marcxml$_{parser}$

Release 1.2.3

March 14, 2016

Contents

This module is used to parse MARC XML and OAI documents. Module provides API to query such records and also to create records from scratch.

Module also contains getters which allows highlevel queries over documents, such as `get_name()` and `get_authors()`, which returns informations scattered over multiple subfields.

Package is developed and maintained by E-deposit team.

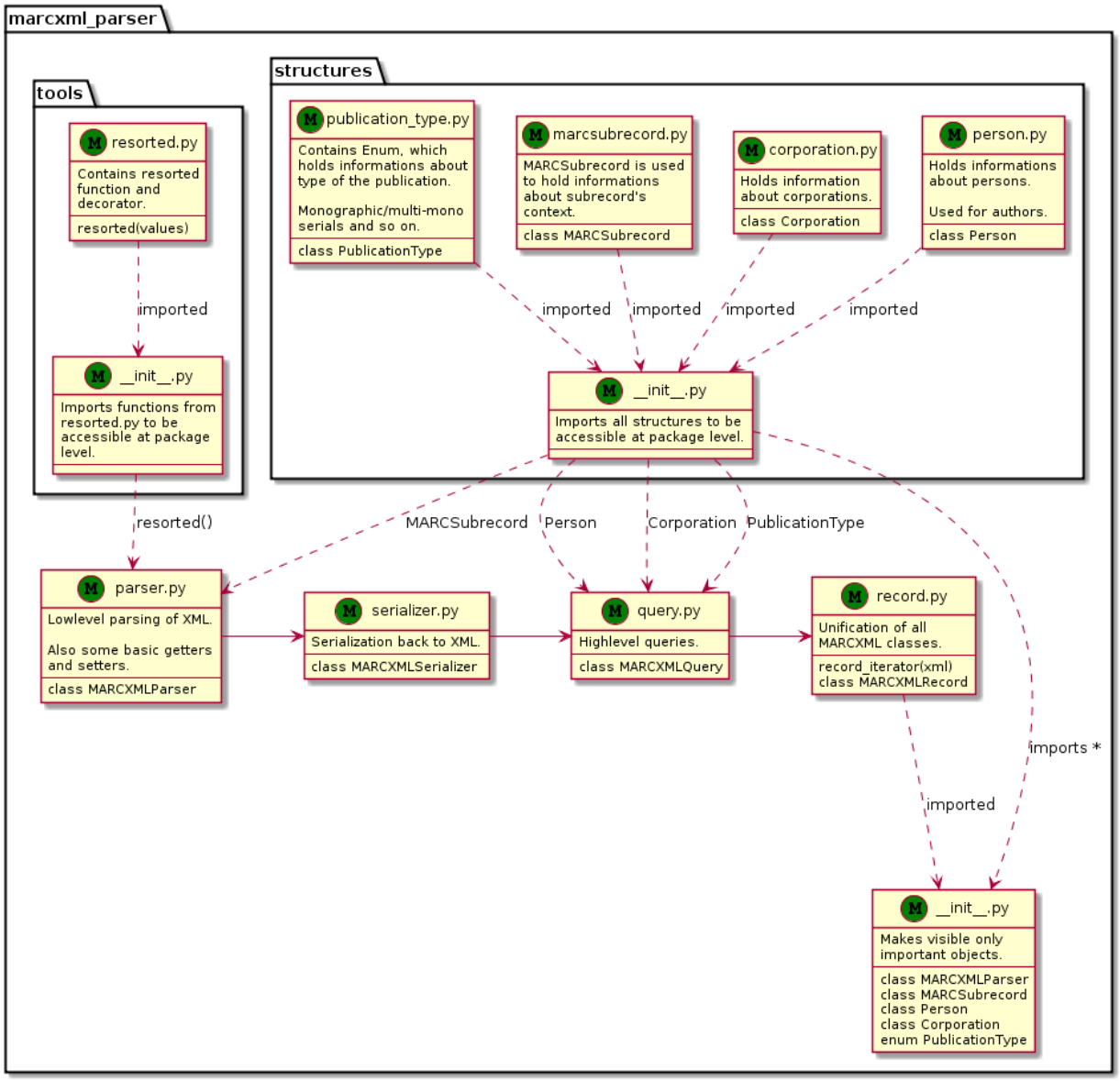
CHAPTER 1

Package structure

Parser is split into multiple classes, which each have own responsibility. Most important is class *MARCXMLRecord*, which contains *MARCXMLParser*, *MARCXMLSerializer* and *MARCXMLQuery*.

