libearth Documentation

Release 0.1.2

Hong Minhee

January 19, 2014

Libearth is the shared common library for various Earth Reader apps. Earth Reader try to support many platforms as possible (e.g. web, mobile apps, desktop apps), so there must be a large part of common concepts and implementations they share like subscription lists, synchronization through cloud storages between several devices, and crawler, that libearth actually implements.

CHAPTER 1

Compatibility & portability

Libearth officially supports the following Python implementations:

- Python 2.6, 2.7, 3.2, 3.3
- CPython, PyPy, IronPython

For environments `setuptools` not available, it has no required dependencies.

See also `tox.ini` file and CI builds.

CHAPTER 2

Installation

You can install it using **pip**:

```
$ pip install libearth
```

See PyPI as well.

CHAPTER 3

References

3.1 libearth — The shared common library for Earth Reader apps

3.1.1 libearth.codecs — Common codecs

This module provides commonly used codecs to parse RSS-related standard formats.

class libearth.codecs.**Boolean**(*true='true'*, *false='false'*, *default_value=None*)
Codec to interpret boolean representation in strings e.g. 'true', 'no', and encode bool values back to string.

Parameters

- **true** (str, tuple) – text to parse as True. 'true' by default
- **false** (str, tuple) – text to parse as False. 'false' by default
- **default_value** (bool) – default value when it cannot be parsed

class libearth.codecs.**Enum**(*values*)
Codec that accepts only predefined fixed types of values:

```
gender = Enum(['male', 'female'])
```

Actually it doesn't any encoding nor decoding, but it simply *validates* all values from XML and Python both.

Note that values have to consist of only strings.

Parameters **values** (collections.Iterable) – any iterable that yields all possible values

class libearth.codecs.**Integer**
Codec to encode and decode integer numbers.

class libearth.codecs.**Rfc3339**(*prefer_utc=False*)
Codec to store datetime.datetime values to **RFC 3339** format.

Parameters **prefer_utc** (bool) – normalize all timezones to UTC. False by default

PATTERN = <_sre.SRE_Pattern object at 0x1393550>
(re.RegexObject) The regular expression pattern that matches to valid **RFC 3339** date time string.

class libearth.codecs.**Rfc822**
Codec to encode/decode datetime.datetime values to/from **RFC 822** format.

3.1.2 libearth.compat — Compatibility layer

This module provides several subtle things to support multiple Python versions (2.6, 2.7, 3.2, 3.3) and VM implementations (CPython, PyPy).

libearth.compat.**IRON_PYTHON = False**
(bool) Whether it is IronPython or not.

libearth.compat.**PY3 = False**
(bool) Whether it is Python 3.x or not.

libearth.compat.**UNICODE_BY_DEFAULT = False**
(bool) Whether the Python VM uses Unicode strings by default. It must be True if PY3 or IronPython.

libearth.compat.**binary**(*string*, *var=None*)
Makes string to str in Python 2. Makes string to bytes in Python 3 or IronPython.

Parameters

- **string** (bytes, str, unicode) – a string to cast it to binary_type
- **var** (str) – an optional variable name to be used for error message

libearth.compat.**binary_type**
(type) Type for representing binary data. str in Python 2 and bytes in Python 3.

alias of str

libearth.compat.**encode_filename**(*filename*)
If filename is a text_type, encode it to binary_type according to filesystem's default encoding.

libearth.compat.**file_types = (<class 'io.RawIOBase'>, <type 'file'>)**
(type, tuple) Types for file objects that have fileno().

libearth.compat.**string_type**
(type) Type for text data. basestring in Python 2 and str in Python 3.

alias of basestring

libearth.compat.**text**(*string*)
Makes string to str in Python 3 or IronPython. Does nothing in Python 2.

Parameters **string** (bytes, str, unicode) – a string to cast it to text_type

libearth.compat.**text_type**
(type) Type for representing Unicode textual data. unicode in Python 2 and str in Python 3.

alias of unicode

class libearth.compat.**xrange**
The xrange() function. Alias for range() in Python 3.

3.1.3 libearth.compat.etree — ElementTree compatibility layer

This proxy module offers a compatibility layer between several ElementTree implementations.

- If there's installed lxml module, use lxml.etree.
- If xml.etree.cElementTree is available, use it.
- If IronPython, use xml.etree.ElementTree with libearth.compat.clrxmlreader.TreeBuilder.
- Otherwise, use xml.etree.ElementTree.

It provides the following two functions:

libearth.compat.etree.**fromstring**(*string*)
Parse the given XML `string`.

Parameters **string** (`str`, `bytes`, `basestring`) – xml string to parse

Returns the element tree object

libearth.compat.etree.**fromstringlist**(*iterable*)
Parse the given chunks of XML string.

Parameters **iterable** (`collections.Iterable`) – chunks of xml string to parse

Returns the element tree object

libearth.compat.etree.**tostring**(*tree*)
Generate an XML string from the given element tree.

Parameters **tree** – an element tree object to serialize

Returns an xml string

Return type `str`, `bytes`

libearth.compat.etree.**fromstring**(*text*, *parser=None*, *base_url=None*)
Parses an XML document or fragment from a string. Returns the root node (or the result returned by a parser target).

To override the default parser with a different parser you can pass it to the `parser` keyword argument.

The `base_url` keyword argument allows to set the original base URL of the document to support relative Paths when looking up external entities (DTD, XInclude, ...).

libearth.compat.etree.**fromstringlist**(*strings*, *parser=None*)
Parses an XML document from a sequence of strings. Returns the root node (or the result returned by a parser target).

To override the default parser with a different parser you can pass it to the `parser` keyword argument.

3.1.4 libearth.xml.compat.clrxmlreader — XML parser implementation for CLR

Python `xml.sax` parser implementation and ElementTree builder using CLR `System.Xml.XmlReader`.

See also:

- XmlReader Class
- Comparing XmlReader to SAX Reader

libearth.compat.clrxmlreader.**XMLNS_XMLNS = 'http://www.w3.org/2000/xmlns/'**
(`str`) The reserved namespace URI for XML namespace.

class libearth.compat.clrxmlreader.**IteratorStream**(*iterable*)
`System.IO.Stream` implementation that takes a Python iterable and then transforms it into CLR stream.

Parameters **iterable** (`collections.Iterable`) – a Python iterable to transform

class libearth.compat.clrxmlreader.**TreeBuilder**
ElementTree builder using `System.Xml.XmlReader`.

class libearth.compat.clrxmlreader.**XmlReader**
SAX `PullReader` implementation using CLR `System.Xml.XmlReader`.

libearth.compat.clrxmlreader.**create_parser**()
Create a new XmlReader() parser instance.

Returns a new parser instance

Return type XmlReader

3.1.5 libearth.compat.parallel — Threading-related compatibility layer

libearth.compat.parallel.**cpu_count**()
Get the number of CPU cores.

Returns the number of cpu cores

Return type numbers.Integral

libearth.compat.parallel.**parallel_map**(*pool_size*, *function*, *iterable*, **iterables*)
Parallel vesion of builtin map() except of some differences:

- It takes a more argument at first: pool_size.
- The function applications will be done in parallel.
- The order of arguments to results are not maintained. You should treat these as a set.
- The result is a lazy iterable. Although the function immediately returns an iterable, it might block if some results are not completely ready when it's iterated.

Parameters

- **pool_size** (numbers.Integral) – the number of workers
- **function** (collections.Callable) – the function to apply iterables as its arguments
- **iterable** (collections.Iterable) – function argument values

Returns a promise iterable to future results

Return type collections.Iterable

Changed in version 0.1.1: Errored values are raised at the lastest.

libearth.compat.parallel.**cpu_count**()
Returns the number of CPUs in the system

3.1.6 libearth.compat.xmlpullreader — Pulling SAX parser

class libearth.compat.xmlpullreader.**PullReader**
SAX parser interface which provides similar but slightly less power than IncremenetalParser.

IncrementalParser can feed arbitrary length of bytes while it can't determine how long bytes to feed.

close()
This method is called when the entire XML document has been passed to the parser through the feed method, to notify the parser that there are no more data. This allows the parser to do the final checks on the document and empty the internal data buffer.

The parser will not be ready to parse another document until the reset method has been called.

close() may raise SAXException.

Raises xml.sax.SAXException when something goes wrong

feed()
This method makes the parser to parse the next step node, emitting the corresponding events.

`feed()` may raise `SAXException`.

Returns whether the stream buffer is not empty yet

Return type `bool`

Raises **xml.sax.SAXException** when something goes wrong

prepareParser(*iterable*)
This method is called by the parse implementation to allow the SAX 2.0 driver to prepare itself for parsing.

Parameters **iterable** (`collections.Iterable`) – iterable of `bytes`

reset()
This method is called after close has been called to reset the parser so that it is ready to parse new documents. The results of calling parse or feed after close without calling reset are undefined.

3.1.7 `libearth.crawler` — Crawler

Crawl feeds.

`libearth.crawler.`**crawl**(*feeds*, *pool_size*)
Crawl feeds in feed list using thread.

Parameters **feeds** – feeds

Returns set of pairs (*~libearth.feed.Feed*, crawler hint)

Return type `collections.Iterable`

3.1.8 `libearth.feed` — Feeds

`libearth` internally stores archive data as Atom format. It's exactly not a complete set of **RFC 4287**, but a subset of the most of that. Since it's not intended for crawling but internal representation, it does not follow robustness principle or such thing. It simply treats stored data are all valid and well-formed.

`libearth.feed.`**ATOM_XMLNS = 'http://www.w3.org/2005/Atom'**
(`str`) The XML namespace name used for Atom (**RFC 4287**).

