EULxml Documentation

Release 0.22.1

Emory University Libraries

February 17, 2016

Contents

EULxml is an extensible library for reading and writing XML documents in idiomatic Python. It allows developers to map predictable XML node structures to *XmlObject* subclasses, using field definitions to map XPath expressions directly to Python attributes.

For projects using Django, it also provides utilities for exposing *XmlObject* instances to web users with `XmlObjectForm`. As a bonus, EULxml happens to include an XPath parser in *eulxml.xpath*.

CHAPTER 1

Contents

1.1 Change & Version Information

The following is a summary of changes and improvements to `eulxml`. New features in each version should be listed, with any necessary information about installation or upgrade notes.

1.1.1 0.22.1

- bugfix: workaround for resolver error failing to load schemas in some cases

1.1.2 0.22

- New method (`eulxml.xmlmap.load_xslt()`) to load and precompile XSLT that will be used more than once.
- Re-worked *`eulxml.xmlmap.XmlObject.xsl_transform()`* to avoid malloc errors and segmentation faults and to allow using precompiled XSLT documents.
- Support for float field types in eulxml.xmlmap. Contributed by jilott.

1.1.3 0.21.2

- Bug fix: correctly support parameters in *`eulxml.xmlmap.XmlObject.xsl_transform()`*
- Automatically encode string parameter values passed to *`xsl_transform()`* as lxml string parameters (`lxml.etree.XSLT.strparam`)

1.1.4 0.21.1

- Bug fix: *`eulxml.xmlmap.XmlObject.xsl_transform()`* now recognizes text output as a valid, non-empty XSL result

1.1.5 0.21

- Add default unicode output of date value for MODS date fields (`Date` and all date variants)

- Bug fix: `XmlObjectForm` now uses the field order as defined on the form when updating the XML instance (fix for XML where schema requires fields in a specific order)

1.1.6 0.20.3

- Revert unused namespace cleanup change to serialization it generates less optimal output in certain cases.
- Minor updates to *eadmap*:
  - Added mapping for *xlink:show* attribute to *DigitalArchivalObject*
  - Added mapping for *note* field *Index*
  - Changed `Note` paragraph content from string list to node list, to support formatting.
  - Added mapping for `processinfo` to *ArchivalDescription* and *Component*

1.1.7 0.20.2

- Unused namespaces will now be cleaned up before serialization in :meth:‘eulxml.xmlmap.XmlObject.serialize’ and :meth:‘eulxml.xmlmap.XmlObject.serializeDocument’
- *eulxml.xmlmap.eadmap* have been updated with root element names where possible, to better support using *eadmap* to update or modify EAD documents.

1.1.8 0.20.1

- Adjust *eulxml.xmlmap* fields for better results when inspected by sphinx autodoc or other similar tools.

1.1.9 0.20.0

- Update *eulxml.xmlmap.mods* with support for id attribute on top-level MODS element. Contributed by bcail.
- Update *eulxml.xmlmap.eadmap* with support for digital archival object tags.
- Add *eulxml.xmlmap.fields.DateField* to support date fields separately from *eulxml.xmlmap.fields.DateTimeField*; also includes minimal support for date fields in `eulxml.forms.xmlobject.XmlObjectForm`.

1.1.10 0.19.1

- Pinned MODS version to 3.4 to guard against new versions of the schema breaking validation

1.1.11 0.19.0

- Corrected a minor bug where schema validation errors were not cleared between multiple validations.
- To avoid permission denied warning for auto-generated parser files, parsetab files are now created in python tempdir if the running user doesn’t have write permission in the package installation directory. [Issue 1]

- When an XSLT transformation results in an empty document, `eulxml.xmlap.XmlObject.xsl_transform()` now returns None. [Issue 6]
- Development requirements can now be installed as an optional requirement of the eulxml package (`pip install "eulxml[dev]"`).
- Unit tests have been updated to use `nose`
- New functionality in *`eulxml.xmlmap.cerp`* for parsing email dates and generating CERP xml from a Python email message object.

1.1.12 0.18.0 - Formset Ordering and DateTime

- `eulxml.forms.xmlobject.XmlObjectForm` formsets now support `can_order`.
- *`eulxml.xmlmap.fields.DateTimeField`* is now included in available *`eulxml.xmlmap`* fields. This replaces the previously officially-unreleased, under-documented and -tested and misnamed `DateField`. Datetime format handling and whitespace normalization contributed by jheath-.

1.1.13 0.17.1 - Bugfix Release

- Fixed an error in eulxml.xpath parse that resulted in parse errors when other lexers are defined.

1.1.14 0.17.0 - Minor Enhancements

- *`eulxml.xmlmap.XmlObject`* now supports lazy-loading for XSD Schemas. To take advantage of this feature, *`XmlObject`* subclasses should define an `XSD_SCHEMA` location but should not set an `xmlschema`.
- When *field* mapped on a *`eulxml.xmlmap.XmlObject`* is deleted, any XPath predicates that could have been automatically constructed when setting the value will be deleted from the *`XmlObject`* where possible, if they are otherwise empty.

1.1.15 0.16.0 - MODS and PREMIS

- Add basic support for MODS in *`eulxml.xmlmap.mods`*.
- Add basic support for PREMIS in *`eulxml.xmlmap.premis`*.
- Minor logging and error handling improvements.

1.1.16 0.15.3 - Minor Enhancement

- Downgrade the lack of an HTTP_PROXY set in the environment from a RuntimeError to a Warning with schema validation disabled.

1.1.17 0.15.2 - Bugfix Release

- Fixed an error in the dependency structure that prevented the package from being used after installation through PyPI.

1.1.18 0.15.1 - Bugfix Release

- Fixed an error in the dependency structure that prevented the package from being installed through PyPI.

1.1.19 0.15.0 - Initial Release

- Split out xml-related components (*xpath*, *xmlmap*, and `forms`) from `eulcore` into `eulxml` for easier re-use.

1.2 `eulxml.xmlmap` – Map XML to Python objects

eulxml.xmlmap makes it easier to map XML to Python objects. The Python DOM does some of this, of course, but sometimes it's prettier to wrap an XML node in a typed Python object and assign attributes on that object to reference subnodes by XPath expressions. This module provides that functionality.

1.2.1 `XmlObject` Instances

`eulxml.xmlmap.eadmap` - Encoded Archival Description (EAD)

General Information

The Encoded Archival Description (EAD) is a a standard xml format for encoding finding aids. For more information, please consult the official Library of Congress EAD site.

This set of xml objects is an attempt to make the major fields of an EAD document accessible for search and display. It is by no means an exhaustive mapping of all EAD elements in all their possible configurations.

Encoded Archival Description

LOC documentation for EAD element

Nearly all fields in all EAD XmlObjects are mapped as `eulxml.xmlmap.XPathString` or `eulxml.xmlmap.XPathStringList`, except for custom EAD sub-objects, which are indicated where in use.

class `eulxml.xmlmap.eadmap.`**`EncodedArchivalDescription`**(*node*[, *context*])

XmlObject for an Encoded Archival Description (EAD) Finding Aid (Schema-based). All XPaths use the EAD namespace; this class can not be used with non-namespaced, DTD-based EAD.

Expects node passed to constructor to be top-level *ead* element.

`abstract` = **<eulxml.xmlmap.fields.NodeField>**
collection level abstract - *archdesc[@level="collection"]/did/abstract*

`archdesc` = **<eulxml.xmlmap.fields.NodeField>**
ArchivalDescription - *archdesc*

`author` = **<eulxml.xmlmap.fields.StringField>**
record author - *eadheader/filedesc/titlestmt/author*

`dsc` = **<eulxml.xmlmap.fields.NodeField>**
SubordinateComponents *archdesc/dsc*; accessible at top-level for convenience

eadid = <eulxml.xmlmap.fields.NodeField>
ead id `EadId` - *eadheader/eadid*

file_desc = <eulxml.xmlmap.fields.NodeField>
`FileDescription` - *filedesc*

id = <eulxml.xmlmap.fields.StringField>
top-level id attribute - *@id*; preferable to use eadid

physical_desc = <eulxml.xmlmap.fields.StringField>
collection level physical description - *archdesc[@level="collection"]/did/physdesc*

profiledesc = <eulxml.xmlmap.fields.NodeField>
`ProfileDescription` - *profiledesc*

title = <eulxml.xmlmap.fields.NodeField>
record title - *eadheader/filedesc/titlestmt/titleproper*

unittitle = <eulxml.xmlmap.fields.NodeField>
unit title for the archive - *archdesc[@level="collection"]/did/unittitle*

Archival Description

LOC documentation for EAD archdesc element

class `eulxml.xmlmap.eadmap.`**ArchivalDescription**(*node*[, *context*])
Archival description, contains the bulk of the information in an EAD document.