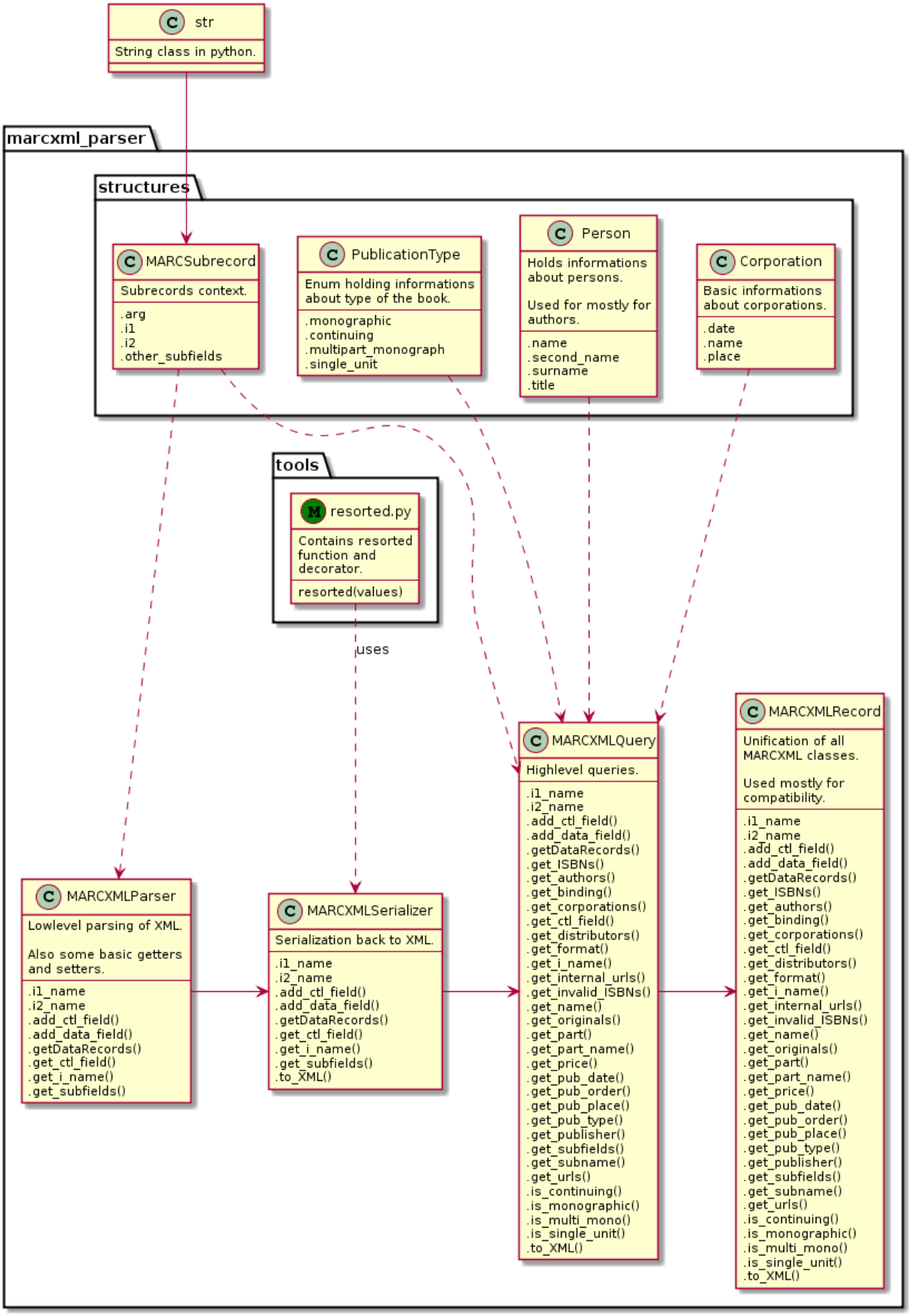
1.1 File relations

Import relations of files in project:



1.2 Class relations

Relations of the classes in project:


str
String class in python.
marcxml_parser
structures
MARCSubrecord
Subrecords context.
.arg
.i1
.i2
.other_subfields
PublicationType
Enum holding informations about type of the book.
.monographic
.continuing
.multipart_monograph
.single_unit
Person
Holds informations about persons.
Used for mostly for authors.
.name
.second_name
.surname
.title
Corporation
Basic informations about corporations.
.date
.name
.place
tools
resorted.py
Contains resorted function and decorator.
resorted(values)
uses
MARCXMLQuery
Highlevel queries.
.i1_name
.i2_name
.add_ctl_field()
.add_data_field()
.getDataRecords()
.get_ISBNs()
.get_authors()
.get_binding()
.get_corporations()
.get_ctl_field()
.get_distributors()
.get_format()
.get_i_name()
.get_internal_urls()
.get_invalid_ISBNs()
.get_name()
.get_originals()
.get_part()
.get_part_name()
.get_price()
.get_pub_date()
.get_pub_order()
.get_pub_place()
.get_pub_type()
.get_publisher()
.get_subfields()
.get_subname()
.get_urls()
.is_continuing()
.is_monographic()
.is_multi_mono()
.is_single_unit()
.to_XML()
MARCXMLRecord
Unification of all MARCXML classes.
Used mostly for compatibility.
.i1_name
.i2_name
.add_ctl_field()
.add_data_field()
.getDataRecords()
.get_ISBNs()
.get_authors()
.get_binding()
.get_corporations()
.get_ctl_field()
.get_distributors()
.get_format()
.get_i_name()
.get_internal_urls()
.get_invalid_ISBNs()
.get_name()
.get_originals()
.get_part()
.get_part_name()
.get_price()
.get_pub_date()
.get_pub_order()
.get_pub_place()
.get_pub_type()
.get_publisher()
.get_subfields()
.get_subname()
.get_urls()
.is_continuing()
.is_monographic()
.is_multi_mono()
.is_single_unit()
.to_XML()
MARCXMLParser
Lowlevel parsing of XML.
Also some basic getters and setters.
.i1_name
.i2_name
.add_ctl_field()
.add_data_field()
.getDataRecords()
.get_ctl_field()
.get_i_name()
.get_subfields()
MARCXMLSerializer
Serialization back to XML.
.i1_name
.i2_name
.add_ctl_field()
.add_data_field()
.getDataRecords()
.get_ctl_field()
.get_i_name()
.get_subfields()
.to_XML()

