<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   Available formats</td>
<td>3</td>
</tr>
<tr>
<td>2   Conversion stages</td>
<td>5</td>
</tr>
<tr>
<td>3   Converters: Parsers + Builders</td>
<td>7</td>
</tr>
<tr>
<td>4   Usage</td>
<td>9</td>
</tr>
<tr>
<td>4.1  Benker</td>
<td>10</td>
</tr>
<tr>
<td>4.2  Tutorials</td>
<td>12</td>
</tr>
<tr>
<td>4.3  API</td>
<td>46</td>
</tr>
<tr>
<td>4.4  Changelog</td>
<td>87</td>
</tr>
<tr>
<td>5   Indices and tables</td>
<td>95</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>97</td>
</tr>
<tr>
<td>Index</td>
<td>99</td>
</tr>
</tbody>
</table>
The Benker library can be used to convert tables from one format to another.

Yes, it only converts the tables, not the whole document, but it tries to do it well. The document itself is not changed, and the paragraphs inside the cells, neither. It’s your responsibility to do this part of the work.
The Benker library works on XML documents. Currently, it can handle:

<table>
<thead>
<tr>
<th>Format Description</th>
<th>Web Site</th>
<th>Wikipedia Page</th>
</tr>
</thead>
</table>

OOXML (Office Open XML) is an XML-based format for office documents, including word processing documents, spreadsheets, presentations, as well as charts, diagrams, shapes, and other graphical material. This is the XML format used by Microsoft Word documents: *.docx.


CALS (Continuous Acquisition and Life-cycle Support) table model is a standard for representing tables in SGML/XML. Developed as part of the CALS Department of Defence initiative. The DTD of the CALS table model is available in the OASIS (Organization for the Advancement of Structured Information Standards) web site.

- Specification on OASIS web site: https://www.oasis-open.org/specs/tablemodels.php
- Wikipedia page: https://en.wikipedia.org/wiki/CALS_Table_Model
To convert a document, Benker uses several stages:

1. Parse the source document and construct a nodes tree,
2. Search for table elements and construct the table objects,
3. Build the target nodes tree by replacing table nodes,
4. Serialise the target document.
The decoupling between parsing, building and final serialization allows a simplified and modular implementation. This decoupling also allows to multiply the combinations: it is easy to change a builder to another one, and to develop its own parser.

The advantage of this approach is that we avoid having a specific document conversion for each format pair (input, output). Instead, you can build a converter by choosing a parser and a builder, as you assemble the pieces of a puzzle.

The following table show you the available converters which groups parser and builders by pairs.
Table 1: Available converters

<table>
<thead>
<tr>
<th></th>
<th>OOOXML</th>
<th>HTML</th>
<th>CALS</th>
<th>Formex4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOOXML</td>
<td>–</td>
<td>(unavailable)</td>
<td>convert_ooxml2cals()</td>
<td>convert_ooxml2formex()</td>
</tr>
<tr>
<td>HTML</td>
<td>(unavailable)</td>
<td>–</td>
<td>(unavailable)</td>
<td>(unavailable)</td>
</tr>
<tr>
<td>CALS</td>
<td>(unavailable)</td>
<td>(unavailable)</td>
<td>–</td>
<td>convert_cals2formex()</td>
</tr>
<tr>
<td>Formex4</td>
<td>(unavailable)</td>
<td>(unavailable)</td>
<td>convert_formex2cals()</td>
<td>–</td>
</tr>
</tbody>
</table>

You can create your own converter by inheriting the available base classes:

- **BaseConverter**: inherit this class to create your own converter. Set your own parser class to the `parser_cls` class attribute, and your own builder class to the `builder_cls` class attribute.

- **BaseParser**: inherit this class to create your own parser. The method `transform_tables()` is an abstract method, so you need to implement it in your subclass: it must call the method `generate_table_tree()` each time a table node is found and converted to a `Table` object.

- **BaseBuilder**: inherit this class to create your own builder. The method `generate_table_tree()` is an abstract method, so you need to implement it in your subclass: it must convert the `Table` object into a target XML node (the resulting table format). You can also implement the method `finalize_tree()` to do any post-processing to the resulting XML tree.

**Hint**: Contribution is welcome!
Usage

For example, to convert the tables of a .docx document to Formex4 format, you can process as follow:

```python
import os
import zipfile

from benker.converters.ooxml2formex import convert_ooxml2formex

# - Unzip the `.docx` in a temporary directory
src_zip = "/path/to/demo.docx"
tmp_dir = "/path/to/tmp/dir/"
with zipfile.ZipFile(src_zip) as zf:
    zf.extractall(tmp_dir)

# - Source paths
src_xml = os.path.join(tmp_dir, "word/document.xml")
styles_xml = os.path.join(tmp_dir, "word/styles.xml")

# - Destination path
dst_xml = "/path/to/demo.xml"

# - Create some options and convert tables
options = {
    'encoding': 'utf-8',
    'styles_path': styles_xml,
}
convert_ooxml2formex(src_xml, dst_xml, **options)
```

This code produces a table like that:

```
<TBL COLS="7" NO.SEQ="0001">
  <CORPUS>
    <ROW>
      <CELL COL="1" ROWSPAN="2">
        <w:p w:rsidR="00EF2ECA" w:rsidRDefault="00EF2ECA">
          <w:r><w:t><w:t>(continues on next page)</w:t></w:t></w:r>
        </w:p>
      </CELL>
    </ROW>
  </CORPUS>
</TBL>
```

(continues on next page)
4.1 Benker

Easily convert your CALS, HTML, Formex4, Office Open XML (docx) tables from one format to another.

4.1.1 Overview

To convert the tables of a .docx document to CALS format, you can process as follow:

```python
import os
import zipfile
from benker.converters.ooxml2cals import convert_ooxml2cals

# - Unzip the `.docx` in a temporary directory
src_zip = "/path/to/demo.docx"
tmp_dir = "/path/to/tmp/dir/
with zipfile.ZipFile(src_zip) as zf:
    zf.extractall(tmp_dir)
```

The content of the cells still contains OOXML fragments. It’s your own responsibility to convert them to the target format.
# - Source paths
src_xml = os.path.join(tmp_dir, "word/document.xml")
sty"les_xml = os.path.join(tmp_dir, "word/styles.xml")

# - Destination path
dst_xml = "/path/to/demo.xml"

# - Create some options and convert tables
options = {
    'encoding': 'utf-8',
    'styles_path': styles_xml,
    'width_unit': "mm",
    'table_in_tgroup': True,
}
convert_ooxml2cals(src_xml, dst_xml, **options)

4.1.2 Installation

To install this library, you can create and activate a virtualenv, and run:

```
pip install benker
```

Requirements

This library uses lxml library and is tested with the versions 3.x (for Python < 3.7), and 4.x.

Usage in your library/application

You can use this library in your own library/application.
To do so, add this library in your setup.py in your project requirements:

```
setup(
    name="YourApp",
    install_requires=['benker'],
    ...
)
```

To install the dependencies, activate your virtualenv and run:

```
pip install -e .
```

And enjoy!

4.1.3 Licence

This library is distributed according to the MIT licence.
Users have legal right to download, modify, or distribute the library.
4.1.4 Authors

Benker was written by Laurent LAPORTE.

4.2 Tutorials

This section presents the different tutorials available to discover and learn how to use Benker. This library has high-level conversion functions to convert tables from one format to another. All functions have the same API, it’s easier for everyone.

The following tutorials will give you sample files to convert and the expected results. Of course, you will have the opportunity to choose your conversion options according to your needs:

4.2.1 OOXML to Formex 4 converter

Description

The `convert_ooxml2formex()` converter is a function designed to convert tables from an Office Open XML (OOXML) document (which respects the schema defined in Office Open XML File Formats) in the Formex 4 format. The conversion is done in the source XML document by replacing the tables of the OOXML format with those transformed in the Formex format. In other words, the general structure of the source XML document is retained except for tables.

The `Ooxml2FormexConverter` converter is composed of:

- a `OoxmlParser` parser that allows you to parse tables in OOXML format,
  The tutorial `OOXML tables (Word) parser` describes the usage of this parser and gives some examples.
- a `FormexBuilder` builder that allows you to build tables in the Formex format.
  The tutorial `Formex 4 tables builder` describes the usage of this builder and gives some examples.

Conversion options

The tables parsing and building can be parameterized using the options described below:

Common parsing options:

- `encoding` (default: “utf-8”): XML encoding of the destination file.

OOXML parser options:

- `styles_path` (default: None): Path to the stylesheet to use to resolve table styles. In an uncompressed `.docx` tree structure, the stylesheet path is `word/styles.xml`.

Formex 4 builder options:

- `use_cals` (default: False): Generate additional CALS-like elements and attributes to simplify the layout of Formex document in typesetting systems.
- `cals_ns` (default: “https://lib.benker.com/schemas/cals.xsd”): Namespace to use for CALS-like elements and attributes (requires: `use_cals`). Set None (or “”) if you don’t want to use namespace.
- `cals_prefix` (default: “cals”): Namespace prefix to use for CALS-like elements and attributes (requires: `use_cals`).


Examples of conversions

Converting a .docx document

You can use the `convert_ooxml2formex()` converter to convert a Word document, for instance, we have the following annex:

```
ANNEX

Annex 1
Concessions granted by Switzerland

The tariff concessions set out below are granted by Switzerland for the following products originating in the European Union and are, where applicable, subject to an annual quantity:

<table>
<thead>
<tr>
<th>Swiss tariff heading</th>
<th>Description</th>
<th>Customs duty applicable (CHF/100 kg gross weight)</th>
<th>Annual quantity (tonnes net weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101 2991</td>
<td>Live horses (excl. pure-bred horses for breeding and horses for slaughter) (in number of head)</td>
<td>0.00</td>
<td>100 head</td>
</tr>
<tr>
<td>0204 5010</td>
<td>Goat meat, fresh, chilled or frozen</td>
<td>40.00</td>
<td>100</td>
</tr>
<tr>
<td>0207 1401</td>
<td>Breasts of fowls of domestic species, frozen</td>
<td>15.00</td>
<td>2 100</td>
</tr>
<tr>
<td>0207 1401</td>
<td>Cuts and edible offal of fowls of domestic species, including livers (excluding breasts), frozen</td>
<td>15.00</td>
<td>1 200</td>
</tr>
<tr>
<td>0207 2781</td>
<td>Breasts of turkeys of domestic species, frozen</td>
<td>15.00</td>
<td>800</td>
</tr>
<tr>
<td>0207 2791</td>
<td>Cuts and edible offal of turkeys of domestic species, including livers (excluding breasts), frozen</td>
<td>15.00</td>
<td>600</td>
</tr>
<tr>
<td>0207 4210</td>
<td>Ducks of domestic species, not cut in pieces, frozen</td>
<td>15.00</td>
<td>700</td>
</tr>
</tbody>
</table>
```

If you want to convert a .docx file, you need first to decompress it in a temporary directory in order to access the “word/document.xml” and “word/styles.xml” stored in the .docx package.

To decompress the .docx package and convert the tables, you can do:
>> import os
>> import zipfile

>> from benker.converters.ooxml2formex import convert_ooxml2formex

>> src_zip = "docs/_static/converters.ooxml2formex.sample1.docx"
>> with zipfile.ZipFile(src_zip) as zf:
...   zf.extractall(tmp_dir)

>> src_xml = os.path.join(tmp_dir, "word/document.xml")
>> styles_xml = os.path.join(tmp_dir, "word/styles.xml")
>> dst_xml = os.path.join(tmp_dir, "converters.ooxml2formex.sample1.xml")
>> options = {
...   'encoding': 'utf-8',
...   'styles_path': styles_xml,
... }

>> convert_ooxml2formex(src_xml, dst_xml, **options)

The result is the “word/document.xml” document, but with tables replaced by the Formex TBL elements.

Here is a sample of the result XML:

```xml
<?xml version='1.0' encoding='UTF-8'?>
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
  mc:Ignorable="w14 w15 w16se w16cid wp14">
  <w:body>
    <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
      <w:pPr>
        <w:pStyle w:val="Titre1"/>
        <w:jc w:val="center"/>
      </w:pPr>
      <w:r><w:t>ANNEX</w:t></w:r>
    </w:p>
    <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
      <w:pPr>
        <w:pStyle w:val="Titre2"/>
        <w:jc w:val="center"/>
      </w:pPr>
      <w:r><w:t>Annex 1</w:t></w:r>
      <w:r><w:br/><w:t>Concessions granted by Switzerland</w:t></w:r>
    </w:p>
    <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
      <w:pPr>
        <w:pStyle w:val="Corpsdetexte"/>
      </w:pPr>
      <w:r><w:t>The tariff concessions set out below are granted by Switzerland
          for the following products originating in the European Union and are,
          where applicable, subject to an annual quantity:</w:t></w:r>
    </w:p>
    <TBL NO.SEQ="0001" COLS="4">
      <CORPUS>
        <ROW TYPE="HEADER">
          <CELL COL="1">
            <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
              <w:pPr>
                <w:pStyle w:val="Corpsdetexte"/>
              </w:pPr>
            </w:p>
          </CELL>
          <CELL COL="2">
            <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
              <w:pPr>
                <w:pStyle w:val="Corpsdetexte"/>
              </w:pPr>
            </w:p>
          </CELL>
          <CELL COL="3">
            <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
              <w:pPr>
                <w:pStyle w:val="Corpsdetexte"/>
              </w:pPr>
            </w:p>
          </CELL>
          <CELL COL="4">
            <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
              <w:pPr>
                <w:pStyle w:val="Corpsdetexte"/>
              </w:pPr>
            </w:p>
          </CELL>
        </ROW>
      </CORPUS>
    </TBL>
  </w:body>
</w:document>
```

(continues on next page)
Using CALS-like attributes and elements

The Formex table format is good to structure tables. The logical structure is similar to the one used for HTML tables but without CSS.

Some difficulties appears when you want to do the layout of Formex tables in typesetting systems: Formex tables
doesn’t have much layout information:

- no borders,
- no horizontal or vertical alignment of the text,
- no background color,
- no indication of the column width,
- etc.

To solve that, it is possible to generate CALS-like attributes and elements in the Formex. Of course, we can use a namespace and a namespace prefix for the CALS attributes and elements.

To convert the tables using CALS, you can do:

```python
>>> dst_xml = os.path.join(tmp_dir, "converters.oxmxml2formex.sample2.xml")
>>> options = {
...   'encoding': 'utf-8',
...   'styles_path': styles_xml,
...   'use_cals': True,
...   'cals_ns': "http://cals",
...   'cals_prefix': "cals",
... }
>>> convert_oxmxml2formex(src_xml, dst_xml, **options)
```

The result is the “word/document.xml” document, but with tables replaced by the Formex TBL elements.

Here is a sample of the result XML:

```xml
<TBL xmlns:cals="http://cals" NO.SEQ="0001" COLS="4">
  <CORPUS cals:frame="none" cals:colsep="0" cals:rowsep="0" cals:pgwide="1">
    <cals:colspec cals:colname="c1" cals:colwidth="24.04mm"/>
    <cals:colspec cals:colname="c2" cals:colwidth="89.09mm"/>
    <cals:colspec cals:colname="c3" cals:colwidth="31.96mm"/>
    <cals:colspec cals:colname="c4" cals:colwidth="24.91mm"/>
  </CORPUS>
  <ROW TYPE="HEADER">
    <CELL COL="1" cals:rowsep="1" cals:align="center">
      <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
        <w:pPr>
          <w:pStyle w:val="Corpsdetexte"/>
          <w:keepNext/>
          <w:jc w:val="center"/>
          <w:rPr><w:b/><w:bCs/></w:rPr>
        </w:pPr>
        <w:r><w:rPr><w:b/><w:bCs/></w:rPr><w:t>Swiss tariff</w:t></w:r>
        <w:r><w:rPr><w:b/><w:bCs/></w:rPr><w:br/></w:r>
        <w:r><w:rPr><w:b/><w:bCs/></w:rPr><w:t>heading</w:t></w:r>
      </w:p>
    </CELL>
    <CELL COL="2" cals:rowsep="1" cals:align="center">
      <w:p w:rsidR="001E0B3A" w:rsidRDefault="00883CDC">
        <w:pPr>
          <w:pStyle w:val="Corpsdetexte"/>
          <w:jc w:val="center"/>
          <w:rPr><w:b/><w:bCs/></w:rPr>
        </w:pPr>
        <w:r><w:rPr><w:b/><w:bCs/></w:rPr><w:t>Description</w:t></w:r>
      </w:p>
    </CELL>
  </ROW>
</TBL>
```

(continues on next page)
In the result, we can notice:

- the presence of the namespace `xmlns:cals="http://cals"`.
- the additional attributes, like `cals:frame="none",cals:colsep="0",cals:rowsep="0"...`
- the additional `colspec` elements: `<cals:colspec cals:colname="c1" cals:colwidth="24.04mm"/>`.

This kind of information is will be preserved if you use a Formex to CALS conversion (see the *Formex 4 to CALS converter* tutorial).

### 4.2.2 OOXML to CALS converter

#### Description

The `convert_ooxml2cals()` converter is a function designed to convert tables from an Office Open XML (OOXML) document (which respects the schema defined in *Office Open XML File Formats*) in the CALS table format.

The conversion is done in the source XML document by replacing the tables of the OOXML format with those transformed in the CALS format. In other words, the general structure of the source XML document is retained except for tables.

The `Ooxml2CalsConverter` converter is composed of:

- a `OoxmlParser` parser that allows you to parse tables in OOXML format.

  The tutorial *OOXML tables (Word) parser* describes the usage of this parser and gives some examples.

- a `CalsBuilder` builder that allows you to build tables in the CALS format.

  The tutorial *CALS tables builder* describes the usage of this builder and gives some examples.

#### Conversion options

The tables parsing and building can be parameterized using the options described below:

**Common parsing options:**
encoding (default: “utf-8”): XML encoding of the destination file.

OOXML parser options:

styles_path (default: None): Path to the stylesheet to use to resolve table styles. In an uncompressed .docx tree structure, the stylesheet path is word/styles.xml.

CALS builder options:

cals_ns (default: None): Namespace to use for CALS-like elements and attributes to generate. Set None (or “”) if you don’t want to use namespace.

cals_prefix (default: None): Namespace prefix to use for CALS-like elements and attributes to generate.


table_in_tgroup (default: False): Where should we put the table properties:

• False to insert the attributes @colsep, @rowsep, and @tabstyle in the <table> element,
• True to insert the attributes @colsep, @rowsep, and @tgroupstyle in the <tgroup> element.

tgroup_sorting (default: ["header", "footer", "body"]) List used to sort (and group) the rows in a tgroup. The sorting is done according to the row natures which is by default: ["header", "footer", "body"] (this order match the CALS DTD defaults, where the footer is between the header and the body. To move the footer to the end, you can use ["header", "body", "footer"]).