Expected node element passed to constructor: *ead/archdesc*.

access_restriction = <eulxml.xmlmap.fields.NodeField>
access restrictions *`Section`* - *accessrestrict*

acquisition_info = <eulxml.xmlmap.fields.NodeField>
acquistion info *`Section`* - *acqinfo*

alternate_form = <eulxml.xmlmap.fields.NodeField>
alternative form available *`Section`* - *altformavail*

arrangement = <eulxml.xmlmap.fields.NodeField>
arrangement *`Section`* - *arrangement*

bibliography = <eulxml.xmlmap.fields.NodeField>
bibliography *`Section`* - *bibliograhy*

biography_history = <eulxml.xmlmap.fields.NodeField>
biography or history *`Section`* - *bioghist*

controlaccess = <eulxml.xmlmap.fields.NodeField>
`ControlledAccessHeadings` - *controlaccess*; subject terms, names, etc.

custodial_history = <eulxml.xmlmap.fields.NodeField>
custodial history *`Section`* - *custodhist*

dao_list = <eulxml.xmlmap.fields.NodeListField>
list of digital archival object references as *`DigitalArchivalObject`*

did = <eulxml.xmlmap.fields.NodeField>
descriptive identification *`DescriptiveIdentification`* - *did*

extent = <eulxml.xmlmap.fields.StringListField>
extent from the physical description - *did/physdesc/extent*

index = <eulxml.xmlmap.fields.NodeListField>
list of *Index* - *index*; e.g., index of selected correspondents

langmaterial = <eulxml.xmlmap.fields.StringField>
language of the materials - *did/langmaterial*

location = <eulxml.xmlmap.fields.StringField>
physical location - *did/physloc*

originals_location = <eulxml.xmlmap.fields.NodeField>
location of originals *Section* - *originalsloc*

origination = <eulxml.xmlmap.fields.StringField>
origination - *did/origination*

other = <eulxml.xmlmap.fields.NodeField>
other finding aid *Section* - *otherfindaid*

preferred_citation = <eulxml.xmlmap.fields.NodeField>
preferred citation *Section* - *prefercite*

process_info = <eulxml.xmlmap.fields.NodeField>
processing information *Section* - *processinfo*

related_material = <eulxml.xmlmap.fields.NodeField>
related material *Section* - *relatedmaterial*

scope_content = <eulxml.xmlmap.fields.NodeField>
scope and content *Section* - *scopecontent*

separated_material = <eulxml.xmlmap.fields.NodeField>
separated material *Section* - *separatedmaterial*

unitid = <eulxml.xmlmap.fields.NodeField>
Unitid - *did/unitid*

use_restriction = <eulxml.xmlmap.fields.NodeField>
use restrictions *Section* - *userestrict*

Subordinate Components

See also LOC documentation for dsc element , c (component) element

class eulxml.xmlmap.eadmap.**SubordinateComponents**(*node*[, *context*])
Description of Subordinate Components (dsc element); container lists and series.

Expected node element passed to constructor: *ead/archdesc/dsc*.

c = <eulxml.xmlmap.fields.NodeListField>
list of *Component* - *c01*; list of c01 elements directly under this section

hasSeries()
Check if this finding aid has series/subseries.

Determined based on level of first component (series) or if first component has subcomponents present.

Return type boolean

type = <eulxml.xmlmap.fields.StringField>
type of component - *@type*

class eulxml.xmlmap.eadmap.**Component**(*node*[, *context*])
Generic component *cN* (*c1-c12*) element - a subordinate component of the materials

access_restriction = <eulxml.xmlmap.fields.NodeField>
access restrictions *Section* - *accessrestrict*

acquisition_info = <eulxml.xmlmap.fields.NodeField>
acquistion info *Section* - *acqinfo*

alternate_form = <eulxml.xmlmap.fields.NodeField>
alternative form available *Section* - *altformavail*

arrangement = <eulxml.xmlmap.fields.NodeField>
arrangement *Section* - *arrangement*

bibliography = <eulxml.xmlmap.fields.NodeField>
bibliography *Section* - *bibliograhy*

biography_history = <eulxml.xmlmap.fields.NodeField>
biography or history *Section* - *bioghist*

c = <eulxml.xmlmap.fields.NodeListField>
list of *Component* - recursive mapping to any c-level 2-12; *c02|c03|c04|c05|c06|c07|c08|c09|c10|c11|c12*

custodial_history = <eulxml.xmlmap.fields.NodeField>
custodial history *Section* - *custodhist*

dao_list = <eulxml.xmlmap.fields.NodeListField>
list of digital archival object references as *DigitalArchivalObject*

did = <eulxml.xmlmap.fields.NodeField>
DescriptiveIdentification - *did*

hasSubseries()
Check if this component has subseries or not.

Determined based on level of first subcomponent (series or subseries) or if first component has subcomponents present.

rtype boolean

id = <eulxml.xmlmap.fields.StringField>
component id - @*id*

level = <eulxml.xmlmap.fields.StringField>
level of the component - @*level*

originals_location = <eulxml.xmlmap.fields.NodeField>
location of originals *Section* - *originalsloc*

other = <eulxml.xmlmap.fields.NodeField>
other finding aid *Section* - *otherfindaid*

preferred_citation = <eulxml.xmlmap.fields.NodeField>
preferred citation *Section* - *prefercite*

process_info = <eulxml.xmlmap.fields.NodeField>
processing infomration *Section* - *processinfo*

related_material = <eulxml.xmlmap.fields.NodeField>
related material *Section* - *relatedmaterial*

scope_content = <eulxml.xmlmap.fields.NodeField>
scope and content *Section* - *scopecontent*

separated_material = <eulxml.xmlmap.fields.NodeField>
separated material *Section* - *separatedmaterial*

use_restriction = <eulxml.xmlmap.fields.NodeField>
use restrictions *Section* - *userestrict*

Controlled Access Headings

LOC Documentation for controlaccess element

class eulxml.xmlmap.eadmap.**ControlledAccessHeadings**(*node*[, *context*])
Controlled access headings, such as subject terms, family and corporate names, etc.

Expected node element passed to constructor: *contolaccess*.

controlaccess = <eulxml.xmlmap.fields.NodeListField>
list of *ControlledAccessHeadings* - recursive mapping to *controlaccess*

corporate_name = <eulxml.xmlmap.fields.NodeListField>
corporate name *Heading* list - *corpname*

family_name = <eulxml.xmlmap.fields.NodeListField>
family name *Heading* list - *famname*

function = <eulxml.xmlmap.fields.NodeListField>
function *Heading* list - *function*

genre_form = <eulxml.xmlmap.fields.NodeListField>
genre or form *Heading* list - *genreform*

geographic_name = <eulxml.xmlmap.fields.NodeListField>
geographic name *Heading* list - *geogname*

occupation = <eulxml.xmlmap.fields.NodeListField>
occupation *Heading* list - *occupation*

person_name = <eulxml.xmlmap.fields.NodeListField>
person name *Heading* list - *persname*

subject = <eulxml.xmlmap.fields.NodeListField>
subject *Heading* list - *subject*

terms = <eulxml.xmlmap.fields.NodeListField>
list of *Heading* - any allowed control access terms, in whatever order they appear

title = <eulxml.xmlmap.fields.NodeListField>
title *Heading* list - *title*

class eulxml.xmlmap.eadmap.**Heading**(*node*[, *context*])
Generic xml object for headings used under *controlaccess*

source = <eulxml.xmlmap.fields.StringField>
source vocabulary for controlled term - @*source*

value = <eulxml.xmlmap.fields.StringField>
controlled term text value (content of the heading element)

Index and Index Entry

See also LOC Documentation for index element, indexentry element

class eulxml.xmlmap.eadmap.**Index**(*node*[, *context*])
Index (index element); list of key terms and reference information.

Expected node element passed to constructor: *ead/archdesc/index*.

entry = <eulxml.xmlmap.fields.NodeListField>
list of *IndexEntry* - *indexentry*; entry in the index

note = <eulxml.xmlmap.fields.NodeField>
Note

class eulxml.xmlmap.eadmap.**IndexEntry**(*node*[, *context*])
Index entry in an archival description index.

name = <eulxml.xmlmap.fields.NodeField>
access element, e.g. name or subject

ptrgroup = <eulxml.xmlmap.fields.NodeField>
PointerGroup - group of references for this index entry

File Description

See also LOC Documentation for filedesc element, publicationstmt element

class eulxml.xmlmap.eadmap.**FileDescription**(*node*[, *context*])
Bibliographic information about this EAD document.

Expected node element passed to constructor: *ead/eadheader/filedesc*.

publication = <eulxml.xmlmap.fields.NodeField>
publication information - *publicationstmt*

class eulxml.xmlmap.eadmap.**PublicationStatement**(*node*[, *context*])
Publication information for an EAD document.

