictly set to `False` to override an inherited value. Or it could allow the *semiHidden* attribute to be re-set to `True` later while preserving the hide-until-used behavior.

- **qFormat**. The *qFormat* attribute specifies whether the style should appear in the Style Gallery when it appears in the recommended list. A style will never appear in the gallery unless it also appears in the recommended list.

- Latent style attributes are only operative for latent styles. Once a style is defined, the attributes of the definition exclusively determine style behavior; no attributes are inherited from its corresponding latent style definition.

Specimen XML

```
<w:latentStyles w:defLockedState="0" w:defUIPriority="99" w:defSemiHidden="1"
                w:defUnhideWhenUsed="1" w:defQFormat="0" w:count="276">
  <w:lsdException w:name="Normal" w:semiHidden="0" w:uiPriority="0"
                  w:unhideWhenUsed="0" w:qFormat="1"/>
  <w:lsdException w:name="heading 1" w:semiHidden="0" w:uiPriority="9"
                  w:unhideWhenUsed="0" w:qFormat="1"/>
  <w:lsdException w:name="caption" w:uiPriority="35" w:qFormat="1"/>
  <w:lsdException w:name="Default Paragraph Font" w:uiPriority="1"/>
  <w:lsdException w:name="Bibliography" w:uiPriority="37"/>
  <w:lsdException w:name="TOC Heading" w:uiPriority="39" w:qFormat="1"/>
</w:latentStyles>
```

Schema excerpt

```
<xsd:complexType name="CT_Styles">
  <xsd:sequence>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="docDefaults"  type="CT_DocDefaults"  minOccurs="0"/>
    <xsd:element name="latentStyles" type="CT_LatentStyles" minOccurs="0"/>
    <xsd:element name="style"        type="CT_Style"        minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_LatentStyles">
  <xsd:sequence>
    <xsd:element name="lsdException" type="CT_LsdException" minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="defLockedState"    type="s:ST_OnOff"/>
  <xsd:attribute name="defUIPriority"     type="ST_DecimalNumber"/>
  <xsd:attribute name="defSemiHidden"     type="s:ST_OnOff"/>
  <xsd:attribute name="defUnhideWhenUsed" type="s:ST_OnOff"/>
  <xsd:attribute name="defQFormat"        type="s:ST_OnOff"/>
  <xsd:attribute name="count"             type="ST_DecimalNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_LsdException">
  <xsd:attribute name="name"           type="s:ST_String"   use="required"/>
  <xsd:attribute name="locked"         type="s:ST_OnOff"/>
  <xsd:attribute name="uiPriority"     type="ST_DecimalNumber"/>
  <xsd:attribute name="semiHidden"     type="s:ST_OnOff"/>
  <xsd:attribute name="unhideWhenUsed" type="s:ST_OnOff"/>
  <xsd:attribute name="qFormat"        type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>

<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>
```

Word supports the definition of *styles* to allow a group of formatting properties to be easily and consistently applied to a paragraph, run, table, or numbering scheme, all at once. The mechanism is similar to how Cascading Style Sheets (CSS) works with HTML.

Styles are defined in the `styles.xml` package part and are keyed to a paragraph, run, or table using the *styleId* string.

Style visual behavior

- **Sort order.** Built-in styles appear in order of the effective value of their *uiPriority* attribute. By default, a custom style will not receive a *uiPriority* attribute, causing its effective value to default to 0. This will generlly place custom styles at the top of the sort order. A set of styles having the same *uiPriority* value will be sub-sorted in alphabetical order.

  If a *uiPriority* attribute is defined for a custom style, that style is interleaved with the built-in styles, according to their *uiPriority* value. The *uiPriority* attribute takes a signed integer, and accepts negative numbers. Note that Word does not allow the use of negative integers via its UI; rather it allows the *uiPriority* number of built-in types to be increased to produce the desired sorting behavior.

- **Identification.** A style is identified by its name, not its styleId attribute. The styleId is used only for internal linking of an object like a paragraph to a style. The styleId may be changed by the application, and in fact is routinely changed by Word on each save to be a transformation of the name.

  Hypothesis. Word calculates the *styleId* by removing all spaces from the style name.

- **List membership.** There are four style list options in the styles panel:

  - *Recommended.* The recommended list contains all latent and defined styles that have *semiHidden* == `False`.
  - *Styles in Use.* The styles-in-use list contains all styles that have been applied to content in the document (implying they are defined) that also have *semiHidden* == `False`.
  - *In Current Document.* The in-current-document list contains all defined styles in the document having *semiHidden* == `False`.
  - *All Styles.* The all-styles list contains all latent and defined styles in the document.

- **Definition of built-in style.** When a built-in style is added to a document (upon first use), the value of each of the *locked*, *uiPriority* and *qFormat* attributes from its latent style definition (the *latentStyles* attributes overridden by those of any *lsdException* element) is used to override the corresponding value in the inserted style definition from their built-in defaults.

- Each built-in style has default attributes that can be revealed by setting the *latentStyles/@count* attribute to 0 and inspecting the style in the style manager. This may include default behavioral properties.

- Anomaly. Style "No Spacing" does not appear in the recommended list even though its behavioral attributes indicate it should. (Google indicates it may be a legacy style from Word 2003).

- Word has 267 built-in styles, listed here: http://www.thedoctools.com/downloads/DocTools_List_Of_Built-in_Style_English_Danish_German_French.pdf

  Note that at least one other sources has the number at 276 rather than 267.

- **Appearance in the Style Gallery.** A style appears in the style gallery when: *semiHidden* == `False` and *qFormat* == `True`

Glossary

built-in style
: One of a set of standard styles known to Word, such as "Heading 1". Built-in styles are presented in Word's style panel whether or not they are actually defined in the styles part.

latent style
: A built-in style having no definition in a particular document is known as a *latent style* in that document.

style definition
: A <`w:style`> element in the styles part that explicitly defines the attributes of a style.

recommended style list
: A list of styles that appears in the styles toolbox or panel when "Recommended" is selected from the "List:" dropdown box.

Word behavior

If no style having an assigned style id is defined in the styles part, the style application has no effect.

Word does not add a formatting definition (`<w:style>` element) for a built-in style until it is used.

Once present in the styles part, Word does not remove a built-in style definition if it is no longer applied to any content. The definition of each of the styles ever used in a document are accumulated in its `styles.xml`.

Related MS API *(partial)*

- Document.Styles
- Styles.Add, .Item, .Count, access by name, e.g. Styles("Foobar")
- Style.BaseStyle
- Style.Builtin
- Style.Delete()
- Style.Description
- Style.Font
- Style.Linked
- Style.LinkStyle
- Style.LinkToListTemplate()
- Style.ListLevelNumber
- Style.ListTemplate
- Style.Locked
- Style.NameLocal
- Style.NameParagraphStyle
- Style.NoSpaceBetweenParagraphsOfSameStyle
- Style.ParagraphFormat
- Style.Priority
- Style.QuickStyle
- Style.Shading
- Style.Table(Style)
- Style.Type
- Style.UnhideWhenUsed
- Style.Visibility

Enumerations

- WdBuiltinStyle

Example XML

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<w:styles
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    mc:Ignorable="w14"
    >
  <w:docDefaults>
    <w:rPrDefault>
      <w:rPr>
        <w:rFonts w:asciiTheme="minorHAnsi" w:eastAsiaTheme="minorEastAsia"
                  w:hAnsiTheme="minorHAnsi" w:cstheme="minorBidi"/>
        <w:sz w:val="24"/>
        <w:szCs w:val="24"/>
        <w:lang w:val="en-US" w:eastAsia="en-US" w:bidi="ar-SA"/>
      </w:rPr>
    </w:rPrDefault>
    <w:pPrDefault/>
  </w:docDefaults>
  <w:latentStyles w:defLockedState="0" w:defUIPriority="99" w:defSemiHidden="1"
                  w:defUnhideWhenUsed="1" w:defQFormat="0" w:count="276">
    <w:lsdException w:name="Normal" w:semiHidden="0" w:uiPriority="0"
                    w:unhideWhenUsed="0" w:qFormat="1"/>
    <w:lsdException w:name="heading 1" w:semiHidden="0" w:uiPriority="9"
                    w:unhideWhenUsed="0" w:qFormat="1"/>
    <w:lsdException w:name="heading 2" w:uiPriority="9" w:qFormat="1"/>
    <w:lsdException w:name="Default Paragraph Font" w:uiPriority="1"/>
  </w:latentStyles>
  <w:style w:type="paragraph" w:default="1" w:styleId="Normal">
    <w:name w:val="Normal"/>
    <w:qFormat/>
  </w:style>
  <w:style w:type="character" w:default="1" w:styleId="DefaultParagraphFont">
    <w:name w:val="Default Paragraph Font"/>
    <w:uiPriority w:val="1"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>
  <w:style w:type="table" w:default="1" w:styleId="TableNormal">
    <w:name w:val="Normal Table"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
    <w:tblPr>
      <w:tblInd w:w="0" w:type="dxa"/>
      <w:tblCellMar>
```

(continues on next page)

(continued from previous page)