CHAPTER 2

API

/api/marcxml_parser:

2.1 Parser submodule

class marcxml_parser.parser.**MARCXMLParser**(*xml=None*, *resort=True*)

Bases: object

This class parses everything between <root> elements. It checks, if there is root element, so please, give it full XML.

controlfields is simple dictionary, where keys are field identificators (string, 3 chars). Value is always string.

datafields is little more complicated; it is dictionary made of arrays of dictionaries, which consists of arrays of MARCSubrecord objects and two special parameters.

It sounds horrible, but it is not that hard to understand:

```
    .datafields = {
        "011": ["ind1": " ", "ind2": " "]  # array of 0 or more dicts
        "012": [
            {
                "a": ["a) subsection value"],
                "b": ["b) subsection value"],
                "ind1": " ",
                "ind2": " "
            },
            {
                "a": [
                    "multiple values in a) subsections are possible!",
                    "another value in a) subsection"
                ],
                "c": [
                    "subsection identificator is always one character long"
                ],
                "ind1": " ",
                "ind2": " "
            }
        ]
    }
```

leader
string – Leader of MARC XML document.

oai_marc
bool – True/False, depending if doc is OAI doc or not

controlfields
dict – Controlfields stored in dict.

datafields
dict of arrays of dict of arrays of strings – Datafileds stored in nested dicts/arrays.

Constructor.

Parameters

- **xml** (*str/file, default None*) – XML to be parsed. May be file-like object.
- **resort** (*bool, default True*) – Sort the output alphabetically?

add_ctl_field(*name*, *value*)
Add new control field *value* with under *name* into control field dictionary *controlfields*.

add_data_field(*name*, *i1*, *i2*, *subfields_dict*)
Add new datafield into *datafields* and take care of OAI MARC differencies.

Parameters

- **name** (*str*) – Name of datafield.
- **i1** (*char*) – Value of i1/ind1 parameter.
- **i2** (*char*) – Value of i2/ind2 parameter.
- **subfields_dict** (*dict*) – Dictionary containing subfields (as list).

subfields_dict is expected to be in this format:

```
{
    "field_id": ["subfield data",],
    ...
    "z": ["X0456b"]
}
```

Warning: For your own good, use OrderedDict for *subfields_dict*, or constructor's *resort* parameter set to `True` (it is by default).

Warning: `field_id` can be only one character long!

get_i_name(*num*, *is_oai=None*)
This method is used mainly internally, but it can be handy if you work with with raw MARC XML object and not using getters.

Parameters

- **num** (*int*) – Which indicator you need (1/2).
- **is_oai** (*bool/None*) – If None, *oai_marc* is used.

Returns current name of `i1`/`ind1` parameter based on *oai_marc* property.

Return type str

i1_name
Property getter / alias for `self.get_i_name(1)`.

i2_name
Property getter / alias for `self.get_i_name(2)`.

get_ctl_field(*controlfield*, *alt=None*)
Method wrapper over *controlfields* dictionary.

Parameters

- **controlfield** (*str*) – Name of the controlfield.
- **alt** (*object, default None*) – Alternative value of the *controlfield* when *controlfield* couldn't be found.

Returns record from given *controlfield*

Return type str

getDataRecords(*datafield*, *subfield*, *throw_exceptions=True*)
Deprecated since version Use: *get_subfields()* instead.

get_subfields(*datafield*, *subfield*, *i1=None*, *i2=None*, *exception=False*)
Return content of given *subfield* in *datafield*.

Parameters

- **datafield** (*str*) – Section name (for example "001", "100", "700").
- **subfield** (*str*) – Subfield name (for example "a", "1", etc..).
- **i1** (*str, default None*) – Optional i1/ind1 parameter value, which will be used for search.
- **i2** (*str, default None*) – Optional i2/ind2 parameter value, which will be used for search.
- **exception** (*bool*) – If `True`, `KeyError` is raised when method couldn't found given *datafield* / *subfield*. If `False`, blank array `[]` is returned.

Returns of *MARCSubrecord*.

Return type list

Raises `KeyError` – If the subfield or datafield couldn't be found.

Note: MARCSubrecord is practically same thing as string, but has defined `MARCSubrecord.i1()` and `MARCSubrecord.i2` methods.