Expected node element passed to constructor: *ead/eadheader/filedesc/publicationstmt*.

address = <eulxml.xmlmap.fields.NodeField>
address of publication/publisher - *address*

date = <eulxml.xmlmap.fields.NodeField>
DateField - *date*

publisher = <eulxml.xmlmap.fields.StringField>
publisher - *publisher*

Miscellaneous

See also LOC documentation for did element , container element

class eulxml.xmlmap.eadmap.**DescriptiveIdentification**(*node*[, *context*])
Descriptive Information (*did* element) for materials in a component

abstract = <eulxml.xmlmap.fields.NodeField>
abstract - *abstract*

container = <eulxml.xmlmap.fields.NodeListField>
Container - *container*

dao_list = <eulxml.xmlmap.fields.NodeListField>
list of digital archival object references as *DigitalArchivalObject*

langmaterial = <eulxml.xmlmap.fields.StringField>
language of materials - *langmaterial*

location = <eulxml.xmlmap.fields.StringField>
physical location - *physloc*

origination = <eulxml.xmlmap.fields.StringField>
origination - *origination*

physdesc = <eulxml.xmlmap.fields.StringField>
physical description - *physdesc*

unitdate = <eulxml.xmlmap.fields.NodeField>
unit date - *.//unitdate* can be anywhere under the DescriptiveIdentification

unitid = <eulxml.xmlmap.fields.NodeField>
`Unitid` - *unitid*

unittitle = <eulxml.xmlmap.fields.NodeField>
unit title - *unittitle*

class `eulxml.xmlmap.eadmap.`**`Container`**(*node*[, *context*])
Container - `DescriptiveIdentification` subelement for locating materials.

Expected node element passed to constructor: *did/container*.

type = <eulxml.xmlmap.fields.StringField>
type - *@type*

value = <eulxml.xmlmap.fields.StringField>
text value - (contents of the container element)

class `eulxml.xmlmap.eadmap.`**`Section`**(*node*[, *context*])
Generic EAD section. Currently only has mappings for head, paragraph, and note.

content = <eulxml.xmlmap.fields.NodeListField>
list of paragraphs - *p*

head = <eulxml.xmlmap.fields.NodeField>
heading - *head*

note = <eulxml.xmlmap.fields.NodeField>
`Note`

class `eulxml.xmlmap.eadmap.`**`Address`**(*node*[, *context*])
Address information.

Expected node element passed to constructor: *address*.

lines = <eulxml.xmlmap.fields.StringListField>
list of lines in an address - *line*

class `eulxml.xmlmap.eadmap.`**`PointerGroup`**(*node*[, *context*])
Group of pointer or reference elements in an index entry

Expected node element passed to constructor: *ptrgrp*.

ref = <eulxml.xmlmap.fields.NodeListField>
list of `Reference` - references

class `eulxml.xmlmap.eadmap.`**`Reference`**(*node*[, *context*])
Internal linking element that may contain text.

Expected node element passed to constructor: *ref*.

target = <eulxml.xmlmap.fields.StringField>
link target

type = <eulxml.xmlmap.fields.StringField>
link type - *xlink:type*

value = <eulxml.xmlmap.fields.NodeField>
text content of the reference

class eulxml.xmlmap.eadmap.**ProfileDescription**(*dom_node*[, *context*])
Profile Descriptor for an EAD document. Expected node element passed to constructor: 'ead/eadheader/profiledesc'.

date = <eulxml.xmlmap.fields.NodeField>
DateField - *creation/date*

language_codes = <eulxml.xmlmap.fields.StringListField>
language codes - *langusage/language/@langcode*

languages = <eulxml.xmlmap.fields.StringListField>
language information - *langusage/language*

class eulxml.xmlmap.eadmap.**DigitalArchivalObject**(*dom_node*[, *context*])
Digital Archival Object (*dao* element)

audience = <eulxml.xmlmap.fields.StringField>
audience (internal or external)

href = <eulxml.xmlmap.fields.StringField>
url where the digital archival object can be accessed

id = <eulxml.xmlmap.fields.StringField>
identifier

show = <eulxml.xmlmap.fields.StringField>
attribute to determine how the resource should be displayed

title = <eulxml.xmlmap.fields.StringField>
title

eulxml.xmlmap.dc - Dublin Core

General Information

Thorough documentation of Dublin Core and all the elements included in simple, unqaulified DC is available from the Dublin Core Metadata Initiative. In particular, see Dublin Core Metadata Element Set, Version 1.1.

Dublin Core

All elements in Dublin Core are optional and can be repeated, so each field has been mapped as a single element (eulxml.xmlmap.StringField) and as a list (eulxml.xmlmap.StringListField), named according to the DC element.

Because the DC elements are thoroughly and clearly documented at http://dublincore.org, element descriptions have not been repeated here.

class eulxml.xmlmap.dc.**DublinCore**([*node*[, *context*]])
XmlObject for Simple (unqualified) Dublin Core metadata.

If no node is specified when initialized, a new, empty Dublin Core XmlObject will be created.

DCMI_TYPES_RDF = 'http://dublincore.org/2010/10/11/dctype.rdf'

DCMI_TYPE_URI = rdflib.term.URIRef(u'http://purl.org/dc/dcmitype/')

ROOT_NAME = 'dc'

XSD_SCHEMA = 'http://www.openarchives.org/OAI/2.0/oai_dc.xsd'

contributor = <eulxml.xmlmap.fields.StringField>

contributor_list = <eulxml.xmlmap.fields.StringListField>

coverage = <eulxml.xmlmap.fields.StringField>

coverage_list = <eulxml.xmlmap.fields.StringListField>

creator = <eulxml.xmlmap.fields.StringField>

creator_list = <eulxml.xmlmap.fields.StringListField>

date = <eulxml.xmlmap.fields.StringField>

date_list = <eulxml.xmlmap.fields.StringListField>

dcmi_types
DCMI Type Vocabulary (recommended), as documented at http://dublincore.org/documents/dcmi-type-vocabulary/

dcmi_types_graph
DCMI Types Vocabulary as an rdflib.Graph

description = <eulxml.xmlmap.fields.StringField>

description_list = <eulxml.xmlmap.fields.StringListField>

elements = <eulxml.xmlmap.fields.NodeListField>
list of all DC elements as instances of DublinCoreElement

format = <eulxml.xmlmap.fields.StringField>

format_list = <eulxml.xmlmap.fields.StringListField>

identifier = <eulxml.xmlmap.fields.StringField>

identifier_list = <eulxml.xmlmap.fields.StringListField>

language = <eulxml.xmlmap.fields.StringField>

language_list = <eulxml.xmlmap.fields.StringListField>

publisher = <eulxml.xmlmap.fields.StringField>

publisher_list = <eulxml.xmlmap.fields.StringListField>

relation = <eulxml.xmlmap.fields.StringField>

relation_list = <eulxml.xmlmap.fields.StringListField>

rights = <eulxml.xmlmap.fields.StringField>

rights_list = <eulxml.xmlmap.fields.StringListField>

source = <eulxml.xmlmap.fields.StringField>

source_list = <eulxml.xmlmap.fields.StringListField>

subject = <eulxml.xmlmap.fields.StringField>

subject_list = <eulxml.xmlmap.fields.StringListField>

title = <eulxml.xmlmap.fields.StringField>

title_list = <eulxml.xmlmap.fields.StringListField>

type = <eulxml.xmlmap.fields.StringField>

type_list = <eulxml.xmlmap.fields.StringListField>

eulxml.xmlmap.cerp - Collaborative Electronic Records Project

General Information

The Collaborative Celectronic Records Project, or CERP is a digital preservation project from the Smithsonian Institution Archives and the Rockefeller Archive Center. One particular product of that project was an XML format for email accounts. This module maps those XML objects to Python objects.

The schema produced by the project will validate only *Account* objects, though this module also allows the creation of subelements.

Account and Associated Objects

class eulxml.xmlmap.cerp.**Account**(*node=None*, *context=None*, ***kwargs*)

A single email account associated with a single email address and composed of multiple *Folder* objects and additional metadata.

http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#element_Account

ROOT_NAME = 'Account'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = 'http://www.archives.ncdcr.gov/mail-account.xsd'

email_address = <eulxml.xmlmap.fields.StringField>

folders = <eulxml.xmlmap.fields.NodeListField>

global_id = <eulxml.xmlmap.fields.StringField>

is_empty()

Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()

Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

references_accounts = <eulxml.xmlmap.fields.NodeListField>

schema_valid()

Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of XmlObject.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of lxml.etree._LogEntry instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of lxml.etree.XMLSchema; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by load_xslt() or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by load_xslt() (optional)

- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**ReferencesAccount**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_ref-account-type

REF_TYPE_CHOICES = ['PreviousContent', 'SubsequentContent', 'Supplemental', 'SeeAlso', 'SeeInstead']

ROOT_NAME = 'ReferencesAccount'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

email_address = <eulxml.xmlmap.fields.StringField>

href = <eulxml.xmlmap.fields.StringField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

reference_type = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

`Folder` and Associated Objects

class `eulxml.xmlmap.cerp.`**Folder**(*node=None*, *context=None*, ***kwargs*)
A single email folder in an *Account*, composed of multiple *Message* objects and associated metadata.

http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_folder-type

ROOT_NAME = 'Folder'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

mboxes = <eulxml.xmlmap.fields.NodeListField>

messages = <eulxml.xmlmap.fields.NodeListField>

name = <eulxml.xmlmap.fields.StringField>

node = None

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

subfolders = <eulxml.xmlmap.fields.NodeListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
        XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
        xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**Mbox**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_mbox-type