```
        <w:top w:w="0" w:type="dxa"/>
        <w:left w:w="108" w:type="dxa"/>
        <w:bottom w:w="0" w:type="dxa"/>
        <w:right w:w="108" w:type="dxa"/>
      </w:tblCellMar>
    </w:tblPr>
  </w:style>
  <w:style w:type="numbering" w:default="1" w:styleId="NoList">
    <w:name w:val="No List"/>
    <w:uiPriority w:val="99"/>
    <w:semiHidden/>
    <w:unhideWhenUsed/>
  </w:style>
  <w:style w:type="paragraph" w:customStyle="1" w:styleId="Foobar">
    <w:name w:val="Foobar"/>
    <w:qFormat/>
    <w:rsid w:val="004B54E0"/>
  </w:style>
</w:styles>
```

Schema excerpt

```
<xsd:complexType name="CT_Styles">
  <xsd:sequence>
    <xsd:element name="docDefaults"  type="CT_DocDefaults"  minOccurs="0"/>
    <xsd:element name="latentStyles" type="CT_LatentStyles" minOccurs="0"/>
    <xsd:element name="style"        type="CT_Style"        minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_DocDefaults">
  <xsd:sequence>
    <xsd:element name="rPrDefault" type="CT_RPrDefault" minOccurs="0"/>
    <xsd:element name="pPrDefault" type="CT_PPrDefault" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_LatentStyles">
  <xsd:sequence>
    <xsd:element name="lsdException" type="CT_LsdException" minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="defLockedState"    type="s:ST_OnOff"/>
  <xsd:attribute name="defUIPriority"     type="ST_DecimalNumber"/>
  <xsd:attribute name="defSemiHidden"     type="s:ST_OnOff"/>
  <xsd:attribute name="defUnhideWhenUsed" type="s:ST_OnOff"/>
  <xsd:attribute name="defQFormat"        type="s:ST_OnOff"/>
  <xsd:attribute name="count"             type="ST_DecimalNumber"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_Style">
  <xsd:sequence>
    <xsd:element name="name"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="aliases"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="basedOn"         type="CT_String"        minOccurs="0"/>
    <xsd:element name="next"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="link"            type="CT_String"        minOccurs="0"/>
    <xsd:element name="autoRedefine"    type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="hidden"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="uiPriority"      type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="semiHidden"      type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="unhideWhenUsed"  type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="qFormat"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="locked"          type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personal"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalCompose" type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="personalReply"   type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rsid"            type="CT_LongHexNumber" minOccurs="0"/>
    <xsd:element name="pPr"             type="CT_PPrGeneral"    minOccurs="0"/>
    <xsd:element name="rPr"             type="CT_RPr"           minOccurs="0"/>
    <xsd:element name="tblPr"           type="CT_TblPrBase"     minOccurs="0"/>
    <xsd:element name="trPr"            type="CT_TrPr"          minOccurs="0"/>
    <xsd:element name="tcPr"            type="CT_TcPr"          minOccurs="0"/>
    <xsd:element name="tblStylePr"      type="CT_TblStylePr"    minOccurs="0" maxOccurs=
↪"unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type"        type="ST_StyleType"/>
  <xsd:attribute name="styleId"     type="s:ST_String"/>
  <xsd:attribute name="default"     type="s:ST_OnOff"/>
  <xsd:attribute name="customStyle" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_LsdException">
  <xsd:attribute name="name"           type="s:ST_String"   use="required"/>
  <xsd:attribute name="locked"         type="s:ST_OnOff"/>
  <xsd:attribute name="uiPriority"     type="ST_DecimalNumber"/>
  <xsd:attribute name="semiHidden"     type="s:ST_OnOff"/>
  <xsd:attribute name="unhideWhenUsed" type="s:ST_OnOff"/>
  <xsd:attribute name="qFormat"        type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_OnOff">
  <xsd:attribute name="val" type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_String">
  <xsd:attribute name="val" type="s:ST_String" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_OnOff">
  <xsd:union memberTypes="xsd:boolean ST_OnOff1"/>
</xsd:simpleType>
```

(continues on next page)

(continued from previous page)

```
<xsd:simpleType name="ST_OnOff1">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="on"/>
    <xsd:enumeration value="off"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ST_StyleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="paragraph"/>
    <xsd:enumeration value="character"/>
    <xsd:enumeration value="table"/>
    <xsd:enumeration value="numbering"/>
  </xsd:restriction>
</xsd:simpleType>
```

Shapes (in general)

A graphical object that appears in a Word document is known as a *shape*. A shape can be *inline* or *floating*. An inline shape appears on a text baseline as though it were a character glyph and affects the line height. A floating shape appears at an arbitrary location on the document and text may wrap around it. Several types of shape can be placed, including a picture, a chart, and a drawing canvas.

The graphical object itself is placed in a container, and it is the container that determines the placement of the graphic. The same graphical object can be placed inline or floating by changing its container. The graphic itself is unaffected.

In addition to this overview, there are the following more specialized feature analyses:

Inline shape

Word allows a graphical object to be placed into a document as an inline object. An inline shape appears as a <w:drawing> element as a child of a <w:r> element.

Candidate protocol – inline shape access

The following interactive session illustrates the protocol for accessing an inline shape:

```
>>> inline_shapes = document.body.inline_shapes
>>> inline_shape = inline_shapes[0]
>>> assert inline_shape.type == MSO_SHAPE_TYPE.PICTURE
```

Resources

- Document Members (Word) on MSDN
- InlineShape Members (Word) on MSDN
- Shape Members (Word) on MSDN

MS API

The Shapes and InlineShapes properties on Document hold references to things like pictures in the MS API.

- Height and Width
- Borders
- Shadow
- Hyperlink
- PictureFormat (providing brightness, color, crop, transparency, contrast)
- ScaleHeight and ScaleWidth
- HasChart
- HasSmartArt
- Type (Chart, LockedCanvas, Picture, SmartArt, etc.)

Spec references

- 17.3.3.9 drawing (DrawingML Object)
- 20.4.2.8 inline (Inline DrawingML Object)
- 20.4.2.7 extent (Drawing Object Size)

Minimal XML

This XML represents my best guess of the minimal inline shape container that Word will load:

```
<w:r>
  <w:drawing>
    <wp:inline>
      <wp:extent cx="914400" cy="914400"/>
      <wp:docPr id="1" name="Picture 1"/>
      <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
        <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">

          <!-- might not have to put anything here for a start -->

        </a:graphicData>
      </a:graphic>
    </wp:inline>
  </w:drawing>
</w:r>
```

Specimen XML

A `CT_Drawing` (`<w:drawing>`) element can appear in a run, as a peer of, for example, a `<w:t>` element. This element contains a DrawingML object. WordprocessingML drawings are discussed in section 20.4 of the ISO/IEC spec.

This XML represents an inline shape inserted inline on a paragraph by itself. The particulars of the graphical object itself are redacted:

```
<w:p>
  <w:r>
    <w:rPr/>
      <w:noProof/>
    </w:rPr>
    <w:drawing>
      <wp:inline distT="0" distB="0" distL="0" distR="0" wp14:anchorId="1BDE1558"␣
↪wp14:editId="31E593BB">
        <wp:extent cx="859536" cy="343814"/>
        <wp:effectExtent l="0" t="0" r="4445" b="12065"/>
        <wp:docPr id="1" name="Picture 1"/>
        <wp:cNvGraphicFramePr>
          <a:graphicFrameLocks xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/
↪main" noChangeAspect="1"/>
        </wp:cNvGraphicFramePr>
        <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
          <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">

