feedparser

Release 5.1.2

Oct 29, 2017

Contents

This documentation claims to describe the behavior of **Universal Feed Parser** 5.1.2. It does not claim to describe the behavior of any other version.

This documentation lives at http://packages.python.org/feedparser/. If you're reading it somewhere else, you may not have the latest version.

This documentation is provided by the author "as is" without any express or implied warranties. See *the documentation license* for more details.

CHAPTER 1

Basic Features

Introduction

Universal Feed Parser is a **Python** module for downloading and parsing syndicated feeds. It can handle RSS (Rich Site Summary) 0.90, Netscape RSS 0.91, Userland RSS 0.91, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, Atom 1.0, and CDF (Channel Definition Format) feeds. It also parses several popular extension modules, including Dublin Core and Apple's **iTunes** extensions.

To use **Universal Feed Parser**, you will need **Python** 2.4 or later (Python 3 is supported). **Universal Feed Parser** is not meant to run standalone; it is a module for you to use as part of a larger **Python** program.

Universal Feed Parser is easy to use; the module is self-contained in a single file, `feedparser.py`, and it has one primary public function, `parse`. `parse` takes a number of arguments, but only one is required, and it can be a URL (Uniform Resource Locator), a local filename, or a raw string containing feed data in any format.

Parsing a feed from a remote URL

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d['feed']['title']
u'Sample Feed'
```

The following example assumes you are on Windows, and that you have saved a feed at `c:\incoming\atom10.xml`.

Note: **Universal Feed Parser** works on any platform that can run **Python**; use the path syntax appropriate for your platform.

Parsing a feed from a local file

```
>>> import feedparser
>>> d = feedparser.parse(r'c:\incoming\atom10.xml')
>>> d['feed']['title']
u'Sample Feed'
```

Universal Feed Parser can also parse a feed in memory.

Parsing a feed from a string

```
>>> import feedparser
>>> rawdata = """<rss version="2.0">
<channel>
<title>Sample Feed</title>
</channel>
</rss>"""
>>> d = feedparser.parse(rawdata)
>>> d['feed']['title']
u'Sample Feed'
```

Values are returned as **Python** Unicode strings (except when they're not – see *Character Encoding Detection* for all the gory details).

See also:

Introduction to Python Unicode strings

Common RSS Elements

The most commonly used elements in RSS feeds (regardless of version) are title, link, description, publication date, and entry ID. The publication date comes from the pubDate element, and the entry ID comes from the guid element.

This sample RSS feed is at http://feedparser.org/docs/examples/rss20.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
<channel>
<title>Sample Feed</title>
<description>For documentation &lt;em&gt;only&lt;/em&gt;</description>
<link>http://example.org/</link>
<pubDate>Sat, 07 Sep 2002 00:00:01 GMT</pubDate>
<!-- other elements omitted from this example -->
<item>
<title>First entry title</title>
<link>http://example.org/entry/3</link>
<description>Watch out for &lt;span style="background-image:
url(javascript:window.location='http://example.org/')"&gt;nasty
tricks&lt;/span&gt;</description>
<pubDate>Thu, 05 Sep 2002 00:00:01 GMT</pubDate>
<guid>http://example.org/entry/3</guid>
<!-- other elements omitted from this example -->
</item>
</channel>
</rss>
```

The channel elements are available in `d.feed`.

Accessing Common Channel Elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.feed.title
u'Sample Feed'
>>> d.feed.link
u'http://example.org/'
>>> d.feed.description
u'For documentation <em>only</em>'
>>> d.feed.published
u'Sat, 07 Sep 2002 00:00:01 GMT'
>>> d.feed.published_parsed
(2002, 9, 7, 0, 0, 1, 5, 250, 0)
```

The items are available in `d.entries`, which is a list. You access items in the list in the same order in which they appear in the original feed, so the first item is available in `d.entries[0]`.

Accessing Common Item Elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.entries[0].title
u'First item title'
>>> d.entries[0].link
u'http://example.org/item/1'
>>> d.entries[0].description
u'Watch out for <span>nasty tricks</span>'
>>> d.entries[0].published
u'Thu, 05 Sep 2002 00:00:01 GMT'
>>> d.entries[0].published_parsed
(2002, 9, 5, 0, 0, 1, 3, 248, 0)
>>> d.entries[0].id
u'http://example.org/guid/1'
```

Tip: You can also access data from RSS feeds using Atom terminology. See *Content Normalization* for details.

Common Atom Elements

Atom feeds generally contain more information than RSS feeds (because more elements are required), but the most commonly used elements are still title, link, subtitle/description, various dates, and ID.

This sample Atom feed is at http://feedparser.org/docs/examples/atom10.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
xml:base="http://example.org/"
xml:lang="en">
<title type="text">Sample Feed</title>
```

```
<subtitle type="html">
For documentation &lt;em&gt;only&lt;/em&gt;
</subtitle>
<link rel="alternate" href="/"/>
<link rel="self"
type="application/atom+xml"
href="http://www.example.org/atom10.xml"/>
<rights type="html">
&lt;p>Copyright 2005, Mark Pilgrim&lt;/p>&lt;
</rights>
<id>tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml</id>
<generator
uri="http://example.org/generator/"
version="4.0">
Sample Toolkit
</generator>
<updated>2005-11-09T11:56:34Z</updated>
<entry>
<title>First entry title</title>
<link rel="alternate"
href="/entry/3"/>
<link rel="related"
type="text/html"
href="http://search.example.com/"/>
<link rel="via"
type="text/html"
href="http://toby.example.com/examples/atom10"/>
<link rel="enclosure"
type="video/mpeg4"
href="http://www.example.com/movie.mp4"
length="42301"/>
<id>tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml:3</id>
<published>2005-11-09T00:23:47Z</published>
<updated>2005-11-09T11:56:34Z</updated>
<summary type="text/plain" mode="escaped">Watch out for nasty tricks</summary>
<content type="application/xhtml+xml" mode="xml"
xml:base="http://example.org/entry/3" xml:lang="en-US">
<div xmlns="http://www.w3.org/1999/xhtml">Watch out for
<span style="background: url(javascript:window.location='http://example.org/')">
nasty tricks</span></div>
</content>
</entry>
</feed>
```

The feed elements are available in `d.feed`.

Accessing Common Feed Elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.feed.title
u'Sample feed'
>>> d.feed.link
u'http://example.org/'
>>> d.feed.subtitle
u'For documentation <em>only</em>'
```

```
>>> d.feed.updated
u'2005-11-09T11:56:34Z'
>>> d.feed.updated_parsed
(2005, 11, 9, 11, 56, 34, 2, 313, 0)
>>> d.feed.id
u'tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml'
```

Entries are available in `d.entries`, which is a list. You access entries in the order in which they appear in the original feed, so the first entry is `d.entries[0]`.

Accessing Common Entry Elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.entries[0].title
u'First entry title'
>>> d.entries[0].link
u'http://example.org/entry/3
>>> d.entries[0].id
u'tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml:3'
>>> d.entries[0].published
u'2005-11-09T00:23:47Z'
>>> d.entries[0].published_parsed
(2005, 11, 9, 0, 23, 47, 2, 313, 0)
>>> d.entries[0].updated
u'2005-11-09T11:56:34Z'
>>> d.entries[0].updated_parsed
(2005, 11, 9, 11, 56, 34, 2, 313, 0)
>>> d.entries[0].summary
u'Watch out for nasty tricks'
>>> d.entries[0].content
[{'type': u'application/xhtml+xml',
'base': u'http://example.org/entry/3',
'language': u'en-US',
'value': u'<div>Watch out for <span>nasty tricks</span></div>'}]
```

Note: The parsed summary and content are not the same as they appear in the original feed. The original elements contained dangerous HTML (HyperText Markup Language) markup which was sanitized. See *Sanitization* for details.

Because Atom entries can have more than one content element, `d.entries[0].content` is a list of dictionaries. Each dictionary contains metadata about a single content element. The two most important values in the dictionary are the content type, in `d.entries[0].content[0].type`, and the actual content value, in `d.entries[0].content[0].value`.

You can get this level of detail on other Atom elements too.

Getting Detailed Information on Atom Elements

Several Atom elements share the Atom content model: title, subtitle, rights, summary, and of course content. (Atom 0.3 also had an info element which shared this content model.) **`Universal Feed Parser`** captures all relevant metadata about these elements, most importantly the content type and the value itself.

Detailed Information on Feed Elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.feed.title_detail
{'type': u'text/plain',
'base': u'http://example.org/',
'language': u'en',
'value': u'Sample Feed'}
>>> d.feed.subtitle_detail
{'type': u'text/html',
'base': u'http://example.org/',
'language': u'en',
'value': u'For documentation <em>only</em>'}
>>> d.feed.rights_detail
{'type': u'text/html',
'base': u'http://example.org/',
'language': u'en',
'value': u'<p>Copyright 2004, Mark Pilgrim</p>'}
>>> d.entries[0].title_detail
{'type': 'text/plain',
'base': u'http://example.org/',
'language': u'en',
'value': u'First entry title'}
>>> d.entries[0].summary_detail
{'type': u'text/plain',
'base': u'http://example.org/',
'language': u'en',
'value': u'Watch out for nasty tricks'}
>>> len(d.entries[0].content)
1
>>> d.entries[0].content[0]
{'type': u'application/xhtml+xml',
'base': u'http://example.org/entry/3',
'language': u'en-US'
'value': u'<div>Watch out for <span> nasty tricks</span></div>'}
```

Uncommon RSS Elements

These elements are less common, but are useful for niche applications and may be present in any RSS feed.

An RSS feed can specify a small image which some aggregators display as a logo.

Accessing feed image

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.feed.image
{'title': u'Example banner',
'href': u'http://example.org/banner.png',
'width': 80,
'height': 15,
'link': u'http://example.org/'}
```

Feeds and entries can be assigned to multiple categories, and in some versions of RSS, categories can be associated with a "domain". Both are free-form strings. For historical reasons, **Universal Feed Parser** makes multiple categories available as a list of tuples, rather than a list of dictionaries.

Accessing multiple categories

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.feed.categories
[(u'Syndic8', u'1024'),
(u'dmoz', 'Top/Society/People/Personal_Homepages/P/')]
```

Each item in an RSS feed can have an "enclosure", a delightful misnomer that is simply a link to an external file (usually a music or video file, but any type of file can be "enclosed"). Once rare, this element has recently gained popularity due to the rise of podcasting. Some clients (such as Apple's **iTunes**) may automatically download enclosures; others (such as the web-based Bloglines) may simply render each enclosure as a link.

The RSS specification states that there can be at most one enclosure per item. However, Atom entries may contain more than one enclosure per entry, so **Universal Feed Parser** captures all of them and makes them available as a list.

Accessing enclosures

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> e = d.entries[0]
>>> len(e.enclosures)
1
>>> e.enclosures[0]
{'type': u'audio/mpeg',
'length': u'1069871',
'href': u'http://example.org/audio/demo.mp3'}
```

Accessing feed cloud

No one is quite sure what a cloud is.

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.feed.cloud
{'domain': u'rpc.example.com',
'port': u'80',
'path': u'/RPC2',
'registerprocedure': u'pingMe',
'protocol': u'soap'}
```

Note: For more examples of accessing RSS elements, see the annotated examples: *RSS 1.0*, *RSS 2.0*, and *RSS 2.0 with Namespaces*.

Uncommon Atom Elements

These elements are less common, but are useful for niche applications and may be present in any Atom feed.

Besides an author, each Atom feed or entry can have an arbitrary number of contributors. **Universal Feed Parser** makes these available as a list.

Accessing contributors

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> e = d.entries[0]
>>> len(e.contributors)
2
>>> e.contributors[0]
{'name': u'Joe',
'href': u'http://example.org/joe/',
'email': u'joe@example.org'}
>>> e.contributors[1]
{'name': u'Sam',
'href': u'http://example.org/sam/',
'email': u'sam@example.org'}
```

Besides an alternate link, each Atom feed or entry can have an arbitrary number of other links. Each link is distinguished by its type attribute, which is a MIME-style content type, and its rel attribute.

Accessing multiple links

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> e = d.entries[0]
>>> len(e.links)
4
>>> e.links[0]
{'rel': u'alternate',
'type': u'text/html',
'href': u'http://example.org/entry/3'}
>>> e.links[1]
{'rel': u'related',
'type': u'text/html',
'href': u'http://search.example.com/'}
>>> e.links[2]
{'rel': u'via',
'type': u'text/html',
'href': u'http://toby.example.com/examples/atom10'}
>>> e.links[3]
{'rel': u'enclosure',
'type': u'video/mpeg4',
'href': u'http://www.example.com/movie.mp4',
'length': u'42301'}
```

Note: For more examples of accessing Atom elements, see the annotated examples *Atom 1.0* and *Atom 0.3*.

Testing for Existence

Feeds in the real world may be missing elements, even elements that are required by the specification. You should always test for the existence of an element before getting its value. Never assume an element is present.

Use standard **Python** dictionary functions such as `has_key` to test whether an element exists.

Testing if elements are present

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.feed.has_key('title')
True
>>> d.feed.has_key('ttl')
False
>>> d.feed.get('title', 'No title')
u'Sample feed'
>>> d.feed.get('ttl', 60)
60
```

CHAPTER 2

Advanced Features

Date Parsing

Different feed types and versions use wildly different date formats. **`Universal Feed Parser`** will attempt to auto-detect the date format used in any date element, and parse it into a standard **`Python`** 9-tuple, as documented in the Python time module.

The following elements are parsed as dates:

- *feed.updated* is parsed into *feed.updated_parsed*.
- *entries[i].published* is parsed into *entries[i].published_parsed*.
- *entries[i].updated* is parsed into *entries[i].updated_parsed*.
- *entries[i].created* is parsed into *entries[i].created_parsed*.
- *entries[i].expired* is parsed into *entries[i].expired_parsed*.

History of Date Formats

Here is a brief history of feed date formats:

- CDF states that all date values must conform to ISO 8601:1988. ISO 8601:1988 is not a freely available specification, but a brief (non-normative) description of the date formats it describes is available here: ISO 8601:1988 Date/Time Representations.
- RSS 0.90 has no date elements.
- Netscape RSS 0.91 does not specify a date format, but examples within the specification show RFC (Request For Comments) 822-style dates with 4-digit years.
- Userland RSS 0.91 states, "All date-times in RSS conform to the Date and Time Specification of RFC 822." RFC 822 mandates 2-digit years; it does not allow 4-digit years.
- RSS 1.0 states that all date elements must conform to W3CDTF, which is a profile of ISO 8601:1988.

- RSS 2.0 states, "All date-times in RSS conform to the Date and Time Specification of RFC 822, with the exception that the year may be expressed with two characters or four characters (four preferred)."
- Atom 0.3 states that all date elements must conform to W3CDTF.
- Atom 1.0 states that all date elements "MUST conform to the date-time production in RFC 3339. In addition, an uppercase T character MUST be used to separate date and time, and an uppercase Z character MUST be present in the absence of a numeric time zone offset."

Recognized Date Formats

Here is a representative list of the formats that **`Universal Feed Parser`** can recognize in any date element:

Recognized Date Formats

Description	Example	Parsed Value
valid RFC 822 (2-digit year)	Thu, 01 Jan 04 19:48:21 GMT	(2004, 1, 1, 19, 48, 21, 3, 1, 0)
valid RFC 822 (4-digit year)	Thu, 01 Jan 2004 19:48:21 GMT	(2004, 1, 1, 19, 48, 21, 3, 1, 0)
invalid RFC 822 (no time)	01 Jan 2004	(2004, 1, 1, 0, 0, 0, 3, 1, 0)
invalid RFC 822 (no seconds)	01 Jan 2004 00:00 GMT	(2004, 1, 1, 0, 0, 0, 3, 1, 0)
valid W3CDTF (numeric timezone)	2003-12-31T10:14:55-08:00	(2003, 12, 31, 18, 14, 55, 2, 365, 0)
valid W3CDTF (UTC timezone)	2003-12-31T10:14:55Z	(2003, 12, 31, 10, 14, 55, 2, 365, 0)
valid W3CDTF (yyyy)	2003	(2003, 1, 1, 0, 0, 0, 2, 1, 0)
valid W3CDTF (yyyy-mm)	2003-12	(2003, 12, 1, 0, 0, 0, 0, 335, 0)
valid W3CDTF (yyyy-mm-dd)	2003-12-31	(2003, 12, 31, 0, 0, 0, 2, 365, 0)
valid ISO 8601 (yyyymmdd)	20031231	(2003, 12, 31, 0, 0, 0, 2, 365, 0)
valid ISO 8601 (-yy-mm)	-03-12	(2003, 12, 1, 0, 0, 0, 0, 335, 0)
valid ISO 8601 (-yymm)	-0312	(2003, 12, 1, 0, 0, 0, 0, 335, 0)
valid ISO 8601 (-yy-mm-dd)	-03-12-31	(2003, 12, 31, 0, 0, 0, 2, 365, 0)
valid ISO 8601 (yymmdd)	031231	(2003, 12, 31, 0, 0, 0, 2, 365, 0)
valid ISO 8601 (yyyy-o)	2003-335	(2003, 12, 1, 0, 0, 0, 0, 335, 0)
valid ISO 8601 (yyo)	03335	(2003, 12, 1, 0, 0, 0, 0, 335, 0)
valid asctime	Sun Jan 4 16:29:06 PST 2004	(2004, 1, 5, 0, 29, 6, 0, 5, 0)
bogus RFC 822 (invalid day/month)	Thu, 31 Jun 2004 19:48:21 GMT	(2004, 7, 1, 19, 48, 21, 3, 183, 0)
bogus RFC 822 (invalid month)	Mon, 26 January 2004 16:31:00 EST	(2004, 1, 26, 21, 31, 0, 0, 26, 0)
bogus RFC 822 (invalid timezone)	Mon, 26 Jan 2004 16:31:00 ET	(2004, 1, 26, 21, 31, 0, 0, 26, 0)
bogus W3CDTF (invalid hour)	2003-12-31T25:14:55Z	(2004, 1, 1, 1, 14, 55, 3, 1, 0)
bogus W3CDTF (invalid minute)	2003-12-31T10:61:55Z	(2003, 12, 31, 11, 1, 55, 2, 365, 0)
bogus W3CDTF (invalid second)	2003-12-31T10:14:61Z	(2003, 12, 31, 10, 15, 1, 2, 365, 0)
bogus (MSSQL)	2004-07-08 23:56:58.0	(2004, 7, 8, 14, 56, 58, 3, 190, 0)
bogus (MSSQL-ish, without fractional second)	2004-07-08 23:56:58	(2004, 7, 8, 14, 56, 58, 3, 190, 0)
bogus (Korean)	2004-05-25 11:23:17	(2004, 5, 25, 14, 23, 17, 1, 146, 0)
bogus (Greek)	Κυρ, 11 Ιολ 2004 12:00:00 EST	(2004, 7, 11, 17, 0, 0, 6, 193, 0)
bogus (Hungarian)	július-13T9:15-05:00	(2004, 7, 13, 14, 15, 0, 1, 195, 0)

Universal Feed Parser recognizes all character-based timezone abbreviations defined in RFC 822. In addition, **Universal Feed Parser** recognizes the following invalid timezones:

- AT is treated as AST
- ET is treated as EST
- CT is treated as CST
- MT is treated as MST
- PT is treated as PST

Supporting Additional Date Formats

Universal Feed Parser supports many different date formats, but there are probably many more in the wild that are still unsupported. If you find other date formats, you can support them by registering them with registerDateHandler. It takes a single argument, a callback function. The callback function should take a single argument, a string, and return a single value, a 9-tuple **Python** date in UTC.

Registering a third-party date handler