Examples of conversions

4.2.3 Formex 4 to CALS converter

Description

The convert_formex2cals() converter is a function designed to convert tables from an Formex 4 document (which respects the schema defined in Formex 4 format) in the CALS table format.

The conversion is done in the source XML document by replacing the tables of the Formex 4 format with those transformed in the CALS format. In other words, the general structure of the source XML document is retained except for tables.

The Formex2CalsConverter converter is composed of:

• a FormexParser parser that allows you to parse tables in Formex 4 format,

The tutorial Formex 4 tables parser describes the usage of this parser and gives some examples.

• a CalsBuilder builder that allows you to build tables in the CALS format.

The tutorial CALS tables builder describes the usage of this builder and gives some examples.

Conversion options

The tables parsing and building can be parameterized using the options described below:

Common parsing options:

encoding (default: “utf-8”): XML encoding of the destination file.

Formex parser options:
**formex_ns (default None):** Namespace to use for Formex elements and attributes parsing. Set None (or ””) if you don’t use namespace.

**cals_ns (default None):** Namespace to use for CALS-like elements and attributes parsing. Set None (or ””) if you don’t use namespace.

**CALS builder options:**

**cals_ns (default: None):** Namespace to use for CALS-like elements and attributes to generate. Set None (or ””) if you don’t want to use namespace.

**cals_prefix (default: None):** Namespace prefix to use for CALS-like elements and attributes to generate.


**table_in_tgroup (default: False):** Where should we put the table properties:

- False to insert the attributes @colsep, @rowsep, and @tabstyle in the <table> element.
- True to insert the attributes @colsep, @rowsep, and @tgroupstyle in the <tgroup> element.

**tgroup_sorting (default: ["header", "footer", "body"])**: List used to sort (and group) the rows in a tgroup. The sorting is done according to the row natures which is by default: ["header", "footer", "body"] (this order match the CALS DTD defaults, where the footer is between the header and the body. To move the footer to the end, you can use ["header", "body", "footer"]).

### Examples of conversions

#### 4.2.4 CALS to Formex 4 converter

**Description**

The `convert_cals2formex()` converter is a function designed to convert tables from an CALS document (which respects the schema defined in CALS table format) in the Formex 4 format.

The conversion is done in the source XML document by replacing the tables of the CALS format with those transformed in the Formex 4 format. In other words, the general structure of the source XML document is retained except for tables.

The `Cals2FormexConverter` converter is composed of:

- a `CalsParser` parser that allows you to parse tables in CALS format,
  
The tutorial `CALS tables parser` describes the usage of this parser and gives some examples.
- a `FormexBuilder` builder that allows you to build tables in the Formex 4 format.
  
The tutorial `Formex 4 tables builder` describes the usage of this builder and gives some examples.

### Conversion options

The tables parsing and building can be parameterized using the options described below:

**Common parsing options:**

- **encoding (default: “utf-8”):** XML encoding of the destination file.

**CALS parser options:**

- **cals_ns (default None):** Namespace to use for CALS elements and attributes parsing. Set None (or ””) if you don’t use namespace in your XML.
Formex 4 builder options:

use_cals (default: False): Generate additional CALS-like elements and attributes to simplify the layout of Formex document in typesetting systems.

cals_ns (default: “https://lib.benker.com/schemas/cals.xsd”): Namespace to use for CALS-like elements and attributes (requires: use_cals). Set None (or “”) if you don’t want to use namespace.

cals_prefix (default: “cals”): Namespace prefix to use for CALS-like elements and attributes (requires: use_cals).


Examples of conversions

Note: future converters are coming.

All the converters available in the benker library use parser/builder pairs. There is one parser for each XML document type to read and one builder for each XML document type to write. For example, you have the parser OOXML which will allow you to analyze the tables contained in Word documents (of type .docx); and you can choose the Formex builder to generate valid Formex 4 tables.

The following tutorials will give you some use cases for parsers and builders. You could immerse yourself in the classes usage and available options.

4.2.5 OOXML tables (Word) parser

4.2.6 Formex 4 tables parser

Description

A FormexParser is used to parse the tables (actually, it parses CORPUS elements) of a Formex 4 document and generate a Table instances (memory representation of a table). The instance can then be serialize in another XML format, like CALS.

To use this class, you need to inherit the BaseBuilder class and create an instance of your class to used in the FormexParser parser.

Of course, for the sake of this demonstration we can used an instance of the class BaseBuilder, without implementing the generate_table_tree() method.

```python
>>> from lxml import etree
>>> from benker.builders.base_builder import BaseBuilder
>>> from benker.parsers.formex import FormexParser

>>> builder = BaseBuilder()
>>> parser = FormexParser(builder)

For example, you can parse the following Formex 4 table:

```
<TBL COLS="9" NO.SEQ="0001" PAGE_SIZE="SINGLE.LANDSCAPE">
  <CORPUS>
    <ROW TYPE="HEADER">
```

(continues on next page)
<table>
<thead>
<tr>
<th>Numéro d'ordre</th>
<th>Nom chimique/DCI/XAN</th>
<th>Dénomination commune du glossaire des ingrédients</th>
<th>Numéro CAS</th>
<th>Numéro CE</th>
<th>Type de produit, parties du corps</th>
<th>Concentration maximale dans les préparations prêtes à l’emploi</th>
<th>Autres</th>
<th>Libellé des conditions d’emploi et des avertissements</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>3,3’-(1,4-phénylène)bis(5,6-diphényl-1,2,4-triazine)</td>
<td>phénylène bis-diphényltriazine</td>
<td>55514-22-2</td>
<td>700-823-1</td>
<td>5 %</td>
<td>Ne pas utiliser dans des applications pouvant conduire à l’exposition des poumons de l’utilisateur final par inhalation.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

And generate a *Table* instance:

```python
>>> tree = etree.parse("docs/_static/parsers.formex.sample1.xml")
>>> fmx_table = tree.getroot()
>>> table = parser.parse_table(fmx_table)
>>> print(table)
+-----------+-----------------------------------------------+-------------------------+
<table>
<thead>
<tr>
<th></th>
<th>Identific</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numéro d’</td>
<td>Nom chimique</td>
<td>Dénominat</td>
</tr>
<tr>
<td>ordre</td>
<td>DCI/XAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Numéro CAS</td>
<td>Numéro CE</td>
</tr>
<tr>
<td></td>
<td>Type de produit, parties du corps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Concentration maximale dans les préparations prêtes à l’emploi</td>
<td>Autres</td>
</tr>
<tr>
<td></td>
<td>Libellé des conditions d’emploi et des avertissements</td>
<td></td>
</tr>
</tbody>
</table>
```
The `FormexParser` parser accept the following options:

- **formex_ns** namespace to use for Formex elements and attributes. Usually, a Formex document has no namespace, but in some case, you can have “http://opoce”.

For instance, if you have:

```xml
<TBL COLS="2" xmlns="http://opoce">
  <CORPUS>
    <ROW TYPE="HEADER">
      <CELL COL="1">Région</CELL>
      <CELL COL="2">Vin</CELL>
    </ROW>
    <ROW>
      <CELL COL="1">Alsace</CELL>
      <CELL COL="2">Gewurztraminer</CELL>
    </ROW>
    <ROW>
      <CELL COL="1">Beaujolais</CELL>
      <CELL COL="2">Brouilly</CELL>
    </ROW>
  </CORPUS>
</TBL>
```

To parse this XML document, you can create a parser using the `formex_ns` option:

```python
>>> parser = FormexParser(builder, formex_ns="http://opoce")
>>> tree = etree.parse("docs/_static/parsers.formex.sample2.xml")
>>> fmx_table = tree.getroot()
>>> table = parser.parse_table(fmx_table)
>>> print(table)
+-----------+-----------+
<table>
<thead>
<tr>
<th>Région</th>
<th>Vin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alsace</td>
<td>Gewurztra</td>
</tr>
<tr>
<td>Beaujolais</td>
<td>Brouilly</td>
</tr>
</tbody>
</table>
```

- **cals_ns** namespace to use for CALS-like elements and attributes. For the purpose of typesetting enhancement, a Formex document may contains CALS-like elements and attributes. This elements and attributes may use a different namespace. In order to parse them, you can use the `cals_ns` options.

For instance, if you have:
<TBL COLS="2" xmlns:cals="http://my.cals.ns">
  <CORPUS cals:colsep="1" cals:frame="all" cals:pgwide="1" cals:rowsep="1">
    <cals:colspec cals:colname="c1" cals:colwidth="80mm" cals:align="left"/>
    <cals:colspec cals:colname="c2" cals:colwidth="60mm" cals:align="center"/>
    <ROW TYPE="HEADER">
      <CELL TYPE="HEADER" COL="1">Header 1</CELL>
      <CELL TYPE="HEADER" COL="2">Header 2</CELL>
    </ROW>
    <ROW cals:rowsep="0" cals:valign="middle">
      <CELL COL="1">Cell A1</CELL>
      <CELL COL="2">Cell B1</CELL>
    </ROW>
    <ROW>
      <CELL COL="1" COLSPAN="2" cals:nameend="c2" cals:namest="c1">Cell A2-B2</CELL>
    </ROW>
    <ROW>
      <CELL COL="1" ROWSPAN="2" cals:morerows="1">Cell A3-A4</CELL>
      <CELL COL="2">Cell B3</CELL>
    </ROW>
    <ROW>
      <CELL COL="2">Cell B4</CELL>
    </ROW>
  </CORPUS>
</TBL>

To parse this XML document, you can create a parser using the *cals_ns* option:

```python
>>> parser = FormexParser(builder, cals_ns="http://my.cals.ns")
>>> tree = etree.parse("docs/_static/parsers.formex.sample3.xml")
>>> fmx_table = tree.getroot()
>>> table = parser.parse_table(fmx_table)
>>> print(table)
+-----------+-----------+
| Header 1  | Header 2  |
+-----------+-----------+
| Cell A1   | Cell B1   |
+-----------+-----------+
| Cell A2-B |           |
+-----------+-----------+
| Cell A3-A | Cell B3   |
|           |           |
| Cell B4   |
+-----------+-----------+
```

### Supported values

The *FormexParser* parser can handle the following values: Formex styles.

#### 4.2.7 CALS tables parser

**Description**

A *CalsParser* is used to parse the tables (*table* elements) of a CALS file and generate a *Table* instances (memory representation of a table). The instance can then be serialize in another XML format, like HTML, Formex 4
or even CALS.

To use this class, you need to inherit the `BaseBuilder` class and create an instance of your class to used in the `CalsParser` parser.

Of course, for the sake of this demonstration we can used an instance of the class `BaseBuilder`, without implementing the `generate_table_tree()` method.

```python
>>> from lxml import etree
>>> from benker.builders.base_builder import BaseBuilder
>>> from benker.parsers.cals import CalsParser

>>> builder = BaseBuilder()
>>> parser = CalsParser(builder)
```

For example, you can parse the following CALS table:

```xml
<table frame="all">
    <tgroup cols="2" colsep="0" rowsep="1">
        <colspec colnum="1" colname="c1" colwidth="50mm" align="left"/>
        <colspec colnum="2" colname="c2" colwidth="20mm" align="right"/>
        <colspec colnum="3" colname="c3" colwidth="20mm" align="char" char="." charoff="20"></colspec>
    </tgroup>
    <thead>
        <row bgcolor="#90ee90">
            <entry valign="top">Element</entry>
            <entry valign="top">(Z)</entry>
            <entry valign="top">(A)</entry>
        </row>
    </thead>
    <tbody>
        <row bgcolor="#cdcdcd">
            <entry>Hydrogen</entry>
            <entry>1</entry>
            <entry>1.008</entry>
        </row>
        <row>
            <entry>Helium</entry>
            <entry>2</entry>
            <entry>4.0026</entry>
        </row>
        <row bgcolor="#cdcdcd">
            <entry>Lithium</entry>
            <entry>3</entry>
            <entry>6.94</entry>
        </row>
    </tbody>
</table>
```

And generate a `Table` instance:

```python
>>> tree = etree.parse("docs/_static/parsers.cals.sample1.xml")
>>> cals_table = tree.getroot()
>>> table = parser.parse_table(cals_table)
>>> print(table)
+-----------+-----------+-----------+
| Element   | (Z)       | (A)       |
+-----------+-----------+-----------+
```

(continues on next page)
Options

The CalsParser parser accepts the following options:

- `cals_ns` is used to specify a specific namespace used by your CALS tables.

For instance, if you have:

```xml
<table frame="box" xmlns="http://my.cals.ns">
  <tgroup cols="2" colsep="0" rowsep="1">
    <colspec colnum="1" colname="c1"/>
    <colspec colnum="2" colname="c2"/>
    <colspec colnum="3" colname="c3"/>
    <thead>
      <row><entry><entry>k</entry><entry>unit</entry></row>
    </thead>
    <tbody>
      <row><entry>Meter</entry><entry>km</entry><entry>m</entry></row>
      <row><entry>Liter</entry><entry>KL</entry><entry>L</entry></row>
      <row><entry>Gram</entry><entry>Kg</entry><entry>g</entry></row>
    </tbody>
  </tgroup>
</table>
```

To parse this XML document, you can create a parser using the `cals_ns` option:

```python
>>> parser = CalsParser(builder, cals_ns="http://my.cals.ns")
>>> tree = etree.parse("docs/_static/parsers.cals.sample2.xml")
>>> cals_table = tree.getroot()
>>> table = parser.parse_table(cals_table)
>>> print (table)
+-----------+-----------+-----------+
|           | k         | unit      |
+-----------+-----------+-----------+
| Meter     | km        | m         |
+-----------+-----------+-----------+
| Liter     | KL        | L         |
+-----------+-----------+-----------+
| Gram      | Kg        | g         |
+-----------+-----------+-----------+
```

Supported values

The CalsParser parser can handle the following values: CALS styles.
4.2.8 Formex 4 tables builder

4.2.9 CALS tables builder

Note: future parsers and builders are coming.

Warning: here, we are talking about tables parsers and not documents parsers: only tables are analyzed and converted into an object structure in memory. The rest of the document is not touched (only namespaces).

The Benker Library also has low-level functions in core modules. These core modules are the essential building blocks for having in memory table structures. A set of tutorials is also available for these modules.

4.2.10 Size

Description

A Size is a tuple with width, height coordinates. It represents the width and the height of a cell in a grid.

```python
>>> from benker.size import Size

>>> size = Size(4, 5)
```

The representation of a size is "(width x height)"

```python
>>> print(size)
(4 x 5)
```

Operations

You can change the size of a Size by adding, subtracting of multiplying values.

You can add two sizes, a size and a tuple (x, y), a size and a single quantity (integer):

```python
>>> Size(2, 3) + Size(3, 4)
Size(width=5, height=7)

>>> Size(2, 3) + (3, 4)
Size(width=5, height=7)

>>> Size(2, 3) + 1
Size(width=3, height=4)
```

You can subtract two sizes, a size and a tuple (x, y), a size and a single quantity (integer):

```python
>>> Size(1, 4) - Size(2, 1)
Size(width=-1, height=3)

>>> Size(1, 4) - (2, 1)
Size(width=-1, height=3)
```
You can multiply a size by an integer. This last ability is useful to reverse a size:

```python
>>> Size(3, 4) * 3
Size(width=9, height=12)
>>> Size(3, 4) * -1
Size(width=-3, height=-4)
```

You can also negate a size:

```python
>>> -Size(3, 5)
Size(width=-3, height=-5)
>>> +Size(3, 5)
Size(width=3, height=5)
```

All this operations are useful to do mathematical transformation with `Coord — Operations`, for instance, a translation of a coord is done by adding a coord and a size.

### 4.2.11 Coord

#### Description

A `Coord` is a tuple with \(x\), \(y\) coordinates. It represents the top-left origin of a cell in a grid.

```python
>>> from benker.coord import Coord
>>> coord = Coord(4, 5)
```

We use the Excel convention to represent a `Coord`: columns are represented by letters, rows are represented by numbers.

```python
>>> print(Coord(2, 5))
B5
```

#### Operations

Mathematical operations are an easy way to translate a `Coord` to another locations.

You can use a `Size` to move a coord to another position. You can also use a tuple \((x, y)\) or a single quantity (integer):

```python
>>> from benker.size import Size
>>> Coord(2, 5) + Size(1, 2)
Coord(x=3, y=7)
>>> Coord(2, 5) + (1, 2)
Coord(x=3, y=7)
>>> Coord(2, 5) + 1
Coord(x=3, y=6)
```
The translation can be positive or negative:

```python
>>> Coord(2, 5) - Size(1, 2)
Coord(x=1, y=3)

>>> Coord(2, 5) - (1, 2)
Coord(x=1, y=3)

>>> Coord(2, 5) - 1
Coord(x=1, y=4)
```

You cannot add or subtract two coordinates:

```python
>>> Coord(2, 5) + Coord(2, 1)
Traceback (most recent call last):
  ...  
TypeError: <class 'benker.coord.Coord'>

>>> Coord(2, 5) - Coord(1, 2)
Traceback (most recent call last):
  ...  
TypeError: <class 'benker.coord.Coord'>
```

Again, you cannot add a size and a coord:

```python
>>> Size(2, 5) + Coord(2, 1)
Traceback (most recent call last):
  ...  
TypeError: <class 'benker.coord.Coord'>

>>> Size(2, 5) - Coord(1, 2)
Traceback (most recent call last):
  ...  
TypeError: <class 'benker.coord.Coord'>
```

**Warning:** This constraint must be respected in order to help diagnosing conceptual errors.

### 4.2.12 Box

**Description**

A *Box* is a rectangular area defined by two coordinates:

- the top-left corner of the rectangle: the *min* coord,
- the bottom-right corner of the rectangle: the *max* coord.

The default size of a *Box* is (1, 1), so you can create a box by only specifying the top-left corner of the rectangle.

```python
>>> from benker.box import Box

>>> Box(1, 2, 2, 3)
Box(min=Coord(x=1, y=2), max=Coord(x=2, y=3))

>>> Box(1, 2)
Box(min=Coord(x=1, y=2), max=Coord(x=1, y=2))
```
You can use two coordinates to define a box:

```python
>>> from benker.coord import Coord
>>> Box(Coord(5, 6), Coord(7, 8))
Box(min=Coord(x=5, y=6), max=Coord(x=7, y=8))
```

You can specify the size of a box:

```python
>>> from benker.size import Size
>>> Box(Coord(5, 6), Size(3, 2))
Box(min=Coord(x=5, y=6), max=Coord(x=7, y=7))
```

We use the Excel convention to represent a `Box`: columns are represented by letters, rows are represented by numbers.

```python
>>> print(Box(Coord(2, 5)))
B5
>>> print(Box(Coord(2, 5), Size(3, 2)))
B5:D6
```

**Properties**

You can use the following properties to extract information from a `box`:

- use `min` to get the top-left corner coordinates,
- use `max` to get the bottom-right corner coordinates,
- use `width` to get the width of the box (number of columns),
- use `height` to get the height of the box (number of rows),
- use `size` to get the size (`width` and `height`) of the box.