EOL_CHOICES = ['CR', 'LF', 'CRLF']

ROOT_NAME = 'Mbox'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

create_hash(*xmlobject*)

eol = <eulxml.xmlmap.fields.StringField>

hash = <eulxml.xmlmap.fields.NodeField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

rel_path = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of lxml.etree._LogEntry instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of lxml.etree.XMLSchema; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by load_xslt() or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by load_xslt() (optional)

- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

`Message` and Associated Objects

class `eulxml.xmlmap.cerp.`**Message**(*node=None*, *context=None*, ***kwargs*)

A single email message in a *`Folder`*.

http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_message-type

EOL_CHOICES = ['CR', 'LF', 'CRLF']

ROOT_NAME = 'Message'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

STATUS_FLAG_CHOICES = ['Seen', 'Answered', 'Flagged', 'Deleted', 'Draft', 'Recent']

XSD_SCHEMA = None

bcc_list = <eulxml.xmlmap.fields.StringListField>

body

cc_list = <eulxml.xmlmap.fields.StringListField>

comments_list = <eulxml.xmlmap.fields.StringListField>

create_hash(*xmlobject*)

create_incomplete_list(*xmlobject*)

create_multi_body(*xmlobject*)

create_single_body(*xmlobject*)

eol = <eulxml.xmlmap.fields.StringField>

classmethod **from_email_message**(*message*, *local_id=None*)

Convert an `email.message.Message` or compatible message object into a CERP XML *`eulxml.xmlmap.cerp.Message`*. If an id is specified, it will be stored in the Message <LocalId>.

Parameters

- **message** – *email.message.Message* object
- **id** – optional message id to be set as *local_id*

Returns *`eulxml.xmlmap.cerp.Message`* instance populated with message information

from_list = <eulxml.xmlmap.fields.StringListField>

hash = <eulxml.xmlmap.fields.NodeField>

headers = <eulxml.xmlmap.fields.NodeListField>

in_reply_to_list = <eulxml.xmlmap.fields.StringListField>

incomplete_list = <eulxml.xmlmap.fields.NodeField>

is_empty()

Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

keywords_list = <eulxml.xmlmap.fields.StringListField>

local_id = <eulxml.xmlmap.fields.IntegerField>

message_id = <eulxml.xmlmap.fields.StringField>

message_id_supplied = <eulxml.xmlmap.fields.SimpleBooleanField>

mime_version = <eulxml.xmlmap.fields.StringField>

multi_body = <eulxml.xmlmap.fields.NodeField>

node = None

orig_date_list = <eulxml.xmlmap.fields.StringListField>

references_list = <eulxml.xmlmap.fields.StringListField>

rel_path = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

sender_list = <eulxml.xmlmap.fields.StringListField>

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

single_body = <eulxml.xmlmap.fields.NodeField>

status_flags = <eulxml.xmlmap.fields.StringListField>

subject_list = <eulxml.xmlmap.fields.StringListField>

to_list = <eulxml.xmlmap.fields.StringListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**ChildMessage**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_child-message-type

ROOT_NAME = 'ChildMessage'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

bcc_list = <eulxml.xmlmap.fields.StringListField>

body

cc_list = <eulxml.xmlmap.fields.StringListField>

comments_list = <eulxml.xmlmap.fields.StringListField>

create_incomplete_list(*xmlobject*)

create_multi_body(*xmlobject*)

create_single_body(*xmlobject*)

from_list = <eulxml.xmlmap.fields.StringListField>

headers = <eulxml.xmlmap.fields.NodeListField>

in_reply_to_list = <eulxml.xmlmap.fields.StringListField>

incomplete_list = <eulxml.xmlmap.fields.NodeField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

keywords_list = <eulxml.xmlmap.fields.StringListField>

local_id = <eulxml.xmlmap.fields.IntegerField>

message_id = <eulxml.xmlmap.fields.StringField>

message_id_supplied = <eulxml.xmlmap.fields.SimpleBooleanField>

mime_version = <eulxml.xmlmap.fields.StringField>

multi_body = <eulxml.xmlmap.fields.NodeField>

node = None

orig_date_list = <eulxml.xmlmap.fields.StringListField>

references_list = <eulxml.xmlmap.fields.StringListField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

sender_list = <eulxml.xmlmap.fields.StringListField>

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

single_body = <eulxml.xmlmap.fields.NodeField>

subject_list = <eulxml.xmlmap.fields.StringListField>

to_list = <eulxml.xmlmap.fields.StringListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth‘:load_xslt‘ to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**SingleBody**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_single-body-type

ROOT_NAME = ‘SingleBody’

ROOT_NAMESPACES = {‘xm’: ‘http://www.archives.ncdcr.gov/mail-account’}

ROOT_NS = ‘http://www.archives.ncdcr.gov/mail-account’

XSD_SCHEMA = None

body_content = <eulxml.xmlmap.fields.NodeField>

charset_list = <eulxml.xmlmap.fields.StringListField>

child_message = <eulxml.xmlmap.fields.NodeField>

content

content_id_comments_list = <eulxml.xmlmap.fields.StringListField>

content_id_list = <eulxml.xmlmap.fields.StringListField>

content_name_list = <eulxml.xmlmap.fields.StringListField>

content_type_comments_list = <eulxml.xmlmap.fields.StringListField>

content_type_list = <eulxml.xmlmap.fields.StringListField>

content_type_param_list = <eulxml.xmlmap.fields.NodeListField>

create_body_content(*xmlobject*)

create_child_message(*xmlobject*)

create_ext_body_content(*xmlobject*)

description_comments_list = <eulxml.xmlmap.fields.StringListField>

description_list = <eulxml.xmlmap.fields.StringListField>

disposition_comments_list = <eulxml.xmlmap.fields.StringListField>

disposition_file_name_list = <eulxml.xmlmap.fields.StringListField>

disposition_list = <eulxml.xmlmap.fields.StringListField>

disposition_params = <eulxml.xmlmap.fields.NodeListField>

ext_body_content = <eulxml.xmlmap.fields.NodeField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

other_mime_headers = <eulxml.xmlmap.fields.NodeListField>

phantom_body = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

transfer_encoding_comments_list = <eulxml.xmlmap.fields.StringListField>

transfer_encoding_list = <eulxml.xmlmap.fields.StringListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth‘:load_xslt‘ to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class eulxml.xmlmap.cerp.**MultiBody**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_multi-body-type

ROOT_NAME = 'MultiBody'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

body

create_multi_body(*xmlobject*)

create_single_body(*xmlobject*)

epilogue = <eulxml.xmlmap.fields.StringField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

multi_body = <eulxml.xmlmap.fields.NodeField>

node = None

preamble = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

single_body = <eulxml.xmlmap.fields.NodeField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth‘:load_xslt‘ to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**Incomplete**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_incomplete-parse-type

ROOT_NAME = ‘Incomplete’

ROOT_NAMESPACES = {‘xm’: ‘http://www.archives.ncdcr.gov/mail-account’}

ROOT_NS = ‘http://www.archives.ncdcr.gov/mail-account’

XSD_SCHEMA = None

error_location = <eulxml.xmlmap.fields.StringField>

error_type = <eulxml.xmlmap.fields.StringField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**BodyContent**(*node=None*, *context=None*, ***kwargs*)

http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_int-body-content-type

ROOT_NAME = 'BodyContent'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

charset_list = <eulxml.xmlmap.fields.StringListField>

content = <eulxml.xmlmap.fields.StringField>

is_empty()

Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()

Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

schema_valid()

Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()

Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

transfer_encoding_list = <eulxml.xmlmap.fields.StringListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**ExtBodyContent**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_ext-body-content-type

EOL_CHOICES = ['CR', 'LF', 'CRLF']

ROOT_NAME = 'ExtBodyContent'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

charset_list = <eulxml.xmlmap.fields.StringListField>

create_hash(*xmlobject*)

eol = <eulxml.xmlmap.fields.StringField>

hash = <eulxml.xmlmap.fields.NodeField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

local_id = <eulxml.xmlmap.fields.IntegerField>

node = None

rel_path = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

transfer_encoding_list = <eulxml.xmlmap.fields.StringListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xml_wrapped = <eulxml.xmlmap.fields.SimpleBooleanField>

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

Additional Utility Objects

class `eulxml.xmlmap.cerp.`**Parameter**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_parameter-type

ROOT_NAME = 'Parameter'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

name = <eulxml.xmlmap.fields.StringField>

node = None

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

value = <eulxml.xmlmap.fields.StringField>

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**Header**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_header-type

ROOT_NAME = 'Header'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

comments = <eulxml.xmlmap.fields.StringListField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

name = <eulxml.xmlmap.fields.StringField>

node = None

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

value = <eulxml.xmlmap.fields.StringField>

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.cerp.`**Hash**(*node=None*, *context=None*, ***kwargs*)
http://www.records.ncdcr.gov/emailpreservation/mail-account/mail-account_docs.html#type_hash-type

HASH_FUNCTION_CHOICES = ['MD5', 'WHIRLPOOL', 'SHA1', 'SHA224', 'SHA256', 'SHA384', 'SHA512', 'RIPEMD

ROOT_NAME = 'Hash'

ROOT_NAMESPACES = {'xm': 'http://www.archives.ncdcr.gov/mail-account'}

ROOT_NS = 'http://www.archives.ncdcr.gov/mail-account'

XSD_SCHEMA = None

function = <eulxml.xmlmap.fields.StringField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

value = <eulxml.xmlmap.fields.StringField>

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

eulxml.xmlmap.mods - Metadata Object Description Schema (MODS)

General Information

The Metadata Object Description Standard, or MODS is a schema developed and maintained by the Library of Congress for bibliographic records.