```
import feedparser
import re

_my_date_pattern = re.compile(
    r'(\d{,2})/(\d{,2})/(\d{4}) (\d{,2}):(\d{2}):(\d{2})')

def myDateHandler(aDateString):
    """parse a UTC date in MM/DD/YYYY HH:MM:SS format"""
    month, day, year, hour, minute, second = \
        _my_date_pattern.search(aDateString).groups()
    return (int(year), int(month), int(day), \
        int(hour), int(minute), int(second), 0, 0, 0)

feedparser.registerDateHandler(myDateHandler)
d = feedparser.parse(...)
```

Your newly-registered date handler will be tried before all the other date handlers built into **Universal Feed Parser**. (More specifically, all date handlers are tried in "last in, first out" order; i.e. the last handler to be registered is the first one tried, and so on in reverse order of registration.)

If your date handler returns None, or anything other than a **Python** 9-tuple date, or raises an exception of any kind, the error will be silently ignored and the other registered date handlers will be tried in order. If no date handlers succeed, then the date is not parsed, and the *_parsed value will not be present in the results dictionary. The original date string will still be available in the appropriate element in the results dictionary.

Tip: If you write a new date handler, you are encouraged (but not required) to submit a patch so it can be integrated into the next version of **Universal Feed Parser**.

Sanitization

Most feeds embed HTML markup within feed elements. Some feeds even embed other types of markup, such as SVG (Scalable Vector Graphics) or MATHML (Mathematical Markup Language). Since many feed aggregators use a web browser (or browser component) to display content, **`Universal Feed Parser`** sanitizes embedded markup to remove things that could pose security risks.

These elements are sanitized by default:

- *entries[i].content*
- *entries[i].summary*
- *entries[i].title*
- *feed.info*
- *feed.rights*
- *feed.subtitle*
- *feed.title*

Note: If the content is declared to be (or is determined to be) *`text/plain`*, it will not be sanitized. This is to avoid data loss. It is recommended that you check the content type in e.g. `entries[i].summary_detail.type`. If it is *`text/plain`* then it has not been sanitized (and you should perform HTML escaping before rendering the content).

HTML Sanitization

The following HTML elements are allowed by default (all others are stripped):

- a
- abbr
- acronym
- address
- area
- article
- aside
- audio
- b
- big
- blockquote
- br
- button
- canvas
- caption
- center
- cite
- code
- col
- colgroup
- command
- datagrid
- datalist

- dd
- del
- details
- dfn
- dialog
- dir
- div
- dl
- dt
- em
- event-source
- fieldset
- figure
- font
- footer
- form
- h1
- h2
- h3
- h4
- h5
- h6
- header
- hr
- i
- img
- input
- ins
- kbd
- keygen
- label
- legend
- li
- m
- map
- menu
- meter
- multicol
- nav
- nextid
- noscript
- ol
- optgroup
- option
- output
- p
- pre
- progress
- q
- s
- samp
- section
- select
- small

- sound
- source
- spacer
- span
- strike
- strong
- sub
- sup
- table
- tbody
- td
- textarea
- tfoot
- th
- thead
- time
- tr
- tt
- u
- ul
- var
- video

The following HTML attributes are allowed by default (all others are stripped):

- abbr
- accept
- accept-charset
- accesskey
- action
- align
- alt
- autocomplete
- autofocus
- autoplay
- axis
- background
- balance
- bgcolor
- bgproperties
- border
- bordercolor
- bordercolordark
- bordercolorlight
- bottompadding
- cellpadding
- cellspacing
- ch
- challenge
- char
- charoff
- charset
- checked
- choff
- cite

- class
- clear
- color
- cols
- colspan
- compact
- contenteditable
- coords
- data
- datafld
- datapagesize
- datasrc
- datetime
- default
- delay
- dir
- disabled
- draggable
- dynsrc
- enctype
- end
- face
- for
- form
- frame
- galleryimg
- gutter
- headers
- height
- hidden
- hidefocus
- high
- href
- hreflang
- hspace
- icon
- id
- inputmode
- ismap
- keytype
- label
- lang
- leftspacing
- list
- longdesc
- loop
- loopcount
- loopend
- loopstart
- low
- lowsrc
- max
- maxlength
- media

- method
- min
- multiple
- name
- nohref
- noshade
- nowrap
- open
- optimum
- pattern
- ping
- point-size
- pqg
- prompt
- radiogroup
- readonly
- rel
- repeat-max
- repeat-min
- replace
- required
- rev
- rightspacing
- rows
- rowspan
- rules
- scope
- selected
- shape
- size
- span
- src
- start
- step
- summary
- suppress
- tabindex
- target
- template
- title
- toppadding
- type
- unselectable
- urn
- usemap
- valign
- value
- variable
- volume
- vrml
- vspace
- width
- wrap
- xml:lang

SVG Sanitization

The following SVG elements are allowed by default (all others are stripped):

- a
- animate
- animateColor
- animateMotion
- animateTransform
- circle
- defs
- desc
- ellipse
- font-face
- font-face-name
- font-face-src
- foreignObject
- g
- glyph
- hkern
- line
- linearGradient
- marker
- metadata
- missing-glyph
- mpath
- path
- polygon
- polyline
- radialGradient
- rect
- set
- stop
- svg
- switch
- text
- title
- tspan
- use

The following SVG attributes are allowed by default (all others are stripped):

- accent-height
- accumulate
- additive
- alphabetic
- arabic-form
- ascent
- attributeName
- attributeType
- baseProfile
- bbox
- begin
- by
- calcMode

- cap-height
- class
- color
- color-rendering
- content
- cx
- cy
- d
- descent
- display
- dur
- dx
- dy
- end
- fill
- fill-opacity
- fill-rule
- font-family
- font-size
- font-stretch
- font-style
- font-variant
- font-weight
- from
- fx
- fy
- g1
- g2
- glyph-name
- gradientUnits
- hanging
- height
- horiz-adv-x
- horiz-origin-x
- id
- ideographic
- k
- keyPoints
- keySplines
- keyTimes
- lang
- marker-end
- marker-mid
- marker-start
- markerHeight
- markerUnits
- markerWidth
- mathematical
- max
- min
- name
- offset
- opacity
- orient

- origin
- overline-position
- overline-thickness
- panose-1
- path
- pathLength
- points
- preserveAspectRatio
- r
- refX
- refY
- repeatCount
- repeatDur
- requiredExtensions
- requiredFeatures
- restart
- rotate
- rx
- ry
- slope
- stemh
- stemv
- stop-color
- stop-opacity
- strikethrough-position
- strikethrough-thickness
- stroke
- stroke-dasharray
- stroke-dashoffset
- stroke-linecap
- stroke-linejoin
- stroke-miterlimit
- stroke-opacity
- stroke-width
- systemLanguage
- target
- text-anchor
- to
- transform
- type
- u1
- u2
- underline-position
- underline-thickness
- unicode
- unicode-range
- units-per-em
- values
- version
- viewBox
- visibility
- width
- widths
- x

- x-height
- x1
- x2
- xlink:actuate
- xlink:arcrole
- xlink:href
- xlink:role
- xlink:show
- xlink:title
- xlink:type
- xml:base
- xml:lang
- xml:space
- xmlns
- xmlns:xlink
- y
- y1
- y2
- zoomAndPan

MATHML Sanitization

The following MATHML elements are allowed by default (all others are stripped):

- annotation
- annotation-xml
- maction
- math
- merror
- mfenced
- mfrac
- mi
- mmultiscripts
- mn
- mo
- mover
- mpadded
- mphantom
- mprescripts
- mroot
- mrow
- mspace
- msqrt
- mstyle
- msub
- msubsup
- msup
- mtable
- mtd
- mtext
- mtr
- munder
- munderover
- none

- semantics

The following MATHML attributes are allowed by default (all others are stripped):

- actiontype
- align
- close
- columnalign
- columnlines
- columnspacing
- columnspan
- depth
- display
- displaystyle
- encoding
- equalcolumns
- equalrows
- fence
- fontstyle
- fontweight
- frame
- height
- linethickness
- lspace
- mathbackground
- mathcolor
- mathvariant
- maxsize
- minsize
- open
- other
- rowalign
- rowlines
- rowspacing
- rowspan
- rspace
- scriptlevel
- selection
- separator
- separators
- stretchy
- width
- xlink:href
- xlink:show
- xlink:type
- xmlns
- xmlns:xlink

CSS (Cascading Style Sheets) Sanitization

The following CSS properties are allowed by default in style attributes (all others are stripped):

- azimuth
- background-color
- border-bottom-color

- border-collapse
- border-color
- border-left-color
- border-right-color
- border-top-color
- clear
- color
- cursor
- direction
- display
- elevation
- float
- font
- font-family
- font-size
- font-style
- font-variant
- font-weight
- height
- letter-spacing
- line-height
- overflow
- pause
- pause-after
- pause-before
- pitch
- pitch-range
- richness
- speak
- speak-header
- speak-numeral
- speak-punctuation
- speech-rate
- stress
- text-align
- text-decoration
- text-indent
- unicode-bidi
- vertical-align
- voice-family
- volume
- white-space
- width

Note: Not all possible CSS values are allowed for these properties. The allowable values are restricted by a whitelist and a regular expression that allows color values and lengths. URI (Uniform Resource Identifier)s are not allowed, to prevent platypus attacks. See the _HTMLSanitizer class for more details.

Whitelist, Don't Blacklist

I am often asked why **`Universal Feed Parser`** is so hard-assed about HTML and CSS sanitizing. To illustrate the problem, here is an incomplete list of potentially dangerous HTML tags and attributes:

- script, which can contain malicious script
- applet, embed, and object, which can automatically download and execute malicious code
- meta, which can contain malicious redirects
- onload, onunload, and all other on* attributes, which can contain malicious script
- style, link, and the style attribute, which can contain malicious script

style? Yes, style. CSS definitions can contain executable code.

Embedding Javascript in CSS

This sample is taken from http://feedparser.org/docs/examples/rss20.xml:

```
<description>Watch out for
&lt;span style="background: url(javascript:window.location='http://example.org/')"&gt;
nasty tricks&lt;/span&gt;</description>
```

This sample is more advanced, and does not contain the keyword javascript: that many naive HTML sanitizers scan for:

```
<description>Watch out for
&lt;span style="any: expression(window.location='http://example.org/')"&gt;
nasty tricks&lt;/span&gt;</description>
```

Internet Explorer for Windows will execute the Javascript in both of these examples.

Now consider that in HTML, attribute values may be entity-encoded in several different ways.

Embedding encoded Javascript in CSS

To a browser, this:

```
<span style="any: expression(window.location='http://example.org/')">
```

is the same as this (without the line breaks):

```
<span style="&#97;&#110;&#121;&#58;&#32;&#101;&#120;&#112;&#114;&#101;
&#115;&#115;&#105;&#111;&#110;&#40;&#119;&#105;&#110;&#100;&#111;&#119;
&#46;&#108;&#111;&#99;&#97;&#116;&#105;&#111;&#110;&#61;&#39;&#104;
&#116;&#116;&#112;&#58;&#47;&#47;&#101;&#120;&#97;&#109;&#112;&#108;
&#101;&#46;&#111;&#114;&#103;&#47;&#39;&#41;">
```

which is the same as this (without the line breaks):

```
<span style="&#x61;&#x6e;&#x79;&#x3a;&#x20;&#x65;&#x78;&#x70;&#x72;
&#x65;&#x73;&#x73;&#x69;&#x6f;&#x6e;&#x28;&#x77;&#x69;&#x6e;
&#x64;&#x6f;&#x77;&#x2e;&#x6c;&#x6f;&#x63;&#x61;&#x74;&#x69;
&#x6f;&#x6e;&#x3d;&#x27;&#x68;&#x74;&#x74;&#x70;&#x3a;&#x2f;
&#x2f;&#x65;&#x78;&#x61;&#x6d;&#x70;&#x6c;&#x65;&#x2e;&#x6f;
&#x72;&#x67;&#x2f;&#x27;&#x29;">
```

And so on, plus several other variations, plus every combination of every variation.

The more I investigate, the more cases I find where Internet Explorer for Windows will treat seemingly innocuous markup as code and blithely execute it. This is why **`Universal Feed Parser`** uses a whitelist and not a blacklist. I am reasonably confident that none of the elements or attributes on the whitelist are security risks. I am not at all

confident about elements or attributes that I have not explicitly investigated. And I have no confidence at all in my ability to detect strings within attribute values that Internet Explorer for Windows will treat as executable code.

See also:

How to consume RSS safely Explains the platypus attack.

Content Normalization

Universal Feed Parser can parse many different types of feeds: Atom, CDF, and nine different versions of RSS. You should not be forced to learn the differences between these formats. **Universal Feed Parser** does its best to ensure that you can treat all feeds the same way, regardless of format or version.

You can access the basic elements of an Atom feed using RSS terminology.

Accessing an Atom feed as an RSS feed

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d['channel']['title']
u'Sample Feed'
>>> d['channel']['link']
u'http://example.org/'
>>> d['channel']['description']
u'For documentation <em>only</em>
>>> len(d['items'])
1
>>> e = d['items'][0]
>>> e['title']
u'First entry title'
>>> e['link']
u'http://example.org/entry/3'
>>> e['description']
u'Watch out for nasty tricks'
>>> e['author']
u'Mark Pilgrim (mark@example.org)'
```

The same thing works in reverse: you can access RSS feeds as if they were Atom feeds.

Accessing an RSS feed as an Atom feed

```
>>> import feedparser
>>> d = feedparser.parse(' http://feedparser.org/docs/examples/rss20.xml')
>>> d.feed.subtitle_detail
{'type': 'text/html',
'base': 'http://feedparser.org/docs/examples/rss20.xml',
'language': None,
'value': u'For documentation <em>only</em>'}
>>> len(d.entries)
1
>>> e = d.entries[0]
>>> e.links
[{'rel': 'alternate',
'type': 'text/html',
```

```
'href': u'http://example.org/item/1'}]
>>> e.summary_detail
{'type': 'text/html',
'base': 'http://feedparser.org/docs/examples/rss20.xml',
'language': u'en',
'value': u'Watch out for <span>nasty tricks</span>'}
>>> e.updated_parsed
(2002, 9, 5, 0, 0, 1, 3, 248, 0)
```

Note: For more examples of how **Universal Feed Parser** normalizes content from different formats, see *Annotated Examples*.

Namespace Handling

Universal Feed Parser attempts to expose all possible data in feeds, including elements in extension namespaces.

Some common namespaced elements are mapped to core elements. For further information about these mappings, see *Reference*.

Other namespaced elements are available as `prefixelement`.

The namespaces defined in the feed are available in the parsed results as `namespaces`, a dictionary of {prefix: namespaceURI}. If the feed defines a default namespace, it is listed as `namespaces['']`.