```python
>>> b1 = Box(Coord(5, 6), Size(3, 2))
>>> b1.min
Coord(x=5, y=6)
>>> b1.max
Coord(x=7, y=7)
>>> b1.width
3
>>> b1.height
2
>>> b1.size
Size(width=3, height=2)
```

**Warning:** All properties are non-mutable:

```python
>>> b1.width = 9
Traceback (most recent call last):
  ...
AttributeError: can't set attribute
```
Operations

Contains

You can check if a point, defined by its coordinates (tuple \((x, y)\) or \(Coord\) instance), is contained in a box:

```python
>>> top_left = Coord(5, 6)
>>> top_right = Coord(6, 6)
>>> bottom_left = Coord(5, 8)
>>> bottom_right = Coord(6, 8)

>>> b1 = Box(top_left, bottom_right)

>>> top_left in b1
True
>>> top_right in b1
True
>>> bottom_left in b1
True
>>> bottom_right in b1
True

>>> Coord(7, 6) in b1
False

>>> (5, 7) in b1
True
```

**Warning:** Even if a `Size` object is a subtype of `tuple`, such an object cannot be “contained” in a `Box`.

```python
>>> b1 = Box(Coord(x=5, y=6), Coord(x=6, y=8))
>>> Size(5, 7) in b1
Traceback (most recent call last):
...
TypeError: <class 'benker.size.Size'>
```

You can check if a `Box` is contained in another box:

```python
>>> b1 = Box(Coord(x=5, y=6), Coord(x=6, y=8))
>>> b2 = Box(Coord(x=5, y=7), Coord(x=6, y=7))
>>> b3 = Box(Coord(x=6, y=6), Coord(x=7, y=6))

>>> b1 in b1
True
>>> b2 in b1
True
>>> b3 in b1
False
```

Intersection and Union

You can find if a `Box` intersects another `Box`:
Two boxes are disjoint if they don’t intersect each other:

```python
>>> b1.isdisjoint(b2)
False
>>> b1.isdisjoint(b3)
True
```

You can calculate the intersection of two boxes. You can use the “&” operator to do that:

```python
>>> b1.intersection(b2)
Box(min=Coord(x=2, y=2), max=Coord(x=3, y=3))
```

**Warning:** If the two boxes are disjoint, there is no intersection:

```python
>>> b1 & b3
Traceback (most recent call last):
  ...
ValueError: (Box(min=Coord(x=1, y=1), max=Coord(x=3, y=3)), Box(min=Coord(x=4, y=1),
       max=Coord(x=5, y=1)))
```

You can calculate the union of two boxes. The union of two boxes is the bounding box: You can use the “|” operator to do that:

```python
>>> b1.union(b2)
Box(min=Coord(x=1, y=1), max=Coord(x=4, y=4))
```

**Total ordering**

A total ordering is defined for the boxes. The aim is to order the cells in a grid sorted from left to right and from top to bottom. This order is useful to group the cells by rows.

You can compare boxes:

```python
>>> b1 = Box(Coord(3, 2), Coord(6, 4))
>>> b1 < b1
False
>>> b1 < Box(Coord(3, 2), Coord(6, 5))
True
>>> b1 < Box(Coord(3, 2), Coord(7, 4))
True
>>> b1 < Box(Coord(4, 2), Coord(6, 4))
(continues on next page)
You can sort boxes. The sort order can be defined as below:

- top cells are sorted before bottom cells,
- top-left cells are sorted before top-right cells,
- smaller cells are sorted before bigger.

```
>>> from random import shuffle

>>> boxes = [Box(x, y) for x in range(1, 4) for y in range(1, 3)]

>>> [str(box) for box in boxes]

>>> shuffle(boxes)

>>> [str(box) for box in sorted(boxes)]
```

### 4.2.13 Styled

**Description**

A *Styled* object contains a dictionary of styles. It is mainly used for *Table*, *RowView*, *ColView*, and *Cell*.

```
>>> from benker.styled import Styled

>>> styled = Styled({'text-align': 'justify'}, "body")
```

The representation of a styled is the representation of its dictionary of styles:

```
>>> print(styled)
{'text-align': 'justify'}
```

**Attributes**

A *Styled* object has the following attribute:

- *styles* is the user-defined styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own styles list.

**Note:** The style dictionary is always copied: in other words, key-value pairs are copied but a shallow copy is done for the values (in general, it is not a problem if you use non-mutable values like *str*).

- *nature*: a way to distinguish the body cells, from the header and the footer. The default value is “body”, but you can use “header”, "footer" or whatever is suitable for your needs. This kind of information is in general not stored in the styles, even if it is similar.
Tables can also have a nature, similar to HTML @class attribute, you can use it do identify the styles to apply to your table. For tables, the default value is None.

**Note:** In a Grid, the merging of two natures is done by keeping the first nature and dropping the second one. In other words, the resulting nature is the group of the most top-left nature of the merged cells.

Example of styles initialisation and shallow copy:

```python
>>> css = { 'border-style': 'solid', 'border-width': '5px'}
>>> one = Styled(css, "body")
>>> one.styles['border-width'] = '2px 10px 4px 20px'
>>> two = Styled(one.styles, "body")
>>> two.styles['border-width'] = 'medium'

>>> css
{'border-style': 'solid', 'border-width': '5px'}

>>> one.styles
{'border-style': 'solid', 'border-width': '2px 10px 4px 20px'}

>>> two.styles
{'border-style': 'solid', 'border-width': 'medium'}
```

### 4.2.14 Cell

**Description**

A Cell object stores the content of a Grid cell.

A cell can have styles, a dictionary of key-value properties attached to the cell.

A cell has a type to distinguish between header, body and footer cells. The default type is “body”, but you can also use “header”, “footer” or whatever...

A cell has top-left coordinates: x and y. The default coordinates is (1, 1): this is the top-left coordinate of the cell box. The coordinates x and y cannot be null: grid coordinates are 1-indexed.

A cell has a size: width and height. The default size is (1, 1), you can increase them to represent horizontal or vertical spanning. The width and the height cannot be null.

To instantiate a Cell, you can do:

```python
>>> from benker.cell import Cell

>>> c1 = Cell("c1")
>>> c2 = Cell("c2", styles={"color": "red"})
>>> c3 = Cell("c3", x=2, y=3, nature="footer")
>>> c4 = Cell("c4", width=2)
>>> c5 = Cell("c5", height=2)
```

The string representation of a cell is the string representation of it’s content:

```python
>>> for cell in c1, c2, c3, c4, c5:
...     print(cell)
... c1
```
Attributes

A cell has the following attributes:

- **content** is the user-defined cell content. It can be of any type: None, `str`, `int`, `float`, a container (`list`), a XML element, etc. The same content can be shared by several cells, it’s your own responsibility to handle the copy (or deep copy) of the `content` reference when needed.

  **Note:** In a **Grid**, the **merging** of two cell contents is done with the “+” operator (`__add__()`). You can override this by using a **content_appender**, a two-arguments function which will perform the concatenation of the two contents.

- **styles** is the user-defined cell styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own styles list.

  **Note:** The style dictionary is always copied: in other words, key-value pairs are copied but a shallow copy is done for the values (in general, it is not a problem if you use non-mutable values like `str`).

- **type** is a way to distinguish the body cells, from the header and the footer. The default value is “body”, but you can use “header”, “footer” or whatever is suitable for your needs.

  **Note:** In a **Grid**, the **merging** of two cell types is done by keeping the first cell type and dropping the second one. In other words, the resulting cell type is the type of the most top-left cell type of the merged cells.

Using the cell attributes:

```python
>>> paragraphs = ['Hello', 'How are you?']
>>> css = {'text-align': 'justify', 'vertical-align': 'top'}

>>> c1 = Cell(paragraphs, styles=css, nature="normal")
>>> c2 = Cell(paragraphs, styles=css, nature="normal")

# this will mutate the referenced *paragraphs* list:
>>> c1.content.append("I am well.")
>>> c2.content
['Hello', 'How are you?', 'I am well.']

# this will change only the cell styles:
>>> c1.styles['vertical-align'] = 'middle'
>>> c2.styles == {'text-align': 'justify', 'vertical-align': 'top'}
True
```
Properties

You can use the following properties to extract information from a cell:

- use `min` to get the top-left corner coordinates,
- use `max` to get the bottom-right corner coordinates,
- use `width` to get the width of the box (number of columns),
- use `height` to get the height of the box (number of rows),
- use `size` to get the size (width and height) of the box.

```python
>>> c1 = Cell("Hi", x=5, y=6, width=3, height=2)
>>> c1.min
Coord(x=5, y=6)
>>> c1.max
Coord(x=7, y=7)
>>> c1.width
3
>>> c1.height
2
>>> c1.size
Size(width=3, height=2)
```

**Warning:** All properties are non-mutable:

```python
>>> c1.width = 9
Traceback (most recent call last):
...
AttributeError: can't set attribute
```

Operations

Contains

You can check if a point, defined by its coordinates (tuple `(x, y)` or `Coord` instance), is contained in a Cell. A cell contains a point if it is in its Box. This rule is explained in detail in the section `Box – Contains`.

```python
>>> c1 = Cell("A", x=2, y=3, width=2, height=1)
>>> (3, 3) in c1
True
>>> (7, 9) in c1
False
```

Total ordering

A total ordering is defined for the cells. The aim is to order the cells in a grid sorted from left to right and from top to bottom. This order is useful to group the cells by rows. The total ordering is base on the Box Total ordering.
```python
>>> c1 = Cell("one")
>>> c2 = Cell("two", x=2)
>>> c3 = Cell("three", y=2)

>>> c1 < c2 < c3
True
```

This total ordering allow us to sort the cells:

```python
>>> from random import shuffle

>>> cells = [c1, c2, c3]
>>> shuffle(cells)

>>> [str(cell) for cell in sorted(cells)]
['one', 'two', 'three']
```

**Transformations**

It is possible to change the cell position and size by using two kind of transformations:

- Move a cell to a different coordinates,
- Resize a cell.

```python
>>> from benker.coord import Coord
>>> from benker.size import Size

>>> c1 = Cell("A")

>>> c1
<Cell('A', styles={}, nature=None, x=1, y=1, width=1, height=1)>

>>> c1.move_to(Coord(2, 3))
<Cell('A', styles={}, nature=None, x=2, y=3, width=1, height=1)>

>>> c1.resize(Size(3, 4))
<Cell('A', styles={}, nature=None, x=1, y=1, width=3, height=4)>

>>> c1.transform(Coord(2, 3), Size(3, 4))
<Cell('A', styles={}, nature=None, x=2, y=3, width=3, height=4)>
```

The transformation functions don’t change the current cell but produce a new one with new coordinates/size.

### 4.2.15 Grid

**Description**

A *Grid* is a collection of *Cell* objects ordered in a grid of rows and columns.

You can define a empty grid like this:

```python
>>> from benker.grid import Grid

>>> Grid()
Grid([])
```
You can also define a grid from a collection (list, set...) of cells. Cells are ordered according to the total ordering of the cell boxes:

```python
>>> from benker.cell import Cell

>>> red = Cell('red', x=1, y=1, height=2)
>>> pink = Cell('pink', x=2, y=1, width=2)
>>> blue = Cell('blue', x=2, y=2)

>>> grid = Grid([red, blue, pink])
>>> for cell in grid:
...    print(cell)
red
pink
blue
```

**Warning:** If at least one cell intersect another one, an exception is raised:

```python
>>> Grid([Cell("one"), Cell("two")])
Traceback (most recent call last):
  ...
KeyError: Coord(x=1, y=1)
```

So, it is important to define the coordinates of the cells.

It's easy to copy the cells of another grid.

Remember that:

- cells are copied (not shared between grids),
- cell contents are shared: two different cells share the same content,
- cell styles are copied (but not deeply).

```python
>>> grid1 = Grid([red, blue, pink])
>>> grid2 = Grid(grid1)

>>> tuple(id(cell) for cell in grid1) != tuple(id(cell) for cell in grid2)
True
>>> tuple(id(cell.content) for cell in grid1) == tuple(id(cell.content) for cell in grid2)
True
>>> tuple(id(cell.styles) for cell in grid1) != tuple(id(cell.styles) for cell in grid2)
True
```

You can pretty print a grid:

```python
>>> grid = Grid([red, blue, pink])

>>> print(grid)
+-----------+-----------------------+
| red | pink |
| +-----------+-----------------------+
| blue | |
+---------------------------------------------------------------+
```

4.2. Tutorials 37
Properties

The bounding box of a grid is the bounding box of all cells:

```python
>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[3, 2] = Cell("gray")
>>> print(grid)
+-----------+-----------------------+
| red       | pink                  |
| +-----------+-----------------------+
|           | gray                  |
+-----------+-----------------------+
```

```python
>>> grid.bounding_box
Box(min=Coord(x=1, y=1), max=Coord(x=3, y=2))
```

**Important:** The bounding box is not defined for an empty grid, so `None` is returned in that case (this behavior is preferable to raising an exception, in order to simplify interactive debugging).

```python
>>> grid = Grid()
>>> grid.bounding_box is None
True
```

Operations

Contains

You can check if a point, defined by its coordinates (tuple `(x, y)` or `Coord` instance), is contained in a `Grid`.

The rule is simple: a grid contains a point if it exists a `Cell` of the grid which contains that point. In other words, a point may be contained in the bounding box of a grid but not in any cell if there are some gaps in the grid.

```python
>>> from benker.coord import Coord

>>> red = Cell('red', x=1, y=1, height=2)
>>> pink = Cell('pink', x=2, y=1, width=2)
>>> blue = Cell('blue', x=2, y=2)
>>> grid = Grid([red, blue, pink])

>>> (1, 1) in grid
True
>>> (3, 1) in grid
True
>>> (4, 1) in grid
False
>>> (3, 2) in grid
False
>>> Coord(1, 2) in grid
True
```
Set, Get, Delete cells

A grid is a `MutableMapping`, it works like a dictionary of cells. Keys of the dictionary are coordinates (tuple `(x, y)` or `Coord` instance). The coordinates are the top-left coordinates of the cells.

```python
>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue")
>>> grid[3, 2] = Cell("gray")
```  
```text
<table>
<thead>
<tr>
<th>red</th>
<th>pink</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>blue</td>
<td>gray</td>
</tr>
</tbody>
</table>
+-----+------+
```

**Warning:** Unlike a `dict`, you cannot set a cell to a given location if a cell already exist in that location, an exception is raised in that case.

```python
>>> grid[3, 1] = Cell("purple")
Traceback (most recent call last):
...  
KeyError: Coord(x=3, y=1)
```

You can get a cell at a given location:

```python
>>> grid[1, 1]
<Cell('red', styles={}, nature=None, x=1, y=1, width=1, height=2)>
>>> grid[3, 1]
<Cell('pink', styles={}, nature=None, x=2, y=1, width=2, height=1)>
```

You can delete a cell at a given location:

```python
>>> del grid[3, 1]
>>> print(grid)
+-----------+-----------+-----------+
| red | | |
| +-----------+-----------+
| | blue | gray |
+-----------+-----------+-----------+
```

Merging/expanding

It is possible to merge several cells in the grid. The merging takes the `start` coordinates and the `end` coordinates of the cells to merge.

We can define a `content_appender` to give the content merging operation to use to merge several cell contents.

```python
>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink")
```
>>> grid[3, 1] = Cell("blue")
>>> print(grid)
+-----------+-----------+-----------+
| red | pink | blue |
| +-----------+-----------+
| | | |
+-----------+-----------+-----------+

>>> grid.merge((2, 1), (3, 1), content_appender=lambda a, b: "/".join([a, b]))
<Cell('pink/blue', styles={}, nature=None, x=2, y=1, width=2, height=1)>
>>> print(grid)
+-----------+-----------------------+
| red | pink/blue |
| +-----------+-----------+
| | | |
+-----------+-----------+-----------+

**Warning:** All cells in the bounding box of the merging must be inside of the bounding box. In other words, the bounding box of the merging must not intersect any cell in the grid.

>>> grid.merge((1, 2), (2, 2))
Traceback (most recent call last):
  ... 
ValueError: ((1, 2), (2, 2))

Similar to the merging, you can expand the size of a cell:

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink")
>>> grid[3, 1] = Cell("blue")
>>> print(grid)
+-----------+-----------+-----------+
| red | pink | blue |
| +-----------+-----------+
| | | |
+-----------+-----------+-----------+

>>> grid.expand((2, 1), height=1)
<Cell('pink', styles={}, nature=None, x=2, y=1, width=1, height=2)>
>>> print(grid)
+-----------+-----------+-----------+
| red | pink | blue |
| | +-----------+
| | | |
+-----------+-----------+-----------+

**Iterators**

You can iterate the cells of a grid:

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("hot", width=2)
>>> grid[2, 2] = Cell("chili")
>>> grid[3, 2] = Cell("peppers")
>>> grid[1, 3] = Cell("Californication", width=3)

>>> print(grid)
+-----------+-----------------------+
| red | hot |
| +-----------+-----------+
| | chili | peppers |
+-----------------------------------+
| Californi |
+-----------------------------------+

>>> for cell in grid:
...   print(cell)
red
hot
chili
peppers
Californication

You can iterate over the grid rows with the method `iter_rows()`. Each row is a `tuple` of cells:

```python
>>> for row in grid.iter_rows():
...   print(" / ".join(cell.content for cell in row))
red / hot
chili / peppers
Californication
```

### 4.2.16 Table

**Description**

A `Table` is a data structure used to represent Office Open XML tables, CALS tables or HTML tables.

A `Table` is a `Styled` object, so you can attach a dictionary of styles and a nature ("body" by default). The nature is used to give a default value to the the row/column views.

```python
>>> from benker.table import Table
>>> Table(styles={'frame': 'all'})
<Table({}, 'header')>
```

A table can be initialize with a collection of cells. Make sure all cells are disjoints.

```python
>>> from benker.cell import Cell
>>> red = Cell('red', x=1, y=1, height=2)
>>> pink = Cell('pink', x=2, y=1, width=2)
>>> blue = Cell('blue', x=2, y=2)

>>> table = Table([red, pink, blue], nature='header')
>>> table
<Table({}, 'header')>
```
>>> print(table)
+-----------+-----------------------+
| red | pink |
| +-----------+-----------+
| | blue | |
+-----------+-----------+

Warning: Make sure all cells are disjoints:

>>> red = Cell('overlap', x=1, y=1, width=2)
>>> pink = Cell('oops!', x=2, y=1)
>>> Table([red, pink])
Traceback (most recent call last):
...  
  KeyError: Coord(x=2, y=1)

Properties

You can use the following properties to extract information from a table:

The bounding box of a table is the bounding box of all cells in the grid:

>>> red = Cell('red', x=1, y=1, height=2)
>>> pink = Cell('pink', x=2, y=1, width=2)
>>> blue = Cell('blue', x=2, y=2)
>>> table = Table([red, pink, blue])

>>> table.bounding_box
Box(min=Coord(x=1, y=1), max=Coord(x=3, y=2))

Important: The bounding box is not defined for an empty table, so None is returned in that case (this behavior is preferable to raising an exception, in order to simplify interactive debugging).

