**MsOffice::Word::Surgeon**

*tamper wit the guts of Microsoft docx documents*

Table des matières

[**MsOffice::Word::Surgeon** 1](#_Toc25163441)

[**Table of contents** 1](#_Toc25163442)

[**NAME** 1](#_Toc25163443)

[**SYNOPSIS** 1](#_Toc25163444)

[**DESCRIPTION** 2](#_Toc25163445)

[**Purpose** 2](#_Toc25163446)

[**Operating mode** 2](#_Toc25163447)

[**Status** 2](#_Toc25163448)

[**METHODS** 2](#_Toc25163449)

[**Constructor** 2](#_Toc25163450)

[**Contents restitution** 3](#_Toc25163451)

[**Modifying contents** 3](#_Toc25163452)

[**SEE ALSO** 6](#_Toc25163453)

[**AUTHOR** 6](#_Toc25163454)

[**COPYRIGHT AND LICENSE** 7](#_Toc25163455)

**NAME**

MsOffice::Word::Surgeon -- tamper wit the guts of Microsoft docx documents

**SYNOPSIS**

my $surgeon = MsOffice::Word::Surgeon->new(docx => $filename);

# extract plain text

my $text = $surgeon->plain\_text;

# simplify the internal XML structure -- so that later replacements work better

$surgeon->reduce\_all\_noises;

$surgeon->unlink\_fields;

$surgeon->merge\_runs;

# anonymize

my %alias = ('Claudio MONTEVERDI' => 'A\_\_\_\_\_', 'Heinrich SCHÜTZ' => 'B\_\_\_\_\_');

my $pattern = join "|", keys %alias;

my $replacement\_callback = sub {

my %args = @\_;

return $surgeon->change(to\_delete => $args{matched},

to\_insert => $alias{$args{matched}},

author => \_\_PACKAGE\_\_,

);

};

$surgeon->replace(qr[$pattern], $replacement\_callback);

# save the result

$surgeon->overwrite; # or ->save\_as($new\_filename);

**DESCRIPTION**

**Purpose**

This module supports a few operations for modifying or extracting text from Microsoft Word documents in '.docx' format -- therefore the name 'surgeon'. Since a surgeon does not give life, there is no support for creating fresh documents; if you have such needs, use one of the other packages listed in the [SEE ALSO](http://localhost:8080/MsOffice/Word/Surgeon" \l "SEE_ALSO) section.

Some applications for this module are :

* content extraction in plain text format;
* unlinking fields (equivalent of performing Ctrl-Shift-F9 on the whole document)
* regex replacements within text, for example for :
  + anonymization, i.e. replacement of names or adresses by aliases;
  + templating, i.e. replacement of special markup by contents coming from a data tree
* pretty-printing the internal XML structure

**Operating mode**

The format of Microsoft .docx documents is described in [http://www.ecma-international.org/publications/standards/Ecma-376.htm](http://www.ecma-international.org/publications/standards/Ecma-376.htm" \t "_blank) and [http://officeopenxml.com/](http://officeopenxml.com/" \t "_blank). An excellent introduction can be found at <https://www.toptal.com/xml/an-informal-introduction-to-docx>. Internally, a document is a zipped archive, where the member named word/document.xml stores the main document contents, in XML format.

The present module does not parse all details of the whole XML structure because it only focuses on *text* nodes (those that contain literal text) and *run* nodes (those that contain text formatting properties). All remaining XML information, for example for representing sections, paragraphs, tables, etc., is stored as opaque XML fragments; these fragments are re-inserted at proper places when reassembling the whole document after having modified some text nodes.

**Status**

This is the first release; the software architecture is quite stable but the module is not battle-proofed. Minor changed to the public interface may occur in future versions.

**METHODS**

**Constructor**

**new**

my $surgeon = MsOffice::Word::Surgeon->new(docx => $filename);

# or simply : ->new($filename);

Builds a new surgeon instance, initialized with the contents of the given filename.

**Contents restitution**

**contents**

Returns a Perl string with the current internal XML representation of the document contents.

**original\_contents**

Returns a Perl string with the XML representation of the document contents, as it was in the ZIP archive before any modification.

**indented\_contents**

Returns an indented version of the XML contents, suitable for inspection in a text editor. This is produced by [toString in XML::LibXML::Document](http://localhost:8080/XML/LibXML/Document" \l "toString) and therefore is returned as an encoded byte string, not a Perl string.

**plain\_text**

Returns the text contents of the document, without any markup. Paragraphs are converted to newlines, all other formatting instructions are ignored.

**runs**

Returns a list of [MsOffice::Word::Surgeon::Run](http://localhost:8080/MsOffice/Word/Surgeon/Run) objects. Each of these objects holds an XML fragment; joining all fragments restores the complete document.

my $contents = join "", map {$\_->as\_xml} $self->runs;

**Modifying contents**

**reduce\_noise**

$surgeon->reduce\_noise($regex1, $regex2, ...);

This method is used for removing unnecessary information in the XML markup. It applies the given list of regexes to the whole document, suppressing matches. The final result is put back into $self->contents. Regexes may be given either as qr/.../ references, or as names of builtin regexes (described below). Regexes are applied to the whole XML contents, not only to run nodes.

**noise\_reduction\_regex**

my $regex = $surgeon->noise\_reduction\_regex($regex\_name);

Returns the builtin regex corresponding to the given name. Known regexes are :

proof\_checking => qr(<w:(?:proofErr[^>]+|noProof/)>),

revision\_ids => qr(\sw:rsid\w+="[^"]+"),

complex\_script\_bold => qr(<w:bCs/>),

page\_breaks => qr(<w:lastRenderedPageBreak/>),

language => qr(<w:lang w:val="[^"]+"/>),

empty\_run\_props => qr(<w:rPr></w:rPr>),

**reduce\_all\_noises**

$surgeon->reduce\_all\_noises;

Applies all regexes from the previous method.