Accessing namespaced elements

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/prism.rdf')
>>> d.feed.prism_issn
u'0028-0836'
>>> d.namespaces
{'': u'http://purl.org/rss/1.0/',
'prism': u'http://prismstandard.org/namespaces/1.2/basic/',
'rdf': u'http://www.w3.org/1999/02/22-rdf-syntax-ns#'}
```

The prefix used to construct the variable name is not guaranteed to be the same as the prefix of the namespaced element in the original feed. If **Universal Feed Parser** recognizes the namespace, it will use the namespace's preferred prefix to construct the variable name. It will also list the namespace in the `namespaces` dictionary using the namespace's preferred prefix.

In the previous example, the namespace (http://prismstandard.org/namespaces/1.2/basic/) was defined with the namespace's preferred prefix (prism), so the prism:issn element was accessible as the variable `d.feed.prism_issn`. However, if the namespace is defined with a non-standard prefix, **Universal Feed Parser** will still construct the variable name using the preferred prefix, *not* the actual prefix that is used in the feed.

This will become clear with an example.

Accessing namespaced elements with non-standard prefixes

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/nonstandard_prefix.rdf')
>>> d.feed.prism_issn
u'0028-0836'
>>> d.feed.foo_issn
Traceback (most recent call last):
File "<stdin>", line 1, in ?
File "feedparser.py", line 158, in __getattr__
raise AttributeError, "object has no attribute '%s'" % key
AttributeError: object has no attribute 'foo_issn'
>>> d.namespaces
{'': u'http://purl.org/rss/1.0/',
'prism': u'http://prismstandard.org/namespaces/1.2/basic/',
'rdf': u'http://www.w3.org/1999/02/22-rdf-syntax-ns#'}
```

This is the complete list of namespaces that **Universal Feed Parser** recognizes and uses to construct the variable names for data in these namespaces:

Prefix	Namespace
admin	http://webns.net/mvcb/
ag	http://purl.org/rss/1.0/modules/aggregation/
annotate	http://purl.org/rss/1.0/modules/annotate/
audio	http://media.tangent.org/rss/1.0/
blogChannel	http://backend.userland.com/blogChannelModule
cc	http://web.resource.org/cc/
co	http://purl.org/rss/1.0/modules/company
content	http://purl.org/rss/1.0/modules/content/
cp	http://my.theinfo.org/changed/1.0/rss/
creativeCommons	http://backend.userland.com/creativeCommonsRssModule
dc	http://purl.org/dc/elements/1.1/
dcterms	http://purl.org/dc/terms/
email	http://purl.org/rss/1.0/modules/email/
ev	http://purl.org/rss/1.0/modules/event/
feedburner	http://rssnamespace.org/feedburner/ext/1.0
fm	http://freshmeat.net/rss/fm/
foaf	http://xmlns.com/foaf/0.1/
geo	http://www.w3.org/2003/01/geo/wgs84_pos#
icbm	http://postneo.com/icbm/
image	http://purl.org/rss/1.0/modules/image/
itunes	http://example.com/DTDs/PodCast-1.0.dtd
itunes	http://www.itunes.com/DTDs/PodCast-1.0.dtd
l	http://purl.org/rss/1.0/modules/link/
media	http://search.yahoo.com/mrss
pingback	http://madskills.com/public/xml/rss/module/pingback/
prism	http://prismstandard.org/namespaces/1.2/basic/
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
ref	http://purl.org/rss/1.0/modules/reference/
reqv	http://purl.org/rss/1.0/modules/richequiv/
search	http://purl.org/rss/1.0/modules/search/
slash	http://purl.org/rss/1.0/modules/slash/

Continued on next page

Table 2.1 – continued from previous page

Prefix	Namespace
soap	http://schemas.xmlsoap.org/soap/envelope/
ss	http://purl.org/rss/1.0/modules/servicestatus/
str	http://hacks.benhammersley.com/rss/streaming/
sub	http://purl.org/rss/1.0/modules/subscription/
sy	http://purl.org/rss/1.0/modules/syndication/
szf	http://schemas.pocketsoap.com/rss/myDescModule/
taxo	http://purl.org/rss/1.0/modules/taxonomy/
thr	http://purl.org/rss/1.0/modules/threading/
ti	http://purl.org/rss/1.0/modules/textinput/
trackback	http://madskills.com/public/xml/rss/module/trackback/
wfw	http://wellformedweb.org/CommentAPI/
wiki	http://purl.org/rss/1.0/modules/wiki/
xhtml	http://www.w3.org/1999/xhtml
xlink	http://www.w3.org/1999/xlink
xml	http://www.w3.org/XML/1998/namespace

Note: **`Universal Feed Parser`** treats namespaces as case-insensitive to match the behavior of certain versions of **`iTunes`**.

Warning: Data from namespaced elements is not *sanitized* (even if it contains HTML markup).

Relative Link Resolution

Many feed elements and attributes are URIs. **`Universal Feed Parser`** resolves relative URIs according to the XML:Base specification. We'll see how that works in a minute, but first let's talk about which values are treated as URIs.

Which Values Are URIs

These feed elements are treated as URIs, and resolved if they are relative:

- *entries[i].author_detail.href*
- *entries[i].comments*
- *entries[i].contributors[j].href*
- *entries[i].enclosures[j].href*
- *entries[i].id*
- *entries[i].license*
- *entries[i].link*
- *entries[i].links[j].href*
- *entries[i].publisher_detail.href*
- *entries[i].source.author_detail.href*

- *entries[i].source.contributors[j].href*
- *entries[i].source.links[j].href*
- *feed.author_detail.href*
- *feed.contributors[i].href*
- *feed.docs*
- *feed.generator_detail.href*
- *feed.id*
- *feed.image.href*
- *feed.image.link*
- *feed.license*
- *feed.link*
- *feed.links[i].href*
- *feed.publisher_detail.href*
- *feed.textinput.link*

In addition, several feed elements may contain HTML or XHTML (Extensible HyperText Markup Language) markup. Certain elements and attributes in HTML can be relative URIs, and **Universal Feed Parser** will resolve these URIs according to the same rules as the feed elements listed above.

These feed elements may contain HTML or XHTML markup. In Atom feeds, whether these elements are treated as HTML depends on the value of the type attribute. In RSS feeds, these values are always treated as HTML.

- *entries[i].content[j].value*
- *entries[i].summary* (*entries[i].summary_detail.value*)
- *entries[i].title* (*entries[i].title_detail.value*)
- *feed.info* (*feed.info_detail.value*)
- *feed.rights* (*feed.rights_detail.value*)
- *feed.subtitle* (*feed.subtitle_detail.value*)
- *feed.title* (*feed.title_detail.value*)

When any of these feed elements contains HTML or XHTML markup, the following HTML elements are treated as URIs and are resolved if they are relative:

- <a href="...">
- <applet codebase="...">
- <area href="...">
- <blockquote cite="...">
- <body background="...">
- <del cite="...">
- <form action="...">
- <frame longdesc="...">
- <frame src="...">

- <head profile="...">
- <iframe longdesc="...">
- <iframe src="...">
- <img longdesc="...">
- <img src="...">
- <img usemap="...">
- <input src="...">
- <input usemap="...">
- <ins cite="...">
- <link href="...">
- <object classid="...">
- <object codebase="...">
- <object data="...">
- <object usemap="...">
- <q cite="...">
- <script src="...">

How Relative URIs Are Resolved

Universal Feed Parser resolves relative URIs according to the XML:Base specification. This defines a hierarchical inheritance system, where one element can define the base URI for itself and all of its child elements, using an xml:base attribute. A child element can then override its parent's base URI by redeclaring xml:base to a different value.

If no xml:base is specified, the feed has a default base URI defined in the Content-Location HTTP (Hypertext Transfer Protocol) header.

If no Content-Location HTTP header is present, the URL used to retrieve the feed itself is the default base URI for all relative links within the feed. If the feed was retrieved via an HTTP redirect (any HTTP 3xx status code), then the final URL of the feed is the default base URI.

For example, an xml:base on the root-level element sets the base URI for all URIs in the feed.

xml:base on the root-level element

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/base.xml")
>>> d.feed.link
u'http://example.org/index.html'
>>> d.feed.generator_detail.href
u'http://example.org/generator/'
```

An xml:base attribute on an <entry> overrides the xml:base on the parent <feed>.

Overriding xml:base on an <entry>

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/base.xml")
>>> d.entries[0].link
u'http://example.org/archives/000001.html'
>>> d.entries[0].author_detail.href
u'http://example.org/about/'
```

An xml:base on <content> overrides the xml:base on the parent <entry>. In addition, whatever the base URI is for the <content> element (whether defined directly on the <content> element, or inherited from the parent element) is used as the base URI for the embedded HTML or XHTML markup within the content.

Relative links within embedded HTML

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/base.xml")
>>> d.entries[0].content[0].value
u'<p id="anchor1"><a href="http://example.org/archives/000001.html#anchor2">skip to␣
↪anchor 2</a></p>
<p>Some content</p>
<p id="anchor2">This is anchor 2</p>'
```

The xml:base affects other attributes in the element in which it is declared.

xml:base and sibling attributes

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/base.xml")
>>> d.entries[0].links[1].rel
u'service.edit'
>>> d.entries[0].links[1].href
u'http://example.com/api/client/37'
```

If no xml:base is specified on the root-level element, the default base URI is given in the Content-Location HTTP header. This can still be overridden by any child element that declares an xml:base attribute.

Content-Location HTTP header

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/http_base.xml")
>>> d.feed.link
u'http://example.org/index.html'
>>> d.entries[0].link
u'http://example.org/archives/000001.html'
```

Finally, if no root-level xml:base is declared, and no Content-Location HTTP header is present, the URL of the feed itself is the default base URI. Again, this can still be overridden by any element that declares an xml:base attribute.

Feed URL as default base URI

```
>>> import feedparser
>>> d = feedparser.parse("http://feedparser.org/docs/examples/no_base.xml")
>>> d.feed.link
u'http://feedparser.org/docs/examples/index.html
>>> d.entries[0].link
u'http://example.org/archives/000001.html'
```

Disabling Relative URIs Resolution

Though not recommended, it is possible to disable **`Universal Feed Parser`**'s relative URI resolution by setting feedparser.RESOLVE_RELATIVE_URIS to `0`.

How to disable relative URI resolution

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/base.xml')
>>> d.entries[0].content[0].base
u'http://example.org/archives/000001.html'
>>> print d.entries[0].content[0].value
<p id="anchor1"><a href="http://example.org/archives/000001.html#anchor2">skip to␣
↪anchor 2</a></p>
<p>Some content</p>
<p id="anchor2">This is anchor 2</p>
>>> feedparser.RESOLVE_RELATIVE_URIS = 0
>>> d2 = feedparser.parse('http://feedparser.org/docs/examples/base.xml')
>>> d2.entries[0].content[0].base
u'http://example.org/archives/000001.html'
>>> print d2.entries[0].content[0].value
<p id="anchor1"><a href="#anchor2">skip to anchor 2</a></p>
<p>Some content</p>
<p id="anchor2">This is anchor 2</p>
```

Feed Type and Version Detection

`Universal Feed Parser` attempts to autodetect the type and version of the feeds it parses. There are many subtle and not-so-subtle differences between the different versions of RSS, and applications may choose to handle different feed types in different ways.

Accessing feed version

```
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.version
'atom10'
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom03.xml')
>>> d.version
'atom03'
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20.xml')
>>> d.version
```

```
'rss20'
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss20dc.xml')
>>> d.version
'rss20'
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rss10.rdf')
>>> d.version
'rss10'
```

Here is the complete list of known feed types and versions that may be returned in `version`:

rss090 RSS 0.90

rss091n Netscape RSS 0.91

rss091u Userland RSS 0.91 (differences from Netscape RSS 0.91)

rss10 RSS 1.0

rss092 RSS 0.92

rss093 RSS 0.93

rss094 RSS 0.94 (no accurate specification is known to exist)

rss20 RSS 2.0

rss RSS (unknown or unrecognized version)

atom01 Atom 0.1

atom02 Atom 0.2

atom03 Atom 0.3

atom10 Atom 1.0

atom Atom (unknown or unrecognized version)

cdf CDF

If the feed type is completely unknown, `version` will be an empty string.

Character Encoding Detection

Tip: Feeds may be published in any character encoding. **Python** supports only a few character encodings by default. To support the maximum number of character encodings (and be able to parse the maximum number of feeds), you should install `cjkcodecs` and `iconv_codec`. Both are available at http://cjkpython.i18n.org/.

RFC 3023 defines the interaction between XML (Extensible Markup Language) and HTTP as it relates to character encoding. XML and HTTP have different ways of specifying character encoding and different defaults in case no encoding is specified, and determining which value takes precedence depends on a variety of factors.

Introduction to Character Encoding

In XML, the character encoding is optional and may be given in the XML declaration in the first line of the document, like this:

```
<?xml version="1.0" encoding="utf-8"?>
```

If no encoding is given, XML supports the use of a Byte Order Mark to identify the document as some flavor of UTF-32, UTF-16, or UTF-8. Section F of the XML specification outlines the process for determining the character encoding based on unique properties of the Byte Order Mark in the first two to four bytes of the document.

If no encoding is specified and no Byte Order Mark is present, XML defaults to UTF-8.

HTTP uses MIME to define a method of specifying the character encoding, as part of the Content-Type HTTP header, which looks like this:

```
Content-Type: text/html; charset="utf-8"
```

If no charset is specified, HTTP defaults to iso-8859-1, but only for text/* media types. For other media types, the default encoding is undefined, which is where RFC 3023 comes in.

According to RFC 3023, if the media type given in the Content-Type HTTP header is application/xml, application/xml-dtd, application/xml-external-parsed-entity, or any one of the subtypes of application/xml such as application/atom+xml or application/rss+xml or even application/rdf+xml, then the encoding is

1. the encoding given in the `charset` parameter of the Content-Type HTTP header, or
2. the encoding given in the encoding attribute of the XML declaration within the document, or
3. utf-8.

On the other hand, if the media type given in the Content-Type HTTP header is text/xml, text/xml-external-parsed-entity, or a subtype like text/AnythingAtAll+xml, then the encoding attribute of the XML declaration within the document is ignored completely, and the encoding is

1. the encoding given in the charset parameter of the Content-Type HTTP header, or
2. us-ascii.

Handling Incorrectly-Declared Encodings

`Universal Feed Parser` initially uses the rules specified in RFC 3023 to determine the character encoding of the feed. If parsing succeeds, then that's that. If parsing fails, **`Universal Feed Parser`** sets the `bozo` bit to `1` and sets `bozo_exception` to `feedparser.CharacterEncodingOverride`. Then it tries to reparse the feed with the following character encodings:

1. the encoding specified in the XML declaration
2. the encoding sniffed from the first four bytes of the document (as per Section F)
3. the encoding auto-detected by the Universal Encoding Detector, if installed
4. utf-8
5. windows-1252

If the character encoding can not be determined, **`Universal Feed Parser`** sets the `bozo` bit to `1` and sets `bozo_exception` to `feedparser.CharacterEncodingUnknown`. In this case, parsed values will be strings, not Unicode strings.

Handling Incorrectly-Declared Media Types

RFC 3023 only applies when the feed is served over HTTP with a Content-Type that declares the feed to be some kind of XML. However, some web servers are severely misconfigured and serve feeds with a Content-Type of text/plain, application/octet-stream, or some completely bogus media type.

Universal Feed Parser will attempt to parse such feeds, but it will set the `bozo` bit to `1` and set `bozo_exception` to `feedparser.NonXMLContentType`.

See also:

- RFC 3023
- Section F of the XML specification
- On the well-formedness of XML documents served as text/plain
- CJKCodecs and iconv_codec

Bozo Detection

Universal Feed Parser can parse feeds whether they are well-formed XML or not. However, since some applications may wish to reject or warn users about non-well-formed feeds, **Universal Feed Parser** sets the `bozo` bit when it detects that a feed is not well-formed. Thanks to Tim Bray for suggesting this terminology.

Detecting a non-well-formed feed

```
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.bozo
0
>>> d = feedparser.parse('http://feedparser.org/tests/illformed/rss/aaa_illformed.xml
↪')
>>> d.bozo
1
>>> d.bozo_exception
<xml.sax._exceptions.SAXParseException instance at 0x00BAAA08>
>>> exc = d.bozo_exception
>>> exc.getMessage()
"expected '>'\\n"
>>> exc.getLineNumber()
6
```

There are many reasons an XML document could be non-well-formed besides this example (incomplete end tags) See *Character Encoding Detection* for some other ways to trip the bozo bit.

CHAPTER 3

HTTP Features

ETag and Last-Modified Headers

ETags and Last-Modified headers are two ways that feed publishers can save bandwidth, but they only work if clients take advantage of them. **`Universal Feed Parser`** gives you the ability to take advantage of these features, but you must use them properly.

The basic concept is that a feed publisher may provide a special HTTP header, called an ETag, when it publishes a feed. You should send this ETag back to the server on subsequent requests. If the feed has not changed since the last time you requested it, the server will return a special HTTP status code (`304`) and no feed data.

Using ETags to reduce bandwidth

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.etag
'"6c132-941-ad7e3080"'
>>> d2 = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml', etag=d.
↪etag)
>>> d2.status
304
>>> d2.feed
{}
>>> d2.entries
[]
>>> d2.debug_message
'The feed has not changed since you last checked, so
the server sent no data.  This is a feature, not a bug!'
```

There is a related concept which accomplishes the same thing, but slightly differently. In this case, the server publishes the last-modified date of the feed in the HTTP header. You can send this back to the server on subsequent requests, and if the feed has not changed, the server will return HTTP status code `304` and no feed data.

Using Last-Modified headers to reduce bandwidth

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
>>> d.modified
Fri, 11 Jun 2012 23:00:34 GMT
>>> d.modified_parsed
(2004, 6, 11, 23, 0, 34, 4, 163, 0)
>>> d2 = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml',␣
↪modified=d.modified)
>>> d2.status
304
>>> d2.feed
{}
>>> d2.entries
[]
>>> d2.debug_message
'The feed has not changed since you last checked, so
the server sent no data.  This is a feature, not a bug!'
```

Clients should support both ETag and Last-Modified headers, as some servers support one but not the other.