This module defines classes to handle common use cases for MODS metadata, rooted in the *MODSv34* object. It is not a complete mapping, though it will likely grow closer to one as development progresses according to user needs.

Root Classes: MODS and Friends

class eulxml.xmlmap.mods.**MODS**(*node=None*, *context=None*, ***kwargs*)
Top-level *XmlObject* for a MODS metadata record. Inherits all standard top-level MODS fields from `BaseMods` and adds a mapping for *RelatedItem*.

ROOT_NAME = 'mods'

ROOT_NAMESPACES = {'mods': 'http://www.loc.gov/mods/v3'}

ROOT_NS = 'http://www.loc.gov/mods/v3'

XSD_SCHEMA = 'http://www.loc.gov/standards/mods/v3/mods-3-4.xsd'

abstract = <eulxml.xmlmap.fields.NodeField>

access_conditions = <eulxml.xmlmap.fields.NodeListField>

create_abstract(*xmlobject*)

create_name(*xmlobject*)

create_note(*xmlobject*)

create_origin_info(*xmlobject*)

create_physical_description(*xmlobject*)

create_record_info(*xmlobject*)

create_title_info(*xmlobject*)

genres = <eulxml.xmlmap.fields.NodeListField>

id = <eulxml.xmlmap.fields.StringField>

identifiers = <eulxml.xmlmap.fields.NodeListField>

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

languages = <eulxml.xmlmap.fields.NodeListField>

location = <eulxml.xmlmap.fields.StringField>

locations = <eulxml.xmlmap.fields.NodeListField>

name = <eulxml.xmlmap.fields.NodeField>

names = <eulxml.xmlmap.fields.NodeListField>

node = None

note = <eulxml.xmlmap.fields.NodeField>

notes = <eulxml.xmlmap.fields.NodeListField>

origin_info = <eulxml.xmlmap.fields.NodeField>

parts = <eulxml.xmlmap.fields.NodeListField>

physical_description = <eulxml.xmlmap.fields.NodeField>

record_info = <eulxml.xmlmap.fields.NodeField>

related_items = <eulxml.xmlmap.fields.NodeListField>

resource_type = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of lxml.etree._LogEntry instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of cStringIO.StringIO

subjects = <eulxml.xmlmap.fields.NodeListField>

title = <eulxml.xmlmap.fields.StringField>

title_info = <eulxml.xmlmap.fields.NodeField>

title_info_list = <eulxml.xmlmap.fields.NodeListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of lxml.etree.XMLSchema; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by load_xslt() or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)

- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

class `eulxml.xmlmap.mods.`**MODSv34**(*node=None*, *context=None*, ***kwargs*)

XmlObject for MODS version 3.4. Currently consists of all the same fields as *MODS*, but loads the MODS version 3.4 schema for validation.

ROOT_NAME = 'mods'

ROOT_NAMESPACES = {'mods': 'http://www.loc.gov/mods/v3'}

ROOT_NS = 'http://www.loc.gov/mods/v3'

XSD_SCHEMA = 'http://www.loc.gov/standards/mods/v3/mods-3-4.xsd'

abstract = <eulxml.xmlmap.fields.NodeField>

access_conditions = <eulxml.xmlmap.fields.NodeListField>

create_abstract(*xmlobject*)

create_name(*xmlobject*)

create_note(*xmlobject*)

create_origin_info(*xmlobject*)

create_physical_description(*xmlobject*)

create_record_info(*xmlobject*)

create_title_info(*xmlobject*)

genres = <eulxml.xmlmap.fields.NodeListField>

id = <eulxml.xmlmap.fields.StringField>

identifiers = <eulxml.xmlmap.fields.NodeListField>

is_empty()

Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()

Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

languages = <eulxml.xmlmap.fields.NodeListField>

location = <eulxml.xmlmap.fields.StringField>

locations = <eulxml.xmlmap.fields.NodeListField>

name = <eulxml.xmlmap.fields.NodeField>

names = <eulxml.xmlmap.fields.NodeListField>

node = None

note = <eulxml.xmlmap.fields.NodeField>

notes = <eulxml.xmlmap.fields.NodeListField>

origin_info = <eulxml.xmlmap.fields.NodeField>

parts = <eulxml.xmlmap.fields.NodeListField>

physical_description = <eulxml.xmlmap.fields.NodeField>

record_info = <eulxml.xmlmap.fields.NodeField>

related_items = <eulxml.xmlmap.fields.NodeListField>

resource_type = <eulxml.xmlmap.fields.StringField>

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of `XmlObject`.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

subjects = <eulxml.xmlmap.fields.NodeListField>

title = <eulxml.xmlmap.fields.StringField>

title_info = <eulxml.xmlmap.fields.NodeField>

title_info_list = <eulxml.xmlmap.fields.NodeListField>

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
        XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
        xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to `XmlObject`; specify unicode or string for text output

Returns an instance of `XmlObject` or the return_type specified

Title Info

MODS titleInfo

class `eulxml.xmlmap.mods.`**TitleInfo**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'titleInfo'

is_empty()
Returns True if all titleInfo subfields are not set or empty; returns False if any of the fields are not empty.

label = <eulxml.xmlmap.fields.StringField>

non_sort = <eulxml.xmlmap.fields.StringField>

part_name = <eulxml.xmlmap.fields.StringField>

part_number = <eulxml.xmlmap.fields.StringField>

subtitle = <eulxml.xmlmap.fields.StringField>

title = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Name

MODS name

class `eulxml.xmlmap.mods.`**Name**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS name

ROOT_NAME = 'name'

affiliation = <eulxml.xmlmap.fields.StringField>

authority = <eulxml.xmlmap.fields.StringField>

display_form = <eulxml.xmlmap.fields.StringField>

id = <eulxml.xmlmap.fields.StringField>

name_parts = <eulxml.xmlmap.fields.NodeListField>

roles = <eulxml.xmlmap.fields.NodeListField>

type = <eulxml.xmlmap.fields.StringField>

class eulxml.xmlmap.mods.**NamePart**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS namePart

ROOT_NAME = 'namePart'

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

class eulxml.xmlmap.mods.**Role**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS role

ROOT_NAME = 'role'

authority = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Genre

MODS genre

class eulxml.xmlmap.mods.**Genre**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'genre'

authority = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

Origin Info

MODS originInfo

class eulxml.xmlmap.mods.**OriginInfo**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS originInfo element (incomplete)

ROOT_NAME = 'originInfo'

captured = <eulxml.xmlmap.fields.NodeListField>

copyright = <eulxml.xmlmap.fields.NodeListField>

created = <eulxml.xmlmap.fields.NodeListField>

is_empty()
Returns True if all child date elements present are empty and other nodes are not set. Returns False if any child date elements are not empty or other nodes are set.

issued = <eulxml.xmlmap.fields.NodeListField>

modified = <eulxml.xmlmap.fields.NodeListField>

other = <eulxml.xmlmap.fields.NodeListField>

publisher = <eulxml.xmlmap.fields.StringField>

valid = <eulxml.xmlmap.fields.NodeListField>

class eulxml.xmlmap.mods.**DateCreated**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'dateCreated'

class eulxml.xmlmap.mods.**DateIssued**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'dateIssued'

Language

MODS language

class eulxml.xmlmap.mods.**Language**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'language'

terms = <eulxml.xmlmap.fields.NodeListField>

class eulxml.xmlmap.mods.**LanguageTerm**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'languageTerm'

authority = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Physical Description

MODS physicalDescription

class eulxml.xmlmap.mods.**PhysicalDescription**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'physicalDescription'

extent = <eulxml.xmlmap.fields.StringField>

media_type = <eulxml.xmlmap.fields.StringField>

Abstract

MODS abstract

class eulxml.xmlmap.mods.**Abstract**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'abstract'

label = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Note

MODS note

class eulxml.xmlmap.mods.**Note**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS note element

ROOT_NAME = 'note'

label = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

class eulxml.xmlmap.mods.**TypedNote**(*node=None*, *context=None*, ***kwargs*)
Extends *Note* to modify `is_empty()` behavior– considered empty when a type attribute is set without any text.

Subject

MODS subject

class eulxml.xmlmap.mods.**Subject**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'subject'

authority = <eulxml.xmlmap.fields.StringField>

create_name(*xmlobject*)

geographic = <eulxml.xmlmap.fields.StringField>

id = <eulxml.xmlmap.fields.StringField>

name = <eulxml.xmlmap.fields.NodeField>

title = <eulxml.xmlmap.fields.StringField>

topic = <eulxml.xmlmap.fields.StringField>

Related Item

MODS relatedItem

class eulxml.xmlmap.mods.**RelatedItem**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS relatedItem: contains all the top-level MODS fields defined by `BaseMods`, plus a type attribute.