`libearth.feed.`**MARK_XMLNS = 'http://earthreader.org/mark/'**
(`str`) The XML namespace name used for Earth Reader `Mark` metadata.

class `libearth.feed.`**Category**(*_parent=None*, ***attributes*)
Category element defined in **RFC 4287** (section 4.2.2).

label
(`str`) The optional human-readable label for display in end-user applications. It corresponds to `label` attribute of **RFC 4287** (section 4.2.2.3).

scheme_uri
(`str`) The URI that identifies a categorization scheme. It corresponds to `scheme` attribute of **RFC 4287** (section 4.2.2.2).

See also:

- Tag Scheme? by Tim Bray
- Representing tags in Atom by Edward O'Connor

term
(str) The required machine-readable identifier string of the cateogry. It corresponds to term attribute of RFC 4287 (section 4.2.2.1).

class libearth.feed.**Content**(*_parent=None*, ***attributes*)
Content construct defined in RFC 4287 (section 4.1.3).

MIMETYPE_PATTERN = <_sre.SRE_Pattern object at 0x2d37620>
(re.RegexObject) The regular expression pattern that matches with valid MIME type strings.

TYPE_MIMETYPE_MAP = {'text': 'text/plain', 'xhtml': 'application/xhtml+xml', 'html': 'text/html'}
(collections.Mapping) The mapping of type string (e.g. 'text') to the corresponding MIME type (e.g. *text/plain*).

mimetype
(str) The mimetype of the content.

source_uri
(str) An optional remote content URI to retrieve the content.

class libearth.feed.**Entry**(*_parent=None*, ***kwargs*)
Represent an individual entry, acting as a container for metadata and data associated with the entry. It corresponds to atom:entry element of RFC 4287 (section 4.1.2).

content
(Content) It either contains or links to the content of the entry.

It corresponds to atom:content element of RFC 4287 (section 4.1.3).

published_at
(datetime.datetime) The tz-aware datetime indicating an instant in time associated with an event early in the life cycle of the entry. Typically, published_at will be associated with the initial creation or first availability of the resource. It corresponds to atom:published element of RFC 4287 (section 4.2.9).

read
(Mark) Whether and when it's read or unread.

source
(Source) If an entry is copied from one feed into another feed, then the source feed's metadata may be preserved within the copied entry by adding source if it is not already present in the entry, and including some or all of the source feed's metadata as the source's data.

It is designed to allow the aggregation of entries from different feeds while retaining information about an entry's source feed.

It corresponds to atom:source element of RFC 4287 (section 4.2.10).

starred
(Mark) Whether and when it's starred or unstarred.

summary
(Text) The text field that conveys a short summary, abstract, or excerpt of the entry. It corresponds to atom:summary element of RFC 4287 (section 4.2.13).

class libearth.feed.**Feed**(*_parent=None*, ***kwargs*)
Atom feed document, acting as a container for metadata and data associated with the feed.

It corresponds to atom:feed element of RFC 4287 (section 4.1.1).

entries
(collections.MutableSequence) The list of Entry objects that represent an individual entry,

acting as a container for metadata and data associated with the entry. It corresponds to `atom:entry` element of **RFC 4287** (section 4.1.2).

class `libearth.feed.`**`Generator`**(*_parent=None*, ***attributes*)
Identify the agent used to generate a feed, for debugging and other purposes. It's corresponds to `atom:generator` element of **RFC 4287** (section 4.2.4).

`uri`
(`str`) A URI that represents something relavent to the agent.

`value`
(`str`) The human-readable name for the generating agent.

`version`
(`str`) The version of the generating agent.

class `libearth.feed.`**`Link`**(*_parent=None*, ***attributes*)
Link element defined in **RFC 4287** (section 4.2.7).

`byte_size`
(`numbers.Integral`) The optional hint for the length of the linked content in octets. It corresponds to `length` attribute of **RFC 4287** (section 4.2.7.6).

`language`
(`str`) The language of the linked content. It corresponds to `hreflang` attribute of **RFC 4287** (section 4.2.7.4).

`mimetype`
(`str`) The optional hint for the MIME media type of the linked content. It corresponds to `type` attribute of **RFC 4287** (section 4.2.7.3).

`relation`
(`str`) The relation type of the link. It corresponds to `rel` attribute of **RFC 4287** (section 4.2.7.2).

`title`
(`str`) The title of the linked resource. It corresponds to `title` attribute of **RFC 4287** (section 4.2.7.5).

`uri`
(`str`) The link's required URI. It corresponds to `href` attribute of **RFC 4287** (section 4.2.7.1).

class `libearth.feed.`**`LinkList`**
Element list mixin specialized for `Link`.

`filter_by_mimetype`(*pattern*)
Filter links by their `mimetype` e.g.:

```
links.filter_by_mimetype('text/html')
```

`pattern` can include wildcards (`*`) as well e.g.:

```
links.filter_by_mimetype('application/xml+*')
```

Parameters **pattern** (`str`) – the mimetype pattern to filter

Returns the filtered links

Return type `LinkList`

class `libearth.feed.`**`Mark`**(*_parent=None*, ***attributes*)
Represent whether the entry is read, starred, or tagged by user. It's not a part of **RFC 4287** Atom standard, but extension for Earth Reader.

marked
(bool) Whether it's marked or not.

updated_at
(datetime.datetime) Updated time.

class libearth.feed.**Metadata**(*_parent=None*, ***attributes*)
Common metadata shared by Source, Entry, and Feed.

authors
(collections.MutableSequence) The list of Person objects which indicates the author of the entry or feed. It corresponds to atom:author element of **RFC 4287** (section 4.2.1).

categories
(collections.MutableSequence) The list of Category objects that conveys information about categories associated with an entry or feed. It corresponds to atom:category element of **RFC 4287** (section 4.2.2).

contributors
(collections.MutableSequence) The list of Person objects which indicates a person or other entity who contributed to the entry or feed. It corresponds to atom:contributor element of **RFC 4287** (section 4.2.3).

id
(str) The URI that conveys a permanent, universally unique identifier for an entry or feed. It corresponds to atom:id element of **RFC 4287** (section 4.2.6).

links
(collections.LinkList) The list of Link objects that define a reference from an entry or feed to a web resource. It corresponds to atom:link element of **RFC 4287** (section 4.2.7).

rights
(Text) The text field that conveys information about rights held in and of an entry or feed. It corresponds to atom:rights element of **RFC 4287** (section 4.2.10).

title
(Text) The human-readable title for an entry or feed. It corresponds to atom:title element of **RFC 4287** (section 4.2.14).

updated_at
(datetime.datetime) The tz-aware datetime indicating the most recent instant in time when the entry was modified in a way the publisher considers significant. Therefore, not all modifications necessarily result in a changed updated_at value. It corresponds to atom:updated element of **RFC 4287** (section 4.2.15).

class libearth.feed.**Person**(*_parent=None*, ***attributes*)
Person construct defined in **RFC 4287** (section 3.2).

email
(str) The optional email address associated with the person. It corresponds to atom:email element of **RFC 4287** (section 3.2.3).

name
(str) The human-readable name for the person. It corresponds to atom:name element of **RFC 4287** (section 3.2.1).

uri
(str) The optional URI associated with the person. It corresponds to atom:uri element of **RFC 4287** (section 3.2.2).

class libearth.feed.**Source**(*_parent=None*, ***attributes*)
All metadata for Feed excepting Feed.entries. It corresponds to atom:source element of RFC 4287 (section 4.2.10).

generator
(Generator) Identify the agent used to generate a feed, for debugging and other purposes. It corresponds to atom:generator element of RFC 4287 (section 4.2.4).

icon
(str) URI that identifies an image that provides iconic visual identification for a feed. It corresponds to atom:icon element of RFC 4287 (section 4.2.5).

logo
(str) URI that identifies an image that provides visual identification for a feed. It corresponds to atom:logo element of RFC 4287 (section 4.2.8).

subtitle
(Text) A text that conveys a human-readable description or subtitle for a feed. It corresponds to atom:subtitle element of RFC 4287 (section 4.2.12).

class libearth.feed.**Text**(*_parent=None*, ***attributes*)
Text construct defined in RFC 4287 (section 3.1).

sanitized_html
(str) The secure HTML string of the text. If it's a plain text, this becomes entity-escaped HTML string (for example, '<Hello>' becomes '<Hello>'), and if it's a HTML text, the value is sanitized (for example, '<script>alert(1);</script> Hello ' comes ' Hello ').

type
(str) The type of the text. It could be one of 'text' or 'html'. It corresponds to RFC 4287 (section 3.1.1).

Note: It currently does not support 'xhtml'.

---

value
(str) The content of the text. Interpretation for this has to differ according to its type. It corresponds to RFC 4287 (section 3.1.1.1) if type is 'text', and RFC 4287 (section 3.1.1.2) if type is 'html'.

3.1.9 libearth.parser — Parsing various RSS formats

libearth.parser.atom — Atom parser

Parsing Atom feed. Atom specification is RFC 4287

libearth.parser.atom.**XMLNS_ATOM = 'http://www.w3.org/2005/Atom'**
(str) The XML namespace for Atom format.

libearth.parser.atom.**XMLNS_XML = 'http://www.w3.org/XML/1998/namespace'**
(str) The XML namespace for the predefined xml: prefix.

libearth.parser.atom.**parse_atom**(*xml*, *feed_url*, *parse_entry=True*)
Atom parser. It parses the Atom XML and returns the feed data as internal representation.

Parameters

- **xml** (str) – target atom xml to parse

- **feed_url** (str) – the url used to retrieve the atom feed. it will be the base url when there are any relative urls without `xml:base` attribute
- **parse_entry** – whether to parse inner items as well. it's useful to ignore items when retrieve <source> in rss 2.0. `True` by default.