>>> table = Table()
>>> table.bounding_box is None
True

Operations

Cells Insertion

You can insert a row to a table. This row is then used to insert cells.

>>> table = Table()

>>> row = table.rows[1]

>>> row.nature = "header"

>>> row.insert_cell("Astronomer", width=2)
The nature of a cell is inherited from its parent's row. The first row contains the header, so the cell nature is "header":

```python
>>> table.rows[1].nature
'header'
```

```python
>>> [cell.nature for cell in table.rows[1].owned_cells]
['header', 'header', 'header']
```

The other rows have no nature, so the cell nature is None

```python
>>> table.rows[2].nature is None
True
```

```python
>>> all(cell.nature is None for cell in table.rows[2].owned_cells)
True
```

### Cells Merging

You can merge cells by giving the coordinates of the cells to merge or by extending the size of a given cell.
Owned and caught cells

When a cell is merged into a row group, it is always bound to the top row of this group (the first row). In that case, we say that the first row **owns** the cell and the other rows **catch** the cell.
Here are the **owned_cells** of this table:

```python
>>> for pos, row in enumerate(table.rows, 1):
...    cells = ', '.join('{}'.format(cell) for cell in row.owned_cells)
...    print("row #{pos}: {cells}".format(pos=pos, cells=cells))
row #1: merged, A
row #2: B
row #3: C, D
```

Here are the **caught_cells** of this table:

```python
>>> for pos, row in enumerate(table.rows, 1):
...    cells = ', '.join('{}'.format(cell) for cell in row.caught_cells)
...    print("row #{pos}: {cells}".format(pos=pos, cells=cells))
row #1: merged, A
row #2: merged, B
row #3: C, D
```

The same applies to columns: if a cell is merged into several columns then it belongs to the first column (left) of the merged column group.

```python
>>> table = Table()
>>> row = table.rows[1]
>>> row.insert_cell("merged", width=2)
>>> row.insert_cell("A")
>>> row = table.rows[2]
>>> row.insert_cell("B")
>>> row.insert_cell("C")
>>> row.insert_cell("D")
>>> print(table)
+-----------------------+-----------+
| merged | A |
+-----------+-----------+-----------+
| B | C | D |
+-----------+-----------+-----------+
```

Here are the **owned_cells** of this table:

```python
>>> for pos, col in enumerate(table.cols, 1):
...    cells = ', '.join('{}'.format(cell) for cell in col.owned_cells)
...    print("col #{pos}: {cells}".format(pos=pos, cells=cells))
col #1: merged, merged, B
col #2: C
col #3: A, D
```

Here are the **caught_cells** of this table:

```python
>>> for pos, col in enumerate(table.cols, 1):
...    cells = ', '.join('{}'.format(cell) for cell in col.caught_cells)
...    print("col #{pos}: {cells}".format(pos=pos, cells=cells))
```

(continues on next page)
Fill missing cells

When you build a table (from an uncontrolled source), you may have missing cells (holes). For instance, in the table below, the cell C2 is missing:

```python
>>> table = Table()
>>> table.rows[1].insert_cell("one")
>>> table.rows[1].insert_cell("two")
>>> table.rows[1].insert_cell("three")
>>> table.rows[1].insert_cell("four", height=2)
>>> table.rows[2].insert_cell("un-deux", width=2)
>>> print(table)
+-----------+-----------+-----------+-----------+
<table>
<thead>
<tr>
<th>one</th>
<th>two</th>
<th>three</th>
<th>four</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>un-deux</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
```

If you need to fill the missing cells, you can use the `fill_missing()` method, like this:

```python
>>> table.fill_missing(table.bounding_box, "HERE")
>>> print(table)
+-----------+-----------+-----------+-----------+
<table>
<thead>
<tr>
<th>one</th>
<th>two</th>
<th>three</th>
<th>four</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HERE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

4.3 API

Benker - Easily convert your CALS, HTML, Formex 4, Office Open XML (docx) tables from one format to another.

4.3.1 Cell Size

A `Size` object is used to represent the `width` and `height` of a `Cell`. The `width` is the number of spanned columns and the `height` is the number of spanned rows. The default cell size is (1, 1).

This module defines the `Size` tuple and give some classic use cases.

```python
class benker.size.Size
    Bases: benker.size.SizeType

    Size of a cell: width is the number of columns and height is the number of row.

    Usage:
    >>> from benker.size import Size
```
size = Size(2, 1)
size
Size(width=2, height=1)
size.width
2
size.height
1
str(size)
'(2 x 1)'

You can use the “+” or “-” operators to increase or decrease the size:

Size(2, 1) + Size(3, 3)
Size(width=5, height=4)
Size(5, 4) - Size(3, 3)
Size(width=2, height=1)

You can expand the width and height to a given factor using the “*” operator:

Size(2, 1) * 2
Size(width=4, height=2)

You can have negative or positive sizes using the unary operators “-” and “+”:

+Size(3, 2)
Size(width=3, height=2)
Size(width=-3, height=-2)

Note: A Cell object cannot have a negative or null sizes, but you can need this values for calculation, for instance when you want to reduce the cell size.

classmethod from_value(value)
Convert a value of type tuple to a Size tuple.

Parameters
value – tuple of two integers or Size tuple.

Returns
Newly created object.

Raises
TypeError – if the value is not a tuple of integers nor a Size tuple.

class benker.size.SizeType(width, height)
Bases: tuple

height
Alias for field number 1

width
Alias for field number 0

4.3.2 Cell Coordinates

A Coord object is used to represent the x and y positions of a Cell. The x is the left position (column number) and the y is the top position (row number). The default cell coordinates is (1, 1).

This module defines the Coord tuple and give some classic use cases.
class benker.coord.Coord

Bases: benker.coord.CoordTuple

Coordinates of a cell in a grid: $x$ is the left column, $y$ if the top row.

Usage:

```python
>>> from benker.coord import Coord

>>> coord = Coord(5, 3)

>>> coord
Coord(x=5, y=3)

>>> coord.x
5

>>> coord.y
3

>>> str(coord)
'E3'
```

You can use the “+” or “-” operators to move the coordinates:

```python
>>> from benker.size import Size

>>> Coord(2, 1) + Size(3, 3)
Coord(x=5, y=4)

>>> Coord(5, 4) - Size(3, 3)
Coord(x=2, y=1)
```

**Warning:** You cannot add or subtract two coordinates, only a coordinate and a size.

```python
>>> from benker.coord import Coord

>>> Coord(2, 1) + Coord(3, 3)
Traceback (most recent call last):
...
TypeError: <class 'benker.coord.Coord'>
```

classmethod from_value(value)

Convert a value of type tuple to a Coord tuple.

**Parameters** value – tuple of two integers or Coord tuple.

**Returns** Newly created object.

**Raises** TypeError – if the value is not a tuple of integers nor a Coord tuple.

class benker.coord.CoordTuple(x, y)

Bases: tuple

$x$

Alias for field number 0

$y$

Alias for field number 1

### 4.3.3 Box

A Box is a rectangular area defined by two coordinates:
• the top-left corner of the rectangle: the min coord,
• the bottom-right corner of the rectangle: the max coord.

To instantiate a Box, you can do:

```python
>>> b1 = Box(Coord(5, 6), Coord(7, 8))
>>> b2 = Box(Coord(5, 6))
>>> b3 = Box(1, 2, 2, 3)
>>> b4 = Box(1, 2)
>>> b5 = Box(b1)
```

Box objects have a string representation à la Excel:

```python
>>> for box in b1, b2, b3, b4, b5:
...     print(box)
E6:G8
E6
A2:B3
A2
E6:G8
```

You can calculate the width and height of boxes:

```python
>>> b1 = Box(Coord(5, 6), Coord(6, 8))
>>> b1.width, b1.height
(2, 3)
```

```python
>>> b2 = Box(Coord(5, 6))
>>> b2.width, b2.height
(1, 1)
```

You can determine if a Coord is included in a Box:

```python
>>> top_left = Coord(5, 6)
>>> top_right = Coord(6, 6)
>>> bottom_left = Coord(5, 8)
>>> bottom_right = Coord(6, 8)

>>> b1 = Box(top_left, bottom_right)

>>> top_left in b1
True
>>> top_right in b1
True
>>> bottom_left in b1
True
>>> bottom_right in b1
True
>>> Coord(7, 6) in b1
False
>>> (5, 7) in b1
True
```

You can determine if two boxes intersect each other, or are disjoints:
```python
tests.py
>>> b1 = Box(Coord(5, 6), Coord(6, 8))
>>> b2 = Box(Coord(6, 6), Coord(6, 7))
>>> b3 = Box(Coord(7, 6), Coord(7, 8))
>>> b2.intersect(b3)
False
>>> b1.isdisjoint(b2)
False
>>> b2.isdisjoint(b1)
False
>>> b1.isdisjoint(b3)
True
>>> b3.isdisjoint(b1)
True
```

```python

class benker.box.Box

    Bases: benker.box.BoxTuple

    A Box is a rectangular area defined by two coordinates:
    • the top-left corner of the rectangle: the min coord,
    • the bottom-right corner of the rectangle: the max coord.

    Usage:

    >>> from benker.box import Box
    >>> box = Box(1, 1, 5, 3)
    >>> box
    Box(min=Coord(x=1, y=1), max=Coord(x=5, y=3))

    height

    intersect(that)

    intersection(*others)

        Return the intersection of self and all the boxes.

        Usage:

        >>> from benker.box import Box
        >>> from benker.coord import Coord
        >>> b1 = Box(Coord(3, 2), Coord(6, 4))
        >>> b2 = Box(Coord(4, 3), Coord(5, 7))
        >>> b1.intersection(b2)
        Box(min=Coord(x=4, y=3), max=Coord(x=5, y=4))
        >>> b1 & b2
        Box(min=Coord(x=4, y=3), max=Coord(x=5, y=4))

        Parameters others – collections of boxes

        Returns The inner box of all the boxes.

        Raises ValueError – if the two boxes are disjoint.

    isdisjoint(that)

    move_to(coord)
```

50 Chapter 4. Usage
resize \( (size) \)

size

transform \((\text{coord=\text{None}}, \text{size=\text{None}})\)

union \((\ast \text{others})\)

Return the union of \text{self} and all the \text{boxes}.

Usage:

```python
>>> from benker.box import Box
>>> from benker.coord import Coord

>>> b1 = Box(Coord(3, 2), Coord(6, 4))
>>> b2 = Box(Coord(4, 3), Coord(5, 7))
>>> b1.union(b2)
Box(min=Coord(x=3, y=2), max=Coord(x=6, y=7))

>>> b1 | b2
Box(min=Coord(x=3, y=2), max=Coord(x=6, y=7))
```

**Parameters** others – collections of boxes

**Returns** The bounding box of all the boxes.

width

class benker.box.BoxTuple \((\text{min, max})\)

\text{Bases: tuple}

max

Alias for field number 1

min

Alias for field number 0

4.3.4 Styled object

A \text{Styled} object contains a dictionary of styles.

It is mainly used for \text{Table}, \text{RowView}, \text{ColView}, and \text{Cell}.

class benker.styled.Styled \((\text{styles, nature})\)

\text{Bases: object}

Styled object, like Table, Row, Column, or Cell objects.

A styled object stores user-defined styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own list of styles.

**Note:** The style dictionary is always copied: in other words, key-value pairs are copied but a shallow copy is done for the values (in general, it is not a problem if you use non-mutable values like \text{str}).

A styled object stores a nature: a way to distinguish the body cells, from the header and the footer. The default value is \text{None}, but you can use “body”, “header”, “footer” or whatever is suitable for your needs. This kind of information is in general not stored in the styles, even if it is similar.
Tables can also have a nature, similar to HTML @class attribute, you can use it to identify the styles to apply to your table.

**Note:** In a Grid, the merging of two natures is done by keeping the first nature and dropping the second one. In other words, the resulting nature is the group of the most top-left nature of the merged cells.

### nature

Cell nature used to distinguish the body cells, from the header and the footer.

### styles

Dictionary of key-value pairs, where keys are the style names.

## 4.3.5 Grid Cell

A Cell object stores the content of a Grid cell.

A cell can have styles, a dictionary of key-value properties attached to the cell.

A cell has a nature to distinguish between header, body and footer cells. The default nature is `None`, but you can also use “body”, “header”, “footer” or whatever...

A cell has top-left coordinates: x and y. The default coordinates is (1, 1): this is the top-left coordinate of the cell box. The coordinates x and y cannot be null: grid coordinates are 1-indexed.

A cell has a size: width and height. The default size is (1, 1), you can increase them to represent horizontal or vertical spanning. The width and the height cannot be null.

To instantiate a Cell, you can do:

```python
>>> c1 = Cell("c1")
>>> c2 = Cell("c2", styles={'color': 'red'})
>>> c3 = Cell("c3", nature="footer")
>>> c4 = Cell("c4", width=2)
>>> c5 = Cell("c5", height=2)
```

The string representation of a cell is the string representation of it’s content:

```python
>>> for cell in c1, c2, c3, c4, c5:
...     print(cell)
```

On initialization, the cell min position is always (1, 1), a.k.a. the top-left.

```python
>>> c1 = Cell("c1")
>>> c1.min
Coord(x=1, y=1)
>>> c1.size
Size(width=1, height=1)
>>> c1.box
Box(min=Coord(x=1, y=1), max=Coord(x=1, y=1))
```

A cell can be moved to another position:
```python
>>> c1 = Cell("c1", width=3, height=2)
>>> c2 = c1.move_to(Coord(5, 3))
>>> c2.min
Coord(x=5, y=3)
>>> c2.size
Size(width=3, height=2)
>>> c2.box
Box(min=Coord(x=5, y=3), max=Coord(x=7, y=4))
```

You can check if a coord is inside the box:

```python
>>> c1 = Cell("c1", width=3, height=2)
>>> c2 = c1.move_to(Coord(5, 3))
>>> (7, 4) in c2
True
>>> Coord(6, 3) in c2
True
>>> Box(6, 3, 7, 4) in c2
True
```

class benker.cell.Cell (content, styles=None, nature=None, x=1, y=1, width=1, height=1)

Bases: benker.styled.Styled

Cell of a grid.

Variables

- **content** – user-defined cell content. It can be of any type: None, str, int, float, a container (list), a XML element, etc. The same content can be shared by several cells, it’s your own responsibility to handle the copy (or deep copy) of the content reference when needed.

  **Note:** In a Grid, the merging of two cell contents is done with the “+” operator (__add__()). You can override this by using a content_appender, a two-arguments function which will perform the concatenation of the two contents.

  Changed in version 0.4.2: The default value of nature is None (instead of “body”).

- **box**

  Bounding box of the cell.

- **content**

- **height**

  Height of the cell – rows spanning.

- **max**

  Maximum coordinates of the cell – bottom-right coordinates.

- **min**

  Minimum coordinates of the cell – top-left coordinates.

- **move_to** (coord)

  Move the cell to the given coordinates.

  **See:** transform().

  **Parameters**

  - coord (tuple[int, int] or benker.coord.Coord) – new top-left coordinates of the cell, by default it is unchanged.

  **Return type** benker.cell.Cell

  **Returns** a copy of this cell after transformation.

4.3. API
resize(size)
    Resize the cell to the given size.
    See: transform().

    Parameters size(tuple[int, int] or benker.size.Size) – new size of the cell, by default it is unchanged.
    
    Return type benker.cell.Cell
    
    Returns a copy of this cell after transformation.

size
    Size of the cell – (width, height).

transform(coord=None, size=None)
    Transform the bounding box of the cell by making a move and a resize.

    Parameters
    • coord(tuple[int, int] or benker.coord.Coord) – new top-left coordinates of the cell, by default it is unchanged.
    • size(tuple[int, int] or benker.size.Size) – new size of the cell, by default it is unchanged.

    Return type benker.cell.Cell
    
    Returns a copy of this cell after transformation.

width
    Width of the cell – columns spanning.

benker.cell.get_content_text(content)
    Try hard to extract a good string representation of the cell content.

    >>> from benker.cell import get_content_text
         

    >>> get_content_text(None) == ''
    True
    >>> get_content_text('') == ''
    True
    >>> print(get_content_text('Hi'))
    Hi
    >>> print(get_content_text(True))
    True
    >>> print(get_content_text(123))
    123
    >>> print(get_content_text(3.14))
    3.14
    >>> get_content_text([None, None]) == ''
    True
    >>> print(get_content_text(['hello', ' ', 'world!']))
    hello world!

    Parameters content – Cell content.

    Returns
    the cell text:
    • if the content is None: returns "",

54 Chapter 4. Usage
• if the content is a string: return the string unchanged,
• if the content is a number: return the string representation of the number,
• if the content is a list of strings, return the concatenated strings (None is ignored),
• if the content is a list of XML nodes, return the concatenated strings of the elements (the processing-instruction and the comments are ignored),
• else: return a concatenation of the string representation of the content items.

New in version 0.4.1.
Changed in version 0.5.1: Returns the XML Element text and tail text.

4.3.6 Grid

A grid of cells.
Example of grid:

```python
>>> from benker.grid import Grid
>>> from benker.cell import Cell

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue")

>>> print(grid)
+-----------+-----------------------+
| red | pink |
| +-----------+-----------+
| | blue | |
| +-----------+-----------+
```

You can retrieve the grid cells as follow:

```python
>>> from benker.grid import Grid
>>> from benker.cell import Cell

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue")

>>> grid[1, 1]
<Cell('red', styles={}, nature=None, x=1, y=1, width=1, height=2)>
>>> grid[2, 1]
<Cell('pink', styles={}, nature=None, x=2, y=1, width=2, height=1)>
>>> grid[2, 2]
<Cell('blue', styles={}, nature=None, x=2, y=2, width=1, height=1)>
>>> grid[3, 3]
Traceback (most recent call last):
  ...
KeyError: Coord(x=3, y=3)
```
A grid has a bounding box, useful to get the grid sizes:

```python
>>> from benker.grid import Grid
>>> from benker.cell import Cell

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue")

>>> grid.bounding_box
Box(min=Coord(x=1, y=1), max=Coord(x=3, y=2))
>>> grid.bounding_box.size
Size(width=3, height=2)
```

You can expand the cell size horizontally or vertically:

```python
>>> from benker.grid import Grid
>>> from benker.cell import Cell

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue")

>>> grid.expand((2, 2), width=1)
<Cell('blue', styles={}, nature=None, x=2, y=2, width=2, height=1)>
```

The content of the merged cells is merged too:

```python
>>> from benker.grid import Grid
>>> from benker.cell import Cell

>>> grid = Grid()
>>> grid[1, 1] = Cell("red", height=2)
>>> grid[2, 1] = Cell("pink", width=2)
>>> grid[2, 2] = Cell("blue", width=2)

>>> grid.merge((2, 1), (3, 2), content_appender=lambda a, b: "/".join([a, b]))
<Cell('pink/blue', styles={}, nature=None, x=2, y=1, width=2, height=2)>
```

```python
class benker.grid.Grid(cells=none)
Bases: collections.abc.MutableMapping

Collection of Cell objects ordered in a grid of rows and columns.

bounding_box
```
Bounding box of the grid (None if the grid is empty).

**expand** (*coord*, *width=0*, *height=0*, *content_appender=None*)

Expand (or shrink) the width and/or height of a cell, using the *content_appender* to append cell contents.

See also the method **merge()** to merge a group of cells contained in a bounding box.

**Parameters**

- *coord* – Coordinates of the cell to expand (or shrink).
- *width* – Number of columns to add to the cell width.
- *height* – Number of rows to add to the cell height.
- *content_appender* – Function to use to append the cell contents. The function must have the following signature: `f(a, b) -> c`, where `a`, `b` and `c` must be of the same type than the cell content. If not provided, the default function is `operator.__add__()`.

**Returns**

The merged cell.

**Raises** **ValueError** – If the group of cells is empty or if cells cannot be merged.

**iter_rows** ()

Iterate the cells grouped by rows.

**merge** (*start*, *end*, *content_appender=None*)

Merge a group of cells contained in a bounding box, using the *content_appender* to append cell contents.

The coordinates *start* and *end* delimit a group of cells to merge.

**Warning:** All the cells of the group must be included in the group bounding box, no intersection is allowed. If not, **ValueError** is raised.

See also the method **expand()** to expand (or shrink) the width and/or height of a cell.

**Parameters**

- *start* (*Coord* or *tuple[int, int]*) – Top-left coordinates of the group of cells to merge.
- *end* – Bottom-right coordinates of the group of cells to merge (inclusive).
- *content_appender* – Function to use to append the cell contents. The function must have the following signature: `f(a, b) -> c`, where `a`, `b` and `c` must be of the same type than the cell content. If not provided, the default function is `operator.__add__()`.

**Returns**

The merged cell.

**Raises** **ValueError** – If the group of cells is empty or if cells cannot be merged.

### 4.3.7 Table

Generic table structure which simplify the conversion from docx table format to CALS or HTML tables.

**class** `benker.table.ColView (table, pos, styles=None, nature=None)`

**Bases:** `benker.table.TextView`

A view on the table cells for a given column.

**can_catch** (*cell*)

Check if a cell can be caught by that view.
Parameters `cell` (*benker.cell.Cell*) – table cell

Returns True if the cell intercept to this view.

can_own (*cell*)
Check if a cell can be stored it that view.

Parameters `cell` (*benker.cell.Cell*) – table cell

Returns True if the cell belong to this view.

col_pos
Column position in the table (1-based).

`insert_cell` (*content*, `styles=None`, `nature=None`, `width=1`, `height=1*)
Insert a new cell in the column at the next free position, or at the end.

Parameters

- `content` – User-defined cell content. It can be of any type: None, str, int, float, a container (list), a XML element, etc. The same content can be shared by several cells, it’s your own responsibility to handle the copy (or deep copy) of the content reference when needed.
- `styles` (*typing.Dict[str, str]*) – User-defined cell styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own list of styles.
- `width` (*int*) – Width of the cell (columns spanning), default to 1.
- `height` (*int*) – Height of the cell (rows spanning), default to 1.

Variables `nature` – a way to distinguish the body cells, from the header and the footer. The default value is None, but you can use “body”, “header”, “footer” or whatever is suitable for your needs. If set to None, the cell nature is inherited from the column nature.

Changed in version 0.4.2: The nature of a cell is inherited from its parent’s column.

class *benker.table.RowView* (*table*, `pos`, `styles=None`, `nature=None`)

A view on the table cells for a given row.

can_catch (*cell*)
Check if a cell can be caught by that view.

Parameters `cell` (*benker.cell.Cell*) – table cell

Returns True if the cell intercept to this view.

can_own (*cell*)
Check if a cell can be stored it that view.

Parameters `cell` (*benker.cell.Cell*) – table cell

Returns True if the cell belong to this view.

`insert_cell` (*content*, `styles=None`, `nature=None`, `width=1`, `height=1*)
Insert a new cell in the row at the next free position, or at the end.

Parameters

- `content` – User-defined cell content. It can be of any type: None, str, int, float, a container (list), a XML element, etc. The same content can be shared by several cells,
it’s your own responsibility to handle the copy (or deep copy) of the content reference when needed.

- **styles** *(typing.Dict[str, str]*) – User-defined cell styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own list of styles.

- **width** *(int)* – Width of the cell (columns spanning), default to 1.

- **height** *(int)* – Height of the cell (rows spanning), default to 1.

**Variables** *nature* – a way to distinguish the body cells, from the header and the footer. The default value is None, but you can use “body”, “header”, “footer” or whatever is suitable for your needs. If set to None, the cell nature is inherited from the row nature.

Changed in version 0.4.2: The nature of a cell is inherited from its parent’s row.

**row_pos**
Row position in the table (1-based).

**class** benker.table.Table *(cells=None, styles=None, nature=None)*

*Bases:* benkerstyled.Styled, collections.abc.MutableMapping

Table data structure used to simplify conversion to CALS or HTML.

Short demonstration:

```python
global import Cell
global from benker.cell import
from benker.cell import Cell
global from benker.table import
from benker.table import Table
global table = Table(styles={"frame": "all"})
global table[1, 1] = Cell("one")
global table.rows[1].insert_cell("two")
global table[2, 1] = Cell("two", styles={}, nature=None, x=2, y=1, width=1, height=1)
global table.cols[1].insert_cell("alpha")
global table.cols[2].insert_cell("beta")
global (1, 2) in table
True
global del table[(1, 2)]
global (1, 2) in table
False
global len(table)
3
global for cell in table:
... print(cell)
one
two
beta
global for row in table.rows:
... print(row)
[one, two]
[beta]
```
bounding_box
   Bounding box of the table (None if the table is empty).

   Return type  benker.box.Box

   Returns  The bounding box.

cols
   List of views of type ColView

expand(coord, width=0, height=0, content_appender=None)

fill_missing(bounding_box, content, styles=None, nature=None)
   Fill the missing cells in the table.

   This method is useful when some rows has missing cells (holes).

   Parameters

   • bounding_box (Box) – The bounding box delimiting the cells/rows to fill if missing.

   • content – User-defined cell content. It can be of any type: None, str, int, float, a container (list), a XML element, etc. The same content can be shared by several cells, it’s your own responsibility to handle the copy (or deep copy) of the content reference when needed.

   • styles (typing.Dict[str, str]) – User-defined cell styles: a dictionary of key-value pairs. This values are useful to store some HTML-like styles (border-style, border-width, border-color, vertical-align, text-align, etc.). Of course, we are not tied to the HTML-like styles, you can use your own list of styles.

   Variables  nature – a way to distinguish the body cells, from the header and the footer. The default value is None, but you can use “body”, “header”, “footer” or whatever is suitable for your needs. If set to None, the cell nature is inherited from the row nature.

New in version 0.5.0.

merge(start, end, content_appender=None)

table
   List of views of type RowView

class benker.table.TableName (table, pos, styles=None, nature=None)
   Bases: benker.styled.Styled
Base class of `RowView` and `ColView` used to create a view on the table cells for a given row or column.

See also: `TableViewList`

`adopt_cell` *(cell)*
Event handler called by the system when a cell is about to be inserted in the table.

`can_catch` *(cell)*
Check if a cell can be caught by that view.

**Parameters**
- `cell` *(benker.cell.Cell)* – table cell

**Returns**
True if the cell intercept to this view.

`can_own` *(cell)*
Check if a cell can be stored it that view.

**Parameters**
- `cell` *(benker.cell.Cell)* – table cell

**Returns**
True if the cell belong to this view.

`caught_cells`
List of cells caught (intercepted) by this view.

`owned_cells`
List of cells owned by this view.

`table`
Non-mutable reference to the table (instance of `Table`).

**class** `benker.table TableViewList` *(table, view_cls)*
**Bases:** `object`

This class defined a (simplified) list of views.

Short demonstration:

```python
>>> from benker.cell import Cell
>>> from benker.table import Table
>>> from benker.table import ColView
>>> from benker.table import RowView
>>> from benker.table import TableViewList

>>> red = Cell('red', x=1, y=1, height=2)
>>> pink = Cell('pink', x=2, y=1, width=2)
>>> blue = Cell('blue', x=2, y=2)
>>> table = Table([red, pink, blue])
>>> print(table)
<table>
<thead>
<tr>
<th>red</th>
<th>pink</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+-----</td>
</tr>
<tr>
<td>-----</td>
<td>-----+-----</td>
</tr>
</tbody>
</table>

>>> cols = TableViewList(table, ColView)
>>> len(cols)
3
>>> rows = TableViewList(table, RowView)
>>> len(rows)
2
>>> for pos, col in enumerate(cols, 1):
    print(pos, col)
1 red
2 pink
3 blue
```

(continues on next page)
...  print("col #{pos}: {col}".format(pos=pos, col=str(col)))
col #1: [red]
col #2: [pink, blue]
col #3: []

>>> cols[3].insert_cell("yellow")
>>> print(table)
+-----------+-----------------------+
| red | pink |
|--------------------------|
| | blue | yellow |
|--------------------------|
+-----------+-----------------------+

**adopt_cell** *(cell)*
Adopt a new cell in the views.

**Parameters**

*cell* *(benker.cell.Cell)* – New cell to adopt

**refresh_all** *

Cleanup and refresh all the views, taking into account the cells which are in the table grid.

**class** *benker.table.ViewsProperty*(view_cls)*

Bases: object

Descriptor used to define the rows/cols properties in the class *Table*.

### 4.3.8 Available Parsers

This package contains a collection of parsers which you can use in conjunction with builders to convert tables from one format to another.

You can pick a builder in the *Available Builders*.

### 4.3.9 Base Parser

Base class of parsers.

**class** *benker.parsers.base_parser.BaseParser*(builder, encoding='utf-8', **options)*

Bases: object

Abstract base class of the parsers classes.

**parse_file** *(src_xml, dst_xml)*

Parse and convert the tables from one format to another.

**Parameters**

- **src_xml** *(str)* – Source path of the XML file to convert.
- **dst_xml** *(str)* – Destination path of the XML file to produce.

Changed in version 0.5.0: Always generate the XML declaration in the destination file.

**transform_tables** *(tree)*

benker.parsers.base_parser.value_of*(element, xpath, namespaces=None, default=None)*

Take the first value of a xpath evaluation.

**Parameters**


• **element** (*etree._Element*) – Root element used to evaluate the xpath expression.

• **xpath** (*str*) – xpath expression. This expression will be evaluated using the **namespaces**

• **namespaces** (*dict[str, str]*) – Namespace map to use for the xpath evaluation.

• **default** – default value used if the xpath evaluation returns no result.

**Returns** the first result or the **default** value.

### 4.3.10 Office Open XML parser

This module can parse Office Open XML (OOXML) tables.

**Specifications:**

- The documentation about OOXML Table is available online at Word Processing - Table Grid/Column Definition.

```python
class benker.parsers.ooxml.OoxmlParser(builder, styles_path=None, **options)
Bases: benker.parsers.base_parser.BaseParser
```

Office Open XML parser.

**parse_grid_col** (*w_grid_col*)

Parse a `<w:gridCol>` element.

See: Table Grid/Column Definition.

**Parameters**

- **w_grid_col** (*etree._Element*) – Table element.

**parse_table** (*w_tbl*)

Convert a Office Open XML `<w:tbl>` into CALS `<table>`

**Parameters**

- **w_tbl** (*etree._Element*) – Office Open XML element.

**Return type** *etree._Element*

**Returns** CALS element.

**parse_tbl** (*w_tbl*)

Parse a `<w:tbl>` element.

See: Table Properties.

**Parameters**

- **w_tbl** (*etree._Element*) – Table element.

Changed in version 0.4.0: The section width and height are now stored in the ‘x-sect-size’ table style (units in ‘pt’).

**parse_tc** (*w_tc*)

Parse a `<w:tc>` element.

See: Table Cell Properties.

**Parameters**

- **w_tc** (*etree._Element*) – Table element.

Changed in version 0.5.1: XML indentation between cell paragraphs is ignored.

**parse_tr** (*w_tr*)

Parse a `<w:tr>` element.

See: Table Row Properties.

**Parameters**

- **w_tr** (*etree._Element*) – Table element.

**transform_tables** (*tree*)
Take the first value of a xpath evaluation.

Parameters

- `element (etree._Element)` – Root element used to evaluate the xpath expression.
- `xpath (str)` – xpath expression. This expression will be evaluated using the `namespaces` namespaces.
- `namespaces (dict[str, str])` – Namespace map to use for the xpath evaluation.
- `default` – default value used if the xpath evaluation returns no result.

Returns the first result or the `default` value.

4.3.11 OOXML namespaces

Namespace map used for xpath evaluation in Office Open XML documents

Take the first value of a xpath evaluation.

Parameters

- `element (etree._Element)` – Root element used to evaluate the xpath expression.
- `xpath (str)` – xpath expression. This expression will be evaluated using the `namespaces` namespaces.
- `namespaces (dict[str, str])` – Namespace map to use for the xpath evaluation.
- `default` – default value used if the xpath evaluation returns no result.

Returns the first result or the `default` value.

4.3.12 OOXML Standard Types

Utility classes used to store/convert OOXML standard types.

```python
class benker.parsers.ooxml.types.StHexColor (value):
    Bases: benker.parsers.ooxml.types.StValue
    Color Value

style

class benker.parsers.ooxml.types.StPageOrientation (value, default=None):
    Bases: benker.parsers.ooxml.types.StValue
    Page Orientation
```

4.3.13 OOOXML Page size

Page Size dimensions, orientation and printer paper code.

```python
class benker.parsers.ooxml.w_pg_sz.PgSz(w_pg_sz)
    Bases: object
    Page Size. — This element specifies the properties (size and orientation) for all pages in the current section.
    Example: `<w:pgSz w:w="11907" w:h="16839" />`, for A4 portrait.
    styles
    w_code = None
        Printer Paper Code
    w_h = None
        Page Height
    w_orient = None
        Page Orientation (Possible values are “landscape” and “portrait”).
    w_w = None
        Page Width
```

benker.parsers.ooxml.w_pg_sz.value_of(element, xpath, *, namespaces={'w': 'http://schemas.openxmlformats.org/wordprocessingml/2006/main'}, default=None)

Take the first value of a xpath evaluation.

**Parameters**

- `element (etree._Element)` – Root element used to evaluate the xpath expression.
- `xpath (str)` – xpath expression. This expression will be evaluated using the `namespaces` names.
- `namespaces (dict[str, str])` – Namespace map to use for the xpath evaluation.
- `default` – default value used if the xpath evaluation returns no result.

**Returns**

the first result or the `default` value.

4.3.14 OOOXML Shading

Table/Cell shading and Table Shading Exception
class benker.parsers.ooxml.w_shd.Shd(w_shd):
    Bases: object

Table/Cell shading and Table Shading Exception

Example: <w:shd w:val="clear" w:color="auto" w:fill="E6E6E6"/>


styles

w_color = None
    Shading Pattern Color

w_fill = None
    Shading Background Color

w_themeColor = None
    Shading Pattern Theme Color

w_themeFill = None
    Shading Background Theme Color

w_themeFillShade = None
    Shading Background Theme Color Shade

w_themeFillTint = None
    Shading Background Theme Color Tint

w_themeShade = None
    Shading Pattern Theme Color Shade

w_themeTint = None
    Shading Pattern Theme Color Tint

w_val = None
    Shading Pattern

benker.parsers.ooxml.w_shd.value_of(element, xpath, *, namespaces={'w': 'http://schemas.openxmlformats.org/wordprocessingml/2006/main'}, default=None)

Take the first value of a xpath evaluation.

Parameters

- element (etree._Element) – Root element used to evaluate the xpath expression.
- xpath (str) – xpath expression. This expression will be evaluated using the namespaces.
- namespaces (dict[str, str]) – Namespace map to use for the xpath evaluation.
- default – default value used if the xpath evaluation returns no result.

Returns the first result or the default value.

4.3.15 Formex 4 Parser

This module can parse the tables (TBL elements) of a Formex 4 file.

The TBL element is used to mark up a Formex table, which actually contains text structured in columns with related data.
A table usually contains the following information:

- an optional title (TITLE),
- one or more structured text blocks (GR.SEQ) in order to mark up optional explanatory information about the table content, located between the title of the table and the table itself,
- optionally a group of notes called in the table (GR.NOTES),
- the corpus of the table (CORPUS).

When building the internal table object, this builder will:

- interpret the title (TITLE) and structured text blocks (GR.SEQ) like rows. The nature attribute of each row will be “title” and “text-block” respectively.
- interpret the group of notes (GR.NOTES) like a row of nature “footer”
- interpret the corpus of the table (CORPUS) like the body of the table. The nature attribute of each row will be “body”.

Note: Since the Formex table structure is not suitable for typesetting/page layout, this parser is also able to parse CALS-like attributes (for instance frame, cols, colsep, rowsep, ...) and CALS-like elements (for instance colspec). This attributes and elements may be added with the Formex 4 builder, see FormexBuilder.

New in version 0.5.0.

```python
benker.parsers.formex.ElementType
    alias of lxml.etree._Element

class benker.parsers.formex.FormexParser(builder, formex_ns=None, cals_ns=None, embed_gr_notes=False, **options)
    Bases: benker.parsers.base_parser.BaseParser
    Formex 4 tables parser

    contains_iel (fmx_node)
    get_cals_qname(name)
    get_formex_qname(name)

parse_cals_row_styles (fmx_elem)
    Parse the row styles

    Parameters fmx_elem (ElementType) – Formex element: ROW, TI.BLK, STI.BLK or GR.NOTES.

    Returns CSS-like styles

    Changed in version 0.5.1: The “vertical-align” style is built from the @cals:valign attribute.

parse_fmx_cell (fmx_cell)
    Parse a CELL element.

    Parameters fmx_cell (ElementType) – table cell

parse_fmx_colspec (cals_colspec)
    Parse a CALS-like colspec element.