ROOT_NAME = 'relatedItem'

label = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Identifer

MODS identifier

class eulxml.xmlmap.mods.**Identifier**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS identifier

ROOT_NAME = 'identifier'

label = <eulxml.xmlmap.fields.StringField>

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Location

MODS location

class eulxml.xmlmap.mods.**Location**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'location'

physical = <eulxml.xmlmap.fields.StringField>

url = <eulxml.xmlmap.fields.StringField>

Access Condition

MODS accessCondition

class eulxml.xmlmap.mods.**AccessCondition**(*node=None*, *context=None*, ***kwargs*)
XmlObject for MODS accessCondition

ROOT_NAME = 'accessCondition'

text = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

Part

MODS part

class eulxml.xmlmap.mods.**Part**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'part'

create_extent(*xmlobject*)

details = <eulxml.xmlmap.fields.NodeListField>

extent = <eulxml.xmlmap.fields.NodeField>

is_empty()
Returns True if details, extent, and type are not set or return True for `is_empty`; returns False if any of the fields are not empty.

type = <eulxml.xmlmap.fields.StringField>

class eulxml.xmlmap.mods.**PartDetail**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'detail'

is_empty()
Returns False if no number value is set; returns True if any number value is set. Type attribute is ignored for determining whether or not this node should be considered empty.

number = <eulxml.xmlmap.fields.StringField>

type = <eulxml.xmlmap.fields.StringField>

class eulxml.xmlmap.mods.**PartExtent**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'extent'

end = <eulxml.xmlmap.fields.StringField>

is_empty()
Returns False if no extent value is set; returns True if any extent value is set. Unit attribute is ignored for determining whether or not this node should be considered empty.

start = <eulxml.xmlmap.fields.StringField>

total = <eulxml.xmlmap.fields.StringField>

unit = <eulxml.xmlmap.fields.StringField>

Record Info

MODS recordInfo

class eulxml.xmlmap.mods.**RecordInfo**(*node=None*, *context=None*, ***kwargs*)

ROOT_NAME = 'recordInfo'

change_date = <eulxml.xmlmap.fields.StringField>

creation_date = <eulxml.xmlmap.fields.StringField>

record_id = <eulxml.xmlmap.fields.StringField>

record_origin = <eulxml.xmlmap.fields.StringField>

eulxml.xmlmap.premis - PREMIS

eulxml.xmlmap classes for dealing with the PREMIS metadata format for preservation metadata.

---

class eulxml.xmlmap.premis.**BasePremis**(*node=None*, *context=None*, ***kwargs*)
Base PREMIS class with namespace declaration common to all PREMIS XmlObjects.

Note: This class is intended mostly for internal use, but could be useful when extending or adding additional PREMIS *XmlObject* classes. The *PREMIS_NAMESPACE* is mapped to the prefix **p**.

class eulxml.xmlmap.premis.**Event**(*node=None*, *context=None*, ***kwargs*)
Preliminary *XmlObject* for a PREMIS event.

> **Note:** The PREMIS schema requires that elements occur in a specified order, which eulxml does not currently handle or manage. As a work-around, when creating a new *Event* from scratch, you should set the following required fields in this order: identifier (*id* and ad_type

date = <eulxml.xmlmap.fields.StringField>
date/time for the event (*eventDateTime*)

detail = <eulxml.xmlmap.fields.StringField>
event detail (*eventDetail*)

id = <eulxml.xmlmap.fields.StringField>
identifier value (*eventIdentifier/eventIdentifierValue*)

id_type = <eulxml.xmlmap.fields.StringField>
identifier type (*eventIdentifier/eventIdentifierType*)

outcome = <eulxml.xmlmap.fields.StringField>
outcome of the event (*eventOutcomeInformation/eventOutcome*).

> **Note:** In this preliminary implementation, the outcome detail fields are not mapped.

type = <eulxml.xmlmap.fields.StringField>
event type (eventType)

class eulxml.xmlmap.premis.**Object**(*node=None*, *context=None*, ***kwargs*)
Preliminary *XmlObject* for a PREMIS object.

Curently only includes the minimal required fields.

id = <eulxml.xmlmap.fields.StringField>
identifier value (*objectIdentifier/objectIdentifierValue*)

id_type = <eulxml.xmlmap.fields.StringField>
identifier type (*objectIdentifier/objectIdentifierType*)

type = <eulxml.xmlmap.fields.StringField>
type of object (e.g., file, representation, bitstream).

> **Note:** To be schema valid, object types must be in the PREMIS namespace, e.g.:

```
from eulxml.xmlmap import premis
obj = premis.Object()
obj.type = "p:file"
```

eulxml.xmlmap.premis.**PREMIS_NAMESPACE = 'info:lc/xmlns/premis-v2'**
authoritative namespace for PREMIS

eulxml.xmlmap.premis.**PREMIS_SCHEMA = 'http://www.loc.gov/standards/premis/v2/premis-v2-1.xsd'**
authoritative schema location for PREMIS

class eulxml.xmlmap.premis.**Premis**(**args*, ***kwargs*)
Preliminary *XmlObject* for a PREMIS container element that can contain any of the other top-level PREMIS elements.

Curently only includes mappings for a single object and list of events.

events = <eulxml.xmlmap.fields.NodeListField>
list of PREMIS events, as instances of *Event*

object = <eulxml.xmlmap.fields.NodeField>
a single PREMIS *object*

version = <eulxml.xmlmap.fields.StringField>
Version of PREMIS in use; by default, new instances of *Premis* will be initialized with a version of 2.1

class eulxml.xmlmap.premis.**PremisRoot**(*node=None*, *context=None*, ***kwargs*)
Base class with a schema declaration for any of the root/stand-alone PREMIS elements:

- <premis> - *Premis*
- <object> - *Object*
- <event> - *Event*
- <agent>
- <rights>

1.2.2 General Usage

Suppose we have an XML object that looks something like this:

```
<foo>
  <bar>
    <baz>42</baz>
  </bar>
  <bar>
    <baz>13</baz>
  </bar>
  <qux>A</qux>
  <qux>B</qux>
</foo>
```

For this example, we want to access the value of the first <baz> as a Python integer and the second <baz> as a string value. We also want to access all of them (there may be lots on another <foo>) as a big list of integers. We can create an object to map these fields like this:

```
from eulxml import xmlmap

class Foo(xmlmap.XmlObject):
    first_baz = xmlmap.IntegerField('bar[1]/baz')
    second_baz = xmlmap.StringField('bar[2]/baz')
    qux = xmlmap.StringListField('qux')
```

first_baz, second_baz, and all_baz here are attributes of the Foo object. We can access them in later code like this:

```
>>> foo = xmlmap.load_xmlobject_from_file(foo_path, xmlclass=Foo)
>>> foo.first_baz
42
>>> foo.second_baz
'13'
>>> foo.qux
['A', 'B']
>>> foo.first_baz=5
```

```
>>> foo.qux.append('C')
>>> foo.qux[0] = 'Q'
>>> print foo.serialize(pretty=True)
<foo>
  <bar>
    <baz>5</baz>
  </bar>
  <bar>
    <baz>13</baz>
  </bar>
  <qux>Q</qux>
  <qux>B</qux>
<qux>C</qux></foo>
```

1.2.3 Concepts

xmlmap simplifies access to XML data in Python. Programs can define new *XmlObject* subclasses representing a type of XML node with predictable structure. Members of these classes can be regular methods and values like in regular Python classes, but they can also be special *field* objects that associate XPath expressions with Python data elements. When code accesses these fields on the object, the code evaluates the associated XPath expression and converts the data to a Python value.

1.2.4 XmlObject

Most programs will use *xmlmap* by defining a subclass of *XmlObject* containing *field* members.

class eulxml.xmlmap.**XmlObject**([*node*[, *context*]])

A Python object wrapped around an XML node.

Typical programs will define subclasses of *XmlObject* with various field members. Some programs will use *load_xmlobject_from_string()* and *load_xmlobject_from_file()* to create instances of these subclasses. Other programs will create them directly, passing a node argument to the constructor. If the subclass defines a *ROOT_NAME* then this node argument is optional: Programs may then create instances directly with no constructor arguments.

Programs can also pass an optional dictionary to the constructor to specify namespaces for XPath evaluation.

If keyword arguments are passed in to the constructor, they will be used to set initial values for the corresponding fields on the *XmlObject*. (Only currently supported for non-list fields.)

Custom equality/non-equality tests: two instances of *XmlObject* are considered equal if they point to the same lxml element node.

_fields

A dictionary mapping field names to *field* members. This dictionary includes all of the fields defined on the class as well as those inherited from its parents.

ROOT_NAME = None

A default root element name (without namespace prefix) used when an object of this type is created from scratch.