Important: If you do not support ETag and Last-Modified headers, you will repeatedly download feeds that have not changed. This wastes your bandwidth and the publisher's bandwidth, and the publisher may ban you from accessing their server.

Note: You can control the behaviour of HTTP caches between your application and the origin server by using the `extra_headers` parameter. For example, you may want to send `Cache-control: max-age=60` to make the caches revalidate against the origin server unless their cached copy is less than a minute old. Again, this should be used with consideration.

See also:

- HTTP Conditional Get For RSS Hackers
- HTTP Web Services

User-Agent and Referer Headers

Universal Feed Parser sends a default User-Agent string when it requests a feed from a web server.

The default User-Agent string looks like this:

```
UniversalFeedParser/5.0.1 +http://feedparser.org/
```

If you are embedding **Universal Feed Parser** in a larger application, you should change the User-Agent to your application name and URL.

Customizing the User-Agent

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml',
agent='MyApp/1.0 +http://example.com/')
```

You can also set the User-Agent once, globally, and then call the `parse` function normally.

Customizing the User-Agent permanently

```
>>> import feedparser
>>> feedparser.USER_AGENT = "MyApp/1.0 +http://example.com/"
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
```

Universal Feed Parser also lets you set the referrer when you download a feed from a web server. This is discouraged, because it is a violation of RFC 2616. The default behavior is to send a blank referrer, and you should never need to override this.

Customizing the referrer

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml',
referrer='http://example.com/')
```

HTTP Redirects

When you download a feed from a remote web server, **Universal Feed Parser** exposes the HTTP status code. You need to understand the different codes, including permanent and temporary redirects, and feeds that have been marked "gone".

When a feed has temporarily moved to a new location, the web server will return a `302` status code. **Universal Feed Parser** makes this available in `d.status`.

There is nothing special you need to do with temporary redirects; by the time you learn about it, **Universal Feed Parser** has already followed the redirect to the new location (available in `d.href`), downloaded the feed, and parsed it. Since the redirect is temporary, you should continue requesting the original URL the next time you want to parse the feed.

Noticing temporary redirects

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/temporary.xml')
>>> d.status
302
>>> d.href
'http://feedparser.org/docs/examples/atom10.xml'
>>> d.feed.title
u'Sample Feed'
```

When a feed has permanently moved to a new location, the web server will return a `301` status code. Again, **Universal Feed Parser** makes this available in `d.status`.

If you are polling a feed on a regular basis, it is very important to check the status code (`d.status`) every time you download. If the feed has been permanently redirected, you should update your database or configuration file with the new address (`d.href`). Repeatedly requesting the original address of a feed that has been permanently redirected is very rude, and may get you banned from the server.

Noticing permanent redirects

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/permanent.xml')
>>> d.status
301
>>> d.href
'http://feedparser.org/docs/examples/atom10.xml'
>>> d.feed.title
u'Sample Feed'
```

When a feed has been permanently deleted, the web server will return a `410` status code. If you ever receive a `410`, you should stop polling the feed and inform the end user that the feed is gone for good.

Repeatedly requesting a feed that has been marked as "gone" is very rude, and may get you banned from the server.

Noticing feeds marked "gone"

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/gone.xml')
>>> d.status
410
```

Password-Protected Feeds

Universal Feed Parser supports downloading and parsing password-protected feeds that are protected by HTTP authentication. Both basic and digest authentication are supported.

Downloading a feed protected by basic authentication (the easy way)

The easiest way is to embed the username and password in the feed URL itself.

In this example, the username is test and the password is basic.

```
>>> import feedparser
>>> d = feedparser.parse('http://test:basic@feedparser.org/docs/examples/basic_auth.
↪xml')
>>> d.feed.title
u'Sample Feed'
```

The same technique works for digest authentication. (Technically, **Universal Feed Parser** will attempt basic authentication first, but if that fails and the server indicates that it requires digest authentication, **Universal Feed Parser** will automatically re-request the feed with the appropriate digest authentication headers. *This means that this technique will send your password to the server in an easily decryptable form.*)

Downloading a feed protected by digest authentication (the easy but horribly insecure way)

In this example, the username is test and the password is digest.

```
>>> import feedparser
>>> d = feedparser.parse('http://test:digest@feedparser.org/docs/examples/digest_auth.
↪xml')
>>> d.feed.title
u'Sample Feed'
```

You can also construct a HTTPBasicAuthHandler that contains the password information, then pass that as a handler to the `parse` function. HTTPBasicAuthHandler is part of the standard urllib2 module.

Downloading a feed protected by HTTP basic authentication (the hard way)

```
import urllib2, feedparser

# Construct the authentication handler
auth = urllib2.HTTPBasicAuthHandler()

# Add password information: realm, host, user, password.
# A single handler can contain passwords for multiple sites;
# urllib2 will sort out which passwords get sent to which sites
# based on the realm and host of the URL you're retrieving
auth.add_password('BasicTest', 'feedparser.org', 'test', 'basic')

# Pass the authentication handler to the feed parser.
# handlers is a list because there might be more than one
# type of handler (urllib2 defines lots of different ones,
# and you can build your own)
d = feedparser.parse('http://feedparser.org/docs/examples/basic_auth.xml',
                     handlers=[auth])
```

Digest authentication is handled in much the same way, by constructing an HTTPDigestAuthHandler and populating it with the necessary realm, host, user, and password information. This is more secure than *stuffing the username and password in the URL*, since the password will be encrypted before being sent to the server.

Downloading a feed protected by HTTP digest authentication (the secure way)

```
import urllib2, feedparser

auth = urllib2.HTTPDigestAuthHandler()
auth.add_password('DigestTest', 'feedparser.org', 'test', 'digest')
d = feedparser.parse('http://feedparser.org/docs/examples/digest_auth.xml',
                     handlers=[auth])
```

The examples so far have assumed that you know in advance that the feed is password-protected. But what if you don't know?

If you try to download a password-protected feed without sending all the proper password information, the server will return an HTTP status code `401`. **`Universal Feed Parser`** makes this status code available in `d.status`.

Details on the authentication scheme are in `d.headers['www-authenticate']`. **`Universal Feed Parser`** does not do any further parsing on this field; you will need to parse it yourself. Everything before the first

space is the type of authentication (probably `Basic` or `Digest`), which controls which type of handler you'll need to construct. The realm name is given as realm="foo" – so foo would be your first argument to auth.add_password. Other information in the www-authenticate header is probably safe to ignore; the `urllib2` module will handle it for you.

Determining that a feed is password-protected

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/basic_auth.xml')
>>> d.status
401
>>> d.headers['www-authenticate']
'Basic realm="Use test/basic"'
>>> d = feedparser.parse('http://feedparser.org/docs/examples/digest_auth.xml')
>>> d.status
401
>>> d.headers['www-authenticate']
'Digest realm="DigestTest",
nonce="+LV/uLLdAwA=5d77397291261b9ef256b034e19bcb94f5b7992a",
algorithm=MD5,
qop="auth"'
```

Other HTTP Headers

You can specify extra HTTP request headers as a dictionary. When you download a feed from a remote web server, **`Universal Feed Parser`** exposes the complete set of HTTP response headers as a dictionary.

Sending custom HTTP request headers

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom03.xml',
                         extra_headers={'Cache-control': 'max-age=0'})
```

Accessing other HTTP response headers

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/atom03.xml')
>>> d.headers
{'date': 'Fri, 11 Jun 2004 23:57:50 GMT',
'server': 'Apache/2.0.49 (Debian GNU/Linux)',
'last-modified': 'Fri, 11 Jun 2004 23:00:34 GMT',
'etag': '"6c132-941-ad7e3080"',
'accept-ranges': 'bytes',
'vary': 'Accept-Encoding,User-Agent',
'content-encoding': 'gzip',
'content-length': '883',
'connection': 'close',
'content-type': 'application/xml'}
```

CHAPTER 4

Annotated Examples

Atom 1.0

This is a sample Atom 1.0 feed, annotated with links that show how each value can be accessed once the feed is parsed.

Caution: Even though many of these elements are required according to the specification, real-world feeds may be missing any element. If an element is not present in the feed, it will not be present in the parsed results. You should not rely on any particular element being present.

Annotated Atom 1.0 feed

<?xml version="1.0" encoding="*utf-8*"?> <feed xmlns="*http://www.w3.org/2005/Atom*" xml:base="*http://example.org/*" xml:lang="*en*"> <title type="*text*"> *Sample Feed* </title> <subtitle type="*html*"> *For documentation only* </subtitle> <link rel="*alternate*" type="*html*" href="/"/> <link rel="*self*" type="*application/atom+xml*" href="*http://www.example.org/atom10.xml*"/> <rights type="*html*"> *<p>Copyright 2005, Mark Pilgrim</p>* </rights> <generator uri="*http://example.org/generator/*" version="*4.0*"> *Sample Toolkit* </generator> <id>*tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml*</id> <updated>*2005-11-09T11:56:34Z*</updated> <entry> <title>*First entry title*</title> <link rel="*alternate*" href="*/entry/3*"/> <link rel="*related*" type="*text/html*" href="*http://search.example.com/*"/> <link rel="*via*" type="*text/html*" href="*http://toby.example.com/examples/atom10*"/> <link rel="*enclosure*" type="*video/mpeg4*" href="*http://www.example.com/movie.mp4*" length="*42301*"/> <id>*tag:feedparser.org,2005-11-09:/docs/examples/atom10.xml:3*</id> <published>*2005-11-09T00:23:47Z*</published> <updated>*2005-11-09T11:56:34Z*</updated> <author> <name>*Mark Pilgrim*</name> <uri>*http://diveintomark.org/*</uri> <email>*mark@example.org*</email> </author> <contributor> <name>*Joe*</name> <url>*http://example.org/joe/*</url> <email>*joe@example.org*</email> </contributor> <contributor> <name>*Sam*</name> <url>*http://example.org/sam/*</url> <email>*sam@example.org*</email> </contributor> <summary type="*text*"> *Watch out for nasty tricks* </summary> <content type="*xhtml*" xml:base="*http://example.org/entry/3*" xml:lang="*en-US*">* Watch out for nasty tricks * </content> </entry> </feed>

Atom 0.3

This is a sample Atom 0.3 feed, annotated with links that show how each value can be accessed once the feed is parsed.

Caution: Even though many of these elements are required according to the specification, real-world feeds may be missing any element. If an element is not present in the feed, it will not be present in the parsed results. You should not rely on any particular element being present.

Annotated Atom 0.3 feed

<?xml version="1.0" encoding="*utf-8*"?> <feed version="*0.3*" xmlns="http://purl.org/atom/ns#" xml:base="http://example.org/" xml:lang="en"> <title type="*text/plain*" mode="escaped"> *Sample Feed* </title> <tagline type="*text/html*" mode="escaped"> *For documentation only* </tagline> <link rel="*alternate*" type="*text/html*" href="/"/> <copyright type="*text/html*" mode="escaped"> *<p>Copyright 2004, Mark Pilgrim</p><* </copyright> <generator url="*http://example.org/generator/*" version="*3.0*"> *Sample Toolkit* </generator> <id>*tag:feedparser.org,2004-04-20:/docs/examples/atom03.xml*</id> <modified>*2004-04-20T11:56:34Z*</modified> <info type="*application/xhtml+xml*" mode="xml"> * This is an Atom syndication feed. * </info> <entry> <title>*First entry title*</title> <link rel="*alternate*" type="*text/html*" href="*/entry/3*"/> <link rel="*service.edit*" type="*application/atom+xml*" title="*Atom API entrypoint to edit this entry*" href="*/api/edit/3*"/> <link rel="*service.post*" type="*application/atom+xml*" title="*Atom API entrypoint to add comments to this entry*" href="*/api/comment/3*"/> <id>*tag:feedparser.org,2004-04-20:/docs/examples/atom03.xml:3*</id> <created>*2004-04-19T07:45:00Z*</created> <issued>*2004-04-20T00:23:47Z*</issued> <modified>*2004-04-20T11:56:34Z*</modified> <author> <name>*Mark Pilgrim*</name> <url>*http://diveintomark.org/*</url> <email>*mark@example.org*</email> </author> <contributor> <name>*Joe*</name> <url>*http://example.org/joe/*</url> <email>*joe@example.org*</email> </contributor> <contributor> <name>*Sam*</name> <url>*http://example.org/sam/*</url> <email>*sam@example.org*</email> </contributor> <summary type="*text/plain*" mode="escaped"> *Watch out for nasty tricks* </summary> <content type="*application/xhtml+xml*" mode="xml" xml:base="*http://example.org/entry/3*" xml:lang="*en-US*"> * Watch out for nasty tricks * </content> </entry> </feed>

RSS 2.0

This is a sample RSS 2.0 feed, annotated with links that show how each value can be accessed once the feed is parsed.

Caution: Even though many of these elements are required according to the specification, real-world feeds may be missing any element. If an element is not present in the feed, it will not be present in the parsed results. You should not rely on any particular element being present.

Annotated RSS 2.0 feed

<?xml version="1.0" encoding="*utf-8*"?> <rss version="*2.0*"> <channel> <title>*Sample Feed*</title> <description>*For documentation only*</description> <link>*http://example.org/*</link> <language>*en*</language> <copyright>*Copyright 2004, Mark Pilgrim*</copyright> <managingEditor>*editor@example.org*</managingEditor> <webMaster>*webmaster@example.org*</webMaster> <pubDate>*Sat, 07 Sep 2002 0:00:01 GMT*</pubDate> <category>*Examples*</category> <generator>*Sample*

Toolkit</generator> <docs>*http://feedvalidator.org/docs/rss2.html*</docs> <cloud domain="*rpc.example.com*" port="*80*" path="*/RPC2*" registerProcedure="*pingMe*" protocol="*soap*"/> <ttl>*60*</ttl> <image> <url>*http://example.org/banner.png*</url> <title>*Example banner*</title> <link>*http://example.org/*</link> <width>*80*</width> <height>*15*</height> </image> <textInput> <title>*Search*</title> <description>*Search this site:*</description> <name>*q*</name> <link>*http://example.org/mt/mt-search.cgi*</link> </textInput> <item> <title>*First item title*</title> <link>*http://example.org/item/1*</link> <description>*Watch out for nasty tricks* </description> <author>*mark@example.org*</author> <category>*Miscellaneous*</category> <comments>*http://example.org/comments/1*</comments> <enclosure url="*http://example.org/audio/demo.mp3*" length="*1069871*" type="*audio/mpeg*"/> <guid>*http://example.org/guid/1*</guid> <pubDate>*Thu, 05 Sep 2002 0:00:01 GMT*</pubDate> </item> </channel> </rss>

RSS 2.0 with Namespaces

This is a sample RSS 2.0 feed that uses several allowable extension modules in namespaces. The feed is annotated with links that show how each value can be accessed once the feed is parsed.

Caution: Even though many of these elements are required according to the specification, real-world feeds may be missing any element. If an element is not present in the feed, it will not be present in the parsed results. You should not rely on any particular element being present.

Annotated RSS 2.0 feed with namespaces

<?xml version="1.0" encoding="*utf-8*"?> <rss version="*2.0*" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:admin="http://webns.net/mvcb/" xmlns:content="http://purl.org/rss/1.0/modules/content/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"> <channel> <title>*Sample Feed*</title> <link>*http://example.org/*</link> <description>*For documentation only*</description> <dc:language>*en-us*</dc:language> <dc:creator>*Mark Pilgrim (mark@example.org)*</dc:creator> <dc:rights>*Copyright 2004 Mark Pilgrim*</dc:rights> <dc:date>*2004-06-04T17:40:33-05:00*</dc:date> <admin:generatorAgent rdf:resource="*http://www.exampletoolkit.org/*"/> <admin:errorReportsTo rdf:resource="*mailto:mark@example.org*"/> <item> <title>*First of all*</title> <link>*http://example.org/archives/2002/09/04.html#first_of_all*</link> <guid isPermaLink="false">*1983@example.org*</guid> <description> *Americans are fat. Smokers are stupid. People who don't speak Perl are irrelevant.* </description> <dc:subject>*Quotes*</dc:subject> <dc:date>*2002-09-04T13:54:20-05:00*</dc:date> <content:encoded><![CDATA[*<cite>Ian Hickson</cite>: <q> Americans are fat. Smokers are stupid. People who don't speak Perl are irrelevant. </q>*]]> </content:encoded> </item> </channel> </rss>

RSS 1.0

This is a sample RSS 1.0 feed, annotated with links that show how each value can be accessed once the feed is parsed.

Caution: Even though many of these elements are required according to the specification, real-world feeds may be missing any element. If an element is not present in the feed, it will not be present in the parsed results. You should not rely on any particular element being present.