    For instance:
```

4.3. API
<colspec
colname="c1"
colnum="1"
colsep="1"
rowsep="1"
colwidth="30mm"
align="center"/>

Parameters  
cals_colspec(ElementType) – CALS-like colspec element.

parse_fmx_corpus(fmx_corpus)

parse_fmx_row(fmx_row)
  Parse a ROW element which contains CELL elements.
  This element may be in a BLK

Parameters  
fmx_row(ElementType) – table row

parse_fmx_sti_blk(fmx_sti_blk)
  Parse a STI.BLK element, considering it like a row of a single cell.
  For instance:

  <STI.BLK COL.START="1" COL.END="1">
    <P>STI.BLK title</P>
  </STI.BLK>

Parameters  
fmx_sti_blk(ElementType) – subtitle of the BLK.

parse_fmx_ti_blk(fmx_ti_blk)
  Parse a TI.BLK element, considering it like a row of a single cell.
  For instance:

  <TI.BLK COL.START="1" COL.END="2">
    <P><HT TYPE="BOLD">TI.BLK title</HT></P>
  </TI.BLK>

Parameters  
fmx_ti_blk(ElementType) – title of the BLK.

parse_gr_notes(fmx_gr_notes)
  Parse a GR.NOTES element, considering it like a row of a single cell.
  For instance:

  <GR.NOTES>
    <TITLE>
      <TI>
        <P>GR.NOTES Title</P>
      </TI>
    </TITLE>
    <NOTE NOTE.ID="N0001">
      <P>Table note</P>
    </NOTE>
  </GR.NOTES>
Parameters **fmx_gr_notes** (*ElementType*) – group of notes called in the table (*GR. NOTES*)

Changed in version 0.5.1: *GR. NOTES* elements can be embedded if the *embed_gr_notes* option is `True`.

```python
parse_table(fmx_corpus)
```
Convert a `<CORPUS>` Formex element into table object.

**Parameters**
- **fmx_corpus** (*ElementType*) – Formex element.

**Return type** *ElementType*

**Returns** *Table.*

```python
parse_tbl_styles(fmx_tbl)
```
Parse a `TBL` element and extract the styles

**Parameters**
- **fmx_tbl** (*ElementType*) – table

**Returns** dictionary of styles and nature

```python
setup_table(styles=None, nature=None)
```

```python
transform_tables(tree)
```

### 4.3.16 benker.parsers.cals package

**CALS Parser Implementation**

This module can parse the tables (`table` elements) of a CALS file.

Specifications and examples:

- The CALS DTD is available online in the OASIS website: [CALS Table Model Document Type Definition](https://www.oasis-open.org/committees/cals).
- An example of CALS table is available in Wikipedia: [CALS Table Model](https://en.wikipedia.org/wiki/CALS_Table_Model).

The main elements of a CALS table are:

- **table**: a table can contain one or several `tgroup`.
  - **titles**: table titles (*not supported by the CALS parser*)
  - **tgroup**: a portion of table
    - **colspec**: column specifications
    - **spanspec**: spanning specifications (*not supported by the CALS parser*)
    - **thead**: table header
      - **colspec**: header column specifications (*not supported by the CALS parser*)
      - **row**: table row (see `tbody`)
    - **tfoot**: table footer
      - **colspec**: footer column specifications (*not supported by the CALS parser*)
      - **row**: table row (see `tbody`)
    - **tbody**: table body
      - **row**: table row
      - **entry**: table entry which contains paragraphs
entrytbl: table entry which contains a table (\textit{not supported by the CALS parser})

An example of CALS table is available in Wikipedia: CALS Table Model

New in version 0.5.0.

```python
class benker.parsers.cals.CalsParser(builder, cals_ns=None, width_unit='mm', **options)
    Bases: benker.parsers.base_parser.BaseParser

    CALS tables parser

    get_cals_qname(name)

    parse_cals_colspec(cals_colspec)
        Parse a CALS-like colspec element.
        For instance:

        ```xml
        <colspec
            colname="c1"
            colnum="1"
            colsep="1"
            rowsep="1"
            colwidth="30mm"
            align="center"/>
        ```

        Parameters  
        
        cals_colspec\text{(ElementType)} -- CALS-like colspec element.

    parse_cals_entry(cals_entry)
        Parse a entry element.

        Parameters  
        
        cals_entry\text{(ElementType)} -- table entry

        Changed in version 0.5.1: The “vertical-align” style is built from the @cals:valign attribute.

    parse_cals_row(cals_row)
        Parse a row element which contains entry elements.

        This element may be in a BLK`

        Parameters  
        
        cals_row\text{(ElementType)} -- table row

        Changed in version 0.5.1: The “vertical-align” style is built from the @cals:valign attribute.

    parse_cals_table(cals_table)
        Parse a CALS table element.

        Parameters  
        
        cals_table\text{(ElementType)} -- CALS table Element.

        Returns  
        State of the parser (for debug purpose).

        Changed in version 0.5.1: Add support for the @cals:width attribute (table width).

    parse_cals_tgroup(cals_tgroup)

    parse_table(cals_table)
        Convert a <table> CALS element into table object.

        Parameters  
        
        cals_table\text{(ElementType)} -- CALS element.

        Return type  
        benker.table.Table

        Returns  
        Table.

    setup_table(styles=None, nature=None)
```
**transform_tables**(*tree*)

benker.parsers.cals.ElementType
    alias of lxml.etree._Element

Submodules

**CALS - Frame Styles**

New in version 0.5.0.

benker.parsers.cals.frame_styles.BORDER_NONE = 'none'
    Default value for no border (for @cals:frame/@cals:colsep/@cals:rowsep,...)

benker.parsers.cals.frame_styles.BORDER_SOLID = 'solid 1pt black'
    Default value for a solid border (for @cals:frame/@cals:colsep/@cals:rowsep,...)

benker.parsers.cals.frame_styles.get_frame_styles(*frame*)

### 4.3.17 Available Builders

This package contains a collection of builders which you can use in conjunction with parsers to convert tables from one format to another.

You can pick a parser in the Available Parsers.

### 4.3.18 Base Builder

Base class of Builders.

```python
class benker.builders.base_builder.BaseBuilder(**options)
    Bases: object
    Base class of Builders.

    static append_cell_elements(cell_elem, elements)
        Append XML elements, PIs or texts to a cell element.

        Parameters
        • cell_elem (ElementType) – Cell element
        • elements – list of child elements to append

        New in version 0.5.1.

    finalize_tree(tree)
        Give the opportunity to finalize the resulting tree structure.

        Parameters tree – The resulting tree.

        New in version 0.4.0.

    generate_table_tree(table)
        Build the XML table from the Table instance.

        Parameters table (benker.table.Table) – Table

        Returns Table tree
```
4.3.19 CALS Builder

This module can construct a CALS table from an instance of type `Table`.

Specifications and examples:

- The CALS DTD is available online in the OASIS website: CALS Table Model Document Type Definition.
- An example of CALS table is available in Wikipedia: CALS Table Model

```python
class benker.builders.cals.CalsBuilder(cals_ns=None, cals_prefix=None, width_unit='mm', table_in_tgroup=False, tgroup_sorting=None, **options)
Bases: benker.builders.base_builder.BaseBuilder
```

CALS table builder.

### build_cell(row_elem, cell)

Build the CALS `<entry>` element.

CALS attributes:

- `@colsep` is built from the “border-right” style. Default value is “1” (displayed), so, it is better to always define it. This value is only set if different from the table `@colsep` value.
- `@rowsep` is built from the “border-bottom” style. Default value is “1” (displayed), so, it is better to always define it. This value is only set if different from the table `@rowsep` value.
- `@valign` is built from the “vertical-align” style. Values can be “top”, “middle”, “bottom” (note: “baseline” is not supported). Default value is “bottom”.
- `@align` is built from the “align” style. Values can be “left”, “center”, “right”, or “justify”. Default value is “left”. Note: paragraphs alignment should be preferred to cells alignment.
- `@namest/@nameend` are set when the cell is spanned horizontally.
- `@morerows` is set when the cell is spanned vertically.
- `@bgcolor` is built from the “background-color” style (HTML color).

**Parameters**

- `row_elem` (`ElementType`) – Parent element: `<row>`.
- `cell` (`benker.cell.Cell`) – The cell.

Changed in version 0.5.0: Add support for `bgcolor`.

Changed in version 0.5.1: Preserve processing instruction in cell content.

### build_colspec(group_elem, col)

Build the CALS `<colspec>` element.

CALS attributes:

- `@colnum` is the column number.
- `@colname` is the column name. Its format is “c{col_pos}”.
- `@colwidth` width of the column (with its unit). The unit is defined by the `width_unit` options.
Benker Documentation, Release 0.5.1

- @align horizontal alignment of table entry content. Possible values are: “left”, “right”, “center”, “justify” (“char” is not supported).
- @colsep column separators (vertical ruling). Possible values are “0” or “1”.
- @colsep row separators (horizontal ruling). Possible values are “0” or “1”.

Parameters

- **group_elem** (ElementType) – Parent element: <tgroup>.
- **col** (benker.table.ColView) – Columns

Changed in version 0.5.0: The @colnum and @align attributes are generated.
Changed in version 0.5.1: The @colsep and @rowsep attributes are generated.

**build_row**(tbody_elem, row)

Build the CALS <row> element.

CALS attributes:

- @valign is built from the “vertical-align” style. Values can be “top”, “middle”, “bottom” (note: “baseline” is not supported). Default value is “bottom”.

**Note:** A row can be marked as inserted if “x-ins” is defined in the row styles. Revision marks are inserted before and after a <row> using a couple of processing-instructions. We use the <?change-start?> PI to mark the start of the inserted row, and the <?change-end?> PI to mark the end.

Parameters

- **tbody_elem** (ElementType) – Parent element: <tbody>, <thead>, or <tfoot>.
- **row** (benker.table.RowView) – The row.

New in version 0.5.0: Add support for the @cals:rowstyle attribute (extension).
Changed in version 0.5.1: The @cals:valign attribute is built from the “vertical-align” style.

**build_table**(table)

Build the CALS <table> element.

CALS attributes:

- @colsep is built from the “x-cell-border-right” style. Default value is “0” (not displayed).
- @rowsep is built from the “x-cell-border-bottom” style. Default value is “0” (not displayed).
- @tabstyle is built from the table nature.
- @orient is built from the “x-sect-orient” style (orientation of the current section). Possible values are “port” (portrait, the default) or “land” (landscape).
- @pgwide is built from the “x-sect-cols” style (column number of the current section). Default value is “0” (width of the current column).
- @bgcolor is built from the “background-color” style (HTML color).
- @width is built from the “width” style (percentage or width with unit). This attribute in an extension.
Note: @colsep, @rowsep and @tabstyle attributes are generated only if the table_in_tgroup options is False.

Attention: According to the CALS specification, the default value for @colsep and @rowsep should be “1”. But, having this value as a default is really problematic for conversions: most of nowadays formats, like Office Open XML and CSS, consider that the default value is “no border” (a.k.a: border: none). So, setting “0” as a default value is a better choice.

Parameters

- **table** *(benker.table.Table)* – Table

Returns The newly-created `<table>` element.

Changed in version 0.5.0: Add support for the bgcolor attribute (background color).

Changed in version 0.5.1: Add support for the @width attribute (table width).

**build_tbody** *(group_elem, row_list, nature_tag)*

Build the CALS `<tbody>`, `<thead`', or `<tfoot`' element.

Parameters

- **group_elem** *(ElementType)* – Parent element: `<tgroup>`.
- **row_list** – List of rows
- **nature_tag** – name of the tag: ‘tbody’, ‘thead’ or ‘tfoot’.

**build_tgroup** *(table_elem, table)*

Build the CALS `<tgroup>` element.

CALS attributes:

- @cols is the total number of columns.
- @colsep is built from the “x-cell-border-right” style. Default value is “0” (not displayed).
- @rowsep is built from the “x-cell-border-bottom” style. Default value is “0” (not displayed).
- @tgroupstyle is built from the table nature.

Note: @colsep, @rowsep and @tgroupstyle attributes are generated only if the table_in_tgroup options is True.

Parameters

- **table_elem** *(ElementType)* – Parent element: `<table>`.
- **table** *(benker.table.Table)* – Table

Returns The newly-created `<tgroup>` element.

**generate_table_tree** *(table)*

Build the XML table from the Table instance.

Parameters **table** *(benker.table.Table)* – Table

Returns Table tree
4.3.20 Formex 4 Builder

This module can construct a Formex 4 table from an instance of type `Table`.

FORMEX describes the format for the exchange of data between the Publication Office and its contractors. In particular, it defines the logical markup for documents which are published in the different series of the Official Journal of the European Union.

This builder allow you to convert Word document tables into Formex 4 tables using the Formex 4 schema (formex-05.59-20170418.xd).

Specifications and examples:

- The Formex 4 documentation and schema is available online in the Publication Office: Formex Version 4.
- An example of Formex 4 table is available in the Schema documentation: TBL

Changed in version 0.5.0: Refactoring (rename “Formex4” to “Formex”):

- the class `Formex4Builder` is renamed `FormexBuilder`,

4.3. API 75
When a cell in column ‘A’ is linked to a cell in row ‘B’ located just below row ‘A’, the CELL element of this single cell must provide the ROWSPAN attribute. The value of the ROWSPAN attribute is equal to the number of cells in the group. The CELL element relating to the single cell must be placed within the first ROW element in the group. The ROW elements corresponding to the other rows in the group may not contain any CELL elements for the column containing the single cell ‘A’.

The use of the ROWSPAN attribute is only authorised to relate the value of a cell in several rows. Its value must be at least equal to ‘2’.

If the group of related cells is physically delimited by a horizontal brace, this symbol must be marked up using the ACCH attribute.

If the group of related cells is physically delimited by a vertical brace, this symbol must be marked up using the ACCV attribute.

The TYPE attribute of the CELL element is used to indicate locally the type of contents of the cells. It overrides the value of the TYPE attribute defined for the row (ROW) which contains the cell.

### Parameters
- `row_elem (ElementType)` – Parent element: `<ROW>`.
- `cell (benker.cell.Cell)` – The cell.

**Note:** The `@colnum` attribute (number of column) is not generated because this value is usually implied, and can be deduce from the `@colname` attribute.

### Parameters
- `group_elem (ElementType)` – Parent element: `<tgroup>`.
- `col (benker.table.ColView)` – Columns
Changed in version 0.5.0: Add support for CALS-like elements and attributes.

Changed in version 0.5.1: Add support for CALS-like attributes: @colnum, @align, @colsep, and @rowsep.

**build_corpus**(tbl_elem, table)
Build the Formex 4 `<CORPUS>` element.

**Parameters**

- **tbl_elem**(ElementType) – Parent element: `<TBL>`.
- **table**(benker.table.Table) – Table

Changed in version 0.5.1: If this option `detect_titles` is enable, a title will be generated if the first row contains an unique cell with centered text.

Changed in version 0.5.1: Add support for the @width CALS-like attribute (table width).

**build_row**(corpus_elem, row)
Build the Formex 4 `<ROW>` element.

Formex 4 attributes:

- **@TYPE** The TYPE attribute indicates the specific role of the row in the table. The allowed values are:
  - ALIAS: if the row contains aliases. Such references may be used when the table is included on several pages of a publication. The references are associated to column headers on the first page and are repeated on subsequent pages.
  - HEADER: if the row contains cells which may be considered as a column header. This generally occurs for the first row of a table.
  - NORMAL: if most of the cells of the row contain ‘simple’ or ‘normal’ data. This is the default value.
  - NOTCOL: if the cells of the row contain units of measure relating to subsequent rows.
  - TOTAL: if the row contains data which could be considered as ‘totals’.

Note that this TYPE attribute is also provided for the cells (CELL), which could be used to override the value defined for the row. On the other hand, ‘NORMAL’ is the default value, so it is necessary to specify the TYPE attribute value in each cell of a row which has a specific type in order to avoid the default overriding (see the first row of the example below).

**Parameters**

- **corpus_elem**(ElementType) – Parent element: `<CORPUS>`.
- **row**(benker.table.RowView) – The row.

Changed in version 0.5.0: Add support for CALS-like elements and attributes.

Changed in version 0.5.1: The `@cals:valign` attribute is built from the “vertical-align” style.

**build_tbl**(table)
Build the Formex 4 `<TBL>` element.

Formex 4 attributes:

- **@NO.SEQ** This mandatory attribute provides a sequence number to the table. This number represents the order in which the table appears in the document.
• **@CLASS** The CLASS attribute is mandatory and is used to specify the type of data contained in the table. The allowed values are:
  - GEN: if the table contains general data (default value),
  - SCHEDULE: if it is a schedule,
  - RECAP: if it is a synoptic table.
  These two last values are only used for documents related to the general budget.

• **@COLS** This mandatory attribute provides the actual number of columns of the table.

• **@PAGE.SIZE** The PAGE.SIZE attribute takes one of these values:
  - DOUBLE.LANDSCAPE: table on two A4 pages forming an A3 landscape page,
  - DOUBLE.PORTRAIT: table on two A4 pages forming an A3 portrait page,
  - SINGLE.LANDSCAPE: table on a single A4 page in landscape,
  - SINGLE.PORTRAIT: table on a single A4 page in portrait (default).

### Parameters

**table** (benker.table.Table) – Table

**Returns** The newly-created `<TBL>` element.

Changed in version 0.5.0: Add support for CALS-like elements and attributes. Add support for `bgcolor` (Table background color).

### build_title (tbl_elem, row)

Build the table title using the `<TITLE>` element.

For instance:

```xml
<TITLE>
  <TI>
    <P>Table IV</P>
  </TI>
</TITLE>
```

**Parameters**

- **tbl_elem** (`ElementType`) – Parent element: `<TBL>`.
- **row** (benker.table.RowView) – The row which contains the title.

### cleanup_tbl_in_tbl (fmx_root)

Cleanup the TBL elements when they are direct children of another TBL.

**Parameters**

- **fmx_root** (`ElementType`) – The result tree which contains the TBL elements to remove.

### drop_superfluous_attrs (fmx_root)

Drop superfluous CALS-like attributes at the end of the Formex building.

- `@cals:namest` and `@cals:nameend` are defined by `@COLSPAN`
- `@cals:morerows` is defined by `@ROWSPAN`
- `@cals:rowstyle` is defined by `ROW/@TYPE`, `GR.NOTES`, `TI.BLK` or `STI.BLK`.

**Parameters**

- **fmx_root** (`ElementType`) – Root element of the Formex file.
New in version 0.5.1.

**extract_gr_notes**(fmx_root)

Extract GR.NOTES from the table footers.

This function moves or creates a GR.NOTES just before the CORPUS.

**Parameters** fmx_root (ElementType) – The result tree with GR.NOTES.

Changed in version 0.5.1: If the ROW contains a GR.NOTES, we move it before the CORPUS, else we create it.

**finalize_tree**(tree)

Finalize the resulting tree structure:

- Calculate the @NO.SEQ values: sequence number of each table;
- Cleanup the TBL elements when they are direct children of another TBL;
- Extract GR.NOTES from the table footers;
- Group ROW elements by BLK based on the @cals:rowstyle attribute (CALS extension).

**Parameters** tree (ElementTreeType) – The resulting tree.

Changed in version 0.5.1: Drop superfluous CALS-like attributes at the end of the Formex building.

**generate_table_tree**(table)

Build the XML table from the Table instance.

**Parameters** table (benker.table.Table) – Table

**Returns** Table tree

**get_cals_qname**(name)

**get_formex_qname**(name)

**insert_blk**(fmx_root)

Group ROW elements by BLK based on the @cals:rowstyle attribute (CALS extension).

**Parameters** fmx_root (ElementType) – The result tree which contains the CORPUS/ROW elements.

**ns_map**

**setup_table**(table)

**update_no_seq**(fmx_root)

Calculate the @NO.SEQ values: sequence number of each table.

**Parameters** fmx_root (ElementType) – The result tree which contains the TBL elements to update.
4.3.21 Namespace

class benker.builders.namespace.Namespace
    Bases: benker.builders.namespace.Namespace
    A namespace is defined by a prefix and an uri.
    New in version 0.5.0.

    get_name(name)
        get the prefixed name

    get_qname(name)
        get the qualified name

4.3.22 Available Converters

This package contains a collection of converters which you can use to convert tables from one format to another.
To do that, you need to select a parser and a builder:

- You can pick a parser in the Available Parsers.
- You can pick a builder in the Available Builders.

4.3.23 Base Converter

Bas class of all converters.

class benker.converters.base_converter.BaseConverter
    Bases: object
    Bas class of all converters.

    builder_cls
        alias of benker.builders.base_builder.BaseBuilder

    convert_file(src_xml, dst_xml, **options)
        Convert the tables from one format to another.

        Parameters

            - src_xml (str) – Source path of the XML file to convert.
            - dst_xml (str) – Destination path of the XML file to produce.

            Common parsing options:

                - encoding (default: “utf-8”): XML encoding of the destination file.

    parser_cls
        alias of benker.parsers.base_parser.BaseParser
**4.3.24 CALS to Formex 4 converter**

New in version 0.5.0.