You may need to be able to get this, because MARC XML depends on i/ind parameters from time to time (names of authors for example).

2.2 Serializer sub-module

class marcxml_parser.serializer.**MARCXMLSerializer**(*xml=None*, *resort=True*)
Bases: *marcxml_parser.parser.MARCXMLParser*

Class which holds all the data from parser, but contains also XML serialization methods.

to_XML()
Serialize object back to XML string.

Returns String which should be same as original input, if everything works as expected.

Return type str

__str__()
Alias for *to_XML()*.

2.3 Query sub-module

class marcxml_parser.query.**MARCXMLQuery**(*xml=None*, *resort=True*)
Bases: *marcxml_parser.serializer.MARCXMLSerializer*

This class defines highlevel getters over MARC XML / OAI records.

get_name(**args*, ***kwargs*)

Returns Name of the book.

Return type str

Raises KeyError – When name is not specified.

get_subname(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *subname* record is not found.

Returns Subname of the book or *undefined* if *subname* is not found.

Return type str

get_price(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *price* record is not found.

Returns Price of the book (with currency) or *undefined* if *price* is not found.

Return type str

get_part(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *part* record is not found.

Returns Which part of the book series is this record or *undefined* if *part* is not found.

Return type str

get_part_name(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *part_name* record is not found.

Returns Name of the part of the series. or *undefined* if *part_name* is not found.

Return type str

get_publisher(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *publisher* record is not found.

Returns Name of the publisher ("Grada" for example) or *undefined* if *publisher* is not found.

Return type str

get_pub_date(*undefined=''*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *pub_date* record is not found.

Returns Date of publication (month and year usually) or *undefined* if *pub_date* is not found.

Return type str

get_pub_order(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *pub_order* record is not found.

Returns Information about order in which was the book published or *undefined* if *pub_order* is not found.

Return type str

get_pub_place(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *pub_place* record is not found.

Returns Name of city/country where the book was published or *undefined* if *pub_place* is not found.

Return type str

get_format(**args*, ***kwargs*)

Parameters **undefined** (*optional*) – Argument, which will be returned if the *format* record is not found.

Returns

Dimensions of the book ('23 cm' for example) or *undefined* if *format* is not found.

Return type str

get_authors()

Returns Authors represented as *Person* objects.

Return type list

get_corporations(*roles=['dst']*)

Parameters **roles** (*list, optional*) – Specify which types of corporations you need. Set to `["any"]` for any role, `["dst"]` for distributors, etc..

Note: See http://www.loc.gov/marc/relators/relaterm.html for details.

Returns *Corporation* objects specified by roles parameter.

Return type list

get_distributors()

Returns Distributors represented as *Corporation* object.

Return type list

get_invalid_ISBNs()
Get list of invalid ISBN (`020z`).

Returns List with INVALID ISBN strings.

Return type list

get_ISBNs()
Get list of VALID ISBN.

Returns List with *valid* ISBN strings.

Return type list

get_invalid_ISSNs()
Get list of invalid ISSNs (`022z` + `022y`).

Returns List with INVALID ISSN strings.

Return type list

get_ISSNs()
Get list of VALID ISSNs (`022a`).

Returns List with *valid* ISSN strings.

Return type list

get_linking_ISSNs()
Get list of linking ISSNs (`022l`).

Returns List with linking ISSN strings.

Return type list

get_binding()

Returns Array of strings with bindings (`["brož."]`) or blank list.

Return type list

get_originals()

Returns List of strings with names of original books (names of books in original language, before translation).

Return type list

get_urls()
Content of field `856u42`. Typically URL pointing to producers homepage.

Returns List of URLs defined by producer.

Return type list

get_internal_urls()
URL's, which may point to edeposit, aleph, kramerius and so on.

Fields `856u40`, `998a` and `URLu`.

Returns List of internal URLs.

Return type list

get_pub_type()

Returns *PublicationType* enum **value**.

Return type *PublicationType*

is_monographic()

Returns True if the record is monographic.

Return type bool

is_multi_mono()

Returns True if the record is multi_mono.

Return type bool

is_continuing()

Returns True if the record is continuing.

Return type bool

is_single_unit()

Returns True if the record is single unit.

Return type bool

__getitem__(*item*)
Query inteface shortcut for `MARCXMLParser.get_ctl_fields()` and *MARCXMLParser.get_subfields()*.

First three characters are considered as *datafield*, next character as *subfield* and optionaly, two others as *i1* / *i2* parameters.

Returned value is str/None in case of `len(item) == 3` (ctl_fields) or list (or blank list) in case of `len(item) >= 4`.

Returns See *MARCXMLParser.get_subfields()* for details, or None in case that nothing was found.

Return type list/str

get(*item*, *alt=None*)
Standard dict-like .get() method.

Parameters

- **item** (*str*) – See *__getitem__()* for details.
- **alt** (*default None*) – Alternative value, if item is not found.

Returns *item* or *alt*, if item is not found.