**unlink\_fields**

Removes all fields from the document, just leaving the current value stored in each field. This is the equivalent of performing Ctrl-Shift-F9 on the whole document.

**merge\_runs**

$surgeon->merge\_runs(no\_caps => 1); # optional arg

Walks through all runs of text within the document, trying to merge adjacent runs when possible (i.e. when both runs have the same properties, and there is no other XML node inbetween).

This operation is a prerequisite before performing replace operations, because documents edited in MsWord often have run boundaries across sentences or even in the middle of words; so regex searches can only be successful if those artificial boundaries have been removed.

If the argument no\_caps => 1 is present, the merge operation will also convert runs with the w:caps property, putting all letters into uppercase and removing the property; this makes more merges possible.

**replace**

my $xml = $surgeon->replace($pattern, $replacement\_callback, %replacement\_args);

Replaces all occurrences of $pattern regex within the document by a new string computed by $replacement\_callback, and returns a new xml string corresponding to the whole document contents after all these replacements.

$pattern should be a reference to a regular expression.

The $replacement\_callback will be called for for each text node within each run node. It will receive a copy of %replacement\_args, enriched with three entries :

* **matched**

The string that has been matched by $pattern.

* **run**

The run object in which this text resides.

* **xml\_before**

An optional XML fragment found before the matched text.

The replacement callback should return an XML string. See the [SYNOPSIS](http://localhost:8080/MsOffice/Word/Surgeon#SYNOPSIS) for an example of a replacement callback.

**change**

my $xml = $surgeon->change(

to\_delete => $text\_to\_delete,

to\_insert => $text\_to\_insert,

author => $author\_string,

date => $date\_string,

run => $run\_object,

xml\_before => $xml\_string,

);

This method generates markup for MsWord tracked changes. Users can then manually review those changes within MsWord and accept or reject them. This is best used in collaboration with the [replace](http://localhost:8080/MsOffice/Word/Surgeon" \l "replace) method : the replacement callback can call $self->change(...) to generate tracked change marks in the document.

All parameters are optional, but either to\_delete or to\_insert (or both) must be present. The parameters are :

* **to\_delete**

The string of text to delete (usually this will be the matched argument passed to the replacement callback).

* **to\_insert**

The string of new text to insert.

* **author**

A short string that will be displayed by MsWord as the "author" of this tracked change.

* **date**

A date (and optional time) in ISO format that will be displayed by MsWord as the date of this tracked change. The current date and time will be used by default.

* **run**

A reference to the [MsOffice::Word::Surgeon::Run](http://localhost:8080/MsOffice/Word/Surgeon/Run) object surrounding this tracked change. The formatting properties of that run will be copied into the <w:r> nodes of the deleted and inserted text fragments.

* **xml\_before**

An optional XML fragment to be inserted before the <w:t> node of the inserted text

This method delegates to the [MsOffice::Word::Surgeon::Change](http://localhost:8080/MsOffice/Word/Surgeon/Change) class for generating the XML markup.

**SEE ALSO**

The <https://metacpan.org/pod/Document::OOXML> distribution on CPAN also manipulates docx documents, but with another approach : internally it uses [XML::LibXML](http://localhost:8080/XML/LibXML) and XPath expressions for manipulating XML nodes. The API has some intersections with the present module, but there are also some differences : Document::OOXML has more support for styling, while MsOffice::Word::Surgeon has more flexible mechanisms for replacing text fragments.

Other programming languages also have packages for dealing with docx documents; here are some references :

* **[https://docs.microsoft.com/en-us/office/open-xml/word-processing](https://docs.microsoft.com/en-us/office/open-xml/word-processing" \t "_blank)**

The C# Open XML SDK from Microsoft

* **[http://www.ericwhite.com/blog/open-xml-powertools-developer-center/](http://www.ericwhite.com/blog/open-xml-powertools-developer-center/" \t "_blank)**

Additional functionalities built on top of the XML SDK.

* **[https://www.docx4java.org/trac/docx4j](https://www.docx4java.org/trac/docx4j" \t "_blank)**

An open source Java library.

* **[https://phpword.readthedocs.io/en/latest/](https://phpword.readthedocs.io/en/latest/" \t "_blank)**

A PHP library dealing not only with Microsoft OOXML documents but also with OASIS and RTF formats.

* **[https://pypi.org/project/python-docx/](https://pypi.org/project/python-docx/" \t "_blank)**

A Python library, documented at <https://python-docx.readthedocs.io/en/latest/>.

As far as I can tell, most of these libraries provide objects and methods that closely reflect the complete XML structure : for example they have classes for paragraphes, styles, fonts, inline shapes, etc.

The present module is much simpler but also much more limited: it was optimised for dealing with the text contents and offers no support for presentation or paging features.

**AUTHOR**

Laurent Dami, <dami AT cpan DOT org<gt>

**COPYRIGHT AND LICENSE**

Copyright 2019 by Laurent Dami.

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

==============

At 1st or 2nd sight, surgery on docx files is an original approach.

This paragraph starts with an **initial TAB**, and also has many internal TABS.

Field test (ASK) : do you prefer Foo ? Really ? Even a 2nd time like this : Foo ?Foo

This paragraph  
contains a soft line break

Condamne small & caps ltd à verser en mains des curatrices de NOM la somme de 1'200 fr. à titre de contribution à son entretien dès le prononcé du présent arrêt jusqu'à la majorité

And here is another ASK field: perduta ho la pace

This paragraph is justified; it contains just a few soft hy­phens that should really be remo­ved when examining the text contents.perduta ho la pace