ROOT_NAMESPACES = {}

A dictionary whose keys are namespace prefixes and whose values are namespace URIs. These namespaces are used to create the root element when an object of this type is created from scratch; should include the namespace and prefix for the root element, if it has one. Any additional namespaces will be added to the root element.

ROOT_NS = None
The default namespace used when an object of this type is created from scratch.

XSD_SCHEMA = None
URI or file path to the XSD schema associated with this *XmlObject*, if any. If configured, will be used for optional validation when calling *load_xmlobject_from_string()* and *load_xmlobject_from_file()*, and with *is_valid()*.

is_empty()
Returns True if the root node contains no child elements, no attributes, and no text. Returns False if any are present.

is_valid()
Determine if the current document is valid as far as we can determine. If there is a schema associated, check for schema validity. Otherwise, return True.

Return type boolean

node = None
The top-level xml node wrapped by the object

schema_valid()
Determine if the current document is schema-valid according to the configured XSD Schema associated with this instance of *XmlObject*.

Return type boolean

Raises Exception if no XSD schema is defined for this XmlObject instance

schema_validate = True
Override for schema validation; if a schema must be defined for the use of `xmlmap.fields.SchemaField` for a sub-xmlobject that should not be validated, set to False.

schema_validation_errors()
Retrieve any validation errors that occured during schema validation done via *is_valid()*.

Returns a list of `lxml.etree._LogEntry` instances

Raises Exception if no XSD schema is defined for this XmlObject instance

serialize(*stream=None*, *pretty=False*)
Serialize the contents of the XmlObject to a stream. Serializes current node only; for the entire XML document, use *serializeDocument()*.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

serializeDocument(*stream=None*, *pretty=False*)
Serialize the contents of the entire XML document (including Doctype declaration, if there is one), with an XML declaration, for the current XmlObject to a stream.

If no stream is specified, returns a string. :param stream: stream or other file-like object to write content to (optional) :param pretty: pretty-print the XML output; boolean, defaults to False :rtype: stream passed in or an instance of `cStringIO.StringIO`

validation_errors()
Return a list of validation errors. Returns an empty list if the xml is schema valid or no schema is defined. If a schema is defined but *schema_validate* is False, schema validation will be skipped.

Currently only supports schema validation.

Return type list

xmlschema
A parsed XSD schema instance of `lxml.etree.XMLSchema`; will be loaded the first time it is requested on any instance of this class if XSD_SCHEMA is set and xmlchema is None. If you wish to load and parse the schema at class definition time, instead of at class instance initialization time, you may want to define your schema in your subclass like this:

```
XSD_SCHEMA = "http://www.openarchives.org/OAI/2.0/oai_dc.xsd"
xmlschema = xmlmap.loadSchema(XSD_SCHEMA)
```

xsl_transform(*filename=None*, *xsl=None*, *return_type=None*, ***params*)
Run an xslt transform on the contents of the XmlObject.

XSLT can be passed in as an XSLT object generated by `load_xslt()` or as filename or string. If a params dictionary is specified, its items will be passed as parameters to the XSL transformation, and any string values will automatically be encoded as XSL string parameters.

Note: If XSL is being used multiple times, it is recommended to use :meth':load_xslt' to load and compile the XSLT once.

Parameters

- **filename** – xslt filename (optional, one of file and xsl is required)
- **xsl** – xslt as string OR compiled XSLT object as returned by `load_xslt()` (optional)
- **return_type** – type of object to return; optional, defaults to *XmlObject*; specify unicode or string for text output

Returns an instance of *XmlObject* or the return_type specified

1.2.5 XmlObjectType

class `eulxml.xmlmap.core.`**XmlObjectType**
A metaclass for *XmlObject*.

Analogous in principle to Django's `ModelBase`, this metaclass functions rather differently. While it'll likely get a lot closer over time, we just haven't been growing ours long enough to demand all of the abstractions built into Django's models. For now, we do three things:

1. take any `Field` members and convert them to descriptors,
2. store all of these fields and all of the base classes' fields in a `_fields` dictionary on the class, and
3. if any local (non-parent) fields look like self-referential `eulxml.xmlmap.NodeField` objects then patch them up to refer to the newly-created *XmlObject*.

1.2.6 Field types

There are several predefined field types. All of them evaluate XPath expressions and map the resultant XML nodes to Python types. They differ primarily in how they map those XML nodes to Python objects as well as in whether they expect their XPath expression to match a single XML node or a whole collection of them.

Field objects are typically created as part of an *XmlObject* definition and accessed with standard Python object attribute syntax. If a `Foo` class defines a `bar` attribute as an *xmlmap* field object, then an object will reference it simply as `foo.bar`.

class eulxml.xmlmap.fields.**StringField**(*xpath*, *normalize=False*, *choices=None*, **args*, ***kwargs*)

Map an XPath expression to a single Python string. If the XPath expression evaluates to an empty NodeList, a StringField evaluates to *None*.

Takes an optional parameter to indicate that the string contents should have whitespace normalized. By default, does not normalize.

Takes an optional list of choices to restrict possible values.

Supports setting values for attributes, empty nodes, or text-only nodes.

class eulxml.xmlmap.fields.**StringListField**(*xpath*, *normalize=False*, *choices=None*, **args*, ***kwargs*)

Map an XPath expression to a list of Python strings. If the XPath expression evaluates to an empty NodeList, a StringListField evaluates to an empty list.

Takes an optional parameter to indicate that the string contents should have whitespace normalized. By default, does not normalize.

Takes an optional list of choices to restrict possible values.

Actual return type is `NodeList`, which can be treated like a regular Python list, and includes set and delete functionality.

class eulxml.xmlmap.fields.**IntegerField**(*xpath*, **args*, ***kwargs*)

Map an XPath expression to a single Python integer. If the XPath expression evaluates to an empty NodeList, an IntegerField evaluates to *None*.

Supports setting values for attributes, empty nodes, or text-only nodes.

class eulxml.xmlmap.fields.**IntegerListField**(*xpath*, **args*, ***kwargs*)

Map an XPath expression to a list of Python integers. If the XPath expression evaluates to an empty NodeList, an IntegerListField evaluates to an empty list.

Actual return type is `NodeList`, which can be treated like a regular Python list, and includes set and delete functionality.

class eulxml.xmlmap.fields.**NodeField**(*xpath*, *node_class*, *instantiate_on_get=False*, **args*, ***kwargs*)

Map an XPath expression to a single *XmlObject* subclass instance. If the XPath expression evaluates to an empty NodeList, a NodeField evaluates to *None*.

Normally a `NodeField`'s `node_class` is a class. As a special exception, it may be the string `"self"`, in which case it recursively refers to objects of its containing *XmlObject* class.

If an *XmlObject* contains a NodeField named `foo`, then the object will automatically have a `create_foo()` method in addition to its `foo` property. Code can call this `create_foo()` method to create the child element if it doesn't exist; the method will have no effect if the element is already present.

Deprecated `instantiate_on_get` flag: set to True if you need a non-existent node to be created when the NodeField is accessed. This feature is deprecated: Instead, create your node explicitly with `create_foo()` as described above.

class eulxml.xmlmap.fields.**NodeListField**(*xpath*, *node_class*, **args*, ***kwargs*)

Map an XPath expression to a list of *XmlObject* subclass instances. If the XPath expression evalues to an empty NodeList, a NodeListField evaluates to an empty list.

Normally a `NodeListField`'s `node_class` is a class. As a special exception, it may be the string `"self"`, in which case it recursively refers to objects of its containing *XmlObject* class.

Actual return type is `NodeList`, which can be treated like a regular Python list, and includes set and delete functionality.

class eulxml.xmlmap.fields.**ItemField**(*xpath*, **args*, ***kwargs*)
Access the results of an XPath expression directly. An ItemField does no conversion on the result of evaluating the XPath expression.

class eulxml.xmlmap.fields.**SimpleBooleanField**(*xpath*, *true*, *false*, **args*, ***kwargs*)
Map an XPath expression to a Python boolean. Constructor takes additional parameter of true, false values for comparison and setting in xml. This only handles simple boolean that can be read and set via string comparison.

Supports setting values for attributes, empty nodes, or text-only nodes.

class eulxml.xmlmap.fields.**DateTimeField**(*xpath*, *format=None*, *normalize=False*, **args*, ***kwargs*)
Map an XPath expression to a single Python datetime.datetime. If the XPath expression evaluates to an empty NodeList, a *DateTimeField* evaluates to *None*.

Parameters

- **format** – optional date-time format. Used with datetime.datetime.strptime() and datetime.datetime.strftime() to convert between XML text and Python datetime.datetime objects. If no format is specified, XML dates are converted from full ISO date time format, with or without microseconds, and dates are written out to XML in ISO format via datetime.datetime.isoformat().
- **normalize** – optional parameter to indicate string contents should have whitespace normalized before converting to datetime. By default, no normalization is done.