            <!-- graphical object, such as pic:pic, goes here -->

          </a:graphicData>
        </a:graphic>
      </wp:inline>
    </w:drawing>
  </w:r>
</w:p>
```

Schema definitions

```
<xsd:complexType name="CT_Drawing">
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element ref="wp:anchor" minOccurs="0"/>
    <xsd:element ref="wp:inline" minOccurs="0"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="CT_Inline">
  <xsd:sequence>
    <xsd:element name="extent"            type="a:CT_PositiveSize2D"/>
    <xsd:element name="effectExtent"      type="CT_EffectExtent"                    ␣
↪minOccurs="0"/>
    <xsd:element name="docPr"             type="a:CT_NonVisualDrawingProps"/>
    <xsd:element name="cNvGraphicFramePr" type="a:CT_NonVisualGraphicFrameProperties"␣
↪minOccurs="0"/>
    <xsd:element name="graphic"           type="CT_GraphicalObject"/>
  </xsd:sequence>
  <xsd:attribute name="distT" type="ST_WrapDistance"/>
  <xsd:attribute name="distB" type="ST_WrapDistance"/>
  <xsd:attribute name="distL" type="ST_WrapDistance"/>
  <xsd:attribute name="distR" type="ST_WrapDistance"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_PositiveSize2D">
  <xsd:attribute name="cx" type="ST_PositiveCoordinate" use="required"/>
  <xsd:attribute name="cy" type="ST_PositiveCoordinate" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_EffectExtent">
  <xsd:attribute name="l" type="a:ST_Coordinate" use="required"/>
  <xsd:attribute name="t" type="a:ST_Coordinate" use="required"/>
  <xsd:attribute name="r" type="a:ST_Coordinate" use="required"/>
  <xsd:attribute name="b" type="a:ST_Coordinate" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_NonVisualDrawingProps">
  <xsd:sequence>
    <xsd:element name="hlinkClick" type="CT_Hyperlink"              minOccurs="0"/>
    <xsd:element name="hlinkHover" type="CT_Hyperlink"              minOccurs="0"/>
    <xsd:element name="extLst"     type="CT_OfficeArtExtensionList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="id"     type="ST_DrawingElementId" use="required"/>
  <xsd:attribute name="name"   type="xsd:string"          use="required"/>
  <xsd:attribute name="descr"  type="xsd:string"          default=""/>
  <xsd:attribute name="hidden" type="xsd:boolean"         default="false"/>
  <xsd:attribute name="title"  type="xsd:string"          default=""/>
</xsd:complexType>

<xsd:complexType name="CT_NonVisualGraphicFrameProperties">
  <xsd:sequence>
    <xsd:element name="graphicFrameLocks" type="CT_GraphicalObjectFrameLocking"␣
↪minOccurs="0"/>
    <xsd:element name="extLst"            type="CT_OfficeArtExtensionList"     ␣
↪minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_GraphicalObject">
  <xsd:sequence>
    <xsd:element name="graphicData" type="CT_GraphicalObjectData"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_GraphicalObjectData">
  <xsd:sequence>
    <xsd:any minOccurs="0" maxOccurs="unbounded" processContents="strict"/>
  </xsd:sequence>
  <xsd:attribute name="uri" type="xsd:token" use="required"/>
</xsd:complexType>
```

Inline shape size

The position of an inline shape is completely determined by the text it is inline with, however its dimensions can be specified. For some shape types, both the contained shape and the shape container specify a width and height. In the case of a picture, the dimensions of the inline shape (container) determine the display size while the dimension of the

pic element determine the "original size" of the image.

Candidate protocol – inline shape access

The following interactive session illustrates the protocol for accessing and changing the size of an inline shape:

```
>>> inline_shape = inline_shapes[0]
>>> assert inline_shape.type == MSO_SHAPE_TYPE.PICTURE
>>> inline_shape.width
914400
>>> inline_shape.height
457200
>>> inline_shape.width = 457200
>>> inline_shape.height = 228600
>>> inline_shape.width, inline_shape.height
457200, 228600
```

Resources

- InlineShape Members (Word) on MSDN
- Shape Members (Word) on MSDN

Specimen XML

This XML represents an inline shape inserted inline on a paragraph by itself:

```
<w:p>
  <w:r>
    <w:rPr/>
      <w:noProof/>
    </w:rPr>
    <w:drawing>
      <wp:inline distT="0" distB="0" distL="0" distR="0" wp14:anchorId="1BDE1558"␣
↪wp14:editId="31E593BB">
        <wp:extent cx="859536" cy="343814"/>
        <wp:effectExtent l="0" t="0" r="4445" b="12065"/>
        <wp:docPr id="1" name="Picture 1"/>
        <wp:cNvGraphicFramePr>
          <a:graphicFrameLocks xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/
↪main" noChangeAspect="1"/>
        </wp:cNvGraphicFramePr>
        <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
          <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">

            <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture
↪">
              <pic:nvPicPr>
                <pic:cNvPr id="1" name="python-powered.png"/>
                <pic:cNvPicPr/>
              </pic:nvPicPr>
              <pic:blipFill>
                <a:blip r:embed="rId7">
                  <a:alphaModFix/>
```

(continues on next page)

(continued from previous page)

```
                <a:extLst>
                  <a:ext uri="{28A0092B-C50C-407E-A947-70E740481C1C}">
                    <a14:useLocalDpi xmlns:a14="http://schemas.microsoft.com/office/
↪drawing/2010/main" val="0"/>
                  </a:ext>
                </a:extLst>
              </a:blip>
              <a:stretch>
                <a:fillRect/>
              </a:stretch>
            </pic:blipFill>
            <pic:spPr>
              <a:xfrm>
                <a:off x="0" y="0"/>
                <a:ext cx="859536" cy="343814"/>
              </a:xfrm>
              <a:prstGeom prst="rect">
                <a:avLst/>
              </a:prstGeom>
            </pic:spPr>
          </pic:pic>

        </a:graphicData>
      </a:graphic>
    </wp:inline>
  </w:drawing>
 </w:r>
</w:p>
```

Picture

Word allows a picture to be placed in a graphical object container, either an inline shape or a floating shape.

Candidate protocol

```
>>> run = paragraph.add_run()
>>> inline_shape = run.add_picture(file_like_image, MIME_type=None)
>>> inline_shape.width = width
>>> inline_shape.height = height
```

Minimal XML

This XML represents the working hypothesis of the minimum XML that must be inserted to add a working picture to a document:

```
<pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
  <pic:nvPicPr>
    <pic:cNvPr id="1" name="python-powered.png"/>
    <pic:cNvPicPr/>
  </pic:nvPicPr>
  <pic:blipFill>
```

(continues on next page)

(continued from previous page)

```
    <a:blip r:embed="rId7"/>
    <a:stretch>
      <a:fillRect/>
    </a:stretch>
  </pic:blipFill>
  <pic:spPr>
    <a:xfrm>
      <a:off x="0" y="0"/>
      <a:ext cx="859536" cy="343814"/>
    </a:xfrm>
    <a:prstGeom prst="rect"/>
  </pic:spPr>
</pic:pic>
```

Required parameters:

- unique DrawingML object id (document-wide, pretty sure it's just the part)
- name, either filename or generic if file-like object.
- rId for rel to image part
- size (cx, cy)

Specimen XML

This XML represents a picture inserted inline on a paragraph by itself:

```
<a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
  <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
    <pic:nvPicPr>
      <pic:cNvPr id="1" name="python-powered.png"/>
      <pic:cNvPicPr/>
    </pic:nvPicPr>
    <pic:blipFill>
      <a:blip r:embed="rId7">
        <a:alphaModFix/>
        <a:extLst>
          <a:ext uri="{28A0092B-C50C-407E-A947-70E740481C1C}">
            <a14:useLocalDpi xmlns:a14="http://schemas.microsoft.com/office/drawing/2010/
↪main" val="0"/>
          </a:ext>
        </a:extLst>
      </a:blip>
      <a:stretch>
        <a:fillRect/>
      </a:stretch>
    </pic:blipFill>
    <pic:spPr>
      <a:xfrm>
        <a:off x="0" y="0"/>
        <a:ext cx="859536" cy="343814"/>
      </a:xfrm>
      <a:prstGeom prst="rect">
```

(continues on next page)

(continued from previous page)

```
        <a:avLst/>
      </a:prstGeom>
    </pic:spPr>
  </pic:pic>
</a:graphicData>
```

Schema definitions