Returns a pair of (`Feed`, crawler hint)

Return type `tuple`

libearth.parser.autodiscovery — Autodiscovery

This module provides functions to autodiscovery feed url in document.

libearth.parser.autodiscovery.**ATOM_TYPE = 'application/atom+xml'**
(str) The MIME type of Atom format.

libearth.parser.autodiscovery.**RSS_TYPE = 'application/rss+xml'**
(str) The MIME type of RSS 2.0 format.

libearth.parser.autodiscovery.**TYPE_TABLE = {<function parse_atom at 0x2e28488>: 'application/atom+xml', <fun**
(collections.Mapping) The mapping table of feed types

class libearth.parser.autodiscovery.**AutoDiscovery**
Parse the given HTML and try finding the actual feed urls from it.

class libearth.parser.autodiscovery.**FeedLink**
Namedtuple which is a pair of `type` and `url`

type
Alias for field number 0

url
Alias for field number 1

exception libearth.parser.autodiscovery.**FeedUrlNotFoundError**(*msg*)
Exception raised when feed url cannot be found in html.

libearth.parser.autodiscovery.**autodiscovery**(*document*, *url*)
If the given url refers an actual feed, it returns the given url without any change.

If the given url is a url of an ordinary web page (i.e. *text/html*), it finds the urls of the corresponding feed. It returns feed urls in feed types' lexicographical order.

If autodiscovery failed, it raise `FeedUrlNotFoundError`.

Parameters

- **document** (str) – html, or xml strings
- **url** (str) – the url used to retrieve the `document`. if feed url is in html and represented in relative url, it will be rebuilt on top of the `url`

Returns list of `FeedLink` objects

Return type `collections.MutableSequence`

libearth.parser.heuristic — Guessing

Guess the syndication format of given arbitrary XML documents.

libearth.parser.heuristic.**TYPE_ATOM**(*xml*, *feed_url*, *parse_entry=True*)
(str) The document type value for Atom format.

libearth.parser.heuristic.**TYPE_RSS2** (*xml*, *feed_url=None*, *parse_entry=True*)
(str) THe document type value for RSS 2.0 format.

libearth.parser.heuristic.**get_format** (*document*)
Guess the syndication format of an arbitrary document.

Parameters **document** (str) – document string to guess

libearth.parser.rss2 — RSS 2.0 parser

Parsing RSS 2.0 feed.

libearth.parser.rss2.**parse_rss** (*xml*, *feed_url=None*, *parse_entry=True*)
Parse RSS 2.0 XML and translate it into Atom.

To make the feed data valid in Atom format, id and link[rel=self] fields would become the url of the feed.

If pubDate is not present, updated field will be from the latest entry's updated time, or the time it's crawled instead.

Parameters

- **xml** (str) – rss 2.0 xml string to parse
- **parse_item** (bool) – whether to parse items (entries) as well. it's useful when to ignore items when retrieve <source>. True by default

Returns a pair of (Feed, crawler hint)

Return type tuple

3.1.10 libearth.repository — Repositories

Repository abstracts storage backend e.g. filesystem. There might be platforms that have no chance to directly access file system e.g. iOS, and in that case the concept of repository makes you to store data directly to Dropbox or Google Drive instead of filesystem. However in the most cases we will simply use FileSystemRepository even if data are synchronized using Dropbox or **rsync**.

In order to make the repository highly configurable it provides the way to lookup and instantiate the repository from url. For example, the following url will load FileSystemRepository which sets path to /home/dahlia/.earthreader/:

```
file:///home/dahlia/.earthreader/
```

For extensibility every repository class has to implement from_url() and to_url() methods, and register it as an entry point of libearth.repositories group e.g.:

```
[libearth.repositories]
file = libearth.repository:FileSystemRepository
```

Note that the entry point name (file in the above example) becomes the url scheme to lookup the corresponding repository class (libearth.repository.FileSystemRepository in the above example).

class libearth.repository.**FileIterator** (*path*, *buffer_size*)
Read a file through Iterator protocol, with automatic closing of the file when it ends.

Parameters

- **path** (str) – the path of file

- **buffer_size** (numbers.Integral) – the size of bytes that would be produced each step

exception libearth.repository.**FileNotFoundError**
Raised when a given path does not exist.

class libearth.repository.**FileSystemRepository**(*path*, *mkdir=True*, *atomic=False*)
Builtin implementation of Repository interface which uses the ordinary file system.

Parameters

- **path** (str) – the directory path to store keys
- **mkdir** (bool) – create the directory if it doesn't exist yet. True by default
- **atomic** – make the update invisible until it's complete. False by default

Raises

- **FileNotFoundError** – when the path doesn't exist
- **NotADirectoryError** – when the path is not a directory

path = None
(str) The path of the directory to read and write data files. It should be readable and writable.

exception libearth.repository.**NotADirectoryError**
Raised when a given path is not a directory.

class libearth.repository.**Repository**
Repository interface agnostic to its underlying storage implementation. Stage objects can deal with documents to be stored using the interface.

Every content in repositories is accessible using *keys*. It actually abstracts out "filenames" in "file systems", hence keys share the common concepts with filenames. Keys are hierarchical, like file paths, so consists of multiple sequential strings e.g. ['dir', 'subdir', 'key']. You can list() all subkeys in the upper key as well e.g.:

```
repository.list(['dir', 'subdir'])
```

exists(*key*)
Return whether the key exists or not. It returns False if it doesn't exist instead of raising RepositoryKeyError.

Parameters **key** (collections.Sequence) – the key to find whether it exists

Returns True only if the given key exists, or False if not exists

Return type bool

Note: Every subclass of Repository has to override exists() method to implement details.

---

classmethod from_url(*url*)
Create a new instance of the repository from the given url. It's used for configuring the repository in plain text e.g. *.ini.

Note: Every subclass of Repository has to override from_url() static/class method to implement details.

Parameters **url** (urllib.parse.ParseResult) – the parsed url tuple

Returns a new repository instance

Return type Repository

Raises ValueError when the given url is not invalid

list(*key*)
List all subkeys in the `key`.

Parameters **key** (`collections.Sequence`) – the incomplete key that might have subkeys

Returns the set of subkeys (set of strings, not set of string lists)

Return type `collections.Set`

Raises **RepositoryKeyError** the `key` cannot be found in the repository, or it's not a directory

Note: Every subclass of `Repository` has to override `list()` method to implement details.

read(*key*)
Read the content from the `key`.

Parameters **key** (`collections.Sequence`) – the key which stores the content to read

Returns byte string chunks

Return type `collections.Iterable`

Raises **RepositoryKeyError** the `key` cannot be found in the repository, or it's not a file

Note: Every subclass of `Repository` has to override `read()` method to implement details.

to_url(*scheme*)
Generate a url that `from_url()` can accept. It's used for configuring the repository in plain text e.g. `*.ini`. URL `scheme` is determined by caller, and given through argument.

Note: Every subclass of `Repository` has to override `to_url()` method to implement details.

Parameters **scheme** – a determined url scheme

Returns a url that `from_url()` can accept

Return type `str`

write(*key*, *iterable*)
Write the `iterable` into the `key`.

Parameters

- **key** (`collections.Sequence`) – the key to stores the `iterable`
- **iterable** (`collections.Iterable`) – the iterable object yiels chunks of the whole content. every chunk has to be a byte string

Note: Every subclass of `Repository` has to override `write()` method to implement details.

exception `libearth.repository.`**RepositoryKeyError**(*key*, **args*, ***kwargs*)
Exception which rises when the requested key cannot be found in the repository.

key = None
(`collections.Sequence`) The requested key.

`libearth.repository.`**from_url**(*url*)
Load the repository instance from the given configuration `url`.

Note: If `setuptools` is not installed it will only support `file://` scheme and `FileSystemRepository`.

Parameters **url** (`str`, `urllib.parse.ParseResult`) – a repository configuration url

Returns the loaded repository instance

Return type `Repository`

Raises

- **LookupError** – when the corresponding repository type to the given `url` scheme cannot be found
- **ValueError** – when the given `url` is invalid

3.1.11 `libearth.sanitizer` — Sanitize HTML tags

class `libearth.sanitizer.`**`HtmlSanitizer`**
HTML parser that is internally used by `sanitize_html()` function.

`DISALLOWED_SCHEMES` **= frozenset(['about', 'jscript', 'livescript', 'javascript', 'mocha', 'vbscript', 'data'])**
(`collections.Set`) The set of disallowed URI schemes e.g. `javascript:`.

`DISALLOWED_STYLE_PATTERN` **= <_sre.SRE_Pattern object at 0x2df2830>**
(`re.RegexObject`) The regular expression pattern that matches to disallowed CSS properties.

class `libearth.sanitizer.`**`MarkupTagCleaner`**
HTML parser that is internally used by `clean_html()` function.

`libearth.sanitizer.`**`clean_html`**(*html*)
Strip *all* markup tags from `html` string. That means, it simply makes the given `html` document a plain text.

Parameters **html** (`str`) – html string to clean

Returns cleaned plain text

Return type `str`

`libearth.sanitizer.`**`sanitize_html`**(*html*)
Sanitize the given `html` string. It removes the following tags and attributes that are not secure nor useful for RSS reader layout:

- `<script>` tags
- `display: none;` styles
- JavaScript event attributes e.g. `onclick`, `onload`
- `href` attributes that start with `javascript:`, `jscript:`, `livescript:`, `vbscript:`, `data:`, `about:`, or `mocha:`.

Parameters **html** (`str`) – html string to sanitize

Returns cleaned plain text

Return type `str`

3.1.12 `libearth.schema` — Declarative schema for pulling DOM parser of XML

There are well-known two ways to parse XML:

Document Object Model It reads the whole XML and then makes a tree in memory. You can easily treverse the document as a tree, but the parsing can't be streamed. Moreover it uses memory for data you don't use.