Annotated RSS 1.0 feed

<?xml version="1.0" encoding="*utf-8*"?> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:admin="http://webns.net/mvcb/" xmlns:content="http://purl.org/rss/1.0/modules/content/" xmlns:cc="http://web.resource.org/cc/" xmlns="*http://purl.org/rss/1.0/*"> <channel rdf:about="http://www.example.org/index.rdf"> <title>*Sample Feed*</title> <link>*http://www.example.org/*</link> <description>*For documentation only*</description> <dc:language>*en*</dc:language> <cc:license rdf:resource="*http://web.resource.org/cc/PublicDomain*"/> <dc:creator>*Mark Pilgrim (mark@example.org)*</dc:creator> <dc:date>*2004-06-04T17:40:33-05:00*</dc:date> <admin:generatorAgent rdf:resource="*http://www.exampletoolkit.org/*"/> <admin:errorReportsTo rdf:resource="*mailto:mark@example.org*"/> <items> <rdf:Seq> <rdf:li rdf:resource="http://www.example.org/1" /> </rdf:Seq> </items> </channel> <item rdf:about="*http://www.example.org/1*"> <title>*First of all*</title> <link>*http://example.org/archives/2002/09/04.html#first_of_all*</link> <description> *Americans are fat. Smokers are stupid. People who don't speak Perl are irrelevant.* </description> <dc:subject>*Quotes*</dc:subject> <dc:date>*2004-05-30T14:23:54-06:00*</dc:date> <content:encoded><![CDATA[*<cite>Ian Hickson</cite>: <q> Americans are fat. Smokers are stupid. People who don't speak Perl are irrelevant. </q>]]>* </content:encoded> </item> </rdf:RDF>

CHAPTER 5

Revision history

Changes in version 4.2

Universal Feed Parser 4.2 was released on 2008-03-12.

- Support for *parsing microformats*, including *rel=enclosure*, *rel=tag*, *XFN*, and *hCard*.
- Updated the whitelist of *acceptable HTML elements and attributes* based on the latest draft of the HTML 5 specification.
- Support for *CSS Sanitization*. (Previous versions of **Universal Feed Parser** simply stripped all inline styles.) Many thanks to Sam Ruby for implementing this, despite my insistence that it was impossible.
- Support for *SVG Sanitization*.
- Support for *MathML Sanitization*. Many thanks to Jacques Distler for patiently debugging this feature.
- IRI (International Resource Identifier) support for every element that can contain a URI.
- Ability to *disable relative URI resolution*.
- Command-line arguments and alternate serializers, for manipulating **Universal Feed Parser** from shell scripts or other non-Python sources.
- More robust parsing of author email addresses, misencoded win-1252 content, rel=self links, and better detection of HTML content in elements with ambiguous content types.

Changes in version 4.1

Universal Feed Parser 4.1 was released on January 11, 2006.

- Support for the Universal Encoding Detector to autodetect character encoding of feeds that declare their encoding incorrectly or don't declare it at all. See *Character Encoding Detection* for details of when this gets called.

- **Universal Feed Parser** no longer sets a default socket timeout (SourceForge bug 1392140). If you were relying on this feature, you will need to call socket.setdefaulttimeout(TIMEOUT_IN_SECONDS) yourself.

Changes in version 4.0.2

Universal Feed Parser 4.0.2 was released on December 24, 2005.

- cleared `_debug` flag.

Changes in version 4.0.1

Universal Feed Parser 4.0.1 was released on December 24, 2005.

- bug fixes for **Python** 2.1 compatibility.

Changes in version 4.0

Universal Feed Parser 4.0 was released on December 23, 2005.

- Support for *Atom 1.0*.
- Support for **iTunes** extensions.
- Support for dc:contributor.
- **Universal Feed Parser** now captures the feed's *namespaces*. See *Namespace Handling* for details.
- Lots of things have been renamed to match Atom 1.0 terminology. issued is now *entries[i].published*, modified is now *entries[i].updated*, and url is now href everywhere. You can still access these elements with the old names, so you shouldn't need to change any existing code, but don't be surprised if you can't find them during debugging.
- category and categories have been replaced by tags, see *feed.tags* and *entries[i].tags*. The old names still work.
- mode is gone from all detail and content dictionaries. It was never terribly useful, since **Universal Feed Parser** unescapes content automatically.
- *entries[i].source* is now a dictionary of feed metadata as per section 4.2.11 of RFC 4287. **Universal Feed Parser** no longer supports the RSS 2.0's source element.
- Content in unknown namespaces is no longer discarded (bug 993305)
- Lots of other bug fixes.

Changes in version 3.3

Universal Feed Parser 3.3 was released on July 15, 2004.

- optimized EBCDIC to ASCII conversion
- fixed obscure problem tracking xml:base and xml:lang if element declares it, child doesn't, first grandchild redeclares it, and second grandchild doesn't
- refactored date parsing

- defined public `registerDateHandler` so callers can add support for additional date formats at runtime
- added support for OnBlog, Nate, MSSQL, Greek, and Hungarian dates (ytrewq1)
- added `zopeCompatibilityHack()` which turns FeedParserDict into a regular dictionary, required for **Zope** compatibility, and also makes command-line debugging easier because pprint module formats real dictionaries better than dictionary-like objects
- added NonXMLContentType exception, which is stored in `bozo_exception` when a feed is served with a non-XML media type such as `'text/plain'`
- respect `Content-Language` as default language if no xml:lang is present
- `cloud` dict is now FeedParserDict
- generator dict is now FeedParserDict
- better tracking of xml:lang, including support for xml:lang='' to unset the current language
- recognize RSS 1.0 feeds even when RSS 1.0 namespace is not the default namespace
- don't overwrite final status on redirects (scenarios: redirecting to a URI that returns `304`, redirecting to a URI that redirects to another URI with a different type of redirect)
- add support for `HTTP 303` redirects

Changes in version 3.2

Universal Feed Parser 3.2 was released on July 3, 2004.

- use `cjkcodecs` and `iconv_codec` if available
- always convert feed to UTF-8 before passing to XML parser
- completely revamped logic for determining character encoding and attempting XML parsing (much faster)
- increased default timeout to 20 seconds
- test for presence of `Location` header on redirects
- added tests for many alternate character encodings
- support various EBCDIC encodings
- support UTF-16BE and UTF16-LE with or without a BOM (Byte Order Mark)
- support UTF-8 with a BOM
- support UTF-32BE and UTF-32LE with or without a BOM
- fixed crashing bug if no XML parsers are available
- added support for `Content-encoding: deflate`
- send blank `Accept-encoding` header if neither `gzip` nor `zlib` modules are available

Changes in version 3.1

Universal Feed Parser 3.1 was released on June 28, 2004.

- added and passed tests for converting HTML entities to Unicode equivalents in illformed feeds (aaronsw)
- added and passed tests for converting character entities to Unicode equivalents in illformed feeds (aaronsw)

- test for valid parsers when setting `XML_AVAILABLE`
- make version and encoding available when server returns a `304`
- add `handlers` parameter to pass arbitrary `urllib2` handlers (like digest auth or proxy support)
- add code to parse username/password out of url and send as basic authentication
- expose downloading-related exceptions in `bozo_exception` (aaronsw)
- added __contains__ method to FeedParserDict (aaronsw)
- added `publisher_detail` (aaronsw)

Changes in version 3.0.1

`Universal Feed Parser` 3.0.1 was released on June 21, 2004.

- default to `us-ascii` for all text/* content types
- recover from malformed `content-type` header parameter with no equals sign ("text/xml; charset:iso-8859-1")
- docs: added `reference-feed.html` and `reference-entry.html` (bug #977723)
- docs: fixed `entry[i]` in documentation (should be `entries[i]`) (bug #977722)
- docs: added note about Unicode string usage (bug #977716)
- docs: added `basic-existence.html` (bug #977704)
- docs: fixed description of feed title (bug #977685)
- docs: fixed typo in annotated RSS 1.0 feed (bug #977682)

Changes in version 3.0

`Universal Feed Parser` 3.0 was released on June 21, 2004.

- don't try `iso-8859-1` (can't distinguish between `iso-8859-1` and `windows-1252` anyway, and most incorrectly marked feeds are `windows-1252`)
- fixed regression that could cause the same encoding to be tried twice (even if it failed the first time)

`Universal Feed Parser` 3.0fc3 was released on June 18, 2004.

- fixed bug in `_changeEncodingDeclaration` that failed to parse UTF-16 encoded feeds
- made `source` into a FeedParserDict
- duplicate admin:generatorAgent/@rdf:resource in `generator_detail.url`
- added support for image
- refactored `parse()` fallback logic to try other encodings if SAX parsing fails (previously it would only try other encodings if re-encoding failed)
- remove `unichr` madness in normalize_attrs now that we're properly tracking encoding in and out of BaseHTMLProcessor
- set `feed.language` from root-level xml:lang
- set `entry.id` from rdf:about

- send `Accept` header

Universal Feed Parser 3.0fc2 was released on May 10, 2004.

- added and passed Sam's amp tests
- added and passed my blink tag tests

Universal Feed Parser 3.0fc1 was released on April 23, 2004.

- made `results.entries[0].links[0]` and `results.entries[0].enclosures[0]` into FeedParserDict
- fixed typo that could cause the same encoding to be tried twice (even if it failed the first time)
- fixed DOCTYPE stripping when DOCTYPE contained entity declarations
- better textinput and image tracking in illformed RSS 1.0 feeds

Universal Feed Parser 3.0b23 was released on April 21, 2004.

- fixed `UnicodeDecodeError` for feeds that contain high-bit characters in attributes in embedded HTML in description (thanks Thijs van de Vossen)
- moved `guid`, `date`, and `date_parsed` to mapped keys in FeedParserDict
- tweaked FeedParserDict.has_key to return `True` if asking about a mapped key

Universal Feed Parser 3.0b22 was released on April 19, 2004.

- changed `channel` to `feed`, `item` to `entries` in `results` dict
- changed `results` dict to allow getting values with `results.key` as well as `results[key]`
- work around embedded illformed HTML with half a DOCTYPE
- work around malformed `Content-Type` header
- if character encoding is wrong, try several common ones before falling back to regexes (if this works, `bozo_exception` is set to `CharacterEncodingOverride`
- fixed character encoding issues in BaseHTMLProcessor by tracking encoding and converting from Unicode to raw strings before feeding data to sgmllib.SGMLParser
- convert each value in results to Unicode (if possible), even if using regex-based parsing

Universal Feed Parser 3.0b21 was released on April 14, 2004.

- added Hot RSS support

Universal Feed Parser 3.0b20 was released on April 7, 2004.

- added CDF support

Universal Feed Parser 3.0b19 was released on March 15, 2004.

- fixed bug exploding author information when author name was in parentheses
- removed ultra-problematic `mxTidy` support
- patch to workaround crash in PyXML/expat when encountering invalid entities (MarkMoraes)
- support for textinput/textInput

Universal Feed Parser 3.0b18 was released on February 17, 2004.

- always map description to `summary_detail` (Andrei)
- use `libxml2` (if available)

Universal Feed Parser 3.0b17 was released on February 13, 2004.

- determine character encoding as per RFC 3023

Universal Feed Parser 3.0b16 was released on February 12, 2004.

- fixed support for RSS 0.90 (broken in b15)

Universal Feed Parser 3.0b15 was released on February 11, 2004.

- fixed bug resolving relative links in wfw:commentRSS
- fixed bug capturing author and contributor URI
- fixed bug resolving relative links in author and contributor URI
- fixed bug resolving relative links in generator URI
- added support for recognizing RSS 1.0
- passed Simon Fell's namespace tests, and included them permanently in the test suite with his permission
- fixed namespace handling under **Python** 2.1

Universal Feed Parser 3.0b14 was released on February 8, 2004.

- fixed CDATA handling in non-wellformed feeds under **Python** 2.1

Universal Feed Parser 3.0b13 was released on February 8, 2004.

- better handling of empty HTML tags (br, hr, img, etc.) in embedded markup, in either HTML or XHTML form (
,
,
)

Universal Feed Parser 3.0b12 was released on February 6, 2004.

- fiddled with `decodeEntities` (still not right)
- added support to Atom 0.2 subtitle
- added support for Atom content model in copyright
- better sanitizing of dangerous HTML elements with end tags (script, frameset)

Universal Feed Parser 3.0b11 was released on February 2, 2004.

- added rights to list of elements that can contain dangerous markup
- fiddled with `decodeEntities` (not right)
- liberalized date parsing even further

Universal Feed Parser 3.0b10 was released on January 31, 2004.

- incorporated ISO-8601 date parsing routines from `xml.util.iso8601`

Universal Feed Parser 3.0b9 was released on January 29, 2004.

- fixed check for presence of `dict` function
- added support for summary

Universal Feed Parser 3.0b8 was released on January 28, 2004.

- added support for contributor

Universal Feed Parser 3.0b7 was released on January 28, 2004.

- support Atom-style author element in `author_detail` (dictionary of `name`, `url`, `email`)
- map `author` to `author_detail` if `author` contains name + email address

Universal Feed Parser 3.0b6 was released on January 27, 2004.

- added feed type and version detection, `result['version']` will be one of `SUPPORTED_VERSIONS.keys()` or empty string if unrecognized
- added support for creativeCommons:license and cc:license
- added support for full Atom content model in title, tagline, info, copyright, summary
- fixed bug with gzip encoding (not always telling server we support it when we do)

Universal Feed Parser 3.0b5 was released on January 26, 2004.

- fixed bug parsing multiple links at feed level

Universal Feed Parser 3.0b4 was released on January 26, 2004.

- fixed xml:lang inheritance
- fixed multiple bugs tracking xml:base URI, one for documents that don't define one explicitly and one for documents that define an outer and an inner xml:base that goes out of scope before the end of the document

Universal Feed Parser 3.0b3 was released on January 23, 2004.

- parse entire feed with real XML parser (if available)
- added several new supported namespaces
- fixed bug tracking naked markup in description
- added support for enclosure
- added support for source
- re-added support for cloud which got dropped somehow
- added support for expirationDate

Universal Feed Parser 3.0b2 and 3.0b1 have been lost in the mists of time.

Changes in version 2.7.x

The 2.7 series was a brief but necessary transition towards some of the core ideas in version 3.0.

Ultra-liberal Feed Parser 2.7.6 was released on January 16, 2004.

- fixed bug with `StringIO` importing

Ultra-liberal Feed Parser 2.7.5 was released on January 15, 2004.

- added workaround for malformed DOCTYPE (seen on many `blogspot.com` sites)
- added `_debug` variable

Ultra-liberal Feed Parser 2.7.4 was released on January 14, 2004.

- added workaround for improperly formed
 tags in encoded HTML (skadz)
- fixed unicode handling in normalize_attrs (ChrisL)
- fixed relative URI processing for guid (skadz)
- added ICBM support
- added `base64` support

Ultra-liberal Feed Parser 2.7.3 was released on January 14, 2004.

- reverted all changes made in 2.7.2

Ultra-liberal Feed Parser 2.7.2 was released on January 13, 2004.

- "Version 2.7.2 of my feed parser, released today, will by default refuse to parse this feed. It does a first-pass check for wellformedness, and when that fails it sets the 'bozo' bit in the result to `1` and immediately terminates. You can revert to the previous behavior by passing `disableWellFormedCheck=1`, but it will print arrogant warning messages to stderr to the effect that anyone who can't create a well-formed XML feed is a bozo and an incompetent fool." source

Ultra-liberal Feed Parser 2.7.1 was released on January 9, 2004.

- fixed bug handling " and '
- fixed memory leak not closing url opener (JohnD)
- added dc:publisher support (MarekK)
- added admin:errorReportsTo support (MarekK)
- **Python** 2.1 `dict` support (MarekK)

Ultra-liberal Feed Parser 2.7 was released on January 5, 2004.

- really added support for trackback and pingback namespaces, as opposed to 2.6 when I said I did but didn't really
- sanitize HTML markup within some elements
- added `mxTidy` support (if installed) to tidy HTML markup within some elements
- fixed indentation bug in `_parse_date` (FazalM)
- use `socket.setdefaulttimeout` if available (FazalM)
- universal date parsing and normalization (FazalM): `created`, `modified`, `issued` are parsed into 9-tuple date format and stored in `created_parsed`, `modified_parsed`, and `issued_parsed`
- `date` is duplicated in `modified` and vice-versa
- `date_parsed` is duplicated in `modified_parsed` and vice-versa

Changes in version 2.6

Ultra-liberal Feed Parser 2.6 was released on January 1, 2004.

- dc:author support (MarekK)
- fixed bug tracking nested divs within content (JohnD)
- fixed missing `sys` import (JohanS)
- fixed regular expression to capture XML character encoding (Andrei)
- added support for Atom 0.3-style links
- fixed bug with textInput tracking
- added support for cloud (MartijnP)
- added support for multiple category/dc:subject (MartijnP)
- normalize content model: `description` gets description (which can come from `<description>`, `<summary>`, or full content if no `<description>`), `content` gets dict of

`base/language/type/value` (which can come from `<content:encoded>`, `<xhtml:body>`, `<content>`, or `<fullitem>`)

- fixed bug matching arbitrary Userland namespaces
- added xml:base and xml:lang tracking
- fixed bug tracking unknown tags
- fixed bug tracking content when `<content>` element is not in default namespace (like Pocketsoap feed)
- resolve relative URLs in `<link>`, `<guid>`, `<docs>`, `<url>`, `<comments>`, `<wfw:comment>`, `<wfw:commentRSS>`
- resolve relative URI markup in `<description>`, `<xhtml:body>`, `<content>`, `<content:encoded>`, `<title>`, `<subtitle>`, `<summary>`, `<info>`, `<tagline>`, and `<copyright>`
- added support for pingback and trackback namespaces

Changes in earlier versions

Universal Feed Parser began as an "ultra-liberal RSS parser" named `rssparser.py`. It was written as a weapon for battles that no one remembers, to work around problems that no longer exist.