```
<xsd:element name="pic" type="CT_Picture"/>

<xsd:complexType name="CT_Picture">
  <xsd:sequence>
    <xsd:element name="nvPicPr"  type="CT_PictureNonVisual"/>
    <xsd:element name="blipFill" type="a:CT_BlipFillProperties"/>
    <xsd:element name="spPr"     type="a:CT_ShapeProperties"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_PictureNonVisual">
  <xsd:sequence>
    <xsd:element name="cNvPr"    type="a:CT_NonVisualDrawingProps"/>
    <xsd:element name="cNvPicPr" type="a:CT_NonVisualPictureProperties"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_BlipFillProperties">
  <xsd:sequence>
    <xsd:element name="blip"    type="CT_Blip"         minOccurs="0"/>
    <xsd:element name="srcRect" type="CT_RelativeRect" minOccurs="0"/>
    <xsd:choice minOccurs="0">
      <xsd:element name="tile"    type="CT_TileInfoProperties"/>
      <xsd:element name="stretch" type="CT_StretchInfoProperties"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="dpi"          type="xsd:unsignedInt"/>
  <xsd:attribute name="rotWithShape" type="xsd:boolean"/>
</xsd:complexType>

<xsd:complexType name="CT_ShapeProperties">
  <xsd:sequence>
    <xsd:element name="xfrm"    type="CT_Transform2D"           minOccurs="0"/>
    <xsd:group   ref="EG_Geometry"                              minOccurs="0"/>
    <xsd:group   ref="EG_FillProperties"                        minOccurs="0"/>
    <xsd:element name="ln"      type="CT_LineProperties"        minOccurs="0"/>
    <xsd:group   ref="EG_EffectProperties"                      minOccurs="0"/>
    <xsd:element name="scene3d" type="CT_Scene3D"               minOccurs="0"/>
    <xsd:element name="sp3d"    type="CT_Shape3D"               minOccurs="0"/>
    <xsd:element name="extLst"  type="CT_OfficeArtExtensionList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="bwMode" type="ST_BlackWhiteMode"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_Blip">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="alphaBiLevel" type="CT_AlphaBiLevelEffect"/>
      <xsd:element name="alphaCeiling" type="CT_AlphaCeilingEffect"/>
      <xsd:element name="alphaFloor"   type="CT_AlphaFloorEffect"/>
      <xsd:element name="alphaInv"     type="CT_AlphaInverseEffect"/>
      <xsd:element name="alphaMod"     type="CT_AlphaModulateEffect"/>
      <xsd:element name="alphaModFix"  type="CT_AlphaModulateFixedEffect"/>
      <xsd:element name="alphaRepl"    type="CT_AlphaReplaceEffect"/>
      <xsd:element name="biLevel"      type="CT_BiLevelEffect"/>
      <xsd:element name="blur"         type="CT_BlurEffect"/>
      <xsd:element name="clrChange"    type="CT_ColorChangeEffect"/>
      <xsd:element name="clrRepl"      type="CT_ColorReplaceEffect"/>
      <xsd:element name="duotone"      type="CT_DuotoneEffect"/>
      <xsd:element name="fillOverlay"  type="CT_FillOverlayEffect"/>
      <xsd:element name="grayscl"      type="CT_GrayscaleEffect"/>
      <xsd:element name="hsl"          type="CT_HSLEffect"/>
      <xsd:element name="lum"          type="CT_LuminanceEffect"/>
      <xsd:element name="tint"         type="CT_TintEffect"/>
    </xsd:choice>
    <xsd:element name="extLst" type="CT_OfficeArtExtensionList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute  ref="r:embed" type="ST_RelationshipId"  default=""/>
  <xsd:attribute  ref="r:link"  type="ST_RelationshipId"  default=""/>
  <xsd:attribute name="cstate"  type="ST_BlipCompression" default="none"/>
</xsd:complexType>

<xsd:simpleType name="ST_RelationshipId">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="CT_NonVisualDrawingProps">
  <xsd:sequence>
    <xsd:element name="hlinkClick" type="CT_Hyperlink"              minOccurs="0"/>
    <xsd:element name="hlinkHover" type="CT_Hyperlink"              minOccurs="0"/>
    <xsd:element name="extLst"     type="CT_OfficeArtExtensionList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="id"     type="ST_DrawingElementId" use="required"/>
  <xsd:attribute name="name"   type="xsd:string"          use="required"/>
  <xsd:attribute name="descr"  type="xsd:string"          default=""/>
  <xsd:attribute name="hidden" type="xsd:boolean"         default="false"/>
  <xsd:attribute name="title"  type="xsd:string"          default=""/>
</xsd:complexType>

<xsd:complexType name="CT_NonVisualPictureProperties">
  <xsd:sequence>
    <xsd:element name="picLocks" type="CT_PictureLocking"         minOccurs="0"/>
    <xsd:element name="extLst"   type="CT_OfficeArtExtensionList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="preferRelativeResize" type="xsd:boolean" default="true"/>
```

(continues on next page)

(continued from previous page)

```
</xsd:complexType>

<xsd:complexType name="CT_Point2D">
  <xsd:attribute name="x" type="ST_Coordinate" use="required"/>
  <xsd:attribute name="y" type="ST_Coordinate" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_PositiveSize2D">
  <xsd:attribute name="cx" type="ST_PositiveCoordinate" use="required"/>
  <xsd:attribute name="cy" type="ST_PositiveCoordinate" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_PresetGeometry2D">
  <xsd:sequence>
    <xsd:element name="avLst" type="CT_GeomGuideList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="prst" type="ST_ShapeType" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_RelativeRect">
  <xsd:attribute name="l" type="ST_Percentage" default="0%"/>
  <xsd:attribute name="t" type="ST_Percentage" default="0%"/>
  <xsd:attribute name="r" type="ST_Percentage" default="0%"/>
  <xsd:attribute name="b" type="ST_Percentage" default="0%"/>
</xsd:complexType>

<xsd:complexType name="CT_StretchInfoProperties">
  <xsd:sequence>
    <xsd:element name="fillRect" type="CT_RelativeRect" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Transform2D">
  <xsd:sequence>
    <xsd:element name="off" type="CT_Point2D"        minOccurs="0"/>
    <xsd:element name="ext" type="CT_PositiveSize2D" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="rot"   type="ST_Angle"    default="0"/>
  <xsd:attribute name="flipH" type="xsd:boolean" default="false"/>
  <xsd:attribute name="flipV" type="xsd:boolean" default="false"/>
</xsd:complexType>

<xsd:group name="EG_FillModeProperties">
  <xsd:choice>
    <xsd:element name="tile"    type="CT_TileInfoProperties"/>
    <xsd:element name="stretch" type="CT_StretchInfoProperties"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_Geometry">
  <xsd:choice>
    <xsd:element name="custGeom" type="CT_CustomGeometry2D"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="prstGeom" type="CT_PresetGeometry2D"/>
  </xsd:choice>
</xsd:group>
```

MS API

Access to shapes is provided by the Shapes and InlineShapes properties on the Document object.

The API for a floating shape overlaps that for an inline shapes, but there are substantial differences. The following properties are some of those common to both:

- Fill
- Glow
- HasChart
- HasSmartArt
- Height
- Shadow
- Hyperlink
- PictureFormat (providing brightness, color, crop, transparency, contrast)
- Type (Chart, LockedCanvas, Picture, SmartArt, etc.)
- Width

Resources

- Document Members (Word) on MSDN
- InlineShape Members (Word) on MSDN
- InlineShapes Members (Word) on MSDN
- Shape Members (Word) on MSDN

Core Document Properties

The Open XML format provides for a set of descriptive properties to be maintained with each document. One of these is the *core file properties*. The core properties are common to all Open XML formats and appear in document, presentation, and spreadsheet files. The 'Core' in core document properties refers to Dublin Core, a metadata standard that defines a core set of elements to describe resources.

The core properties are described in Part 2 of the ISO/IEC 29500 spec, in Section 11. The names of some core properties in `python-docx` are changed from those in the spec to conform to the MS API.

Other properties such as company name are custom properties, held in `app.xml`.

Candidate Protocol

```
>>> document = Document()
>>> core_properties = document.core_properties
>>> core_properties.author
'python-docx'
```

(continues on next page)

(continued from previous page)