Return type obj

2.4 Record sub-module

class marcxml_parser.record.**MARCXMLRecord**(*xml=None*, *resort=True*)
Bases: *marcxml_parser.query.MARCXMLQuery*

Syndication of *MARCXMLParser*, *MARCXMLSerializer* and *MARCXMLQuery* into one class for backward compatibility.

marcxml_parser.record.**record_iterator**(*xml*)
Iterate over all <record> tags in *xml*.

Parameters **xml** (*str/file*) – Input string with XML. UTF-8 is prefered encoding, unicode should be ok.

Yields *MARCXMLRecord* – For each corresponding `<record>`.

`/api/structures/structures`:

2.5 Person structure

class `marcxml_parser.structures.person.`**Person**
Bases: *marcxml_parser.structures.person.Person*

This class represents informations about persons as they are defined in MARC standards.

name
str

second_name
str

surname
str

title
str

2.6 Corporation structure

class `marcxml_parser.structures.corporation.`**Corporation**
Bases: *marcxml_parser.structures.corporation.Corporation*

Informations about corporations (fields 110, 610, 710, 810).

name
str – Name of the corporation.

place
str – Location of the corporation/action.

date
str – Date in unspecified format.

2.7 MARCSubrecord structure

class `marcxml_parser.structures.marcsubrecord.`**MARCSubrecord**(*val*, *i1*, *i2*, *other_subfields*)
Bases: `str`

This class is used to store data returned from `MARCXMLParser.get_datafield()`.

It may look like overshot, but when you are parsing the MARC XML, values from *subrecords*, you need to know the context in which the *subrecord* is put.

This context is provided by the `i1`/`i2` values, but sometimes it is also useful to have access to the other subfields from this *subrecord*.

val
str – Value of *subrecord*.

ind1
char – Indicator one.

ind2
char – Indicator two.

other_subfields
dict – Dictionary with other subfields from the same *subrecord*.

2.8 PublicationType enum

class marcxml_parser.structures.publication_type.**PublicationType**
Bases: enum.Enum

Enum used to decide type of the publication.

monographic = <PublicationType.monographic: 0>

continuing = <PublicationType.continuing: 1>

multipart_monograph = <PublicationType.multipart_monograph: 2>

single_unit = <PublicationType.single_unit: 3>

/api/tools/tools:

2.9 Resorted sub-module

marcxml_parser.tools.resorted.**resorted**(*values*)
Sort values, but put numbers after alphabetically sorted words.

This function is here to make outputs diff-compatible with Aleph.

Example::

```
>>> sorted(["b", "1", "a"])
['1', 'a', 'b']
>>> resorted(["b", "1", "a"])
['a', 'b', '1']
```

Parameters **values** (*iterable*) – any iterable object/list/tuple/whatever.