Ultra-liberal Feed Parser 2.5.3 was released on August 3, 2003.

- track whether we're inside an image or textInput (TvdV)
- return the character encoding, if specified

Ultra-liberal Feed Parser 2.5.2 was released on July 28, 2003.

- entity-decode inline XML properly
- added support for inline <xhtml:body> and <xhtml:div> as used in some RSS 2.0 feeds

Ultra-liberal Feed Parser 2.5.1 was released on July 26, 2003.

- clear `opener.addheaders` so we only send our custom `User-Agent` (otherwise `urllib2` sends two, which confuses some servers) (RMK)

Ultra-liberal Feed Parser 2.5 was released on July 25, 2003.

- changed to **Python** license (all contributors agree)
- removed unnecessary `>urllib` code – `urllib2` should always be available anyway
- return actual `url`, `status`, and full HTTP headers (as `result['url']`, `result['status']`, and `result['headers']`) if parsing a remote feed over HTTP. This should pass all the *Aggregator client :abbr:'HTTP (Hypertext Transfer Protocol)* tests <http://diveintomark.org/tests/client/http/>'_.
- added the latest namespace-of-the-week for RSS 2.0

Ultra-liberal Feed Parser 2.4 was released on July 9, 2003.

- added preliminary Pie/Atom/Echo support based on Sam Ruby's snapshot of July 1
- changed project name

Ultra-liberal RSS Parser 2.3.1 was released on June 12, 2003.

- if item has both link and guid, return both as-is

Ultra-liberal RSS Parser 2.3 was released on June 11, 2003.

- added `USER_AGENT` for default (if caller doesn't specify)
- make sure we send the `User-Agent` even if `urllib2` isn't available
- Match any variation of `backend.userland.com/rss` namespace

Ultra-liberal RSS Parser 2.2 was released on January 27, 2003.

- added attribute support and admin:generatorAgent. start_admingeneratoragent is an example of how to handle elements with only attributes, no content.

Ultra-liberal RSS Parser 2.1 was released on November 14, 2002.

- added gzip support

Ultra-liberal RSS Parser 2.0.2 was released on October 21, 2002.

- added the `inchannel` to the `if` statement, otherwise it's useless. Fixes the problem JD was addressing by adding it. (JB)

Ultra-liberal RSS Parser 2.0.1 was released on October 21, 2002.

- changed `parse()` so that if we don't get anything because of `etag`/`modified`, return the old `etag`/`modified` to the caller to indicate why nothing is being returned

Ultra-liberal RSS Parser 2.0 was released on October 19, 2002.

- use `inchannel` to watch out for image and textinput elements which can also contain title, link, and description elements (JD)
- check for isPermaLink='false' attribute on guid elements (JD)
- replaced `openAnything` with `open_resource` supporting `ETag` and `If-Modified-Since` request headers (JD)
- `parse` now accepts `etag`, `modified`, `agent`, and `referrer` optional arguments (JD)
- modified `parse` to return a dictionary instead of a tuple so that any `etag` or `modified` information can be returned and cached by the caller

Ultra-liberal RSS Parser 1.1 was released on September 27, 2002.

- fixed infinite loop on incomplete CDATA sections

Ultra-liberal RSS Parser 1.0 was released on September 27, 2002.

- fixed namespace processing on prefixed RSS 2.0 elements
- added Simon Fell's namespace test suite

Ultra-liberal RSS Parser was first released on August 13, 2002.

Announcement:

> Aaron Swartz has been looking for an ultra-liberal RSS parser. Now that I'm experimenting with a homegrown RSS-to-email news aggregator, so am I. You see, most RSS feeds suck. Invalid characters, unescaped ampersands (Blogger feeds), invalid entities (Radio feeds), unescaped and invalid HTML (The Register's feed most days). Or just a bastardized mix of RSS 0.9x elements with RSS 1.0 elements (Movable Type feeds).
>
> Then there are feeds, like Aaron's feed, which are too bleeding edge. He puts an excerpt in the description element but puts the full text in the content:encoded element (as CDATA). This is valid RSS 1.0, but nobody actually uses it (except Aaron), few news aggregators support it, and many parsers choke on it. Other parsers are confused by the new elements (guid) in RSS 0.94 (see Dave Winer's feed for an example). And then there's Jon Udell's feed, with the fullitem element that he just sort of made up.
>
> `rssparser.py`. GPL-licensed. Tested on 5000 active feeds.

CHAPTER 6

Microformats

An emerging trend in feed syndication is the inclusion of microformats. Besides the semantics defined by individual feed formats, publishers can add additional semantics using rel and class attributes in embedded HTML content.

Note: To parse microformats. **`Universal Feed Parser`** relies on a third-party library called Beautiful Soup, which is distributed separately. If Beautiful Soup is not installed, **`Universal Feed Parser`** will silently skip microformats parsing.

The following elements are parsed for microformats:

- *entries[i].summary_detail.value*
- *entries[i].content[j].value*

rel=enclosure

The rel=enclosure microformat provides a way for embedded HTML content to specify that a certain link should be treated as an *enclosure*. **`Universal Feed Parser`** looks for links within embedded markup that meet any of the following conditions:

- rel attribute contains enclosure (note: rel attributes can contain a list of space-separated values)
- type attribute starts with audio/
- type attribute starts with video/
- type attribute starts with application/ but does not end with xml
- href attribute ends with one of the following file extensions: `.7z`, `.avi`, `.bin`, `.bz2`, `.bz2`, `.deb`, `.dmg`, `.exe`, `.gz`, `.hqx`, `.img`, `.iso`, `.jar`, `.m4a`, `.m4v`, `.mp2`, `.mp3`, `.mp4`, `.msi`, `.ogg`, `.ogm`, `.rar`, `.rpm`, `.sit`, `.sitx`, `.tar`, `.tbz2`, `.tgz`, `.wma`, `.wmv`, `.z`, `.zip`

When **`Universal Feed Parser`** finds a link that satisfies any of these conditions, it adds it to *entries[i].enclosures*.

Parsing embedded enclosures

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rel-enclosure.xml')
>>> d.entries[0].enclosures
[{u'href': u'http://example.com/movie.mp4', 'title': u'awesome movie'}]
```

rel=tag

The rel=tag microformat allows you to define *tags* within embedded HTML content. **Universal Feed Parser** looks for these attribute values in embedded markup and maps them to *entries[i].tags*.

Parsing embedded tags

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/rel-tag.xml')
>>> d.entries[0].tags
[{'term': u'tech', 'scheme': u'http://del.icio.us/tag/', 'label': u'Technology'}]
```

XFN (XHTML Friends Network)

The XFN microformat allows you to define human relationships between URIs. For example, you could link from your weblog to your spouse's weblog with the `rel="spouse"` relation. It is intended primarily for "blogrolls" or other static lists of links, but the relations can occur anywhere in HTML content. If found, **Universal Feed Parser** will return the XFN information in *entries[i].xfn*.

Universal Feed Parser supports all of the relationships listed in the XFN 1.1 profile, as well as the following variations:

- `coworker` in addition to `co-worker`
- `coresident` in addition to `co-resident`
- `relative` in addition to `kin`
- `brother` and `sister` in addition to `sibling`
- `husband` and `wife` in addition to `spouse`

Parsing XFN relationships

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/xfn.xml')
>>> person = d.entries[0].xfn[0]
>>> person.name
u'John Doe'
>>> person.href
u'http://example.com/johndoe'
>>> person.relationships
[u'coworker', u'friend']
```

hCard

The hCard microformat allows you to embed address book information within HTML content. If **Universal Feed Parser** finds an hCard within supported elements, it converts it into an RFC 2426-compliant vCard and returns it in *entries[i].vcard*.

Converting embedded hCard markup into a vCard