Simple API for XML It's an event-based sequential access parser. It means you need to listen events from it and then utilize its still unstructured data by yourself. In other words, you don't need to pay memory to data you never use if you simply do nothing for them when you listen the event.

Pros and cons between these two ways are obvious, but there could be another way to parse XML: *mix them*.

The basic idea of this pulling DOM parser (which this module implements) is that the parser can consume the stream just in time when you actually reach the child node. There should be an assumption for that: parsed XML has a schema for it. If the document is schema-free, this heuristic approach loses the most of its efficiency.

So the parser should have the information about the schema of XML document it'd parser, and we can declare the schema by defining classes. It's a thing like ORM for XML. For example, suppose there is a small XML document:

```
<?xml version="1.0"?>
<person version="1.0">
  <name>Hong Minhee</name>
  <url>http://dahlia.kr/</url>
  <url>https://github.com/dahlia</url>
  <url>https://bitbucket.org/dahlia</url>
  <dob>1988-08-04</dob>
</person>
```

You can declare the schema for this like the following class definition:

```
class Person(DocumentElement):
    __tag__ = 'person'
    format_version = Attribute('version')
    name = Text('name')
    url = Child('url', URL, multiple=True)
    dob = Child('dob', Date)
```

`libearth.schema.`**`PARSER_LIST`** **= []**
(`collections.Sequence`) The list of `xml.sax` parser implementations to try to import.

class `libearth.schema.`**`Attribute`**(*name*, *codec=None*, *xmlns=None*, *required=False*, *default=None*, *encoder=None*, *decoder=None*)
Declare possible element attributes as a descriptor.

Parameters

- **name** (`str`) – the XML attribute name
- **codec** (`Codec`, `collections.Callable`) – an optional codec object to use. if it's callable and not an instance of `Codec`, its return value will be used instead. it means this can take class object of `Codec` subtype that is not instantiated yet unless the constructor require any arguments
- **xmlns** (`str`) – an optional XML namespace URI
- **required** (`bool`) – whether the child is required or not. `False` by default
- **encoder** (`collections.Callable`) – an optional function that encodes Python value into XML text value e.g. `str()`. the encoder function has to take an argument
- **decoder** (`collections.Callable`) – an optional function that decodes XML text value into Python value e.g. `int()`. the decoder function has to take a string argument

key_pair = None
(tuple) The pair of (xmlns, name).

name = None
(str) The XML attribute name.

required = None
(bool) Whether it is required for the element.

xmlns = None
(str) The optional XML namespace URI.

class libearth.schema.**Child**(*tag*, *element_type*, *xmlns=None*, *required=False*, *multiple=False*, *sort_key=None*, *sort_reverse=None*)
Declare a possible child element as a descriptor.

In order to have Child of the element type which is not defined yet (or self-referential) pass the class name of the element type to contain. The name will be lazily evaluated e.g.:

```
class Person(Element):
    '''Everyone can have their children, that also are a Person.'''

    children = Child('child', 'Person', multiple=True)
```

Parameters

- **tag** (str) – the tag name
- **xmlns** (str) – an optional XML namespace URI
- **element_type** (type, str) – the type of child element(s). it has to be a subtype of Element. if it's a string it means referring the class name which is going to be lazily evaluated
- **required** (bool) – whether the child is required or not. it's exclusive to multiple. False by default
- **multiple** (bool) – whether the child can be multiple. it's exclusive to required. False by default
- **sort_key** (collections.Callable) – an optional function to be used for sorting multiple child elements. it has to take a child as Element and return a value for sort key. it is the same to key option of sorted() built-in function. note that *it doesn't guarantee that all elements must be sorted in runtime*, but all elements become sorted when it's written using write() function. it's available only when multiple is True. use sort_reverse for descending order.
- **sort_reverse** (bool) – ehether to reverse elements when they become sorted. it is the same to reverse option of sorted() built-in function. it's available only when sort_key is present

element_type
(type) The class of this child can contain. It must be a subtype of Element.

class libearth.schema.**Codec**
Abstract base class for codecs to serialize Python values to be stored in XML and deserialize XML texts to Python values.

In most cases encoding and decoding are implementation details of *format* which is well-defined, so these two functions could be paired. The interface rely on that idea.

To implement a codec, you have to subclass `Codec` and override a pair of methods: `encode()` and `decode()`.

Codec objects are acceptable by `Attribute`, `Text`, and `Content` (all they subclass `CodecDescriptor`).

decode (*text*)
Decode the given XML `text` to Python value.

Parameters **text** (`str`) – XML text to decode

Returns the decoded Python value

Raises DecodeError when decoding the given XML `text` goes wrong

---

Note: Every `Codec` subtype has to override this method.

---

encode (*value*)
Encode the given Python `value` into XML text.

Parameters **value** – Python value to encode

Returns the encoded XML text

Return type `str`

Raises EncodeError when encoding the given `value` goes wrong

Note: Every `Codec` subtype has to override this method.

---

class `libearth.schema.`**CodecDescriptor** (*codec=None*, *encoder=None*, *decoder=None*)
Mixin class for descriptors that provide `decoder()` and `encoder()`.

`Attribute`, `Content` and `Text` can take `encoder` and `decoder` functions for them. It's used for encoding from Python values to XML string and decoding raw values from XML to natural Python representations.

It can take a `codec`, or `encode` and `decode` separately. (Of course they all can be present at a time.) In most cases, you'll need only `codec` parameter that encoder and decoder are coupled:

```
Text('dob', Rfc3339(prefer_utc=True))
```

Encoders can be specified using `encoder` parameter of descriptor's constructor, or `encoder()` decorator.

Decoders can be specified using `decoder` parameter of descriptor's constructor, or `decoder()` decorator:

```
class Person(DocumentElement):
    __tag__ = 'person'
    format_version = Attribute('version')
    name = Text('name')
    url = Child('url', URL, multiple=True)
    dob = Text('dob',
               encoder=datetime.date.strftime.isoformat,
               decoder=lambda s: datetime.date.strptime(s, '%Y-%m-%d'))

    @format_version.encoder
    def format_version(self, value):
        return '.'.join(map(str, value))

    @format_version.decoder
    def format_version(self, value):
        return tuple(map(int, value.split('.')))
```

Parameters

- **codec** (`Codec`, `collections.Callable`) – an optional codec object to use. if it's callable and not an instance of `Codec`, its return value will be used instead. it means this can take class object of `Codec` subtype that is not instantiated yet unless the constructor require any arguments
- **encoder** (`collections.Callable`) – an optional function that encodes Python value into XML text value e.g. `str()`. the encoder function has to take an argument
- **decoder** (`collections.Callable`) – an optional function that decodes XML text value into Python value e.g. `int()`. the decoder function has to take a string argument

decode (*text*, *instance*)

Decode the given `text` as it's programmed.

Parameters

- **text** (`str`) – the raw text to decode. xml attribute value or text node value in most cases
- **instance** (`Element`) – the instance that is associated with the descriptor

Returns decoded value

Note: Internal method.

---

decoder (*function*)

Decorator which sets the decoder to the decorated function:

```
import datetime

class Person(DocumentElement):
    '''Person.dob will be a datetime.date instance.'''

    __tag__ = 'person'
    dob = Text('dob')

    @dob.decoder
    def dob(self, dob_text):
        return datetime.date.strptime(dob_text, '%Y-%m-%d')

>>> p = Person('<person><dob>1987-07-26</dob></person>')
>>> p.dob
datetime.date(1987, 7, 26)
```

If it's applied multiple times, all decorated functions are piped in the order:

```
class Person(Element):
    '''Person.age will be an integer.'''

    age = Text('dob', decoder=lambda text: text.strip())

    @age.decoder
    def age(self, dob_text):
        return datetime.date.strptime(dob_text, '%Y-%m-%d')

    @age.decoder
    def age(self, dob):
        now = datetime.date.today()
        d = now.month < dob.month or (now.month == dob.month and
```

```
                                        now.day < dob.day)
        return now.year - dob.year - d

>>> p = Person('<person>\n\t<dob>\n\t\t1987-07-26\n\t</dob>\n</person>')
>>> p.age
26
>>> datetime.date.today()
datetime.date(2013, 7, 30)
```

Note: This creates a copy of the descriptor instance rather than manipulate itself in-place.

encoder (*function*)

Decorator which sets the encoder to the decorated function:

```
import datetime

class Person(DocumentElement):
    '''Person.dob will be written to ISO 8601 format'''

    __tag__ = 'person'
    dob = Text('dob')

    @dob.encoder
    def dob(self, dob):
        if not isinstance(dob, datetime.date):
            raise TypeError('expected datetime.date')
        return dob.strftime('%Y-%m-%d')

>>> isinstance(p, Person)
True
>>> p.dob
datetime.date(1987, 7, 26)
>>> ''.join(write(p, indent='', newline=''))
'<person><dob>1987-07-26</dob></person>'
```

If it's applied multiple times, all decorated functions are piped in the order:

```
class Person(Element):
    '''Person.email will have mailto: prefix when it's written
    to XML.

    '''

    email = Text('email', encoder=lambda email: 'mailto:' + email)

    @age.encoder
    def email(self, email):
        return email.strip()

    @email.encoder
    def email(self, email):
        login, host = email.split('@', 1)
        return login + '@' + host.lower()

>>> isinstance(p, Person)
True
>>> p.email
'  earthreader@librelist.com  '
```

```
>>> ''.join(write(p, indent='', newline=''))
>>> '<person><email>mailto:earthreader@librelist.com</email></person>')
```

Note: This creates a copy of the descriptor instance rather than manipulate itself in-place.

exception `libearth.schema.`**`CodecError`**
Rise when encoding/decoding between Python values and XML data goes wrong.

class `libearth.schema.`**`Content`**(*codec=None*, *encoder=None*, *decoder=None*)
Declare possible text nodes as a descriptor.