```python
class benker.converters.cals2formex.Cals2FormexConverter
    Bases: benker.converters.base_converter.BaseConverter

    CALS to Formex 4 converter

    builder_cls
        alias of benker.builders.formex.FormexBuilder

    parser_cls
        alias of benker.parsers.cals.CalsParser
```

```python
benker.converters.cals2formex.convert_cals2formex(src_xml, dst_xml, **options)
```

Convert CALS 4 tables to Formex tables.

**Parameters**

- `src_xml (str)` – Source path of the XML file to convert.
- `dst_xml (str)` – Destination path of the XML file to produce.

**Common parsing options:**

- `encoding` (default: “utf-8”): XML encoding of the destination file.

**CALS parser options:**

- `cals_ns` (default None): Namespace to use for CALS elements and attributes parsing. Set None (or “”) if you don’t use namespace in your XML.

**Formex 4 builder options:**

- `use_cals` (default: False): Generate additional CALS-like elements and attributes to simplify the layout of Formex document in typesetting systems.
- `cals_ns` (default: “https://lib.benker.com/schemas/cals.xsd”): Namespace to use for CALS-like elements and attributes (requires: use_cals). Set None (or “”) if you don’t want to use namespace.
- `cals_prefix` (default: “cals”): Namespace prefix to use for CALS-like elements and attributes (requires: use_cals).

**4.3.25 Formex 4 to CALS converter**

New in version 0.5.0.

```python
class benker.converters.formex2cals.Formex2CalsConverter
    Bases: benker.converters.base_converter.BaseConverter

    Formex 4 to CALS converter

    builder_cls
        alias of benker.builders.cals.CalsBuilder
```

4.3. API
parser_cls
alias of `benker.parsers.formex.FormexParser`

`benker.converters.formex2cals.convert_formex2cals(src_xml, dst_xml, **options)`

Convert Formex 4 tables to Cals tables.

**Parameters**

- `src_xml (str)` – Source path of the XML file to convert.
- `dst_xml (str)` – Destination path of the XML file to produce.

**Common parsing options:**

- `encoding (default: “utf-8”)`: XML encoding of the destination file.

**Formex parser options:**

- `formex_ns (default None)`: Namespace to use for Formex elements and attributes parsing. Set None (or “”) if you don’t use namespace.
- `cals_ns (default None)`: Namespace to use for CALS-like elements and attributes parsing. Set None (or “”) if you don’t use namespace.
- `embed_gr_notes (default False)`: If True, Embed the GR.NOTES in a row/cell, else only copy the content (not the GR.NOTES tag).

**CALS builder options:**

- `cals_ns (default: None)`: Namespace to use for CALS-like elements and attributes to generate. Set None (or “”) if you don’t want to use namespace.
- `cals_prefix (default: None)`: Namespace prefix to use for CALS-like elements and attributes to generate.
- `table_in_tgroup (default: False)`: Where should we put the table properties:
  - False to insert the attributes @colsep, @rowsep, and @tabstyle in the `<table>` element,
  - True to insert the attributes @colsep, @rowsep, and @tgroupstyle in the `<tgroup>` element.
- `tgroup_sorting (default: ["header", "footer", "body"])`: List used to sort (and group) the rows in a tgroup. The sorting is done according to the row natures which is by default: ["header", "footer", "body"] (this order match the CALS DTD defaults, where the footer is between the header and the body. To move the footer to the end, you can use ["header", "body", "footer"]).

Changed in version 0.5.0: Add the options cals_ns, cals_prefix, tgroup_sorting.

### 4.3.26 Office Open XML to CALS converter

`class benker.converters.ooxml2cals.Ooxml2CalsConverter`  
**Bases:** `benker.converters.base_converter.BaseConverter`

Office Open XML to CALS converter
**builder_cls**
apias of `benker.builders.cals.CalsBuilder`

**parser_cls**
apias of `benker.parsers.ooxml.OoxmlParser`

`benker.converters.ooxml2cals.convert_ooxml2cals(src_xml, dst_xml, **options)`

Convert Office Open XML (OOXML) tables to CALS tables.

**Parameters**

- **src_xml** *(str)* – Source path of the XML file to convert.
  
  This must be an XML file, for instance, if you want to convert a Word document (.docx), you first need to unzip the .docx file, and then, run this function on the file word/document.xml. You can also use the `styles_path` option to parse and use the table styles defined in the file word/styles.xml.

- **dst_xml** *(str)* – Destination path of the XML file to produce.

- **options** – Dictionary of parsing/building options.

  **Common parsing options:**

  - `encoding` *(default: “utf-8”)*: XML encoding of the destination file.

  **OOXML parser options:**

  - `styles_path` *(default: None)*: Path to the stylesheet to use to resolve table styles. In an uncompressed .docx tree structure, the stylesheet path is word/styles.xml.

  **CALS builder options:**

  - `cals_ns` *(default: None)*: Namespace to use for CALS-like elements and attributes to generate. Set `None` (or “”) if you don’t want to use namespace.

  - `cals_prefix` *(default: None)*: Namespace prefix to use for CALS-like elements and attributes to generate.


  - `table_in_tgroup` *(default: False)*: Where should we put the table properties:
    
    - False to insert the attributes @colsep, @rowsep, and @tabstyle in the `<table>` element,
    - True to insert the attributes @colsep, @rowsep, and @tgroupstyle in the `<tgroup>` element.

  - `tgroup_sorting` *(default: ["header", "footer", "body"])*: List used to sort (and group) the rows in a tgroup. The sorting is done according to the row natures which is by default: ["header", "footer", "body"] (this order match the CALS DTD defaults, where the footer is between the header and the body. To move the footer to the end, you can use ["header", "body", "footer"]).

  Changed in version 0.5.0: Add the options `cals_ns`, `cals_prefix`, `tgroup_sorting`.

### 4.3.27 Office Open XML to Formex 4 converter

Changed in version 0.5.0: Refactoring (rename “Formex4” to “Formex”):

- the class `Ooxml2Formex4Converter` is renamed `Ooxml2FormexConverter`,

---

**4.3. API**

---
Benker Documentation, Release 0.5.1

- the function `convert_ooxml2formex4` is renamed `convert_ooxml2formex`,

```python
class benker.converters.ooxml2formex.Ooxml2FormexConverter
    Bases: benker.converters.base_converter.BaseConverter
    Office Open XML to Formex 4 converter

    builder_cls
        alias of benker.builders.formex.FormexBuilder

    parser_cls
        alias of benker.parsers.ooxml.OoxmParser
```

`benker.converters.ooxml2formex.convert_ooxml2formex(src_xml, dst_xml, **options)`

Convert Office Open XML (OOXML) tables to Formex 4 tables.

**Parameters**

- `src_xml` (*str*) – Source path of the XML file to convert.
  - This must be an XML file, for instance, if you want to convert a Word document (.docx), you first need to unzip the .docx file, and then, run this function on the file word/document.xml. You can also use the `styles_path` option to parse and use the table styles defined in the file word/styles.xml.

- `dst_xml` (*str*) – Destination path of the XML file to produce.

  - **Common parsing options:**
    - `encoding` (default: “utf-8”): XML encoding of the destination file.
  - **OOXML parser options:**
    - `styles_path` (default: None): Path to the stylesheet to use to resolve table styles. In an uncompressed .docx tree structure, the stylesheet path is word/styles.xml.
  - **Formex 4 builder options:**
    - `detect_titles` (default: False): If this option is enable, a title will be generated if the first row contains an unique cell with centered text.
    - `use_cals` (default: False): Generate additional CALS-like elements and attributes to simplify the layout of Formex document in typesetting systems.
    - `cals_ns` (default: “https://lib.benker.com/schemas/cals.xsd”): Namespace to use for CALS-like elements and attributes (requires: `use_cals`). Set None (or “”) if you don’t want to use namespace.
    - `cals_prefix` (default: “cals”): Namespace prefix to use for CALS-like elements and attributes (requires: `use_cals`).

4.3.28 benker.common package

Common libraries

Submodules
lxml Iterators

Python alternative to \texttt{lxml.etree.iterwalk} for lxml $<$ 4.2.1

\begin{verbatim}
benker.common.lxml_iterwalk.ElementTreeType
    alias of lxml.etree._ElementTree

benker.common.lxml_iterwalk.ElementType
    alias of lxml.etree._Element
\end{verbatim}

lxml - QName

Python alternative to \texttt{lxml.etree.QName} for lxml $<$ 4

New in version 0.5.0.

4.3.29 Alphabet

Utility functions to convert integer into a base-26 “number”, and vis versa.

\begin{verbatim}
benker.alphabet.alphabet_to_int (letters, alphabet='ABCDEFGHIJKLMNOPQRSTUVWXYZ')

    Convert a base-26 “number” using uppercase ASCII letters into an integer.

>>> from benker.alphabet import alphabet_to_int

>>> alphabet_to_int("A")
1
>>> alphabet_to_int("B")
2
>>> alphabet_to_int("AA")
27
>>> alphabet_to_int("AB")
28
>>> alphabet_to_int("ZZZ")
18278
>>> alphabet_to_int("")
0
>>> alphabet_to_int("AA@")
Traceback (most recent call last):
  ... ValueError: AA@
\end{verbatim}

Parameters

- \texttt{letters} – string representing a “number” in the base-26.
- \texttt{alphabet} – alphabet to use for the conversion.

Returns

Integer value of the “number”.

\begin{verbatim}
benker.alphabet.int_to_alphabet (value, alphabet='ABCDEFGHIJKLMNOPQRSTUVWXYZ')

    Convert a non-nul integer into a base-26 “number” using uppercase ASCII letters.

Usage:
\end{verbatim}
```python
>>> from benker.alphabet import int_to_alphabet
>>> int_to_alphabet(1)
'A'
>>> int_to_alphabet(2)
'B'
>>> int_to_alphabet(26)
'Z'
>>> int_to_alphabet(27)
'AA'
>>> int_to_alphabet(28)
'AB'
>>> int_to_alphabet(52)
'AZ'
>>> int_to_alphabet(53)
'BA'
>>> int_to_alphabet(18278)
'ZZZ'
>>> int_to_alphabet(-5)
Traceback (most recent call last):
  ... ValueError: -5
```

**Parameters**

- **value** (`int`) – value to convert
- **alphabet** – alphabet to use for the conversion.

**Returns** string representing this “number” in the base-26.

### 4.3.30 Drawing

Utility functions used to draw a grid.

**benker.drawing**.TILES = {((False, False, False, False)): ' 
 XXXXXXXXX 
', (False, False, False, True): ' 
 XXXXXXXXX 
------------
', (False, True, True, False): '+-----------+
| XXXXXXXXX |
+-----------+
(False, True, True, True): '+-----------+
| XXXXXXXXX |
+-----------+
(0.5, True, False, False): ' 
 XXXXXXXXX 
', (False, True, False, True): ' 
 XXXXXXXXX 
------------
', (False, True, True, True): '+-----------+
| XXXXXXXXX |
+-----------+
(0.5, True, True, False): ' 
 XXXXXXXXX 
', (False, True, True, True): ' 
 XXXXXXXXX 
------------
', (True, False, True, False): ' 
 XXXXXXXXX 
', (False, True, True, True): ' 
 XXXXXXXXX 
------------
', (True, False, False, False): ' 
 XXXXXXXXX 
', (False, False, True, False): ' 
 XXXXXXXXX 
', (True, False, False, True): ' 
 XXXXXXXXX 
------------
', (True, False, True, True): ' 
 XXXXXXXXX 
', (False, False, True, True): ' 
 XXXXXXXXX 
------------
', (True, True, False, True): ' 
 XXXXXXXXX 
', (False, False, True, True): ' 
 XXXXXXXXX 
', (True, True, False, False): ' 
 XXXXXXXXX 
', (False, False, False, True): ' 
 XXXXXXXXX 
', (True, False, False, False): ' 
 XXXXXXXXX 
', (False, False, False, False): ' 
 XXXXXXXXX 

Default tiles used to draw a grid.

Keys are tuples of (left, top, right, bottom) which represent the presence (if True) or absence (if False) of the border. Values are the string representation of the tiles, “XXXXXXXXX” will be replaced by the cell content.

**benker.drawing**.draw (grid, tiles=None)

Draw a grid using a collection of tiles.

**Parameters**

- **grid** (`benker.grid.grid`) – Grid to draw.
- **tiles** – Collection of tiles, use TILES if not provided.

**Returns** String representation of the grid.

**benker.drawing**.iter_lines (grid, tiles=None)

**benker.drawing**.iter_tiles (grid, tiles=None)
4.3.31 Units

Utility functions to convert values from one unit to another.

```python
benker.units.UNITS = {'cm': 0.01, 'dm': 0.1, 'ft': 0.3048, 'in': 0.0254, 'm': 1.0, 'mm': 0.001, 'pc': 0.0021166666666666664, 'pt': 0.00035277777777777776, 'px': 0.00035277777777777776}

Usual units Lengths in meter
```

```python
benker.units.convert_value(value, unit_in, unit_out)
```

Convert a value from one unit to another.

To convert 1pt to 'mm', you can do:

```python
>>> from benker.units import convert_value
>>> convert_value(1, 'pt', 'mm')
0.353
```

Parameters

- `value (int or float)` – Value to convert
- `unit_in` – Current unit of the value.
- `unit_out` – Expected unit of the value.

Return type float

Returns The converted value

```python
benker.units.parse_width(width, default_unit='mm')
```

Parse a width and return the value and its unit.

```python
>>> from benker.units import parse_width

>>> parse_width("210")
(210.0, 'mm')

>>> parse_width("210mm")
(210.0, 'mm')