```
>>> core_properties.author = 'Brian'
>>> core_properties.author
'Brian'
```

Properties

15 properties are supported. All unicode values are limited to 255 characters (not bytes).

author ***(unicode)***
Note: named 'creator' in spec. An entity primarily responsible for making the content of the resource. (Dublin Core)

category ***(unicode)***
A categorization of the content of this package. Example values for this property might include: Resume, Letter, Financial Forecast, Proposal, Technical Presentation, and so on. (Open Packaging Conventions)

comments ***(unicode)***
Note: named 'description' in spec. An explanation of the content of the resource. Values might include an abstract, table of contents, reference to a graphical representation of content, and a free-text account of the content. (Dublin Core)

content_status ***(unicode)***
The status of the content. Values might include "Draft", "Reviewed", and "Final". (Open Packaging Conventions)

created ***(datetime)***
Date of creation of the resource. (Dublin Core)

identifier ***(unicode)***
An unambiguous reference to the resource within a given context. (Dublin Core)

keywords ***(unicode)***
A delimited set of keywords to support searching and indexing. This is typically a list of terms that are not available elsewhere in the properties. (Open Packaging Conventions)

language ***(unicode)***
The language of the intellectual content of the resource. (Dublin Core)

last_modified_by ***(unicode)***
The user who performed the last modification. The identification is environment-specific. Examples include a name, email address, or employee ID. It is recommended that this value be as concise as possible. (Open Packaging Conventions)

last_printed ***(datetime)***
The date and time of the last printing. (Open Packaging Conventions)

modified ***(datetime)***
Date on which the resource was changed. (Dublin Core)

revision ***(int)***
The revision number. This value might indicate the number of saves or revisions, provided the application updates it after each revision. (Open Packaging Conventions)

subject ***(unicode)***
The topic of the content of the resource. (Dublin Core)

title ***(unicode)***
The name given to the resource. (Dublin Core)

version *(unicode)*
The version designator. This value is set by the user or by the application. (Open Packaging Conventions)

Specimen XML

core.xml produced by Microsoft Word:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
    xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:dcmitype="http://purl.org/dc/dcmitype/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title>Core Document Properties Exploration</dc:title>
  <dc:subject>PowerPoint core document properties</dc:subject>
  <dc:creator>Steve Canny</dc:creator>
  <cp:keywords>powerpoint; open xml; dublin core; microsoft office</cp:keywords>
  <dc:description>
    One thing I'd like to discover is just how line wrapping is handled
    in the comments. This paragraph is all on a single
    line._x000d__x000d_This is a second paragraph separated from the
    first by two line feeds.
  </dc:description>
  <cp:lastModifiedBy>Steve Canny</cp:lastModifiedBy>
  <cp:revision>2</cp:revision>
  <dcterms:created xsi:type="dcterms:W3CDTF">2013-04-06T06:03:36Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2013-06-15T06:09:18Z</dcterms:modified>
  <cp:category>analysis</cp:category>
</cp:coreProperties>
```

Schema Excerpt

```
<xs:schema
  targetNamespace="http://schemas.openxmlformats.org/package/2006/metadata/core-
↪properties"
  xmlns="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  elementFormDefault="qualified"
  blockDefault="#all">

  <xs:import
    namespace="http://purl.org/dc/elements/1.1/"
    schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd"/>
  <xs:import
    namespace="http://purl.org/dc/terms/"
    schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dcterms.xsd"/>
  <xs:import
    id="xml"
    namespace="http://www.w3.org/XML/1998/namespace"/>
```

(continues on next page)

(continued from previous page)

```
  <xs:element name="coreProperties" type="CT_CoreProperties"/>

  <xs:complexType name="CT_CoreProperties">
    <xs:all>
      <xs:element name="category"        type="xs:string"   minOccurs="0"/>
      <xs:element name="contentStatus"   type="xs:string"   minOccurs="0"/>
      <xs:element ref="dcterms:created"                     minOccurs="0"/>
      <xs:element ref="dc:creator"                          minOccurs="0"/>
      <xs:element ref="dc:description"                      minOccurs="0"/>
      <xs:element ref="dc:identifier"                       minOccurs="0"/>
      <xs:element name="keywords"        type="CT_Keywords" minOccurs="0"/>
      <xs:element ref="dc:language"                         minOccurs="0"/>
      <xs:element name="lastModifiedBy"  type="xs:string"   minOccurs="0"/>
      <xs:element name="lastPrinted"     type="xs:dateTime" minOccurs="0"/>
      <xs:element ref="dcterms:modified"                    minOccurs="0"/>
      <xs:element name="revision"        type="xs:string"   minOccurs="0"/>
      <xs:element ref="dc:subject"                          minOccurs="0"/>
      <xs:element ref="dc:title"                            minOccurs="0"/>
      <xs:element name="version"         type="xs:string"   minOccurs="0"/>
    </xs:all>
  </xs:complexType>

  <xs:complexType name="CT_Keywords" mixed="true">
    <xs:sequence>
      <xs:element name="value" minOccurs="0" maxOccurs="unbounded" type="CT_Keyword"/>
    </xs:sequence>
    <xs:attribute ref="xml:lang" use="optional"/>
  </xs:complexType>

  <xs:complexType name="CT_Keyword">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="xml:lang" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```

Numbering Part

… having to do with numbering sequences for ordered lists, etc. …

Schema excerpt

```
<xsd:complexType name="CT_Numbering">
  <xsd:sequence>
    <xsd:element name="numPicBullet"      type="CT_NumPicBullet"  minOccurs="0"␣
↪maxOccurs="unbounded"/>
    <xsd:element name="abstractNum"       type="CT_AbstractNum"   minOccurs="0"␣
↪maxOccurs="unbounded"/>
    <xsd:element name="num"               type="CT_Num"           minOccurs="0"␣
```

(continues on next page)

(continued from previous page)

```
↪maxOccurs="unbounded"/>
    <xsd:element name="numIdMacAtCleanup" type="CT_DecimalNumber" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Num">
  <xsd:sequence>
    <xsd:element name="abstractNumId" type="CT_DecimalNumber"/>
    <xsd:element name="lvlOverride"   type="CT_NumLvl"        minOccurs="0" maxOccurs="9
↪"/>
  </xsd:sequence>
  <xsd:attribute name="numId" type="ST_DecimalNumber" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_NumLvl">
  <xsd:sequence>
    <xsd:element name="startOverride" type="CT_DecimalNumber" minOccurs="0"/>
    <xsd:element name="lvl"           type="CT_Lvl"           minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="ilvl" type="ST_DecimalNumber" use="required"/>
</xsd:complexType>

<xsd:complexType name="CT_NumPr">
  <xsd:sequence>
    <xsd:element name="ilvl"            type="CT_DecimalNumber"        minOccurs="0"/>
    <xsd:element name="numId"           type="CT_DecimalNumber"        minOccurs="0"/>
    <xsd:element name="numberingChange" type="CT_TrackChangeNumbering" minOccurs="0"/>
    <xsd:element name="ins"             type="CT_TrackChange"          minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_DecimalNumber">
  <xsd:attribute name="val" type="ST_DecimalNumber" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_DecimalNumber">
  <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>
```

Sections

Word supports the notion of a *section*, having distinct page layout settings. This is how, for example, a document can contain some pages in portrait layout and others in landscape. Section breaks are implemented completely differently from line, page, and column breaks. The former adds a <w:pPr><w:sectPr> element to the last paragraph in the new section. The latter inserts a <w:br> element in a run.

The last section in a document is specified by a <w:sectPr> element appearing as the last child of the <w:body> element. While this element is optional, it appears that Word creates it for all files. Since most files have only a single section, the most common case is where this is the only <w:sectPr> element.

Additional sections are specified by a w:p/w:pPr/w:sectPr element in the last paragraph of the section. Any content in that paragraph is part of the section defined by its <w:sectPr> element. The subsequent section begins with the following paragraph.

When a section break is inserted using the Word UI, the following steps occur:

1. The next-occurring <w:sectPr> element is copied and added to the current paragraph. (It would be interesting to see what happens when that paragraph already has a <w:sectPr> element.)
2. A new paragraph is inserted after the current paragraph. The text occuring after the cursor position is moved to the new paragraph.
3. The start-type (e.g. next page) of the next-occuring <w:sectPr> element is changed to reflect the type chosen by the user from the UI.

Word behavior

- A paragraph containing a section break (<w:sectPr> element) does not produce a ¶ glyph in the Word UI.
- The section break indicator/double-line appears directly after the text of the paragraph in which the <w:sectPr> appears. If the section break paragraph has no text, the indicator line appears immediately after the paragraph mark of the prior paragraph.

Before and after analysis

Baseline document containing two paragraphs:

```
<w:body>
  <w:p>
    <w:r>
      <w:t>Paragraph 1</w:t>
    </w:r>
  </w:p>
  <w:p>
    <w:r>
      <w:t>Paragraph 2</w:t>
    </w:r>
  </w:p>
  <w:sectPr>
    <w:pgSz w:w="12240" w:h="15840"/>
    <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
             w:header="720" w:footer="720" w:gutter="0"/>
    <w:cols w:space="720"/>
    <w:docGrid w:linePitch="360"/>
  </w:sectPr>
</w:body>
```

Odd-page section inserted before paragraph mark in Paragraph 1:

```
<w:body>
  <w:p>
    <w:pPr>
      <w:sectPr>
        <w:pgSz w:w="12240" w:h="15840"/>
        <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
                 w:header="720" w:footer="720" w:gutter="0"/>
        <w:cols w:space="720"/>
        <w:docGrid w:linePitch="360"/>
      </w:sectPr>
    </w:pPr>
```

(continues on next page)

(continued from previous page)

```
    <w:r>
      <w:t>Paragraph 1</w:t>
    </w:r>
  </w:p>
  <w:p/>
  <w:p>
    <w:r>
      <w:t>Paragraph 2</w:t>
    </w:r>
  </w:p>
  <w:sectPr w:rsidR="00F039D0" w:rsidRsidSect="006006E7">
    <w:type w:val="oddPage"/>
    <w:pgSz w:w="12240" w:h="15840"/>
    <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
             w:header="720" w:footer="720" w:gutter="0"/>
    <w:cols w:space="720"/>
    <w:docGrid w:linePitch="360"/>
  </w:sectPr>
</w:body>
```

UI shows empty ¶ mark in first position of new next page. Section break indicator appears directly after Paragraph 1 text, with no intervening ¶ mark.

Even-page section break inserted before first character in Paragraph 2:

```
<w:body>
  <w:p>
    <w:r>
      <w:t>Paragraph 1</w:t>
    </w:r>
  </w:p>
  <w:p>
    <w:pPr>
      <w:sectPr>
        <w:type w:val="oddPage"/>
        <w:pgSz w:w="12240" w:h="15840"/>
        <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
                 w:header="720" w:footer="720" w:gutter="0"/>
        <w:cols w:space="720"/>
        <w:docGrid w:linePitch="360"/>
      </w:sectPr>
    </w:pPr>
  </w:p>
  <w:p>
    <w:r>
      <w:lastRenderedPageBreak/>
      <w:t>Paragraph 2</w:t>
    </w:r>
  </w:p>
  <w:sectPr>
    <w:type w:val="evenPage"/>
    <w:pgSz w:w="12240" w:h="15840"/>
    <w:pgMar w:top="1440" w:right="1800" w:bottom="1440" w:left="1800"
```

(continues on next page)

(continued from previous page)

```
            w:header="720" w:footer="720" w:gutter="0"/>
    <w:cols w:space="720"/>
    <w:docGrid w:linePitch="360"/>
  </w:sectPr>
</w:body>
```

Enumerations

WD_SECTION_START

alias: **WD_SECTION**

WdSectionStart Enumeration on MSDN

CONTINUOUS (0)
Continuous section break.

NEW_COLUMN (1)
New column section break.

NEW_PAGE (2)
New page section break.

EVEN_PAGE (3)
Even pages section break.

ODD_PAGE (4)
Odd pages section break.

WD_ORIENTATION

alias: **WD_ORIENT**

WdOrientation Enumeration on MSDN

LANDSCAPE (1)
Landscape orientation.

PORTRAIT (0)
Portrait orientation.

Schema excerpt

```
<xsd:complexType name="CT_PPr">  <!-- denormalized -->
  <xsd:sequence>
    <!-- 34 others ... -->
    <xsd:element name="sectPr"    type="CT_SectPr"    minOccurs="0"/>
    <xsd:element name="pPrChange" type="CT_PPrChange" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_SectPr">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="6"/>
      <xsd:element name="headerReference" type="CT_HdrFtrRef"/>
      <xsd:element name="footerReference" type="CT_HdrFtrRef"/>
```

(continues on next page)

(continued from previous page)

```
    </xsd:choice>
    <xsd:element name="footnotePr"      type="CT_FtnProps"      minOccurs="0"/>
    <xsd:element name="endnotePr"       type="CT_EdnProps"      minOccurs="0"/>
    <xsd:element name="type"            type="CT_SectType"      minOccurs="0"/>
    <xsd:element name="pgSz"            type="CT_PageSz"        minOccurs="0"/>
    <xsd:element name="pgMar"           type="CT_PageMar"       minOccurs="0"/>
    <xsd:element name="paperSrc"        type="CT_PaperSource"   minOccurs="0"/>
    <xsd:element name="pgBorders"       type="CT_PageBorders"   minOccurs="0"/>
    <xsd:element name="lnNumType"       type="CT_LineNumber"    minOccurs="0"/>
    <xsd:element name="pgNumType"       type="CT_PageNumber"    minOccurs="0"/>
    <xsd:element name="cols"            type="CT_Columns"       minOccurs="0"/>
    <xsd:element name="formProt"        type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="vAlign"          type="CT_VerticalJc"    minOccurs="0"/>
    <xsd:element name="noEndnote"       type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="titlePg"         type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="textDirection"   type="CT_TextDirection" minOccurs="0"/>
    <xsd:element name="bidi"            type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="rtlGutter"       type="CT_OnOff"         minOccurs="0"/>
    <xsd:element name="docGrid"         type="CT_DocGrid"       minOccurs="0"/>
    <xsd:element name="printerSettings" type="CT_Rel"           minOccurs="0"/>
    <xsd:element name="sectPrChange"    type="CT_SectPrChange"  minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr"  type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel"  type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"    type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidSect" type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:complexType name="CT_HdrFtrRef">
  <xsd:attribute  ref="r:id"                  use="required"/>
  <xsd:attribute name="type" type="ST_HdrFtr" use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_HdrFtr">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="even"/>
    <xsd:enumeration value="default"/>
    <xsd:enumeration value="first"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CT_SectType">
  <xsd:attribute name="val" type="ST_SectionMark"/>
</xsd:complexType>

<xsd:simpleType name="ST_SectionMark">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="nextPage"/>
    <xsd:enumeration value="nextColumn"/>
    <xsd:enumeration value="continuous"/>
    <xsd:enumeration value="evenPage"/>
    <xsd:enumeration value="oddPage"/>
```

(continues on next page)

(continued from previous page)

```
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CT_PageSz">
  <xsd:attribute name="w"      type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="h"      type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="orient" type="ST_PageOrientation"/>
  <xsd:attribute name="code"   type="ST_DecimalNumber"/>
</xsd:complexType>

<xsd:simpleType name="ST_PageOrientation">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="portrait"/>
    <xsd:enumeration value="landscape"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CT_PageMar">
  <xsd:attribute name="top"    type="ST_SignedTwipsMeasure" use="required"/>
  <xsd:attribute name="right"  type="s:ST_TwipsMeasure"     use="required"/>
  <xsd:attribute name="bottom" type="ST_SignedTwipsMeasure" use="required"/>
  <xsd:attribute name="left"   type="s:ST_TwipsMeasure"     use="required"/>
  <xsd:attribute name="header" type="s:ST_TwipsMeasure"     use="required"/>
  <xsd:attribute name="footer" type="s:ST_TwipsMeasure"     use="required"/>
  <xsd:attribute name="gutter" type="s:ST_TwipsMeasure"     use="required"/>
</xsd:complexType>

<xsd:simpleType name="ST_SignedTwipsMeasure">
  <xsd:union memberTypes="xsd:integer s:ST_UniversalMeasure"/>
</xsd:simpleType>

<xsd:complexType name="CT_Columns">
  <xsd:sequence minOccurs="0">
    <xsd:element name="col" type="CT_Column" maxOccurs="45"/>
  </xsd:sequence>
    <xsd:attribute name="equalWidth" type="s:ST_OnOff"/>
    <xsd:attribute name="space"      type="s:ST_TwipsMeasure"/>
    <xsd:attribute name="num"        type="ST_DecimalNumber"/>
    <xsd:attribute name="sep"        type="s:ST_OnOff"/>
</xsd:complexType>