For example, given the field definition:

```
last_update = DateTimeField('last_update', format="%d-%m-%Y %H:%M:%S",
    normalize=True)
```

and the XML:

```
<last_update>
    21-04-2012 00:00:00
</last_update>
```

accessing the field would return:

```
>>> myobj.last_update
datetime.datetime(2012, 4, 21, 0, 0)
```

class eulxml.xmlmap.fields.**DateTimeListField**(*xpath*, *format=None*, *normalize=False*, **args*, ***kwargs*)
Map an XPath expression to a list of Python datetime.datetime objects. If the XPath expression evaluates to an empty NodeList, a *DateTimeListField* evaluates to an empty list. Date formatting is as described in *DateTimeField*.

Actual return type is NodeList, which can be treated like a regular Python list, and includes set and delete functionality.

Parameters

- **format** – optional date-time format. See *DateTimeField* for more details.
- **normalize** – optional parameter to indicate string contents should have whitespace normalized before converting to datetime. By default, no normalization is done.

class eulxml.xmlmap.fields.**DateField**(*xpath*, *format=None*, *normalize=False*, **args*, ***kwargs*)
Map an XPath expression to a single Python datetime.date, roughly comparable to *DateTimeField*.

Parameters

- **format** – optional date-time format. Used to convert between XML and Python `datetime.date`; if no format, then the ISO format YYYY-MM-DD (%Y-%m-%d) will be used.
- **normalize** – optional parameter to indicate string contents should have whitespace normalized before converting to `date`. By default, no normalization is done.

class `eulxml.xmlmap.fields.`**DateListField**(*xpath*, *format=None*, *normalize=False*, **args*, ***kwargs*)

Map an XPath expression to a list of Python `datetime.date` objects. See *DateField* and *DateTimeListField* for more details.

class `eulxml.xmlmap.fields.`**SchemaField**(*xpath*, *schema_type*, **args*, ***kwargs*)

Schema-based field. At class definition time, a SchemaField will be **replaced** with the appropriate `eulxml.xmlmap.fields.Field` type based on the schema type definition.

Takes an xpath (which will be passed on to the real Field init) and a schema type definition name. If the schema type has enumerated restricted values, those will be passed as choices to the Field.

For example, to define a resource type based on the MODS schema, `resourceTypeDefinition` is a simple type with an enumeration of values, so you could add something like this:

```
resource_type  = xmlmap.SchemaField("mods:typeOfResource", "resourceTypeDefinition")
```

Currently only supports simple string-based schema types.

get_field(*schema*)

Get the requested type definition from the schema and return the appropriate `Field`.

Parameters **schema** – instance of `eulxml.xmlmap.core.XsdSchema`

Return type `eulxml.xmlmap.fields.Field`

class `eulxml.xmlmap.fields.`**FloatField**(*xpath*, **args*, ***kwargs*)

Map an XPath expression to a single Python float. If the XPath expression evaluates to an empty NodeList, an FloatField evaluates to *None*.

Supports setting values for attributes, empty nodes, or text-only nodes.

class `eulxml.xmlmap.fields.`**FloatListField**(*xpath*, **args*, ***kwargs*)

Map an XPath expression to a list of Python floats. If the XPath expression evaluates to an empty NodeList, an IntegerListField evaluates to an empty list.

Actual return type is `NodeList`, which can be treated like a regular Python list, and includes set and delete functionality.

1.2.7 Other facilities

`eulxml.xmlmap.`**load_xmlobject_from_string**(*string*, *xmlclass=<class 'eulxml.xmlmap.core.XmlObject'>*, *validate=False*, *resolver=None*)

Initialize an XmlObject from a string.

If an xmlclass is specified, construct an instance of that class instead of *XmlObject*. It should be a subclass of XmlObject. The constructor will be passed a single node.

If validation is requested and the specified subclass of `XmlObject` has an XSD_SCHEMA defined, the parser will be configured to validate against the specified schema. Otherwise, the parser will be configured to use DTD validation, and expect a Doctype declaration in the xml content.

Parameters

- **string** – xml content to be loaded, as a string
- **xmlclass** – subclass of *XmlObject* to initialize
- **validate** – boolean, enable validation; defaults to false

Return type instance of *XmlObject* requested

eulxml.xmlmap.**load_xmlobject_from_file**(*filename*, *xmlclass=<class 'eulxml.xmlmap.core.XmlObject'>*, *validate=False*, *resolver=None*)

Initialize an XmlObject from a file.

See `load_xmlobject_from_string()` for more details; behaves exactly the same, and accepts the same parameters, except that it takes a filename instead of a string.

Parameters **filename** – name of the file that should be loaded as an xmlobject. `etree.lxml.parse()` will accept a file name/path, a file object, a file-like object, or an HTTP or FTP url, however file path and URL are recommended, as they are generally faster for lxml to handle.

eulxml.xmlmap.**parseString**(*string*, *uri=None*)

Read an XML document provided as a byte string, and return a `lxml.etree` document. String cannot be a Unicode string. Base_uri should be provided for the calculation of relative URIs.

eulxml.xmlmap.**parseUri**(*stream*, *uri=None*)

Read an XML document from a URI, and return a `lxml.etree` document.

eulxml.xmlmap.**loadSchema**(*uri*, *base_uri=None*)

Load an XSD XML document (specified by filename or URL), and return a `lxml.etree.XMLSchema`.

Note that frequently loading a schema without using a web proxy may introduce significant network resource usage as well as instability if the schema becomes unavailable. Thus this function will fail if the `HTTP_PROXY` environment variable is not set.

1.3 eulxml.forms - Forms for XmlObjects

1.4 eulxml.xpath – Parse and Serialize XPath

Functions and classes for parsing XPath expressions into abstract syntax trees and serializing them back to strings.

This module exports two key functions, *parse()* and *serialize()*.

eulxml.xpath.**parse**(*xpath_str*)

Parse a string XPath expression into an abstract syntax tree. The AST will be built from the classes defined in *eulxml.xpath.ast*.

eulxml.xpath.**serialize**(*xpath_ast*)

Serialize an XPath AST expressed in terms of *eulxml.xpath.ast* objects into a valid XPath string.

This module does not support evaluating XPath expressions.

1.4.1 eulxml.xpath.ast – Abstract syntax trees for XPath

Abstract Syntax Tree nodes for parsed XPath.

This module contains basic nodes for representing parsed XPath expressions. The parser provided by this module creates its parsed XPath representation from the classes defined in this module. Library callers will mostly not use this

module directly, unless they need to produce XPath ASTs from scratch or perhaps introspect ASTs returned by the parser.

eulxml.xpath.ast.**serialize**(*xp_ast*)
Serialize an XPath AST as a valid XPath expression.

class eulxml.xpath.ast.**UnaryExpression**(*op*, *right*)
A unary XPath expression. Practially, this means -foo.

op = None
the operator used in the expression

right = None
the expression the operator is applied to

class eulxml.xpath.ast.**BinaryExpression**(*left*, *op*, *right*)
Any binary XPath expression. a/b; a and b; a | b.

left = None
the left side of the binary expression

op = None
the operator of the binary expression

right = None
the right side of the binary expression

class eulxml.xpath.ast.**PredicatedExpression**(*base*, *predicates=None*)
A filtered XPath expression. $var[1]; (a or b)[foo][@bar].

base = None
the base expression to be filtered

predicates = None
a list of filter predicates

class eulxml.xpath.ast.**AbsolutePath**(*op='/'*, *relative=None*)
An absolute XPath path. /a/b/c; //a/ancestor:b/@c.

op = None
the operator used to root the expression

relative = None
the relative path after the absolute root operator

class eulxml.xpath.ast.**Step**(*axis*, *node_test*, *predicates*)
A single step in a relative path. a; @b; text(); parent::foo:bar[5].

axis = None
the step's axis, or @ or None if abbreviated or undefined

node_test = None
a NameTest or NodeType object describing the test represented

predicates = None
a list of predicates filtering the step

class eulxml.xpath.ast.**NameTest**(*prefix*, *name*)
An element name node test for a Step.

name = None
the node name used for the test, or *

prefix = None
the namespace prefix used for the test, or None if unset

class eulxml.xpath.ast.**NodeType**(*name*, *literal=None*)
A node type node test for a Step.

literal = None
the argument to the node specifier. XPath allows these only for processing-instruction() node tests.

name = None
the node type name, such as node or text

class eulxml.xpath.ast.**AbbreviatedStep**(*abbr*)
An abbreviated XPath step. . or ..

abbr = None
the abbreviated step

class eulxml.xpath.ast.**VariableReference**(*name*)
An XPath variable reference. $foo; $myns:foo.

name = None
a tuple (prefix, localname) containing the variable name

class eulxml.xpath.ast.**FunctionCall**(*prefix*, *name*, *args*)
An XPath function call. foo(); my:foo(1); foo(1, 'a', $var).

args = None
a list of argument expressions

name = None
the local function name

prefix = None
the namespace prefix, or None if unspecified

1.4.2 Notes

- The re standard library module in Python had a bug prior to 2.6.4 that made it reject patterns with Unicode characters above U+FFFF. As a result, XPath expressions including these characters in node names, namespace abbreviations, or function names will not work correctly in those versions of Python.

  If you don't know what this means, you almost certainly don't need to worry about it.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

X

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

X