>>> parse_width("210pt")
(210.0, 'pt')
```

Parameters

- `width` – width string to parse, for instance: “247mm”.
- `default_unit` – default unit to use if it is not specified

Return type (float, str)

Returns the value and its unit.

New in version 0.5.1.

4.4 Changelog

All notable changes to this project will be documented in this file.
The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

4.4.1 v0.5.1 (2019-11-12)

Bug fix release

**Changed**

Add the `parse_width()` function used to parse a width and return the value and its unit.

**Fixed**

- Documentation: add missing link to `convert_cals2formex` in the main page.
- Fix #4: Remove superfluous attributes in `cals2formex`.
  
  Change in the `FormexBuilder` class: Add the `drop_superfluous_attrs()` method: drop superfluous CALS-like attributes at the end of the Formex building.
- Fix #5: The title generation should be optional.
  
  Change in the `Formex4Builder` class: Add the `detect_titles` option: if this option is enable, a title will be generated if the first row contains an unique cell with centered text. The `detect_titles` options is disable by default.
- Change in the `Formex4Builder` class: Allow empty strings for `cals_ns` and `cals_prefix` options.
- Fix #6: Formex 2 Cals conversion: missing `entry/@valign`.
  
  Change in the `FormexParser` class: The “vertical-align” style is built from the `@cals:valign` attribute.
  Change in the `CalsParser` class: The “vertical-align” style is built from the `@cals:valign` attribute.
  Change in the `FormexBuilder` class: The `@cals:valign` attribute is built from the “vertical-align” style.
- Change in the `CalsBuilder` class: The `@cals:valign` attribute is built from the “vertical-align” style.
- Fix #7: Formex 2 Cals conversion: missing `table/@width`.
  
  Change in the `CalsBuilder` class: Add support for the `@width` attribute (table width).
  Change in the `FormexBuilder` class: Add support for the `@width` CALS-like attribute (table width).
- Minor change in the `OoxmlParser` class: XML indentation between cell paragraphs is ignored.
- Fix #9: Cals 2 Formex conversion: Text and PIs lost in entries.
  
  Add the `append_cell_elements()` method: Append XML elements, PIs or texts to a cell element.
  Change in the `CalsBuilder` and `FormexBuilder` classes: Preserve processing instruction in cell content.
- Fix #10: Formex 2 Cals conversion: GR.NOTES should be preserved.
  
  Change in `FormexParser` class: GR.NOTES elements can be embedded if the `embed_gr_notes` options is True.
  Change in the `FormexBuilder` class: During GR.NOTES extraction, existing GR.NOTES are moved before the CORPUS (or created if missing).
  Change in the `convert_formex2cals()` function: Add the `embed_gr_notes` options to allow GR.NOTES element embedding.
• **Fix #11**: Cals 2 Formex conversion: missing CORPUS/@width.
  
  Change in the **CalsParser** class: Add the `width_unit` option, and add support for the `@cals:width` attribute (table width).

• **Fix #12**: Cals 2 Formex conversion: missing `colspec` attributes.
  
  Change in the **FormexBuilder** class: Add support for CALS-like attributes: `@colnum`, `@align`, `@colsep`, and `@rowsep` in the `colspec` element.

  Change in the **CalsBuilder** class: The `@colsep` and `@rowsep` attributes are generated.

**Other**

• Change link to the Formex documentation to “https://op.europa.eu/en/web/eu-vocabularies/formex”.

• Change Tox & AppVeyor configuration to use lxml v4.3.3 on Windows (for Python 3.4), because lxml v4.3.5 is not available for this platform.

**4.4.2 v0.5.0 (2019-09-25)**

Minor release

**Changed**

• Refactoring (rename “Formex4” to “Formex”):
  
  – the module `benker/builders/formex4.py` is renamed `benker/builders/formex.py`,
  
  – the module `benker/converters/ooxml2formex4.py` is renamed `benker/converters/ooxml2formex.py`,
  
  – the module `benker/parsers/formex4.py` is renamed `benker/parsers/formex.py`,
  
  – the class `Formex4Builder` is renamed `FormexBuilder`,
  
  – the class `Ooxml2Formex4Converter` is renamed `Ooxml2FormexConverter`,
  
  – the function `convert_ooxml2formex4` is renamed `convert_ooxml2formex`,
  
  – the class `Formex4Parser` is renamed `FormexParser`,

• Change in the class **Table**: add the method `fill_missing()` to fill the missing cells in a table.

• Change in the class **CalsBuilder**: Add support for the `@cals:rowstyle` attribute (extension). The `@colnum` and `@align` attributes are generated for the `<colspec>` element. The new options `cals_ns` and `cals_prefix` allow the used of namespaces in CALS. The option `tgroup_sorting` can be used to sort the `thead`, `tbody` and `tfoot` elements.

• Change in the method **parse_file**: Always generate the XML declaration in the destination file.

**Added**

• Change in the converter: `convert_ooxml2formex()`: Add the option `use_cals` (and related options: `cals_ns`, `cals_prefix` and `width_unit`): This options is used to generate additional CALS-like elements and attributes to simplify the layout of Formex document in typesetting systems.

• Add support for the Table/Cell shading in the OOXML parser.
• Add support for bgcolor (Table/Cell background color) in the CALS builder.
• Add support for bgcolor (Table/Cell background color) in the Formex 4 builder (only with the use_cals option).
• New parser: CalsParser: CALS tables parser.

**Fixed**

• Change in the builder CalsBuilder: the possible values for row/cell nature is “header”, “body” and “footer” (instead of “head”, “body”, “foot”).
• Fix in the class OoxmlParser: rows with missing cells are filled with empty cells of the same nature as the row.

**Other**

• Fix an issue with the AppVeyor build: upgrade setuptools version in appveyor.yml, change the Tox configuration: set py27,py34,py35: pip >= 9.0.3, < 19.2.
• Change the project’s slogan: “Easily convert your CALS, HTML, Formex 4, Office Open XML (docx) tables from one format to another.”
• Change Tox configuration file to test the library with lxml v4.3 on Python 3.4 (support for Python 3.4 was removed in lxml v4.4).
• Change Tox configuration file to test the library on Python 3.8.
• Change the Travis CI configuration to build on Python 3.7 and 3.8-dev.

4.4.3 v0.4.3 (unreleased)

Bug fix release

**Fixed**

• Fix #5: The title generation should be optional.
  Change in the Formex4Builder class: Add the detect_titles option: if this option is enable, a title will be generated if the first row contains an unique cell with centered text. The detect_titles options is disable by default.

4.4.4 v0.4.2 (2019-06-06)

Bug fix release

**Fixed**

• Fix #1: Cell nature should inherit row nature by default.
  Change in the class Styled: The default value of the nature parameter is None (instead of “body”).
  Change in the methods insert_cell() and insert_cell() The nature of a cell is inherited from its parent’s row (or column).
Other

- Change the requirements for Sphinx: add `requests[security]` for Python 2.7.
- Fix an issue with the AppVeyor build: change the Tox configuration: set `py27,py34,py35: pip >= 9.0.3`.

4.4.5 v0.4.1 (2019-04-24)

Bug fix release

Fixed

- Change in the parser `OoxmlParser`: fix the ‘x-sect-cols’ value extraction when the `w:sectPr` is missing (use “1” by default).
- Fix the Formex 4 builder `FormexBuilder`: Generate a `<IE/>` element if the cell content (the string representation) is empty.

4.4.6 v0.4.0 (2019-04-23)

Feature release

Added

- New converter: `convert_ooxml2formex()`: Convert Office Open XML (OOXML) tables to Formex 4 tables.
- New builder: `FormexBuilder`: Formex 4 builder used to convert tables into `TBL` elements.
- Change in the parser `OoxmlParser`:
  - The section width and height are now stored in the ‘x-sect-size’ table style (units in ‘pt’).
- Change in the builder `BaseBuilder`: Add the method `finalize_tree()`: Give the opportunity to finalize the resulting tree structure.

4.4.7 v0.3.0 (2019-02-16)

Feature release

Added

- Change in the parser `OoxmlParser`:
  - Parse cell `w:tcPr/w:vAlign` values.
  - Parse paragraph alignments to calculate cell horizontal alignments.
  - Parse cell `w:tcPr/w:tcBorders` values to extract border styles.
- Change in the builder `benker.builders.cals.CalsBuilder`:
  - Generate `entry/@valign` attributes.
– Generate entry/@align attributes.
– Generate entry/@colsep and entry/@rowsep attributes.

Changed

• Change in the parser OoxmlParser:
  – Add more supported border styles

4.4.8 v0.2.2 (2018-12-15)

Bug fix release

Added

• Add a Python alternative to lxml.etree.iterwalk if using lxml < 4.2.1. See lxml changelog v4.2.1.

Fixed

• Fix the implementation of parse_table(): use a new implementation of lxml.etree.iterwalk if using lxml < 4.2.1.

Other

• Change Tox configuration file to test the library with lxml v3 and v4.
• Add a changelog in the documentation.

4.4.9 v0.2.1 (2018-11-27)

Fixed

• Fix Coverage configuration file.
• Fix and improve configuration for Tox.
• Fix docstring in ooxml2cals.
• Fix calculation of the @frame attribute in the method benker.builders.cals.CalsBuilder.build_table().

Other

• Change link to PyPi project to “https://pypi.org/project/Benker/”.
• Add the README to the documentation.
• Add configuration files for TravisCI and AppVeyor.
4.4.10 v0.2.0 (2018-11-26)

Changed

• Update project configuration
• Add missing `__init__.py` file in `tests` directory: it is required for test modules import.

Fixed

• Fix unit tests (Python 2.7).
• Fix flake8 problems.
• Fix implementation of the `Grid` class for Python 2.7 (remove annotation). And minor fixes.
• Remove pipenv configuration files.
• Fix project configuration.

4.4.11 v0.1.0 (2018-11-26)

• First version of Benker.
CHAPTER 5

Indices and tables

- genindex
- modindex
- search
Python Module Index

b
 benker, 46
 benkeralphabet, 85
 benkerbox, 48
 benkerbuilders, 71
 benkerbuildersbase_builder, 71
 benkerbuilderscals, 72
 benkerbuildersformex, 75
 benkerbuildersnamespace, 80
 benkercell, 52
 benkercommon, 84
 benkercommonlxml_iterwalk, 84
 benkercommonlxml_qname, 85
 benkerconverters, 80
 benkerconvertersbase_converter, 80
 benkerconverterscals2formex, 80
 benkerconvertersformex2cals, 81
 benkerconvertersooxml2cals, 82
 benkerconvertersooxml2formex, 83
 benkercord, 47
 benkerdrawing, 86
 benkergid, 55
 benkerparsers, 62
 benkerparsersbase_parser, 62
 benkerparserscals, 69
 benkerparserscalsframesetyles, 71
 benkerparsersformex, 66
 benkerparsersooxml, 63
 benkerparsersooxmlnamespaces, 64
 benkerparsersooxmltypes, 64
 benkerparsersooxmlw_pg_sz, 65
 benkerparsersooxmlw_shd, 65
 benkersize, 46
 benkerstyled, 51
 benkerstyle, 57
 benkerunit, 86
adopt_cell() (benkertable.TableView method), 61
adopt_cell() (benkertable.TableViewList method), 62
alphabet_to_int () (in module benker.alphabet), 85
append_cell_elements ()
  (benker.builders.base_builder.BaseBuilder static method), 71

B
BaseBuilder (class in benker.builders.base_builder), 71
BaseConverter (class in benker.converters.base_converter), 80
BaseParser (class in benker.parsers.base_parser), 62
benker (module), 46
benker.alphabet (module), 85
benker.box (module), 48
benker.builders (module), 71
benker.builders.base_builder (module), 71
benker.builders.cals (module), 72
benker.builders.formex (module), 75
benker.builders.namespace (module), 80
benker.cell (module), 52
benker.common (module), 84
benker.common.lxml_iterwalk (module), 84
benker.common.lxml_qname (module), 85
benker.converters (module), 80
benker.converters.base_converter (module), 80
benker.converters.cals2formex (module), 80
benker.converters.formex2cals (module), 81
benker.converters.ooxml2cals (module), 82
benker.converters.ooxml2formex (module), 83
benker.coord (module), 47
benker.drawing (module), 86
benker.grid (module), 55
benker.parsers (module), 62
benker.parsers.base_parser (module), 62
benker.parsers.cals (module), 69
benker.parsers.cals.frame_styles (module), 71
benker.parsers.formex (module), 66
benker.parsers.ooxml (module), 63
benker.parsers.ooxml.namespaces (module), 64
benker.parsers.ooxml.types (module), 64
benker.parsers.ooxml.w.pg_sz (module), 65
benker.parsers.ooxml.w.shd (module), 65
benker.size (module), 46
benker.styled (module), 51
benker.table (module), 57
benker.units (module), 86
BORDER_NONE (in module benker.parsers.cals.frame_styles), 71
BORDER_SOLID (in module benker.parsers.cals.frame_styles), 71
bounding_box (benker.grid.Grid attribute), 56
bounding_box (benker.table.Table attribute), 60
box (benker.cell.Cell attribute), 53
Box (class in benker.box), 50
BoxTuple (class in benker.box), 51
build_cell() (benker.builders.cals.CalsBuilder method), 72
build_cell() (benker.builders.formex.FormexBuilder method), 75
build_colspec() (benker.builders.cals.CalsBuilder method), 72
build_colspec() (benker.builders.formex.FormexBuilder method), 76
build_corpus() (benker.builders.formex.FormexBuilder method), 77
build_row() (benker.builders.cals.CalsBuilder method), 73
build_row() (benker.builders.formex.FormexBuilder method), 77
build_table() (benker.builders.cals.CalsBuilder method), 73
get_colsep_attr() (in module benker.builders.cals), 75
get_content_text() (in module benker.cell), 54
get_formex_qname() (benker.builders.formex.FormexBuilder method), 79
get_formex_qname() (benker.parsers.formex.FormexParser method), 67
get_frame_attr() (in module benker.builders.cals), 75
get_frame_styles() (in module benker.parsers.cals.frame_styles), 71
get_name() (benker.builders.namespace.Namespace method), 80
get_qname() (benker.builders.namespace.Namespace method), 80
get_rowsep_attr() (in module benker.builders.cals), 75
get_frame_styles() (in module benker.parsers.cals.frame_styles), 71
get_name() (benker.builders.namespace.Namespace method), 80
get_qname() (benker.builders.namespace.Namespace method), 80
get_rowsep_attr() (in module benker.builders.cals), 75
Grid (class in benker.grid), 56

H

height (benker.box.Box attribute), 50
height (benker.cell.Cell attribute), 53
height (benker.size.SizeType attribute), 47

I

insert_blk() (benker.builders.formex.FormexBuilder method), 79
insert_cell() (benker.table.ColView method), 58
insert_cell() (benker.table.RowView method), 58
int_to_alphabet() (in module benker.alphabet), 85
intersect() (benker.box.Box method), 50
intersection() (benker.box.Box method), 50
isdisjoint() (benker.box.Box method), 50
iter_lines() (in module benker.drawing), 86
iter_rows() (benker.grid.Grid method), 57
iter_tiles() (in module benker.drawing), 86

L

level (benker.builders.formex.RowInfo attribute), 79

M

max (benker.box.BoxTuple attribute), 51
max (benker.cell.Cell attribute), 53
merge() (benker.grid.Grid method), 57
merge() (benker.table.Table method), 60
min (benker.box.BoxTuple attribute), 51
min (benker.cell.Cell attribute), 53
move_to() (benker.box.Box method), 50
move_to() (benker.cell.Cell method), 53

N

Namespace (class in benker.builders.namespace), 80
nature (benkerstyled.Styled attribute), 52
NS (in module benker.parsers.ooxml.namespaces), 64
ns_map (benker.builders.cals.CalsBuilder attribute), 74
ns_map (benker.builders.formex.FormexBuilder attribute), 79
ns_name () (in module benker.parsers.ooxml.namespaces), 64

O

Ooxml2CalsConverter (class in benker.converters.ooxml2cals), 82
Ooxml2FormexConverter (class in benker.converters.ooxml2formex), 84
OoxmlParser (class in benker.parsers.ooxml), 63
owned_cells (benker.table TableView attribute), 61

P

parse_cals_colspec() (benker.parsers.cals.CalsParser method), 70
parse_cals_entry() (benker.parsers.cals.CalsParser method), 70
parse_cals_row() (benker.parsers.cals.CalsParser method), 70
parse_cals_row_styles() (benker.parsers.formex.FormexParser method), 67
parse_cals_table() (benker.parsers.cals.CalsParser method), 70
parse_cals_tgroup() (benker.parsers.cals.CalsParser method), 70
parse_file() (benker.parsers.base Parser.BaseParser method), 62
parse_fmx_cell() (benker.parsers.formex.FormexParser method), 67
parse_fmx_colspec() (benker.parsers.formex.FormexParser method), 67
parse_fmx_corpus() (benker.parsers.formex.FormexParser method), 68
parse_fmx_row() (benker.parsers.formex.FormexParser method), 68
parse_gr_notes() (benker.parsers.formex.FormexParser method), 68
parse_grid_col() (benker.parsers.ooxml.OoxmlParser method), 63
parse_grid_col() (benker.parsers.formex.FormexParser method), 63
parse_table() (benker.parsers.cals.CalsParser method), 70
parse_table() (benker.parsers.formex.FormexParser method), 69
parse_table() (benker.parsers.ooxml.OoxmlParser method), 63
parse_table() (benker.parsers.cals.CalsParser method), 70
parse_table() (benker.parsers.ooxml.OoxmlParser method), 63
parse_tbl() (benker.parsers.ooxml.OoxmlParser method), 63
parse_tbl_styles() (benker.parsers.formex.FormexParser method), 69
parse_tc() (benker.parsers.ooxml.OoxmlParser method), 63
parse_tr() (benker.parsers.ooxml.OoxmlParser method), 63
parse_width() (in module benker.units), 87
parser_cls (benker.converters.base_converter.BaseConverter attribute), 80
parser_cls (benker.converters.cals2formex.Cals2FormexConverter attribute), 81
parser_cls (benker.converters.formex2cals.Formex2CalsConverter attribute), 81
parser_cls (benker.converters.ooxml2cals.Ooxml2CalsConverter attribute), 83
parser_cls (benker.converters.ooxml2formex.Ooxml2FormexConverter attribute), 84
PgSz (class in benker.parsers.ooxml.w_pg_sz), 65
ProcessingInstructionType (in module benker.builders.formex), 79
refresh_all() (benker.table.TableViewList method), 62
resize() (benker.box.Box method), 50
resize() (benker.cell.Cell method), 53
revision_mark() (in module benker.builders.cals), 75
revision_mark() (in module benker.builders.formex), 80
row_pos (benker.table.RowView attribute), 59
RowInfo (class in benker.builders.formex), 79
rows (benker.table.Table attribute), 60
RowView (class in benker.table), 58

R
refresh_all() (benker.table.TableViewList method), 62
resize() (benker.box.Box method), 50
resize() (benker.cell.Cell method), 53
revision_mark() (in module benker.builders.cals), 75
revision_mark() (in module benker.builders.formex), 80
row_pos (benker.table.RowView attribute), 59
RowInfo (class in benker.builders.formex), 79
rows (benker.table.Table attribute), 60
RowView (class in benker.table), 58

S
setup_table() (benker.builders.cals.CalsBuilder method), 75
setup_table() (benker.builders.formex.FormexBuilder method), 79
setup_table() (benker.parsers.cals.CalsParser method), 70
setup_table() (benker.parsers.formex.FormexParser method), 69
setup_table() (benker.parsers.ooxml.OoxmlParser method), 63
setup_table() (benker.parsers.ooxml.OoxmlParser method), 63
setup_table() (benker.parsers.cals.CalsParser method), 70
setup_table() (benker.parsers.formex.FormexParser method), 69
StPageOrientation (class in benker.parsers.ooxml.types), 64
StTwipsMeasure (class in benker.parsers.ooxml.types), 65
StValue (class in benker.parsers.ooxml.types), 65
style (benker.parsers.ooxml.types.StHexColor attribute), 64
style (benker.parsers.ooxml.types.StTwipsMeasure attribute), 65
style (benker.parsers.ooxml.types.StValue attribute), 65
Styled (class in benker.styled), 51
Table (class in benker.table), 59
TableView (class in benker.table), 60
TableViewList (class in benker.table), 61
tag (benker.builders.formex.RowInfo attribute), 79
TILES (in module benker.drawing), 86
transform() (benker.box.Box method), 51
transform() (benker.cell.Cell method), 54
transform_tables() (benker.parsers.base_parser.BaseParser method), 62
transform_tables() (benker.parsers.cals.CalsParser method), 70
transformation() (benker.parsers.formex.FormexParser method), 69
transform_tables() (benker.parsers.ooxml.OoxmlParser method), 63
type (benker.builders.formex.RowInfo attribute), 79

U
UNITS (in module benker.units), 87
update_no_seq() (benker.builders.formex.FormexBuilder method), 79

V
value_of() (in module benker.parsers.base Parser), 62
value_of() (in module benker.parsers.ooxml), 64
value_of() (in module benker.parsers.ooxml.namespaces), 64
value_of() (in module benker.parsers.ooxml.w_pg_sz), 65
value_of() (in module benker.parsers.ooxml.w_shd), 66
ViewsProperty (class in benker.table), 62

W
w() (in module benker.parsers.ooxml), 64
w() (in module benker.parsers.ooxml.namespaces), 64
w_code (benker.parsers.ooxml.w_pg_sz.PgSz attribute), 65
w_color (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_fill (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_h (benker.parsers.ooxml.w.pg_sz.PgSz attribute), 65
w_orient (benker.parsers.ooxml.w.pg_sz.PgSz attribute), 65
w_themeColor (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_themeFill (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_themeFillShade (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_themeFillTint (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_themeShade (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_themeTint (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_val (benker.parsers.ooxml.w_shd.Shd attribute), 66
w_w (benker.parsers.ooxml.w_pg_sz.PgSz attribute), 65
width (benker.box.Box attribute), 51
width (benker.cell.Cell attribute), 54
width (benker.size.SizeType attribute), 47

X
x (benker.coord.CoordTuple attribute), 48

Y
y (benker.coord.CoordTuple attribute), 48