Parameters

- **codec** (`Codec`, `collections.Callable`) – an optional codec object to use. if it's callable and not an instance of `Codec`, its return value will be used instead. it means this can take class object of `Codec` subtype that is not instantiated yet unless the constructor require any arguments
- **encoder** (`collections.Callable`) – an optional function that encodes Python value into XML text value e.g. `str()`. the encoder function has to take an argument
- **decoder** (`collections.Callable`) – an optional function that decodes XML text value into Python value e.g. `int()`. the decoder function has to take a string argument

`read`(*element*, *value*)
Read raw `value` from XML, decode it, and then set the attribute for content of the given `element` to the decoded value.

Note: Internal method.

class `libearth.schema.`**`ContentHandler`**(*document*)
Event handler implementation for SAX parser.

It maintains the stack that contains parsing contexts of what element is lastly open, what descriptor is associated to the element, and the buffer for chunks of content characters the element has. Every context is represented as the namedtuple `ParserContext`.

Each time its events (`startElement()`, `characters()`, and `endElement()`) are called, it forwards the data to the associated descriptor. `Descriptor` subtypes implement `start_element()` method and `end_element()`.

exception `libearth.schema.`**`DecodeError`**
Rise when decoding XML data to Python values goes wrong.

class `libearth.schema.`**`Descriptor`**(*tag*, *xmlns=None*, *required=False*, *multiple=False*, *sort_key=None*, *sort_reverse=None*)
Abstract base class for `Child` and `Text`.

`end_element`(*reserved_value*, *content*)
Abstract method that is invoked when the parser meets an end of an element related to the descriptor. It will be called by `ContentHandler`.

Parameters

- **reserved_value** – the value `start_element()` method returned
- **content** (`str`) – the content text of the read element

`key_pair` = None
(`tuple`) The pair of (`xmlns`, `tag`).

multiple = None
(bool) Whether it can be zero or more for the element. If it's True required has to be False.

required = None
(bool) Whether it is required for the element. If it's True multiple has to be False.

sort_key = None
(collections.Callable) An optional function to be used for sorting multiple elements. It has to take an element and return a value for sort key. It is the same to key option of sorted() built-in function.

It's available only when multiple is True.

Use sort_reverse for descending order.

Note: It doesn't guarantee that all elements must be sorted in runtime, but all elements become sorted when it's written using write() function.

---

sort_reverse = None
(bool) Whether to reverse elements when they become sorted. It is the same to reverse option of sorted() built-in function.

It's available only when sort_key is present.

start_element (*element*, *attribute*)
Abstract method that is invoked when the parser meets a start of an element related to the descriptor. It will be called by ContentHandler.

Parameters

- **element** (Element) – the parent element of the read element
- **attribute** (str) – the attribute name of the descriptor

Returns a value to reserve. it will be passed to reserved_value parameter of end_element()

tag = None
(str) The tag name.

xmlns = None
(str) The optional XML namespace URI.

exception libearth.schema.**DescriptorConflictError**
Error which rises when a schema has duplicate descriptors more than one for the same attribute, the same child element, or the text node.

class libearth.schema.**DocumentElement** (*_parent=None*, ***kwargs*)
The root element of the document.

__tag__
(str) Every DocumentElement subtype has to define this attribute to the root tag name.

__xmlns__
(str) A DocumentElement subtype may define this attribute to the XML namespace of the document element.

class libearth.schema.**Element** (*_parent=None*, ***attributes*)
Represent an element in XML document.

It provides the default constructor which takes keywords and initializes the attributes by given keyword arguments. For example, the following code that uses the default constructor:

```
assert issubclass(Person, Element)

author = Person(
    name='Hong Minhee',
    url='http://dahlia.kr/'
)
```

is equivalent to the following code:

```
author = Person()
author.name = 'Hong Minhee'
author.url = 'http://dahlia.kr/'
```

classmethod **__coerce_from__**(*value*)

Cast a value which isn't an instance of the element type to the element type. It's useful when a boxed element type could be more naturally represented using builtin type.

For example, `Mark` could be represented as a boolean value, and `Text` also could be represented as a string.

The following example shows how the element type can be automatically casted from string by implementing `__coerce_from__()` class method:

```
@classmethod
def __coerce_from__(cls, value):
    if isinstance(value, str):
        return Text(value=value)
    raise TypeError('expected a string or Text')
```

__entity_id__()

Identify the entity object. It returns the entity object itself by default, but should be overridden.

Returns any value to identify the entity object

__merge_entities__(*other*)

Merge two entities (`self` and `other`). It can return one of the two, or even a new entity object. This method is used by `Session` objects to merge conflicts between concurrent updates.

Parameters **other** (`Element`) – other entity to merge. it's guaranteed that it's older session's (note that it doesn't mean this entity is older than `self`, but the session's last update is)

Returns on of the two, or even an new entity object that merges two entities

Return type `Element`

Note: The default implementation simply returns `self`. That means the entity of the newer session will always win unless the method is overridden.

---

class `libearth.schema.`**ElementList**(*element*, *descriptor*, *value_type=None*)

List-like object to represent multiple chidren. It makes the parser to lazily consume the buffer when an element of a particular offset is requested.

You can extend methods or properties for a particular element type using `element_list_for()` class decorator e.g.:

```
@element_list_for(Link)
class LinkList(collections.Sequence):
    '''Specialized ElementList for Link elements.'''

    def filter_by_mimetype(self, mimetype):
```

```
        '''Filter links by their mimetype.'''
        return [link for link in self if link.mimetype == mimetype]
```

Extended methods/properties can be used for element lists for the type:

```
assert isinstance(feed.links, LinkList)
assert isinstance(feed.links, ElementList)
feed.links.filter_by_mimetype('text/html')
```

consume_buffer()
Consume the buffer for the parser. It returns a generator, so can be stopped using `break` statement by caller.

Note: Internal method.

classmethod register_specialized_type(*value_type*, *specialized_type*)
Register specialized `collections.Sequence` type for a particular `value_type`.

An imperative version of :func'element_list_for()' class decorator.

Parameters

- **value_type** (`type`) – a particular element type that `specialized_type` would be used for instead of default `ElementList` class. it has to be a subtype of `Element`
- **specialized_type** (`type`) – a `collections.Sequence` type which extends methods and properties for `value_type`

specialized_types = {<class 'libearth.feed.Link'>: (<class 'libearth.feed.LinkList'>, None)}
(`collections.MutableMapping`) The internal table for specialized subtypes used by `register_specialized_type()` method and `element_list_for()` class decorator.

exception `libearth.schema.`**EncodeError**
Rise when encoding Python values into XML data goes wrong.

exception `libearth.schema.`**IntegrityError**
Rise when an element is invalid according to the schema.

exception `libearth.schema.`**SchemaError**
Error which rises when a schema definition has logical errors.

class `libearth.schema.`**Text**(*tag*, *codec=None*, *xmlns=None*, *required=False*, *multiple=False*, *encoder=None*, *decoder=None*, *sort_key=None*, *sort_reverse=None*)
Descriptor that declares a possible child element that only cosists of character data. All other attributes and child nodes are ignored.

Parameters

- **tag** (`str`) – the XML tag name
- **codec** (`Codec`, `collections.Callable`) – an optional codec object to use. if it's callable and not an instance of `Codec`, its return value will be used instead. it means this can take class object of `Codec` subtype that is not instantiated yet unless the constructor require any arguments
- **xmlns** (`str`) – an optional XML namespace URI
- **required** (`bool`) – whether the child is required or not. it's exclusive to `multiple`. `False` by default
- **multiple** (`bool`) – whether the child can be multiple. it's exclusive to `required`. `False` by default

- **encoder** (collections.Callable) – an optional function that encodes Python value into XML text value e.g. str(). the encoder function has to take an argument
- **decoder** (collections.Callable) – an optional function that decodes XML text value into Python value e.g. int(). the decoder function has to take a string argument
- **sort_key** (collections.Callable) – an optional function to be used for sorting multiple child elements. it has to take a child as Element and return a value for sort key. it is the same to key option of sorted() built-in function. note that *it doesn't guarantee that all elements must be sorted in runtime*, but all elements become sorted when it's written using write() function. it's available only when multiple is True. use sort_reverse for descending order.
- **sort_reverse** (bool) – ehether to reverse elements when they become sorted. it is the same to reverse option of sorted() built-in function. it's available only when sort_key is present

libearth.schema.**complete**(*element*)
Completely load the given element.

Parameters **element** (Element) – an element loaded by read()

class libearth.schema.**element_list_for**(*value_type*)
Class decorator which registers specialized ElementList subclass for a particular value_type e.g.:

```
@element_list_for(Link)
class LinkList(collections.Sequence):
    '''Specialized ElementList for Link elements.'''

    def filter_by_mimetype(self, mimetype):
        '''Filter links by their mimetype.'''
        return [link for link in self if link.mimetype == mimetype]
```

Parameters **value_type** (type) – a particular element type that specialized_type would be used for instead of default ElementList class. it has to be a subtype of Element

libearth.schema.**index_descriptors**(*element_type*)
Index descriptors of the given element_type to make them easy to be looked up by their identifiers (pairs of XML namespace URI and tag name).

Parameters **element_type** (type) – a subtype of Element to index its descriptors

---

Note: Internal function.

---

libearth.schema.**inspect_attributes**(*element_type*)
Get the dictionary of Attribute descriptors of the given element_type.

Parameters **element_type** (type) – a subtype of Element to inspect

Returns a dictionary of attribute identifiers (pairs of xml namespace uri and xml attribute name) to pairs of instance attribute name and associated Attribute descriptor

Return type collections.Mapping

Note: Internal function.

---

libearth.schema.**inspect_child_tags**(*element_type*)
Get the dictionary of Descriptor objects of the given element_type.