Returns list of sorted values, but with numbers after words

CHAPTER 3

Usage example

3.1 Example of usage

Lets say, that you have following MARC OAI document, which you need to process:

```
<record>
<metadata>
<oai_marc>
<fixfield id="LDR">-----nas-a22------a-4500</fixfield>
<fixfield id="FMT">SE</fixfield>
<fixfield id="001">nkc20150003059</fixfield>
<fixfield id="003">CZ-PrNK</fixfield>
<fixfield id="005">20150326133612.0</fixfield>
<fixfield id="007">ta</fixfield>
<fixfield id="008">150312c20149999xr--u---------0---b0cze--</fixfield>
<varfield id="BAS" i1=" " i2=" ">
<subfield label="a">01</subfield>
</varfield>
<varfield id="040" i1=" " i2=" ">
<subfield label="a">ABA001</subfield>
<subfield label="b">cze</subfield>
</varfield>
<varfield id="245" i1="0" i2="0">
<subfield label="a">Echa ... :</subfield>
<subfield label="b">[fórum pro literární vědu] /</subfield>
<subfield label="c">Jiří Brabec ... [et al.]</subfield>
</varfield>
<varfield id="246" i1="3" i2=" ">
<subfield label="a">Echa Institutu pro studium literatury ...</subfield>
</varfield>
<varfield id="260" i1=" " i2=" ">
<subfield label="a">Praha :</subfield>
<subfield label="b">Institut pro studium literatury,</subfield>
<subfield label="c">[2014?]-</subfield>
</varfield>
<varfield id="300" i1=" " i2=" ">
<subfield label="a">^^^ online zdroj</subfield>
</varfield>
<varfield id="362" i1="0" i2=" ">
<subfield label="a">2010/2011</subfield>
</varfield>
<varfield id="500" i1=" " i2=" ">
<subfield label="a">Součástí názvu je označení rozmezí let, od r. 2012 součástí názvu označení kalend
```

```
</varfield>
<varfield id="500" i1=" " i2=" ">
<subfield label="a">V některých formátech autoři neuvedeni</subfield>
</varfield>
<varfield id="500" i1=" " i2=" ">
<subfield label="a">Jednotlivé sv. mají ISBN</subfield>
</varfield>
<varfield id="500" i1=" " i2=" ">
<subfield label="a">Popsáno podle: 2010/2011</subfield>
</varfield>
<varfield id="700" i1="1" i2=" ">
<subfield label="a">Brabec, Jiří</subfield>
<subfield label="4">aut</subfield>
</varfield>
<varfield id="856" i1="4" i2="0">
<subfield label="u">http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
<subfield label="z">2010-2011</subfield>
<subfield label="4">N</subfield>
</varfield>
<varfield id="856" i1="4" i2="0">
<subfield label="u">http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
<subfield label="z">2010-2011</subfield>
<subfield label="4">N</subfield>
</varfield>
<varfield id="856" i1="4" i2="0">
<subfield label="u">http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
<subfield label="z">201-2011</subfield>
<subfield label="4">N</subfield>
</varfield>
<varfield id="856" i1="4" i2="0">
<subfield label="u">http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
<subfield label="z">2012</subfield>
<subfield label="4">N</subfield>
</varfield>
<varfield id="856" i1="4" i2="0">
<subfield label="u">http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
<subfield label="z">2013</subfield>
<subfield label="4">N</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-10-6</subfield>
<subfield label="q">(2010/2011 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">pdf)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-09-0</subfield>
<subfield label="q">(2010/2011 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">Mobipocket)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-08-3</subfield>
<subfield label="q">(2010/2011 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">ePub)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
```

```
<subfield label="a">978-80-87899-06-9</subfield>
<subfield label="q">(2012 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">Mobipocket)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-05-2</subfield>
<subfield label="q">(2012 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">ePub)</subfield>
<subfield label="z">978-80-87899-07-6</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-07-6</subfield>
<subfield label="q">(2012 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">pdf)</subfield>
<subfield label="z">978-80-87899-05-2</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-02-1</subfield>
<subfield label="q">(2013 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">ePub)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-03-8</subfield>
<subfield label="q">(2013 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">Mobipocket)</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-04-5</subfield>
<subfield label="q">(2013 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">pdf)</subfield>
</varfield>
<varfield id="910" i1=" " i2=" ">
<subfield label="a">ABA001</subfield>
<subfield label="s">2010/2011, 2012-2013-</subfield>
</varfield>
<varfield id="998" i1=" " i2=" ">
<subfield label="a">http://aleph.nkp.cz/F/?func=direct&amp;doc_number=000003059&amp;local_base=CZE-DI
</varfield>
<varfield id="PSP" i1=" " i2=" ">
<subfield label="a">BK</subfield>
</varfield>
<varfield id="IST" i1="1" i2=" ">
<subfield label="a">jp20150312</subfield>
<subfield label="b">kola</subfield>
</varfield>
</oai_marc>
</metadata>
</record>
```

This document is saved at `tests/data/aleph_epub.xml`. To parse this document, you just open it and create *MARCXMLRecord* object from the string:

```
from marcxml_parser import MARCXMLRecord

with open("tests/data/aleph_epub.xml") as f:
    rec = MARCXMLRecord(f.read())
```