<xsd:complexType name="CT_Column">
  <xsd:attribute name="w"     type="s:ST_TwipsMeasure"/>
  <xsd:attribute name="space" type="s:ST_TwipsMeasure"/>
</xsd:complexType>
```

4.1.2 Schema Analysis

Collected bits and pieces from the XML Schema docs, MSDN web pages, and the ISO/IEC 29500 spec.

CT_Document

Spec Name	Document
Tag(s)	w:document
Namespace	wordprocessingml (wml.xsd)
Spec Section	17.2.3

Spec text

This element specifies the contents of a main document part in a WordprocessingML document.

Consider the basic structure of the main document part in a basic WordprocessingML document, as follows:

```
<w:document>
  <w:body>
    <w:p/>
  </w:body>
</w:document>
```

All of the contents of the main document part are contained beneath the document element.

Schema excerpt

```
<xsd:complexType name="CT_Document">
  <xsd:sequence>
    <xsd:element name="background" type="CT_Background" minOccurs="0"/>
    <xsd:element name="body"       type="CT_Body"       minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="conformance" type="s:ST_ConformanceClass"/>
</xsd:complexType>

<xsd:simpleType name="ST_ConformanceClass">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="strict"/>
    <xsd:enumeration value="transitional"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CT_Background">
  <xsd:sequence>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:any processContents="lax" namespace="urn:schemas-microsoft-com:vml"
               minOccurs="0" maxOccurs="unbounded"/>
      <xsd:any processContents="lax" namespace="urn:schemas-microsoft-com:office:office"
               minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:element name="drawing" type="CT_Drawing" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="color"      type="ST_HexColor"       use="optional"/>
  <xsd:attribute name="themeColor" type="ST_ThemeColor"     use="optional"/>
  <xsd:attribute name="themeTint"  type="ST_UcharHexNumber" use="optional"/>
  <xsd:attribute name="themeShade" type="ST_UcharHexNumber" use="optional"/>
</xsd:complexType>
```

(continues on next page)

(continued from previous page)

```
<xsd:complexType name="CT_Body">  <!-- denormalized -->
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="p"                          type="CT_P"/>
      <xsd:element name="tbl"                        type="CT_Tbl"/>
      <xsd:element name="sdt"                        type="CT_SdtBlock"/>
      <xsd:element name="customXml"                  type="CT_CustomXmlBlock"/>
      <xsd:element name="altChunk"                   type="CT_AltChunk"/>
      <xsd:element name="proofErr"                   type="CT_ProofErr"/>
      <xsd:element name="permStart"                  type="CT_PermStart"/>
      <xsd:element name="permEnd"                    type="CT_Perm"/>
      <xsd:element name="bookmarkStart"              type="CT_Bookmark"/>
      <xsd:element name="bookmarkEnd"                type="CT_MarkupRange"/>
      <xsd:element name="moveFromRangeStart"         type="CT_MoveBookmark"/>
      <xsd:element name="moveFromRangeEnd"           type="CT_MarkupRange"/>
      <xsd:element name="moveToRangeStart"           type="CT_MoveBookmark"/>
      <xsd:element name="moveToRangeEnd"             type="CT_MarkupRange"/>
      <xsd:element name="commentRangeStart"          type="CT_MarkupRange"/>
      <xsd:element name="commentRangeEnd"            type="CT_MarkupRange"/>
      <xsd:element name="customXmlInsRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlInsRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlDelRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlDelRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveFromRangeEnd"  type="CT_Markup"/>
      <xsd:element name="customXmlMoveToRangeStart"  type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveToRangeEnd"    type="CT_Markup"/>
      <xsd:element name="ins"                        type="CT_RunTrackChange"/>
      <xsd:element name="del"                        type="CT_RunTrackChange"/>
      <xsd:element name="moveFrom"                   type="CT_RunTrackChange"/>
      <xsd:element name="moveTo"                     type="CT_RunTrackChange"/>
      <xsd:element ref="m:oMathPara"/>
      <xsd:element ref="m:oMath"/>
    </xsd:choice>
    <xsd:element name="sectPr" minOccurs="0" maxOccurs="1" type="CT_SectPr"/>
  </xsd:sequence>
</xsd:complexType>
```

CT_Body

Schema Name	CT_Body
Spec Name	Document Body
Tag(s)	w:body
Namespace	wordprocessingml (wml.xsd)
Spec Section	17.2.2

Spec text

This element specifies the contents of the body of the document – the main document editing surface.

The document body contains what is referred to as *block-level markup* – markup which can exist as a sibling element to paragraphs in a WordprocessingML document.

Example: Consider a document with a single paragraph in the main document story. This document would require the following WordprocessingML in its main document part:

```
<w:document>
  <w:body>
    <w:p/>
  </w:body>
</w:document>
```

The fact that the paragraph is inside the body element makes it part of the main document story.

Schema excerpt

```
<xsd:complexType name="CT_Body">
  <xsd:sequence>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="p"                          type="CT_P"/>
      <xsd:element name="tbl"                        type="CT_Tbl"/>
      <xsd:element name="customXml"                  type="CT_CustomXmlBlock"/>
      <xsd:element name="sdt"                        type="CT_SdtBlock"/>
      <xsd:element name="proofErr"                   type="CT_ProofErr"/>
      <xsd:element name="permStart"                  type="CT_PermStart"/>
      <xsd:element name="permEnd"                    type="CT_Perm"/>
      <xsd:element name="ins"                        type="CT_RunTrackChange"/>
      <xsd:element name="del"                        type="CT_RunTrackChange"/>
      <xsd:element name="moveFrom"                   type="CT_RunTrackChange"/>
      <xsd:element name="moveTo"                     type="CT_RunTrackChange"/>
      <xsd:element  ref="m:oMathPara"                type="CT_OMathPara"/>
      <xsd:element  ref="m:oMath"                    type="CT_OMath"/>
      <xsd:element name="bookmarkStart"              type="CT_Bookmark"/>
      <xsd:element name="bookmarkEnd"                type="CT_MarkupRange"/>
      <xsd:element name="moveFromRangeStart"         type="CT_MoveBookmark"/>
      <xsd:element name="moveFromRangeEnd"           type="CT_MarkupRange"/>
      <xsd:element name="moveToRangeStart"           type="CT_MoveBookmark"/>
      <xsd:element name="moveToRangeEnd"             type="CT_MarkupRange"/>
      <xsd:element name="commentRangeStart"          type="CT_MarkupRange"/>
      <xsd:element name="commentRangeEnd"            type="CT_MarkupRange"/>
      <xsd:element name="customXmlInsRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlInsRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlDelRangeStart"     type="CT_TrackChange"/>
      <xsd:element name="customXmlDelRangeEnd"       type="CT_Markup"/>
      <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveFromRangeEnd"  type="CT_Markup"/>
      <xsd:element name="customXmlMoveToRangeStart"  type="CT_TrackChange"/>
      <xsd:element name="customXmlMoveToRangeEnd"    type="CT_Markup"/>
      <xsd:element name="altChunk"                   type="CT_AltChunk"/>
    </xsd:choice>
```

(continues on next page)

(continued from previous page)

```
    <xsd:element name="sectPr" type="CT_SectPr" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_Body">
  <xsd:sequence>
    <xsd:group   ref="EG_BlockLevelElts"        minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="sectPr" type="CT_SectPr" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CT_SectPr">
  <xsd:sequence>
    <xsd:group   ref="EG_HdrFtrReferences"                  minOccurs="0" maxOccurs="6"/>
    <xsd:group   ref="EG_SectPrContents"                    minOccurs="0"/>
    <xsd:element name="sectPrChange" type="CT_SectPrChange" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="AG_SectPrAttributes"/>
</xsd:complexType>