Parameters **element_type** (type) – a subtype of Element to inspect

Returns a dictionary of child node identifiers (pairs of xml namespace uri and tag name) to pairs of instance attribute name and associated `Descriptor`

Return type `collections.Mapping`

Note: Internal function.

`libearth.schema.`**`inspect_content_tag`**(*element_type*)
Gets the `Content` descriptor of the given `element_type`.

Parameters **element_type** (`type`) – a subtype of `Element` to inspect

Returns a pair of instance attribute name and associated `Content` descriptor

Return type `tuple`

Note: Internal function.

`libearth.schema.`**`inspect_xmlns_set`**(*element_type*)
Get the set of XML namespaces used in the given `element_type`, recursively including all child elements.

Parameters **element_type** (`type`) – a subtype of `Element` to inspect

Returns a set of uri strings of used all xml namespaces

Return type `collections.Set`

Note: Internal function.

`libearth.schema.`**`is_partially_loaded`**(*element*)
Return whether the given `element` is not completely loaded by `read()` yet.

Parameters **element** (`Element`) – an element

Returns whether `True` if the given `element` is partially loaded

Return type `bool`

`libearth.schema.`**`read`**(*cls*, *iterable*)
Initialize a document in read mode by opening the `iterable` of XML string.

```
with open('doc.xml', 'rb') as f:
    read(Person, f)
```

Returned document element is not fully read but partially loaded into memory, and then lazily (and eventually) loaded when these are actually needed.

Parameters

- **cls** (`type`) – a subtype of `DocumentElement`
- **iterable** (`collections.Iterable`) – chunks of XML string to read

Returns initialized document element in read mode

Return type `DocumentElement`

`libearth.schema.`**`validate`**(*element*, *recurse=True*, *raise_error=True*)
Validate the given `element` according to the schema.

```
from libearth.schema import IntegrityError, validate

try:
```

```
    validate(element)
except IntegrityError:
    print('the element {0!r} is invalid!'.format(element))
```

Parameters

- **element** (`Element`) – the element object to validate
- **recurse** (`bool`) – recursively validate the whole tree (child nodes). `True` by default
- **raise_error** (`bool`) – raise exception when the `element` is invalid. if it's `False` it returns `False` instead of raising an exception. `True` by default

Returns `True` if the `element` is valid. `False` if the `element` is invalid and `raise_error` option is `False`'

Raises IntegrityError when the `element` is invalid and `raise_error` option is `True`

class `libearth.schema.`**`write`**(*document*, *validate=True*, *indent=' '*, *newline='\n'*, *canonical_order=False*, *as_bytes=None*)

Write the given `document` to XML string. The return value is an iterator that yields chunks of an XML string.

```
with open('doc.xml', 'w') as f:
    for chunk in write(document):
        f.write(chunk)
```

Parameters

- **document** (`DocumentElement`) – the document element to serialize
- **validate** (`bool`) – whether validate the `document` or not. `True` by default
- **indent** (`str`) – an optional string to be used for indent. default is four spaces (`'  '`)
- **newline** (`str`) – an optional character to be used for newline. default is `'\n'`
- **canonical_order** (`bool`) – make the order of attributes and child nodes consistent to any python versions and implementations. useful for testing. `False` by default
- **as_bytes** – return chunks as `bytes` (`str` in Python 2) if `True`. return chunks as `str` (`unicode` in Python 3) if `False`. return chunks as default string type (`str`) by default

Returns chunks of an XML string

Return type `collections.Iterable`

3.1.13 `libearth.session` — Isolate data from other installations

This module provides merging facilities to avoid conflict between concurrent updates of the same document/entity from different devices (installations). There are several concepts here.

`Session` abstracts installations on devices. For example, if you have a laptop, a tablet, and a mobile phone, and two apps are installed on the laptop, then there have to be four sessions: *laptop-1*, *laptop-2*, *table-1*, and *phone-1*. You can think of it as branch if you are familiar with DVCS.

`Revision` abstracts timestamps of updated time. An important thing is that it preserves its session as well.

Base revisions (`MergeableDocumentElement.__base_revisions__`) show what revisions the current revision is built on top of. In other words, what revisions were merged into the current revision. `RevisionSet` is a dictionary-like data structure to represent them.

libearth.session.**SESSION_XMLNS = 'http://earthreader.org/session/'**
(str) The XML namespace name used for session metadata.

class libearth.session.**MergeableDocumentElement**(*_parent=None*, ***kwargs*)
Document element which is mergeable using Session.

class libearth.session.**Revision**
The named tuple type of (Session, datetime.datetime) pair.

session
Alias for field number 0

updated_at
Alias for field number 1

class libearth.session.**RevisionCodec**
Codec to encode/decode Revision pairs.

```
>>> from libearth.tz import utc
>>> session = Session('test-identifier')
>>> updated_at = datetime.datetime(2013, 9, 22, 3, 43, 40, tzinfo=utc)
>>> rev = Revision(session, updated_at)
>>> RevisionCodec().encode(rev)
'test-identifier 2013-09-22T03:43:40Z'
```

RFC3339_CODEC = <libearth.codecs.Rfc3339 object at 0x2e22c90>
(Rfc3339) The internally used codec to encode Revision.updated_at time to **RFC 3339** format.

class libearth.session.**RevisionParserHandler**
SAX content handler that picks session metadata (__revision__ and __base_revisions__) from the given document element.

Parsed result goes revision and base_revisions.

Used by parse_revision().

done = None
(bool) Represents whether the parsing is complete.

revision = None
(Revision) The parsed __revision__. It might be None.

class libearth.session.**RevisionSet**(*revisions=[]*)
Set of Revision pairs. It provides dictionary-like mapping protocol.

Parameters **revisions** (collections.Iterable) – the iterable of (Session, datetime.datetime) pairs

contains(*revision*)
Find whether the given revision is already merged to the revision set. In other words, return True if the revision doesn't have to be merged to the revision set anymore.

Parameters **revision** (Revision) – the revision to find whether it has to be merged or not

Returns True if the revision is included in the revision set, or False

Return type bool

copy()
Make a copy of the set.

Returns a new equivalent set

Return type RevisionSet

items()
The list of (Session, datetime.datetime) pairs.

Returns the list of Revision instances

Return type collections.ItemsView

merge(**sets*)
Merge two or more RevisionSets. The latest time remains for the same session.

Parameters ***sets** – one or more RevisionSet objects to merge

Returns the merged set

Return type RevisionSet

class libearth.session.**RevisionSetCodec**
Codec to encode/decode multiple Revision pairs.

```
>>> from datetime import datetime
>>> from libearth.tz import utc
>>> revs = RevisionSet([
...     (Session('a'), datetime(2013, 9, 22, 16, 58, 57, tzinfo=utc)),
...     (Session('b'), datetime(2013, 9, 22, 16, 59, 30, tzinfo=utc)),
...     (Session('c'), datetime(2013, 9, 22, 17, 0, 30, tzinfo=utc))
... ])
>>> encoded = RevisionSetCodec().encode(revs)
>>> encoded
'c 2013-09-22T17:00:30Z,\nb 2013-09-22T16:59:30Z,\na 2013-09-22T16:58:57Z'
>>> RevisionSetCodec().decode(encoded)
libearth.session.RevisionSet([
    Revision(session=libearth.session.Session('b'),
             updated_at=datetime.datetime(2013, 9, 22, 16, 59, 30,
                                          tzinfo=libearth.tz.Utc())),
    Revision(session=libearth.session.Session('c'),
             updated_at=datetime.datetime(2013, 9, 22, 17, 0, 30,
                                          tzinfo=libearth.tz.Utc())),
    Revision(session=libearth.session.Session('a'),
             updated_at=datetime.datetime(2013, 9, 22, 16, 58, 57,
                                          tzinfo=libearth.tz.Utc()))
])
```

SEPARATOR_PATTERN = <_sre.SRE_Pattern object at 0x2e240f8>
(re.RegexObject) The regular expression pattern that matches to separator substrings between revision pairs.

class libearth.session.**Session**
The unit of device (more abstractly, *installation*) that updates the same document (e.g. Feed). Every session must have its own unique identifier to avoid conflict between concurrent updates from different sessions.

Parameters **identifier** (str) – the unique identifier. automatically generated using uuid if not present

IDENTIFIER_PATTERN = <_sre.SRE_Pattern object at 0x2e24030>
(re.RegexObject) The regular expression pattern that matches to allowed identifiers.

identifier = None
(str) The session identifier. It has to be distinguishable from other devices/apps, but consistent for the same device/app.

interns = {}
(collections.MutableMapping) The pool of interned sessions. It's for maintaining single sessions

for the same identifiers.

merge(*a*, *b*, *force=False*)
Merge the given two documents and return new merged document. The given documents are not manipulated in place. Two documents must have the same type.

Parameters

- **a** (`MergeableDocumentElement`) – the first document to be merged
- **b** (`MergeableDocumentElement`) – the second document to be merged
- **force** – by default (`False`) it doesn't merge but simply pull a or b if one already contains other. if `force` is `True` it always merge two. it assumes `b` is newer than `a`

pull(*document*)
Pull the `document` (of possibly other session) to the current session.

Parameters **document** (`MergeableDocumentElement`) – the document to pull from the possibly other session to the current session

Returns the clone of the given `document` with the replaced `__revision__`. note that the `Revision.updated_at` value won't be revised. it could be the same object to the given `document` object if the session is the same

Return type `MergeableDocumentElement`

revise(*document*)
Mark the given `document` as the latest revision of the current session.

Parameters **document** (`MergeableDocumentElement`) – mergeable document to mark

libearth.session.**ensure_revision_pair**(*pair*, *force_cast=False*)
Check the type of the given `pair` and error unless it's a valid revision pair (`Session`, `datetime.datetime`).