3.1.1 Lowlevel access

All the controlfields and datafields were parsed into *controlfields* and *datafields*:

```
>>> rec.controlfields
```

```
OrderedDict([
    ('LDR', '-----nas-a22------a-4500'),
    ('FMT', 'SE'),
    ('001', 'nkc20150003059'),
    ('003', 'CZ-PrNK'),
    ('005', '20150326133612.0'),
    ('007', 'ta'),
    ('008', '150312c20149999xr--u---------0---b0cze--'),
])
```

```
>>> rec.datafields
```

```
OrderedDict([
    ('BAS', [OrderedDict([('i1', ' '), ('i2', ' '), ('a', ['01'])])]),
    ('040', [OrderedDict([
        ('i1', ' '),
        ('i2', ' '),
        ('a', ['ABA001']),
        ('b', ['cze'])
    ])]),
    ('245', [OrderedDict([
        ('i1', '0'),
        ('i2', '0'),
        ('a', ['Echa ... :']),
        ('b', ['[f\xc3\xb3rum pro liter\xc3\xa1rn\xc3\xad v\xc4\x9bdu] /']),
        ('c', ['Ji\xc5\x99\xc3\xad Brabec ... [et al.]'])
    ])]),
    ('246', [OrderedDict([
        ('i1', '3'),
        ('i2', ' '),
        ('a', ['Echa Institutu pro studium literatury ...'])
    ])]),
    ('260', [OrderedDict([
        ('i1', ' '),
        ('i2', ' '),
        ('a', ['Praha :']),
        ('b', ['Institut pro studium literatury,']),
        ('c', ['[2014?]-'])
    ])]),
    ('300', [OrderedDict([
        ('i1', ' '),
        ('i2', ' '),
        ('a', ['^^^ online zdroj'])
    ])]),
    ('362', [OrderedDict([
        ('i1', '0'),
```

```
        ('i2', ' '),
        ('a', ['2010/2011'])
    ])]),
    ('500',
        [OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['Sou\xc4\x8d\xc3\xa1st\xc3\xad n\xc3\xa1zvu je ozna\xc4\x8den\xc3\xad rozmez\xc3\x
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['V n\xc4\x9bkter\xc3\xbdch form\xc3\xa1tech auto\xc5\x99i neuvedeni'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['Jednotliv\xc3\xa9 sv. maj\xc3\xad ISBN'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['Pops\xc3\xa1no podle: 2010/2011'])
        ])
    ]),
    ('700', [OrderedDict([
        ('i1', '1'),
        ('i2', ' '),
        ('a', ['Brabec, Ji\xc5\x99\xc3\xad']),
        ('4', ['aut'])
    ])]),
    ('856', [
        OrderedDict([
            ('i1', '4'),
            ('i2', '0'),
            ('u', ['http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
            ('z', ['2010-2011']),
            ('4', ['N'])
        ]),
        OrderedDict([
            ('i1', '4'),
            ('i2', '0'),
            ('u', ['http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
            ('z', ['2010-2011']),
            ('4', ['N'])
        ]),
        OrderedDict([
            ('i1', '4'),
            ('i2', '0'),
            ('u', ['http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
            ('z', ['201-2011']),
            ('4', ['N'])
        ]),
        OrderedDict([
            ('i1', '4'),
            ('i2', '0'),
            ('u', ['http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
            ('z', ['2012']),
```

```
            ('4', ['N'])
        ]),
        OrderedDict([
            ('i1', '4'),
            ('i2', '0'),
            ('u', ['http://edeposit-test.nkp.cz/producents/nakladatelstvi-delta/epublications/echa-20
            ('z', ['2013']),
            ('4', ['N'])
        ])
    ]),
    ('902', [
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-10-6']),
            ('q', ['(2010/2011 :', 'online :', 'pdf)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-09-0']),
            ('q', ['(2010/2011 :', 'online :', 'Mobipocket)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a',"['978-80-87899-08-3']),
            ('q', ['(2010/2011 :', 'online :', 'ePub)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-06-9']),
            ('q', ['(2012 :', 'online :', 'Mobipocket)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-05-2']),
            ('q', ['(2012 :', 'online :' , 'ePub)']),
            ('z', ['978-80-87899-07-6'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-07-6']),
            ('q', ['(2012 :', 'online :', 'pdf)']),
            ('z', ['978-80-87899-05-2'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-02-1']),
            ('q', ['(2013 :', 'online :' , 'ePub)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
```

```
            ('a', ['978-80-87899-03-8']),
            ('q', ['(2013 :', 'online :' , 'Mobipocket)'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-04-5']),
            ('q', ['(2013 :', 'online :' , 'pdf)'])
        ]),
    ]),
    ('910', [OrderedDict([
        ('i1', ' '),
        ('i2', ' '),
        ('a', ['ABA001']),
        ('s', ['2010/2011, 2012-2013-'])])
    ]),
    ('998', [OrderedDict([
        ('i1', ' '),
        ('i2', ' '),
        ('a', ['http://aleph.nkp.cz/F/?func=direct&doc_number=000003059&local_base=CZE-DEP'])
    ])]),
    ('PSP', [OrderedDict([('i1', ' '), ('i2', ' '), ('a', ['BK'])])]),
    ('IST', [OrderedDict([
        ('i1', '1'),
        ('i2', ' '),
        ('a', ['jp20150312']),
        ('b', ['kola'])
    ])]),
])
```

As you can see, this format is probably too much lowlevel, than you would ever want to use, but it demonstrates one important aspect of the parser; All values are parsed to (ordered) dicts.

That means, that XML:

```
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-05-2</subfield>
<subfield label="q">(2012 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">ePub)</subfield>
<subfield label="z">978-80-87899-07-6</subfield>
</varfield>
<varfield id="902" i1=" " i2=" ">
<subfield label="a">978-80-87899-07-6</subfield>
<subfield label="q">(2012 :</subfield>
<subfield label="q">online :</subfield>
<subfield label="q">pdf)</subfield>
<subfield label="z">978-80-87899-05-2</subfield>
</varfield>
```

Is parsed to:

```
OrderedDict([
    ('902', [
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-05-2']),
            ('q', ['(2012 :', 'online :' , 'ePub)']),
```

```
            ('z', ['978-80-87899-07-6'])
        ]),
        OrderedDict([
            ('i1', ' '),
            ('i2', ' '),
            ('a', ['978-80-87899-07-6']),
            ('q', ['(2012 :', 'online :', 'pdf)']),
            ('z', ['978-80-87899-05-2'])
        ]),
    ]),
])
```

Which is equivalent to following code (without ordered dicts for simplicity):

```
{
    "902": [
        {
            'i1': ' ',
            'i2': ' ',
            'a': ['978-80-87899-05-2'],
            'q': ['(2012 :', 'online :' , 'ePub)'],
            'z': ['978-80-87899-07-6']
        },
        {
            'i1': ' ',
            'i2': ' ',
            'a': ['978-80-87899-07-6'],
            'q': ['(2012 :', 'online :', 'pdf)'],
            'z': ['978-80-87899-05-2']
        }
    ]
}
```

Notice the `q` sub-record, which was three times in original XML and now is stored as list.

This is the reason why most of the getters returns lists and not just simply values - the nature of MARC records are lists.