```
>>> import feedparser
>>> d = feedparser.parse('http://feedparser.org/docs/examples/hcard.xml')
>>> print d.entries[0].vcard
BEGIN:vCard
VERSION:3.0
FN:Frank Dawson
N:Dawson;Frank
ADR;TYPE=work,postal,parcel:;;6544 Battleford Drive;Raleigh;NC;27613-3502;U
.S.A.
TEL;TYPE=WORK,VOICE,MSG:+1-919-676-9515
TEL;TYPE=WORK,FAX:+1-919-676-9564
EMAIL;TYPE=internet,pref:Frank_Dawson at Lotus.com
EMAIL;TYPE=internet:fdawson at earthlink.net
ORG:Lotus Development Corporation
URL:http://home.earthlink.net/~fdawson
END:vCard
BEGIN:vCard
VERSION:3.0
FN:Tim Howes
N:Howes;Tim
ADR;TYPE=work:;;501 E. Middlefield Rd.;Mountain View;CA;94043;U.S.A.
TEL;TYPE=WORK,VOICE,MSG:+1-415-937-3419
TEL;TYPE=WORK,FAX:+1-415-528-4164
EMAIL;TYPE=internet:howes at netscape.com
ORG:Netscape Communications Corp.
END:vCard
```

Note: There are a growing number of microformats, and **Universal Feed Parser** does not parse all of them. However, both the rel and class attributes survive *HTML sanitizing*, so applications built on **Universal Feed Parser** that wish to parse additional microformat content are free to do so.

See also:

- Microformats.org
- rel=enclosure specification
- rel=tag specification
- XFN specification
- hCard specification

CHAPTER 7

Reference

bozo

An integer, either `1` or `0`. Set to `1` if the feed is not well-formed XML, and `0` otherwise.

See *Bozo Detection* for more details on the `bozo` bit.

Tip: `bozo` may not be present. Some platforms, such as Mac OS X 10.2 and some versions of FreeBSD, do not include an XML parser in their **`Python`** distributions. **`Universal Feed Parser`** will still work on these platforms, but it will not be able to detect whether a feed is well-formed. However, it *can* detect whether a feed's character encoding is incorrectly declared. (This is done in **`Python`**, not by the XML parser.) See *Character Encoding Detection* for details.

---

bozo_exception

The exception raised when attempting to parse a non-well-formed feed.

See *Bozo Detection* for more details.

Tip: `bozo_exception` will only be present if `bozo` is `1`.

---

encoding

The character encoding that was used to parse the feed.

Note: The process by which **Universal Feed Parser** determines the character encoding of the feed is explained in *Character Encoding Detection*.

Tip: This element always exists, although it may be an empty string if the character encoding cannot be determined.

entries

A list of dictionaries. Each dictionary contains data from a different entry. Entries are listed in the order in which they appear in the original feed.

Tip: This element always exists, although it may be an empty list.

Comes from

- /atom03:feed/atom03:entry
- /atom10:feed/atom10:entry
- /rdf:RDF/rdf:item
- /rss/channel/item

entries[i].author

The author of this entry.

See also:

- *entries[i].author_detail*

Comes from

- /atom10:feed/atom10:entry/atom10:author
- /atom03:feed/atom03:entry/atom03:author
- /rss/channel/item/dc:creator
- /rss/channel/item/dc:author
- /rss/channel/itunes:author
- /rdf:RDF/rdf:item/dc:creator
- /rdf:RDF/rdf:item/dc:author

entries[i].author_detail

A dictionary with details about the author of this entry.

See also:

- *entries[i].author*

entries[i].author_detail.name

The name of this entry's author.

entries[i].author_detail.href

The URL of this entry's author. This can be the author's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].author_detail.email

The email address of this entry's author.

Comes from

- /atom10:feed/atom10:entry/atom10:author
- /atom03:feed/atom03:entry/atom03:author
- /rss/channel/item/dc:creator
- /rss/channel/item/dc:author
- /rss/channel/itunes:author
- /rdf:RDF/rdf:item/dc:creator
- /rdf:RDF/rdf:item/dc:author

entries[i].comments

A URL of the HTML comment submission page associated with this entry.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /rss/channel/item/comments

entries[i].content

A list of dictionaries with details about the full content of the entry.

Atom feeds may contain multiple content elements. Clients should render as many of them as possible, based on the type and the client's abilities.

entries[i].content[j].value

The value of this piece of content.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

If this contains HTML or XHTML, it will be *parsed for microformats*.

entries[i].content[j].type

The content type of this piece of content.

Most likely values for *type*:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].content[j].language

The language of this piece of content.

language is supposed to be a language code, as specified by **RFC 3066**, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

language may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the *Content-Language* HTTP header. If the feed does not specify a language, language will be None, the **Python** null value.

entries[i].content[j].base

The original base URI for links within this piece of content.

base is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of

the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

Comes from

- /atom03:feed/atom03:entry/atom03:content
- /atom10:feed/atom10:entry/atom10:content
- /rdf:RDF/rdf:item/content:encoded
- /rss/channel/item/body
- /rss/channel/item/content:encoded
- /rss/channel/item/fullitem
- /rss/channel/item/xhtml:body

entries[i].contributors

A list of contributors (secondary authors) to this entry.

entries[i].contributors[j].name

The name of this contributor.

entries[i].contributors[j].href

The URL of this contributor. This can be the contributor's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].contributors[j].email

The email address of this contributor.

Comes from

- /atom03:feed/atom03:entry/atom03:contributor
- /atom10:feed/atom10:entry/atom10:contributor
- /rss/channel/item/dc:contributor

entries[i].created

The date this entry was first created (drafted), as a string in the same format as it was published in the original feed).

This element is *parsed as a date* and stored in *entries[i].created_parsed*.

Comes from

- /atom03:feed/atom03:entry/atom03:created
- /rdf:RDF/rdf:item/dcterms:created
- /rss/channel/item/dcterms:created

See also:

- *entries[i].created_parsed*

entries[i].created_parsed

The date this entry was first created (drafted), as a standard **Python** 9-tuple.

Comes from

- /atom03:feed/atom03:entry/atom03:created
- /rdf:RDF/rdf:item/dcterms:created
- /rss/channel/item/dcterms:created

See also:

- *entries[i].created*

entries[i].enclosures

A list of links to external files associated with this entry.

Some aggregators automatically download enclosures (although this technique has known problems). Some aggregators render each enclosure as a link. Most aggregators ignore them.

The RSS specification states that there can be at most one enclosure per item. However, because some feeds break this rule, **Universal Feed Parser** captures all of them and makes them available as a list.

Comes from

- /atom10:feed/atom10:entry/atom10:link[@rel="enclosure"]
- /rss/channel/item/enclosure
- additionally, *certain links within embedded markup*

entries[i].enclosures[j].href

The URL of the linked file.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].enclosures[j].length

The length of the linked file.

entries[i].enclosures[j].type

The content type of the linked file.

entries[i].expired

The date this entry is set to expire, as a string in the same format as it was published in the original feed).

This element is *parsed as a date* and stored in *entries[i].expired_parsed*.

This element is rare. It only existed in RSS 0.93, and it was never widely implemented by publishers. Most clients ignore it in favor of user-defined expiration algorithms.

Comes from

- /rss/channel/item/expirationDate

See also:

- *entries[i].expired_parsed*

entries[i].expired_parsed

The date this entry is set to expire, as a standard **Python** 9-tuple.

This element is rare. It only existed in RSS 0.93, and it was never widely implemented by publishers. Most clients ignore it in favor of user-defined expiration algorithms.

Comes from

- /rss/channel/item/expirationDate

See also:

- *entries[i].expired*

entries[i].id

A globally unique identifier for this entry.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom03:feed/atom03:entry/atom03:id
- /atom10:feed/atom10:entry/atom10:id
- /rdf:RDF/rdf:item/@rdf:about
- /rss/channel/item/guid

entries[i].license

A URL of the license under which this entry is distributed.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom10:feed/atom10:entry/atom10:link[@rel="license"]/@href
- /rdf:RDF/rdf:item/cc:license/@rdf:resource
- /rss/channel/item/creativeCommons:license

entries[i].link

The primary link of this entry. Most feeds use this as the permanent link to the entry in the site's archives.

If this is a relative URI, it is *resolved according to a set of rules*.

Some RSS feeds use guid when they mean link. guid can also be used as an opaque identifier that has nothing to do with links. If an RSS feed uses guid as the entry link and no link is present, **Universal Feed Parser** detects this and makes the guid available in `entries[i].link`.

In other words, you can always use `entries[i].link` to get the entry link, regardless of how the feed is actually structured.

Comes from

- /atom03:feed/atom03:entry/atom03:link[@rel="alternate"]/@href
- /atom10:feed/atom10:entry/atom10:link[@rel="alternate"]/@href
- /atom10:feed/atom10:entry/atom10:link[not(@rel)]/@href
- /rdf:RDF/rdf:item/rdf:link
- /rss/channel/item/link

See also:

- *entries[i].links*

entries[i].links

A list of dictionaries with details on the links associated with the feed. Each link has a rel (relationship), type (content type), and href (the URL that the link points to). Some links may also have a title.

entries[i].links[j].rel

The relationship of this entry link.

Atom 1.0 defines five standard link relationships and describes the process for registering others. Here are the five standard rel values:

- *alternate*
- *enclosure*
- *related*
- *self*
- *via*

entries[i].links[j].type

The content type of the page that this entry link points to.

entries[i].links[j].href

The URL of the page that this entry link points to.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].links[j].title

The title of this entry link.

Comes from

- /atom03:feed/atom03:entry/atom03:link
- /atom10:feed/atom10:entry/atom10:link
- /rdf:RDF/rdf:item/rdf:link
- /rss/channel/item/link

See also:

- *entries[i].link*

entries[i].published

The date this entry was first published, as a string in the same format as it was published in the original feed.

This element is *parsed as a date* and stored in *entries[i].published_parsed*.

Comes from

- /atom10:feed/atom10:entry/atom10:published
- /atom03:feed/atom03:entry/atom03:issued
- /rss/channel/item/dcterms:issued
- /rss/channel/item/pubDate
- /rdf:RDF/rdf:item/dcterms:issued

See also:

- *entries[i].published_parsed*

entries[i].published_parsed

The date this entry was first published, as a standard **Python** 9-tuple.

Comes from

- /atom10:feed/atom10:entry/atom10:published
- /atom03:feed/atom03:entry/atom03:issued
- /rss/channel/item/dcterms:issued
- /rdf:RDF/rdf:item/dcterms:issued
- /rss/channel/item/pubDate

See also:

- *entries[i].published*

entries[i].publisher

The publisher of the entry.

Comes from

- /rss/item/dc:publisher
- /rss/item/itunes:owner
- /rdf:RDF/rdf:item/dc:publisher

See also:

- *entries[i].publisher_detail*

entries[i].publisher_detail

A dictionary with details about the entry publisher.

entries[i].publisher_detail.name

The name of this entry's publisher.

entries[i].publisher_detail.href

The URL of this entry's publisher. This can be the publisher's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].publisher_detail.email

The email address of this entry's publisher.

Comes from

- /rss/item/dc:publisher
- /rss/item/itunes:owner
- /rdf:RDF/rdf:item/dc:publisher

See also:

- *entries[i].publisher*

entries[i].source

A dictionary with details about the source of the entry.

Comes from

- /atom10:feed/atom10:entry/atom10:source

entries[i].source.author

The author of the source of this entry.

entries[i].source.author_detail

A dictionary containing details about the author of the source of this entry.

entries[i].source.author_detail.name

The name of the author of the source of this entry.

entries[i].source.author_detail.href

The URL of the author of the source of this entry. This can be the author's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].source.author_detail.email

The email address of the author of the source of this entry.

entries[i].source.contributors

A list of contributors to the source of this entry.

entries[i].source.contributors[j].name

The name of a contributor to the source of this entry.

entries[i].source.contributors[j].href

The URL of a contributor to the source of this entry. This can be the contributor's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].source.contributors[j].email

The email address of a contributor to the source of this entry.

entries[i].source.icon

The URL of an icon representing the source of this entry.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].source.id

A globally unique identifier for the source of this entry.

entries[i].source.link

The primary permanent link of the source of this entry

entries[i].source.links

A list of all links defined by the source of this entry.

entries[i].source.links[j].rel

The relationship of a link defined by the source of this entry.

Atom 1.0 defines five standard link relationships and describes the process for registering others. Here are the five standard rel values:

- `alternate`
- `self`
- `related`
- `via`
- `enclosure`

entries[i].source.links[j].type

The content type of the page pointed to by a link defined by the source of this entry.

entries[i].source.links[j].href

The URL of the page pointed to by a link defined by the source of this entry.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].source.links[j].title

The title of a link defined by the source of this entry.

entries[i].source.logo

The URL of a logo representing the source of this entry.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].source.rights

A human-readable copyright statement for the source of this entry.

entries[i].source.rights_detail

A dictionary containing details about the copyright statement for the source of this entry.

entries[i].source.rights_detail.value

Same as *entries[i].source.rights*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].source.rights_detail.type

The content type of the copyright statement for the source of this entry.

Most likely values for `type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].source.rights_detail.language

The language of the copyright statement for the source of this entry.

`language` is supposed to be a language code, as specified by RFC 3066, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

entries[i].source.rights_detail.base

The original base URI for links within the copyright statement for the source of this entry.

`entries[i].source.rights_detail.base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

entries[i].source.subtitle

A subtitle, tagline, slogan, or other short description of the source of this entry.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].source.subtitle_detail

A dictionary containing details about the subtitle for the source of this entry.

entries[i].source.subtitle_detail.value

Same as *entries[i].source.subtitle*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].source.subtitle_detail.type

The content type of the subtitle of the source of this entry.

Most likely values for `type`:

- *`text/plain`*
- *`text/html`*
- *`application/xhtml+xml`*

For Atom feeds, the content type is taken from the type attribute, which defaults to *`text/plain`* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *`text/html`*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *`text/plain`*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].source.subtitle_detail.language

The language of the subtitle of the source of this entry.

`language` is supposed to be a language code, as specified by RFC 3066, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

entries[i].source.subtitle_detail.base

The original base URI for links within the subtitle of the source of this entry.

`entries[i].source.subtitle_detail.base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time

you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

entries[i].source.title

The title of the source of this entry.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].source.title_detail

A dictionary containing details about the title for the source of this entry.

entries[i].source.title_detail.value

Same as *entries[i].source.title*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].source.title_detail.type

The content type of the title of the source of this entry.

Most likely values for `entries[i].source.title_detail.type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].source.title_detail.language

The language of the title of the source of this entry.

`language` is supposed to be a language code, as specified by RFC 3066, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

entries[i].source.title_detail.base

The original base URI for links within the title of the source of this entry.

`entries[i].source.title_detail.base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

entries[i].source.updated

The date the source of this entry was last updated, as a string in the same format as it was published in the original feed.

This element is *parsed as a date* and stored in *entries[i].source.updated_parsed*.

entries[i].source.updated_parsed

The date this entry was last updated, as a standard **Python** 9-tuple.

entries[i].summary

A summary of the entry.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

Some publishing systems auto-generate this value from the first few words or first paragraph of the entry. Other publishing systems misuse it to include the full content. In the latter cases, **Universal Feed Parser** ought to detect it and put the value in *entries[i].content* instead, but it doesn't.

Note: Some feeds include both a summary and description element for each entry. In this case, the first element will be available in `entry['summary']` and the second will be available in `entry['content'][0]`.

Comes from

- /atom10:feed/atom10:entry/atom10:summary
- /atom03:feed/atom03:entry/atom03:summary
- /rss/channel/item/description
- /rss/channel/item/dc:description
- /rdf:RDF/rdf:item/rdf:description
- /rdf:RDF/rdf:item/dc:description

See also:

- *entries[i].summary_detail*

entries[i].summary_detail

A dictionary with details about the entry summary.

Comes from

- /atom10:feed/atom10:entry/atom10:summary
- /atom03:feed/atom03:entry/atom03:summary
- /rss/channel/item/description
- /rss/channel/item/dc:description
- /rdf:RDF/rdf:item/rdf:description
- /rdf:RDF/rdf:item/dc:description

See also:

- *entries[i].summary*

entries[i].summary_detail.value

Same as *entries[i].summary*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

If this contains HTML or XHTML, it will be *parsed for microformats*.

entries[i].summary_detail.type

The content type of the entry summary.

Most likely values for `type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].summary_detail.language

The language of the entry summary.

`language` is supposed to be a language code, as specified by RFC 3066, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

entries[i].summary_detail.base

The original base URI for links within the entry summary.

`base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

entries[i].tags

A list of dictionaries that contain details of the categories for the entry.

Note: Prior to version 4.0, **Universal Feed Parser** exposed categories in `feed.category` (the primary category) and `feed.categories` (a list of tuples containing the domain and term of each category). These uses are still supported for backward compatibility, but you will not see them in the parsed results unless you explicitly ask for them.

---

entries[i].tags[j].term

The category term (keyword).

entries[i].tags[j].scheme

The category scheme (domain).

entries[i].tags[j].label

A human-readable label for the category.

Comes from

- /atom10:feed/atom10:entry/category
- /atom03:feed/atom03:entry/dc:subject

- /rss/channel/item/category
- /rss/channel/item/dc:subject
- /rss/channel/item/itunes:category
- /rss/channel/item/itunes:keywords
- /rdf:RDF/rdf:channel/rdf:item/dc:subject

entries[i].title

The title of the entry.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

Comes from

- /atom03:feed/atom03:entry/atom03:title
- /atom10:feed/atom10:entry/atom10:title
- /rdf:RDF/rdf:item/dc:title
- /rdf:RDF/rdf:item/rdf:title
- /rss/channel/item/dc:title
- /rss/channel/item/title

See also:

- *entries[i].title_detail*

entries[i].title_detail

A dictionary with details about the entry title.

entries[i].title_detail.value

Same as *entries[i].title*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

entries[i].title_detail.type

The content type of the entry title.

Most likely values for `type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

entries[i].title_detail.language

The language of the entry title.

`language` is supposed to be a language code, as specified by RFC 3066, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

entries[i].title_detail.base

The original base URI for links within the entry title.

`base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

Comes from

- /atom10:feed/atom10:entry/atom10:title
- /atom03:feed/atom03:entry/atom03:title
- /rss/channel/item/title
- /rss/channel/item/dc:title
- /rdf:RDF/rdf:item/rdf:title
- /rdf:RDF/rdf:item/dc:title

See also:

- *entries[i].title*

entries[i].updated

The date this entry was last updated, as a string in the same format as it was published in the original feed).

This element is *parsed as a date* and stored in *entries[i].updated_parsed*.

Note: As of version 5.1.1, if this key doesn't exist but `entries[i].published` does, the value of `entries[i].published` will be returned.

In the past the RSS pubDate element was stored in *updated*, but this incorrect behavior was reported in issue 310. However, developers may have come to rely on this incorrect behavior – as was reported in issue 328 – so to help avoid hurting their users' experience, this mapping from *updated* to *published* was temporarily introduced to give developers time to update their software, and to give users time to upgrade.

This mapping is temporary and will be removed in a future version of feedparser.

Comes from

- /atom03:feed/atom03:entry/atom03:modified
- /atom10:feed/atom10:entry/atom10:updated
- /rdf:RDF/rdf:item/dc:date
- /rdf:RDF/rdf:item/dcterms:modified
- /rss/channel/item/dc:date
- /rss/channel/item/dcterms:modified

See also:

- *entries[i].updated_parsed*

entries[i].updated_parsed

The date this entry was last updated, as a standard **Python** 9-tuple.

Note: As of version 5.1.1, if this key doesn't exist but `entries[i].published_parsed` does, the value of `entries[i].published_parsed` will be returned.

In the past the RSS pubDate element was stored in *updated*, but this incorrect behavior was reported in issue 310. However, developers may have come to rely on this incorrect behavior – as was reported in issue 328 – so to help avoid hurting their users' experience, this mapping from *updated_parsed* to *published_parsed* was temporarily introduced to give developers time to update their software, and to give users time to upgrade.

This mapping is temporary and will be removed in a future version of feedparser.

Comes from

- /atom10:feed/atom10:entry/atom10:updated
- /atom03:feed/atom03:entry/atom03:modified

- /rss/channel/item/dc:date
- /rss/channel/item/dcterms:modified
- /rdf:RDF/rdf:item/dc:date
- /rdf:RDF/rdf:item/dcterms:modified

See also:

- *entries[i].updated*

entries[i].vcard

An RFC 2426-compliant vCard derived from *hCard information* found in this entry's HTML content.

Comes from

- /atom03:feed/atom03:entry/atom03:content
- /atom03:feed/atom03:entry/atom03:summary
- /atom10:feed/atom10:entry/atom10:content
- /atom10:feed/atom10:entry/atom10:summary
- /rdf:RDF/rdf:item/content:encoded
- /rdf:RDF/rdf:item/dc:description
- /rdf:RDF/rdf:item/rdf:description
- /rss/channel/item/body
- /rss/channel/item/content:encoded
- /rss/channel/item/dc:description
- /rss/channel/item/description
- /rss/channel/item/fullitem
- /rss/channel/item/xhtml:body

entries[i].xfn

A list of *XFN relationships* found in this entry's HTML content.

Comes from

- /atom03:feed/atom03:entry/atom03:content
- /atom03:feed/atom03:entry/atom03:summary
- /atom10:feed/atom10:entry/atom10:content
- /atom10:feed/atom10:entry/atom10:summary
- /rdf:RDF/rdf:item/content:encoded

- /rdf:RDF/rdf:item/dc:description
- /rdf:RDF/rdf:item/rdf:description
- /rss/channel/item/body
- /rss/channel/item/content:encoded
- /rss/channel/item/dc:description
- /rss/channel/item/description
- /rss/channel/item/fullitem
- /rss/channel/item/xhtml:body

entries[i].xfn is a list. Each list item represents a single person and may contain the following values:

entries[i].xfn[j].relationships

A list of relationships for this person. Each list item is a string, either one of the constants defined in the XFN 1.1 profile or *one of these variations*.

entries[i].xfn[j].href

The URI for this person.

If this is a relative URI, it is *resolved according to a set of rules*.

entries[i].xfn[j].name

The name of this person, a string.

etag

The ETag of the feed, as specified in the HTTP headers.

The purpose of `etag` is explained more fully in *ETag and Last-Modified Headers*.

Tip: `etag` will only be present if the feed was retrieved from a web server, and only if the web server provided an ETag HTTP header for the feed. If the feed was parsed from a local file or from a string in memory, `etag` will not be present.

---

feed

A dictionary of data about the feed.

Comes from

- /atom03:feed
- /atom10:feed
- /rdf:RDF/rdf:channel
- /rss/channel

Tip: This element always exists, although it may be an empty dictionary.

feed.author

The author of this feed.

Comes from

- /atom03:feed/atom03:author
- /atom10:feed/atom10:author
- /rdf:RDF/rdf:channel/dc:author
- /rdf:RDF/rdf:channel/dc:creator
- /rss/channel/dc:author
- /rss/channel/dc:creator
- /rss/channel/itunes:author
- /rss/channel/managingEditor

See also:

- *feed.author_detail*

feed.author_detail

A dictionary with details about the feed author.

feed.author_detail.name

The name of the feed author.

feed.author_detail.href

The URL of the feed author. This can be the author's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.author_detail.email

The email address of the feed author.

Comes from

- /atom03:feed/atom03:author
- /atom10:feed/atom10:author
- /rdf:RDF/rdf:channel/dc:author
- /rdf:RDF/rdf:channel/dc:creator
- /rss/channel/dc:author
- /rss/channel/dc:creator
- /rss/channel/itunes:author
- /rss/channel/managingEditor

See also:

- *feed.author*

feed.cloud

No one really knows what a cloud is. It is vaguely documented in *:abbr:‘SOAP (Simple Object Access Protocol)* meets RSS <http://www.thetwowayweb.com/soapmeetsrss>‘_.

feed.cloud.domain

The domain of the cloud. Should be just the domain name, not including the http:// protocol. All clouds are presumed to operate over HTTP. The cloud specification does not support secure clouds over HTTPS, nor can clouds operate over other protocols.

feed.cloud.port

The port of the cloud. Should be an integer, but **Universal Feed Parser** currently returns it as a string.

feed.cloud.path

The URL path of the cloud.

feed.cloud.registerProcedure

The name of the procedure to call on the cloud.

feed.cloud.protocol

The protocol of the cloud. Documentation differs on what the acceptable values are. Acceptable values definitely include xml-rpc and soap, although only in lowercase, despite both being acronyms.

There is no way for a publisher to specify the version number of the protocol to use. soap refers to SOAP (Simple Object Access Protocol) 1.1; the cloud interface does not support SOAP 1.0 or 1.2.

post or http-post might also be acceptable values; nobody really knows for sure.

Comes from

- /rss/channel/cloud

feed.contributors

A list of contributors (secondary authors) to this feed.

feed.contributors[i].name

The name of this contributor.

feed.contributors[i].href

The URL of this contributor. This can be the contributor's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.contributors[i].email

The email address of this contributor.

Comes from

- /atom03:feed/atom03:contributor
- /atom10:feed/atom10:contributor
- /rss/channel/dc:contributor

feed.docs

A URL pointing to the specification which this feed conforms to.

This element is rare. The reasoning was that in 25 years, someone will stumble on an RSS feed and not know what it is, so we should waste everyone's bandwidth with useless links until then. Most publishers skip it, and all clients ignore it.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /rss/channel/docs

feed.errorreportsto

An email address for reporting errors in the feed itself.

Comes from

- /rdf:RDF/admin:errorReportsTo/@rdf:resource

feed.generator

A human-readable name of the application used to generate the feed.

Comes from

- /atom03:feed/atom03:generator
- /atom10:feed/atom10:generator
- /rdf:RDF/rdf:channel/admin:generatorAgent/@rdf:resource
- /rss/channel/generator

See also:

- *feed.generator_detail*

feed.generator_detail

A dictionary with details about the feed generator.

feed.generator_detail.name

Same as *feed.generator*.

feed.generator_detail.href

The URL of the application used to generate the feed.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.generator_detail.version

The version number of the application used to generate the feed. There is no required format for this, but most applications use a MAJOR.MINOR version number.

Comes from

- /atom03:feed/atom03:generator
- /atom10:feed/atom10:generator
- /rdf:RDF/rdf:channel/admin:generatorAgent/@rdf:resource
- /rss/channel/generator

See also:

- *feed.generator*

feed.icon

A URL to a small icon representing the feed.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom10:feed/atom10:icon

feed.id

A globally unique identifier for this feed.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom03:feed/atom03:id
- /atom10:feed/atom10:id

feed.image

A dictionary with details about the feed image. A feed image can be a logo, banner, or a picture of the author.
feed.image.title ————————-==========

The alternate text of the feed image, which would go in the alt attribute if you rendered the feed image as an HTML img element.

feed.image.href

The URL of the feed image itself, which would go in the src attribute if you rendered the feed image as an HTML img element.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.image.link

The URL which the feed image would point to. If you rendered the feed image as an HTML img element, you would wrap it in an a element and put this in the href attribute.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.image.width

The width of the feed image, which would go in the width attribute if you rendered the feed image as an HTML img element.

feed.image.height

The height of the feed image, which would go in the height attribute if you rendered the feed image as an HTML img element.

feed.image.description

A short description of the feed image, which would go in the title attribute if you rendered the feed image as an HTML img element. This element is rare; it was available in Netscape RSS 0.91 but was dropped from Userland RSS 0.91.

Annotated example

This is a feed image:

```
<image>
<title>Feed logo</title>
<url>http://example.org/logo.png</url>
<link>http://example.org/</link>
<width>80</width>
<height>15</height>
<description>Visit my home page</description>
</image>
```

This feed image could be rendered in HTML as this:

```
<a href="http://example.org/">
<img src="http://example.org/logo.png"
width="80"
height="15"
alt="Feed logo"
title="Visit my home page">
</a>
```

Comes from

- /rdf:RDF/rdf:image
- /rss/channel/image

feed.info

Free-form human-readable description of the feed format itself. Intended for people who view the feed in a browser, to explain what they just clicked on. This element is generally ignored by feed readers.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

Comes from

- /atom03:feed/atom03:info
- /rss/channel/feedburner:browserFriendly

See also:

- *feed.info_detail*

feed.info_detail

A dictionary with details about the feed info.

Comes from

- /atom03:feed/atom03:info

See also:

- *feed.info*

feed.info_detail.value

Same as *feed.info*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

feed.info_detail.type

The content type of the feed info.

Most likely values for `type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

feed.info_detail.language

The language of the feed info.

`language` is supposed to be a language code, as specified by *:abbr:'RFC (Request For Comments)* 3066 <http://www.ietf.org/rfc/rfc3066.txt>'_, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

feed.info_detail.base

The original base URI for links within the feed copyright.

`base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

feed.language

The primary language of the feed.

Comes from

- /atom03:feed/@xml:lang
- /atom10:feed/@xml:lang
- /rdf:RDF/rdf:channel/dc:language
- /rss/channel/dc:language

- /rss/channel/language

feed.license

A URL of the license under which this feed is distributed.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom10:feed/atom10:link[@rel="license"]/@href
- /rdf:RDF/cc:license/@rdf:resource
- /rss/channel/creativeCommons:license

feed.link

The URL of the HTML page associated with this feed.

For site feeds, this is probably the home page of the site. For category feeds, this is probably the category's archive page. For search feeds, this is probably the web page that displays the search results for the given search parameters.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom03:feed/atom03:link[@rel="alternate"]/@href
- /atom10:feed/atom10:link[@rel="alternate"]/@href
- /atom10:feed/atom10:link[not(@rel)]/@href
- /rdf:RDF/rdf:channel/rdf:link
- /rss/channel/link

See also:

- *feed.links*

feed.links

A list of dictionaries with details on the links associated with the feed. Each link has a rel (relationship), type (content type), and href (the URL that the link points to). Some links may also have a title.

feed.links[i].rel

The relationship of this feed link.

Atom 1.0 defines five standard link relationships and describes the process for registering others. Here are the five standard rel values:

- *alternate*
- *enclosure*
- *related*
- *self*
- *via*

feed.links[i].type

The content type of the page that this feed link points to.

feed.links[i].href

The URL of the page that this feed link points to.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.links[i].title

The title of this feed link.

Comes from

- /atom03:feed/atom03:link
- /atom10:feed/atom10:link
- /rdf:RDF/rdf:channel/rdf:link
- /rss/channel/link

See also:

- *feed.link*

feed.logo

A URL to a graphic representing a logo for the feed.

If this is a relative URI, it is *resolved according to a set of rules*.

Comes from

- /atom10:feed/atom10:logo

feed.published

The date the feed was published, as a string in the same format as it was published in the original feed.

This element is *parsed as a date* and stored in *feed.published_parsed*.

Comes from

- /rss/channel/pubDate

See also:

- *feed.published_parsed*

feed.published_parsed

The date the feed was published, as a standard **Python** 9-tuple.

Comes from

- /rss/channel/pubDate

See also:

- *feed.published*

feed.publisher

The publisher of the feed.

Comes from

- /rdf:RDF/rdf:channel/dc:publisher
- /rss/channel/dc:publisher
- /rss/channel/itunes:owner
- /rss/channel/webMaster

See also:

- *feed.publisher_detail*

feed.publisher_detail

A dictionary with details about the feed publisher.

feed.publisher_detail.name

The name of this feed's publisher.

feed.publisher_detail.href

The URL of this feed's publisher. This can be the publisher's home page, or a contact page with a webmail form.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.publisher_detail.email

The email address of this feed's publisher.

Comes from

- /rdf:RDF/rdf:channel/dc:publisher
- /rss/channel/dc:publisher
- /rss/channel/itunes:owner
- /rss/channel/webMaster

See also:

- *feed.publisher*

feed.rights

A human-readable copyright statement for the feed.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

Note: For machine-readable copyright information, see *feed.license*.

Comes from

- /atom03:feed/atom03:copyright
- /atom10:feed/atom10:rights
- /rdf:RDF/rdf:channel/dc:rights
- /rss/channel/copyright
- /rss/channel/dc:rights

See also:

- *feed.rights_detail*

feed.rights_detail

A dictionary with details on the feed copyright.

feed.rights_detail.value

Same as *feed.rights*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

feed.rights_detail.type

The content type of the feed copyright.

Most likely values for `type`:

- *`text/plain`*
- *`text/html`*
- *`application/xhtml+xml`*

For Atom feeds, the content type is taken from the type attribute, which defaults to *`text/plain`* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *`text/html`*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *`text/plain`*, and **`Universal Feed Parser`** should respect this, but it doesn't yet.

feed.rights_detail.language

The language of the feed copyright.

`language` is supposed to be a language code, as specified by *:abbr:'RFC (Request For Comments)* 3066 <http://www.ietf.org/rfc/rfc3066.txt>'_, but publishers have been known to publish random values like "English" or "German". **`Universal Feed Parser`** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **`Python`** null value.

feed.rights_detail.base

The original base URI for links within the feed copyright.

`base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of

the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **`Universal Feed Parser`** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

Comes from

- /atom03:feed/atom03:copyright
- /atom10:feed/atom10:rights
- /rdf:RDF/rdf:channel/dc:rights
- /rss/channel/copyright
- /rss/channel/dc:rights

See also:

- *feed.rights*

feed.subtitle

A subtitle, tagline, slogan, or other short description of the feed.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

Comes from

- /atom03:feed/atom03:tagline
- /atom10:feed/atom10:subtitle
- /rdf:RDF/rdf:channel/dc:description
- /rdf:RDF/rdf:channel/rdf:description
- /rss/channel/dc:description
- /rss/channel/description
- /rss/channel/itunes:subtitle

See also:

- *feed.subtitle_detail*

feed.subtitle_detail

A dictionary with details about the feed subtitle.

Comes from

- /atom03:feed/atom03:tagline
- /atom10:feed/atom10:subtitle
- /rdf:RDF/rdf:channel/dc:description
- /rdf:RDF/rdf:channel/rdf:description
- /rss/channel/dc:description
- /rss/channel/description
- /rss/channel/itunes:subtitle

See also:

- *feed.subtitle*

feed.subtitle_detail.value

Same as *feed.subtitle*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

feed.subtitle_detail.type

The content type of the feed subtitle.

Most likely values for `type`:

- *text/plain*
- *text/html*
- *application/xhtml+xml*

For Atom feeds, the content type is taken from the type attribute, which defaults to *text/plain* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *text/html*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

feed.subtitle_detail.language

The language of the feed subtitle.

`language` is supposed to be a language code, as specified by *:abbr:'RFC (Request For Comments)* 3066 <http://www.ietf.org/rfc/rfc3066.txt>'_, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

`language` may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, `language` will be `None`, the **Python** null value.

feed.subtitle_detail.base

The original base URI for links within the feed subtitle.

`base` is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

feed.tags

A list of dictionaries that contain details of the categories for the feed.

Note: Prior to version 4.0, **Universal Feed Parser** exposed categories in `feed.category` (the primary category) and `feed.categories` (a list of tuples containing the domain and term of each category). These uses are still supported for backward compatibility, but you will not see them in the parsed results unless you explicitly ask for them.

---

feed.tags[i].term

The category term (keyword).

feed.tags[i].scheme

The category scheme (domain).

feed.tags[i].label

A human-readable label for the category.

Comes from

- /atom03:feed/dc:subject
- /atom10:feed/category
- /rdf:RDF/rdf:channel/dc:subject
- /rss/channel/category
- /rss/channel/dc:subject
- /rss/channel/itunes:category
- /rss/channel/itunes:keywords

feed.textinput

A text input form. No one actually uses this. Why are you?

feed.textinput.title

The title of the text input form, which would go in the value attribute of the form's submit button.

feed.textinput.link

The link of the script which processes the text input form, which would go in the action attirbute of the form.

If this is a relative URI, it is *resolved according to a set of rules*.

feed.textinput.name

The name of the text input box in the form, which would go in the name attribute of the form's input box.

feed.textinput.description

A short description of the text input form, which would go in the label element of the form.

Annotated example

This is a text input in a feed:

```
<textInput>
<title>Go!</title>
<link>http://example.org/search</link>
<name>keyword</name>
<description>Search this site:</description>
</textInput>
```

This is how it could be rendered in HTML:

```
<form method="get" action="http://example.org/search">
<label for="keyword">Search this site:</label>
<input type="text" id="keyword" name="keyword" value="">
<input type="submit" value="Go!">
</form>
```

Comes from

- /rdf:RDF/rdf:textinput
- /rss/channel/textInput
- /rss/channel/textinput

feed.title

The title of the feed.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URI (UNIFORM RESOURCE IDENTIFIER)'S. IF SO, THEY ARE :REF:'RESOLVED ACCORDING TO A SET OF RULES <ADVANCED.BASE>.

Comes from

- /atom03:feed/atom03:title
- /atom10:feed/atom10:title
- /rdf:RDF/rdf:channel/dc:title
- /rdf:RDF/rdf:channel/rdf:title
- /rss/channel/dc:title
- /rss/channel/title

See also:

- *feed.title_detail*

feed.title_detail

A dictionary with details about the feed title.

feed.title_detail.value

Same as *feed.title*.

If this contains HTML or XHTML, it is *sanitized* by default.

If this contains HTML or XHTML, certain (X)HTML elements within this value may contain relative URIs. If so, they are *resolved according to a set of rules*.

feed.title_detail.type

The content type of the feed title.

Most likely values for `type`:

- *`text/plain`*
- *`text/html`*
- *`application/xhtml+xml`*

For Atom feeds, the content type is taken from the type attribute, which defaults to *`text/plain`* if not specified. For RSS feeds, the content type is auto-determined by inspecting the content, and defaults to *`text/html`*. Note that this may cause silent data loss if the value contains plain text with angle brackets. There is nothing I can do about this problem; it is a limitation of RSS.

Future enhancement: some versions of RSS clearly specify that certain values default to *text/plain*, and **Universal Feed Parser** should respect this, but it doesn't yet.

feed.title_detail.language

The language of the feed title.

language is supposed to be a language code, as specified by *:abbr:'RFC (Request For Comments)* 3066 <http://www.ietf.org/rfc/rfc3066.txt>'_, but publishers have been known to publish random values like "English" or "German". **Universal Feed Parser** does not do any parsing or normalization of language codes.

language may come from the element's xml:lang attribute, or it may inherit from a parent element's xml:lang, or the Content-Language HTTP header. If the feed does not specify a language, language will be None, the **Python** null value.

feed.title_detail.base

The original base URI for links within the feed title.

base is only useful in rare situations and can usually be ignored. It is the original base URI for this value, as specified by the element's xml:base attribute, or a parent element's xml:base, or the appropriate HTTP header, or the URI of the feed. (See *Relative Link Resolution* for more details.) By the time you see it, **Universal Feed Parser** has already resolved relative links in all values where it makes sense to do so. *Clients should never need to manually resolve relative links.*

Comes from

- /atom03:feed/atom03:title
- /atom10:feed/atom10:title
- /rdf:RDF/rdf:channel/dc:title
- /rdf:RDF/rdf:channel/rdf:title
- /rss/channel/dc:title
- /rss/channel/title

See also:

- *feed.title*

feed.ttl

According to the RSS specification, "None"

No one is quite sure what this means, and no one publishes feeds via file-sharing networks.

Some clients have interpreted this element to be some sort of inline caching mechanism, albeit one that completely ignores the underlying HTTP protocol, its robust caching mechanisms, and the huge amount of HTTP-savvy network infrastructure that understands them. Given the vague documentation, it is impossible to say that this interpretation is any more ridiculous than the element itself.

Comes from

- /rss/channel/ttl

feed.updated

The date the feed was last updated, as a string in the same format as it was published in the original feed.

This element is *parsed as a date* and stored in *feed.updated_parsed*.

Note: As of version 5.1.1, if this key doesn't exist but `feed.published` does, the value of `feed.published` will be returned.

In the past the RSS pubDate element was stored in *updated*, but this incorrect behavior was reported in issue 310. However, developers may have come to rely on this incorrect behavior – as was reported in issue 328 – so to help avoid hurting their users' experience, this mapping from *updated* to *published* was temporarily introduced to give developers time to update their software, and to give users time to upgrade.

This mapping is temporary and will be removed in a future version of feedparser.

Comes from

- /atom03:feed/atom03:modified
- /atom10:feed/atom10:updated
- /rdf:RDF/rdf:channel/dc:date
- /rdf:RDF/rdf:channel/dcterms:modified
- /rss/channel/dc:date

See also:

- *feed.updated_parsed*

feed.updated_parsed

The date the feed was last updated, as a standard **Python** 9-tuple.

Note: As of version 5.1.1, if this key doesn't exist but `feed.published_parsed` does, the value of `feed.published_parsed` will be returned.

In the past the RSS pubDate element was stored in *updated*, but this incorrect behavior was reported in issue 310. However, developers may have come to rely on this incorrect behavior – as was reported in issue 328 – so to help avoid hurting their users' experience, this mapping from *updated_parsed* to *published_parsed* was temporarily introduced to give developers time to update their software, and to give users time to upgrade.

This mapping is temporary and will be removed in a future version of feedparser.

Comes from

- /atom03:feed/atom03:modified
- /atom10:feed/atom10:updated
- /rdf:RDF/rdf:channel/dc:date
- /rdf:RDF/rdf:channel/dcterms:modified
- /rss/channel/dc:date

See also:

- *feed.updated*

headers

A dictionary of all the HTTP headers received from the web server when retrieving the feed.

Tip: `headers` will only be present if the feed was retrieved from a web server. If the feed was parsed from a local file or from a string in memory, `headers` will not be present.

---

href

The final URL of the feed that was parsed.

If the feed was redirected from the original requested address, `href` will contain the final (redirected) address.

Tip: `href` will only be present if the feed was retrieved from a web server. If the feed was parsed from a local file or from a string in memory, `href` will not be present.

---

modified

The last-modified date of the feed, as specified in the HTTP headers.

The purpose of `modified` is explained more fully in *ETag and Last-Modified Headers*.

Tip: `modified` will only be present if the feed was retrieved from a web server, and only if the web server provided a Last-Modified HTTP header for the feed. If the feed was parsed from a local file or from a string in memory, `modified` will not be present.

---

namespaces

A dictionary of all XML namespaces defined in the feed, as `{prefix: namespaceURI}`.

Note: The prefixes listed in the `namespaces` dictionary may not match the prefixes defined in the original feed. See *Namespace Handling* for more details.

Tip: This element always exists, although it may be an empty dictionary if the feed does not define any namespaces (such as an RSS 2.0 feed with no extensions).

status

The HTTP status code that was returned by the web server when the feed was fetched.

If the feed was redirected from its original URL, `status` will contain the redirect status code, not the final status code.

If `status` is `301`, the feed was permanently redirected to a new URL. Clients should update their address book to request the new URL from now on.

If `status` is `410`, the feed is gone. Clients should stop polling the feed.

Tip: `status` will only be present if the feed was retrieved from a web server. If the feed was parsed from a local file or from a string in memory, `status` will not be present.

version

The format and version of the feed.

Here is the complete list of known feed types and versions that may be returned in `version`:

`atom`	Atom (unknown or unrecognized version)
`atom01`	Atom 0.1
`atom02`	Atom 0.2
`atom03`	Atom 0.3
`atom10`	Atom 1.0
`cdf`	CDF
`rss`	RSS (unknown or unrecognized version)
`rss090`	RSS 0.90
`rss091n`	Netscape RSS 0.91
`rss091u`	Userland RSS 0.91
`rss092`	RSS 0.92
`rss093`	RSS 0.93
`rss094`	RSS 0.94 (no accurate specification is known to exist)
`rss10`	RSS 1.0
`rss20`	RSS 2.0

If the feed type is completely unknown, `version` will be an empty string.

Tip: This element always exists, although it may be an empty string if the version can not be determined.

See also:

The Myth of RSS compatibility Mark Pilgrim's excellent analysis of the extraordinary variety of incompatibilities each version of "RSS" introduced.

CHAPTER 8

Documentation license

Redistribution and use in source (Sphinx ReST) and "compiled" forms (HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code (Sphinx ReST) must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in compiled form (converted to HTML, PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Index