Parameters

- **pair** (`collections.Sequence`) – a value to check
- **force_cast** (`bool`) – whether to return the casted value to `Revision` named tuple type

Returns the revision pair

Return type `Revision`, `collections.Sequence`

libearth.session.**parse_revision**(*iterable*)
Efficiently parse only `__revision__` and `__base_revisions__` from the given `iterable` which contains chunks of XML. It reads only head of the given document, and `iterable` will be not completely consumed in most cases.

Note that it doesn't validate the document.

Parameters **iterable** (`collections.Iterable`) – chunks of bytes which contains a `MergeableDocumentElement` element

Returns a pair of (`__revision__`, `__base_revisions__`). it might be `None` if the document is not stamped

Return type `collections.Sequence`

3.1.14 `libearth.stage` — Staging updates and transactions

Stage is a similar concept to Git's one. It's a unit of updates, so every change to the repository should be done through a stage.

It also does more than Git's stage: `Route`. Routing system hide how document should be stored in the repository, and provides the natural object-mapping interface instead.

Stage also provides transactions. All operations on staged documents should be done within a transaction. You can open and close a transaction using `with` statement e.g.:

```
with stage:
    subs = stage.subscriptions
    stage.subscriptions = some_operation(subs)
```

Transaction will merge all simultaneous updates if there are multiple updates when it's committed. You can easily achieve thread safety using transactions.

Note that it however doesn't guarantee data integrity between multiple processes, so *you have to use different session ids when there are multiple processes.*

class `libearth.stage.`**`BaseStage`**(*session*, *repository*)

Base stage class that routes nothing yet. It should be inherited to route document types. See also `Route` class.

It's a context manager, which is possible to be passed to `with` statement. The context maintains a transaction, that is required for all operations related to the stage:

```
with stage:
    v = stage.some_value
    stage.some_value =  operate(v)
```

If any ongoing transaction is not present while the operation requires it, it will raise `TransactionError`.

Parameters

- **session** (`Session`) – the current session to stage
- **repository** (`Repository`) – the repository to stage

`SESSION_DIRECTORY_KEY = ['.sessions']`

(`collections.Sequence`) The repository key of the directory where session list are stored.

`get_current_transaction`(*pop=False*)

Get the current ongoing transaction. If any transaction is not begun yet, it raises `TransactionError`.

Returns the dirty buffer that should be written when the transaction is committed

Return type `DirtyBuffer`

Raises TransactionError if not any transaction is not begun yet

`read`(*document_type*, *key*)

Read a document of `document_type` by the given `key` in the staged `repository`.

Parameters

- **document_type** (`type`) – the type of document to read. it has to be a subclass of `MergeableDocumentElement`
- **key** (`collections.Seqeuence`) – the key to find the `document` in the `repository`

Returns found document instance

Return type `MergeableDocumentElement`

Raises libearth.repository.RepositoryKeyError when the key cannot be found

Note: This method is intended to be internal. Use routed properties rather than this. See also `Route`.

repository = None
(`Repository`) The staged repository.

session = None
(`Session`) The current session of the stage.

sessions
(`collections.Set`) List all sessions associated to the `repository`. It includes the session of the current stage.

touch()
Touch the latest staged time of the current `session` into the `repository`.

Note: This method is intended to be internal.

transactions = None
(`collections.MutableMapping`) Ongoing transactions. Keys are the context identifier (that `get_current_context_id()` returns), and values are pairs of the `DirtyBuffer` that should be written when the transaction is committed, and stack information.

write(*key*, *document*, *merge=True*)
Save the `document` to the `key` in the staged `repository`.

Parameters

- **key** (`collections.Sequence`) – the key to be stored
- **document** (`MergeableDocumentElement`) – the document to save
- **merge** (`bool`) – merge with the previous revision of the same session (if exists). `True` by default

Returns actually written document

Return type `MergeableDocumentElement`

Note: This method is intended to be internal. Use routed properties rather than this. See also `Route`.

class `libearth.stage.`**Directory**(*stage*, *document_type*, *key_spec*, *indices*, *key*)
Mapping object which represents hierarchy of routed key path.

Parameters

- **stage** (`BaseStage`) – the current stage
- **document_type** (`type`) – the same to `Route.document_type`
- **key_spec** (`collections.Sequence`) – the same to `Route.key_spec` value
- **indices** (`collections.Sequence`) – the upper indices that are already completed
- **key** (`collections.Sequence`) – the upper key that are already completed

Note: The constructor is intended to be internal, so don't instantiate it directory. Use `Route` instead.

class libearth.stage.**DirtyBuffer**(*repository*, *lock*)
Memory-buffered proxy for the repository. It's used for transaction buffer which maintains updates to be written until the ongoing transaction is committed.

Parameters

- **repository** (Repository) – the bare repository where the buffer will flush() to
- **lock** (threading.RLock) – the common lock shared between dirty buffers of the same stage

---

Note: This class is intended to be internal.

---

flush(*_dictionary=None*, *_key=None*)
Flush all buffered updates to the repository.

repository = None
(Repository) The bare repository where the buffer will flush() to.

class libearth.stage.**Route**(*document_type*, *key_spec*)
Descriptor that routes a document_type to a particular key path pattern in the repository.

key_spec could contain some format strings. Format strings can take a keyword (session) and zero or more positional arguments.

For example, if you route a document type without any positional arguments in key_spec format:

```
class Stage(BaseStage):
    '''Stage example.'''

    metadata = Route(
        Metadata,
        ['metadata', '{session.identifier}.xml']
    )
```

Stage instance will has a metadata attribute that simply holds Metadata document instance (in the example):

```
>>> stage.metadata  # ['metadata', 'session-id.xml']
<Metadata ...>
```

If you route something with one or more positional arguments in key_spec format, then it works in some different way:

```
class Stage(BaseStage):
    '''Stage example.'''

    seating_chart = Route(
        Student,
        ['students', 'col-{0}', 'row-{1}', '{session.identifier}.xml']
    )
```

In the above routing, two positional arguments were used. It means that the seating_chart property will return two-dimensional mapping object (Directory):

```
>>> stage.seating_chart  # ['students', ...]
<libearth.directory.Directory ['students']>
>>> list(stage.seating_chart)
['A', 'B', 'C', 'D']
>>> b = stage.seating_chart['B']  # ['students', 'col-B', ...]
<libearth.directory.Directory ['students', 'col-B']>
```

```
>>> list(stage.seating_chart['B'])
['1', '2', '3', '4', '5', '6']
>>> stage.seating_chart['B']['6']  \
... # ['students', 'col-B', 'row-6', 'session-id.xml']
<Student B6>
```

Parameters

- **document_type** (`type`) – the type of document to route. it has to be a subclass of `MergeableDocumentElement`
- **key_spec** (`collections.Sequence`) – the repository key pattern that might contain some format strings e.g. `['docs', '{0}', '{session.identifier}.xml']`. positional values are used for directory indices (if present), and ``session keyword value is used for identifying sessions

document_type = None
(`type`) The type of the routed document. It is a subtype of `MergeableDocumentElement`.

key_spec = None
(`collections.Sequence`) The repository key pattern that might contain some format strings.

class `libearth.stage.`**Stage**(*session*, *repository*)
Staged documents of Earth Reader.

feeds
(`collections.MutableMapping`) The map of feed ids to `Feed` objects.

subscriptions
(`SubscriptionList`) The set of subscriptions.

exception `libearth.stage.`**TransactionError**
The error that rises if there's no ongoing transaction while it's needed to update the stage, or if there's already begun ongoing transaction when the new transaction get tried to begin.

`libearth.stage.`**compile_format_to_pattern**(*format_string*)
Compile a `format_string` to regular expression pattern. For example, `'string{0}like{1}this{{2}}'` will be compiled to `/^string(.*?)like(.*?)this\{2\}$/`.

Parameters **format_string** (`str`) – format string to compile

Returns compiled pattern object

Return type `re.RegexObject`

`libearth.stage.`**get_current_context_id**()
Identifies which context it is (greenlet, stackless, or thread).

Returns the identifier of the current context

3.1.15 `libearth.subscribe` — Subscription list

Maintain the subscription list using OPML format, which is de facto standard for the purpose.

class `libearth.subscribe.`**Body**(*_parent=None*, ***attributes*)
Represent `body` element of OPML document.

children
(`collections.MutableSequence`) Child `Outline` objects.

class libearth.subscribe.**Category**(*_parent=None*, ***attributes*)
Category which groups Subscription objects or other Category objects. It implements collections.MutableSet protocol.

children
(collections.MutableSequence) The list of child Outline elements. It's for internal use.

class libearth.subscribe.**CommaSeparatedList**
Encode strings e.g. ['a', 'b', 'c'] into a comma-separated list e.g. 'a,b,c', and decode it back to a Python list. Whitespaces between commas are ignored.

```
>>> codec = CommaSeparatedList()
>>> codec.encode(['technology', 'business'])
'technology,business'
>>> codec.decode('technology, business')
['technology', 'business']
```

class libearth.subscribe.**Head**(*_parent=None*, ***attributes*)
Represent head element of OPML document.

owner_email
(str) The owner's email.

owner_name
(str) The owner's name.

owner_uri
(str) The owner's website url.

title
(str) The title of the subscription list.

class libearth.subscribe.**Outline**(*_parent=None*, ***attributes*)
Represent outline element of OPML document.

created_at
(datetime.datetime) The created time.

label
(str) The human-readable text of the outline.

type
(str) Internally-used type identifier.

class libearth.subscribe.**Subscription**(*_parent=None*, ***attributes*)
Subscription which holds referring feed_uri.

feed_id
(str) The feed identifier to be used for lookup. It's intended to be SHA1 digest of Feed.id value (which is UTF-8 encoded).

feed_uri
(str) The feed url.

alternate_uri
(str) The web page url.

class libearth.subscribe.**SubscriptionList**(*_parent=None*, ***kwargs*)
The set (exactly, tree) of subscriptions. It consists of Subscriptions and Category objects for grouping. It implements collections.MutableSet protocol.

owner
(Person) The owner of the subscription list.

title
(str) The title of the subscription list.

version
(distutils.version.StrictVersion) The OPML version number.

class libearth.subscribe.**SubscriptionSet**
Mixin for SubscriptionList and Category, both can group Subscription object and other Category objects, to implement collections.MutableSet protocol.

categories
(collections.Mapping) Label to Category instance mapping.

children

(collections.MutableSequence) Child Outline objects.