3.1.2 Getters

To access values inside *controlfields* and *datafields*, you can use direct access to internal *dict* structure:

```
>>> rec.datafields["902"][4]["q"]
```

```
['(2012 :', 'online :', 'ePub)']
```

but I can highly recommend to use highlevel getters:

```
>>> rec.get_subfields("902", "q")
```

```
[
    '(2010/2011 :',
    'online :',
    'pdf)',
    '(2010/2011 :',
    'online :',
    'Mobipocket)',
    '(2010/2011 :',
```

```
    'online :',
    'ePub)',
    '(2012 :',
    'online :',
    'Mobipocket)',
    '(2012 :',
    'online :',
    'ePub)',
    '(2012 :',
    'online :',
    'pdf)',
    '(2013 :',
    'online :',
    'ePub)',
    '(2013 :',
    'online :',
    'Mobipocket)',
    '(2013 :',
    'online :',
    'pdf)',
]
```

Whoa. What happened? There weren't specified any more arguments to *get_subfields()*, so all the 902q subrecords were returned.

Lets look at the first returned item:

```
>>> rec.get_subfields("902", "q")[0]
```

```
'(2010/2011 :'
```

It looks like a string. But in fact, it is *MARCSubrecord* instance:

```
>>> type(rec.get_subfields("902", "q")[0])
```

```
<class 'marcxml_parser.structures.marcsubrecord.MARCSubrecord'>
```

That means, that it has more context, than ordinary string:

```
>>> r = rec.get_subfields("902", "q")[0]
>>> r.val
```

```
'(2010/2011 :'
```

```
>>> r.i1
```

```
' '
```

```
>>> r.i2
```

```
' '
```

```
>>> r.other_subfields
```

```
OrderedDict([('i1', ' '), ('i2', ' '), ('a', ['978-80-87899-10-6']), ('q', ['(2010/2011 :', 'online
```

3.1.3 Highlevel getters

Here is the list of all highlevel getters are defined by *MARCXMLQuery*:

- *get_name()*
- *get_subname()*
- *get_price()*
- *get_part()*
- *get_part_name()*
- *get_publisher()*
- *get_pub_date()*
- *get_pub_order()*
- *get_pub_place()*
- *get_format()*
- *get_authors()*
- *get_corporations()*
- *get_distributors()*
- *get_ISBNs()*
- *get_binding()*
- *get_originals()*

You will probably like the indexing operator, which can be used as shortcut for `rec.get_subfields` calls, for example `rec.get_subfields("500", "a")` can be shortened to:

```
>>> rec["500a"]
```

```
[
    'Sou\xc4\x8d\xc3\xa1st\xc3\xad n\xc3\xa1zvu je ...',  # shortened
    'V n\xc4\x9bkter\xc3\xbdch form\xc3\xa1tech auto\xc5\x99i neuvedeni',
    'Jednotliv\xc3\xa9 sv. maj\xc3\xad ISBN',
    'Pops\xc3\xa1no podle: 2010/2011'
]
```

```
>>> rec["001"]
```

```
'nkc20150003059
```

or with `i1`/`i2` arguments:

```
>>> rec["500a 9"]  # equivalent to rec.get_subfields("500", "a", i1=" ", i2="9")
```

```
[]
```

(nothing was returned, because there isn't `i1` == ' ' and `i2` == `9`)

```
>>> rec["902q  "]
```

```
[
    'Sou\xc4\x8d\xc3\xa1st\xc3\xad n\xc3\xa1zvu je ...',  # shortened
    'V n\xc4\x9bkter\xc3\xbdch form\xc3\xa1tech auto\xc5\x99i neuvedeni',
    'Jednotliv\xc3\xa9 sv. maj\xc3\xad ISBN',
    'Pops\xc3\xa1no podle: 2010/2011'
]
```

CHAPTER 4

Installation

Module is hosted at PYPI, and can be easily installed using PIP:

```
sudo pip install marcxml_parser
```

4.1 Source code

Project is released as opensource (MIT) and source code can be found at GitHub:

- https://github.com/edeposit/marcxml_parser

4.2 Unittests

Almost every feature of the project is tested by unittests. You can run those tests using provided `run_tests.sh` script, which can be found in the root of the project.

4.2.1 Requirements

This script expects that pytest is installed. In case you don't have it yet, it can be easily installed using following command:

```
pip install --user pytest
```

or for all users:

```
sudo pip install pytest
```

4.2.2 Example

```
$ ./run_tests.sh
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.26 -- pytest-2.6.4
collected 66 items

tests/test_module.py ..
tests/test_parser.py ............
tests/test_query.py ..............................
```

```
tests/test_record.py .
tests/test_serializer.py .......
tests/structures/test__structures_module.py .
tests/structures/test_corporation.py .
tests/structures/test_marcsubrecord.py .
tests/structures/test_person.py .
tests/structures/test_publication_type.py .
tests/tools/test_resorted.py ........

========================== 66 passed in 1.14 seconds ==========================
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

m

Symbols

A

C

D

G

I

L

M

N

O

P

R

S

T

V