<xsd:group name="EG_BlockLevelElts">
  <xsd:choice>
    <xsd:group    ref="EG_BlockLevelChunkElts"/>
    <xsd:element name="altChunk"               type="CT_AltChunk"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_BlockLevelChunkElts">
  <xsd:choice>
    <xsd:group ref="EG_ContentBlockContent"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_ContentBlockContent">
  <xsd:choice>
    <xsd:element name="customXml"       type="CT_CustomXmlBlock"/>
    <xsd:element name="sdt"             type="CT_SdtBlock"/>
    <xsd:element name="p"               type="CT_P"/>
    <xsd:element name="tbl"             type="CT_Tbl"/>
    <xsd:group    ref="EG_RunLevelElts"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_RunLevelElts">
  <xsd:choice>
    <xsd:element name="proofErr"               type="CT_ProofErr"/>
    <xsd:element name="permStart"              type="CT_PermStart"/>
    <xsd:element name="permEnd"                type="CT_Perm"/>
    <xsd:element name="ins"                    type="CT_RunTrackChange"/>
    <xsd:element name="del"                    type="CT_RunTrackChange"/>
    <xsd:element name="moveFrom"               type="CT_RunTrackChange"/>
    <xsd:element name="moveTo"                 type="CT_RunTrackChange"/>
```

(continues on next page)

(continued from previous page)

```
    <xsd:group    ref="EG_MathContent"/>
    <xsd:group    ref="EG_RangeMarkupElements"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_MathContent">
  <xsd:choice>
    <xsd:element ref="m:oMathPara" type="CT_OMathPara"/>
    <xsd:element ref="m:oMath"     type="CT_OMath"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_RangeMarkupElements">
  <xsd:choice>
    <xsd:element name="bookmarkStart"               type="CT_Bookmark"/>
    <xsd:element name="bookmarkEnd"                 type="CT_MarkupRange"/>
    <xsd:element name="moveFromRangeStart"          type="CT_MoveBookmark"/>
    <xsd:element name="moveFromRangeEnd"            type="CT_MarkupRange"/>
    <xsd:element name="moveToRangeStart"            type="CT_MoveBookmark"/>
    <xsd:element name="moveToRangeEnd"              type="CT_MarkupRange"/>
    <xsd:element name="commentRangeStart"           type="CT_MarkupRange"/>
    <xsd:element name="commentRangeEnd"             type="CT_MarkupRange"/>
    <xsd:element name="customXmlInsRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlInsRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlDelRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlDelRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
    <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
  </xsd:choice>
</xsd:group>
```

CT_P

Spec Name	Paragraph
Tag(s)	w:p
Namespace	wordprocessingml (wml.xsd)
Spec Section	17.3.1.22

Schema excerpt

```
<xsd:complexType name="CT_P">
  <xsd:sequence>
    <xsd:element name="pPr" type="CT_PPr" minOccurs="0"/>
    <xsd:group   ref="EG_PContent"        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="rsidRPr"      type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"        type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel"      type="ST_LongHexNumber"/>
```

(continues on next page)

(continued from previous page)

```
  <xsd:attribute name="rsidP"        type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidRDefault" type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:group name="EG_PContent">  <!-- denormalized -->
  <xsd:choice>
    <xsd:element name="r"               type="CT_R"/>
    <xsd:element name="hyperlink"       type="CT_Hyperlink"/>
    <xsd:element name="fldSimple"       type="CT_SimpleField"/>
    <xsd:element name="sdt"             type="CT_SdtRun"/>
    <xsd:element name="customXml"       type="CT_CustomXmlRun"/>
    <xsd:element name="smartTag"        type="CT_SmartTagRun"/>
    <xsd:element name="dir"             type="CT_DirContentRun"/>
    <xsd:element name="bdo"             type="CT_BdoContentRun"/>
    <xsd:element name="subDoc"          type="CT_Rel"/>
    <xsd:group    ref="EG_RunLevelElts"/>
  </xsd:choice>
</xsd:group>

<xsd:group name="EG_RunLevelElts">
  <xsd:choice>
    <xsd:element name="proofErr"                    type="CT_ProofErr"/>
    <xsd:element name="permStart"                   type="CT_PermStart"/>
    <xsd:element name="permEnd"                     type="CT_Perm"/>
    <xsd:element name="bookmarkStart"               type="CT_Bookmark"/>
    <xsd:element name="bookmarkEnd"                 type="CT_MarkupRange"/>
    <xsd:element name="moveFromRangeStart"          type="CT_MoveBookmark"/>
    <xsd:element name="moveFromRangeEnd"            type="CT_MarkupRange"/>
    <xsd:element name="moveToRangeStart"            type="CT_MoveBookmark"/>
    <xsd:element name="moveToRangeEnd"              type="CT_MarkupRange"/>
    <xsd:element name="commentRangeStart"           type="CT_MarkupRange"/>
    <xsd:element name="commentRangeEnd"             type="CT_MarkupRange"/>
    <xsd:element name="customXmlInsRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlInsRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlDelRangeStart"      type="CT_TrackChange"/>
    <xsd:element name="customXmlDelRangeEnd"        type="CT_Markup"/>
    <xsd:element name="customXmlMoveFromRangeStart" type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveFromRangeEnd"   type="CT_Markup"/>
    <xsd:element name="customXmlMoveToRangeStart"   type="CT_TrackChange"/>
    <xsd:element name="customXmlMoveToRangeEnd"     type="CT_Markup"/>
    <xsd:element name="ins"                         type="CT_RunTrackChange"/>
    <xsd:element name="del"                         type="CT_RunTrackChange"/>
    <xsd:element name="moveFrom"                    type="CT_RunTrackChange"/>
    <xsd:element name="moveTo"                      type="CT_RunTrackChange"/>
    <xsd:group   ref="EG_MathContent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_R">
  <xsd:sequence>
    <xsd:group ref="EG_RPr"             minOccurs="0"/>
    <xsd:group ref="EG_RunInnerContent" minOccurs="0" maxOccurs="unbounded"/>
```

(continues on next page)

(continued from previous page)

```
  </xsd:sequence>
  <xsd:attribute name="rsidRPr" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidDel" type="ST_LongHexNumber"/>
  <xsd:attribute name="rsidR"   type="ST_LongHexNumber"/>
</xsd:complexType>

<xsd:group name="EG_RunInnerContent">
  <xsd:choice>
    <xsd:element name="t"                    type="CT_Text"/>
    <xsd:element name="tab"                  type="CT_Empty"/>
    <xsd:element name="br"                   type="CT_Br"/>
    <xsd:element name="cr"                   type="CT_Empty"/>
    <xsd:element name="sym"                  type="CT_Sym"/>
    <xsd:element name="ptab"                 type="CT_PTab"/>
    <xsd:element name="softHyphen"           type="CT_Empty"/>
    <xsd:element name="contentPart"          type="CT_Rel"/>
    <xsd:element name="noBreakHyphen"        type="CT_Empty"/>
    <xsd:element name="fldChar"              type="CT_FldChar"/>
    <xsd:element name="instrText"            type="CT_Text"/>
    <xsd:element name="dayShort"             type="CT_Empty"/>
    <xsd:element name="monthShort"           type="CT_Empty"/>
    <xsd:element name="yearShort"            type="CT_Empty"/>
    <xsd:element name="dayLong"              type="CT_Empty"/>
    <xsd:element name="monthLong"            type="CT_Empty"/>
    <xsd:element name="yearLong"             type="CT_Empty"/>
    <xsd:element name="annotationRef"        type="CT_Empty"/>
    <xsd:element name="footnoteReference"    type="CT_FtnEdnRef"/>
    <xsd:element name="footnoteRef"          type="CT_Empty"/>
    <xsd:element name="endnoteReference"     type="CT_FtnEdnRef"/>
    <xsd:element name="endnoteRef"           type="CT_Empty"/>
    <xsd:element name="commentReference"     type="CT_Markup"/>
    <xsd:element name="separator"            type="CT_Empty"/>
    <xsd:element name="continuationSeparator" type="CT_Empty"/>
    <xsd:element name="pgNum"                type="CT_Empty"/>
    <xsd:element name="object"               type="CT_Object"/>
    <xsd:element name="pict"                 type="CT_Picture"/>
    <xsd:element name="ruby"                 type="CT_Ruby"/>
    <xsd:element name="drawing"              type="CT_Drawing"/>
    <xsd:element name="delText"              type="CT_Text"/>
    <xsd:element name="delInstrText"         type="CT_Text"/>
    <xsd:element name="lastRenderedPageBreak" type="CT_Empty"/>
  </xsd:choice>
</xsd:group>

<xsd:complexType name="CT_Text">
  <xsd:simpleContent>
    <xsd:extension base="s:ST_String">
      <xsd:attribute ref="xml:space" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W