Note: Every subclass of SubscriptionSet has to override children property to implement details.

subscribe(*feed*)
Add a subscription from Feed instance. Prefer this method over add() method.

Parameters **feed** (Feed) – feed to subscribe

subscriptions
(collections.Set) The subset which consists of only Subscription instances.

3.1.16 libearth.tz — Basic timezone implementations

Almost of this module is from the official documentation of datetime module in Python standard library.

libearth.tz.**utc**
(Utc, datetime.timezone) The tzinfo instance that represents UTC. It's an instance of Utc in Python 2 (which provide no built-in fixed-offset tzinfo implementation), and an instance of timezone with zero offset in Python 3.

class libearth.tz.**FixedOffset**(*offset*, *name=None*)
Fixed offset in minutes east from UTC.

```
>>> kst = FixedOffset(9 * 60, name='Asia/Seoul')  # KST +09:00
>>> current = now()
>>> current
datetime.datetime(2013, 8, 15, 3, 18, 37, 404562, tzinfo=libearth.tz.Utc())
>>> current.astimezone(kst)
datetime.datetime(2013, 8, 15, 12, 18, 37, 404562,
                  tzinfo=<libearth.tz.FixedOffset Asia/Seoul>)
```

class libearth.tz.**Utc**
UTC.

In most cases, it doesn't need to be directly instantiated: there's already the utc value.

libearth.tz.**now**()
Return the current datetime with the proper tzinfo setting.

```
>>> now()
datetime.datetime(2013, 8, 15, 3, 17, 11, 892272, tzinfo=libearth.tz.Utc())
>>> now()
datetime.datetime(2013, 8, 15, 3, 17, 17, 532483, tzinfo=libearth.tz.Utc())
```

3.1.17 `libearth.version` — Version data

`libearth.version.`**`VERSION`** = **'0.1.2'**
(`str`) The version string e.g. `'1.2.3'`.

`libearth.version.`**`VERSION_INFO`** = **(0, 1, 2)**
(`tuple`) The triple of version numbers e.g. `(1, 2, 3)`.

CHAPTER 4

Additional notes

4.1 Goal

Earth Reader aims to decentralize feed reader ecosystem which had been highly centralized to Google Reader. Google Reader had changed the world of news readers, from desktop apps to web-based services.

However Google Reader shut down on July 1, 2013. Everyone panicked, several new feed reader services were born, users had to migrate their data, and the most of alternative services were not able to import starred and read data, but just subscription list through OPML.

Feed readers are actually desktop apps at first. A few years later some people had started to lose their data, because desktop apps had simply stored data to local disk. In those days there were already some web-based feed readers e.g. Bloglines, Google Reader, but they provided worse experience than desktop apps (there were no Chrome, and JavaScript engines were way slower back then). Nevertheless people had gradually moved to web-based services from desktop apps, because they never (until the time at least) lost data, and were easily synchronized between multiple computers.

These feed reader services are enough convenient, but always have some risk that you can't control your own data. If the service you use suddenly shut down without giving you a chance to backup data, you would have to start everything from scratch. Your starred articles would be gone.

The goal of Earth Reader is to achieve the following subgoals at the same time:

- The whole data should be controlled by the owner. It means data will be tangible and reachable on the file system.
- It should be possible to synchronize data between multple devices, without any conflict between simultaneous updates.
- The implementation and data format should be open and free.
- It could provide native apps for the most of major platforms.

4.2 Core concepts

To achieve the *goal* of Earth Reader, its design need to resolve the following subproblems:

1. Data should be stored in tangible format and more specifically, in plain text with well-structured directory layout. It would be much better if data can be easily read and parsed by other softwares.
2. Data should be possible to be synchronized through several existing utilities including Dropbox, Google Drive, and even **rsync**, without any data corruption. In this docs we try to explain core concepts of libearth and what these concepts purpose to resolve.

4.2.1 Schema

All data libearth deals with are based on (de facto) standard formats. For example, it stores subscription list and its category hierarchy to an OPML file. OPML have been a de facto standard format to exchange subscription list by feed readers. It also stores all feed data to Atom format (**RFC 4287**).

Actually the most technologies related to RSS/syndication formats are from early 00's, and it means they had used XML instead of JSON today we use for the same purpose. OPML is an (though poorly structured) XML format, and Atom also is an XML format.

Since we need to deal with several XML data and not need any other formats, we decided to make something first-class model objects to XML like ORM to relational databases. You can find how it can be used for designing model objects at `libearth/feed.py` and `libearth/subscribe.py`. It looks similar to Django ORM and SQLAlchemy, and makes you to deal with XML documents in the same way you use plain Python objects.

Under the hood it does incremental parsing using SAX instead of DOM to reduce memory usage when the document is larger than a hundred megabytes.

See also:

Module `libearth.schema` Declarative schema for pulling DOM parser of XML

4.2.2 Read-time merge

Earth Reader data can be shared by multiple installations e.g. desktop apps, mobile apps, web apps. So there must be simultaneous updates between them that could conflict. An important constraint we have is synchronization isn't done by Earth Reader. We can't lock files nor do atomic operations on them.

Our solution to this is read-time merge. All data are not shared between installations at least in filesystem level. They have isolated files for the same entities, and libearth merges all of them when it's loaded into memory. Merged result doesn't affect to all replicas but only a replica that corresponds to the installation. You can understand the approach similar to DVCS (although there are actually many differences): installations are branches, and updates from others can be pulled to mine. If there are simultaneous changes, these are merged and then committed to mine. If there's no change for me, simply pull changes from others without merge. A big difference is that there's no push. You can only do pull others, or wait others to pull yours. It's because the most of existing synchronization utilities like Dropbox passively works in background. Moreover there could be offline.

4.2.3 Repository

`Repository` abstracts storage backend e.g. filesystem. There might be platforms that have no chance to directly access file system e.g. iOS, and in that case the concept of repository makes you to store data directly to Dropbox or Google Drive instead of filesystem. However in the most cases we will simply use `FileSystemRepository` even if data are synchronized using Dropbox or **rsync**.

See also:

Module `libearth.repository` Repositories

4.2.4 Session

`Session` abstracts installations. Every installation has its own session identifier. To be more exact it purposes to distinguish processes, hence every process has its unique identifier even if they are child processes of the same installation e.g. prefork workers.

Every session makes its own file for a document, for example, if there are two sessions identified *a* and *b*, two files for a document e.g. `doc.xml` will be made `doc.a.xml` and `doc.b.xml` respectively.

For each change a session merges all changes from other sessions when a document is being loaded (read-time merge).

See also:

Module `libearth.session` Isolate data from other installations

4.2.5 Stage

`Stage` is a unit of changes i.e. an atomic changes to be merged. It provides transactions for multi threaded environment. If there are simultaneous changes from other sessions or other transactions, these are automatically merged when the currently ongoing transaction is committed.

Stage also provides `Route`, a convenient interface to access documents. For example, you can read the subscription list by `stage.subscriptions`, and write it by `stage.subscriptions = new_subscriptions`. In the similar way you can read a feed by `stage.feeds[feed_id]`, and write it by `stage.feeds[feed_id] = new_feed`.

See also:

Module `libearth.stage` Staging updates and transactions

4.3 Libearth Changelog

4.3.1 Version 0.1.2

Released on January 19, 2014.

- XML elements in data files are written in canonical order. For example, `<title>` element of the feed was at the back before, but now is in front.
- Fixed a bug that `autodiscovery` raises `AttributeError` when the given HTML contains `<link>` to both *`application/atom+xml`* and *`application/rss+xml`*. [issue #40]
- Fill `<title>` to `<description>` if there's no `<title>` (`rss2`).
- Fill `<id>` to the feed URL if there's no `<id>` (`atom`).

4.3.2 Version 0.1.1

Released on January 2, 2014.

- Added a workaround for thread unsafety `time.strftime()` on CPython. See http://bugs.python.org/issue7980 as well. [issue #32]
- Fixed `UnicodeDecodeError` which is raised when a feed title contains any non-ASCII characters. [issue #34 by Jae-Myoung Yu]
- Now `libearth.parser.rss2` fills `Entry.updated_at` if it's not given. [issue #35]
- Fixed `TypeError` which is raised when any `DocumentElement` with `multiple Child` elements is passed to `validate()` function.
- Fixed the race condition of two `FileSystemRepository` objects creating the same directory. [issue #36 by klutzy]
- `parallel_map()` becomes to raise exceptions at the last, if any errored. [issue #38]

4.3.3 Version 0.1.0

Released on December 13, 2013. Initial alpha version.

CHAPTER 5

Open source

Libearth is an open source software written by Hong Minhee and the Earth Reader team. See also the complete list of contributors as well. Libearth is free software licensed under the terms of the GNU General Public License Version 2 or any later version, and you can find the code at GitHub repository:

```
$ git clone git://github.com/earthreader/libearth.git
```

If you find any bugs, please report them to our issue tracker. Pull requests are always welcome!

We discuss about libearth's development on IRC. Come `#earthreader` channel on Ozinger network. (We will make one on freenode as well soon!)

l